Claudia Alves

PHP AND MYSQL

PHP Programming and MySQL
For Beginners

1st Edition

COMMUNITY SERVICE

2020

PHP and MySQL

PHP Programming and MySQL For Beginners

Claudia Alves

1st edition

2020

---

introduction

PHP and SQL are two important elements of many Internet applications. PHP is one of the programming languages most commonly used for creating websites. It is ideal for creating dynamic content. These allow the pages to be individually adapted to the user or to the context. The data used to create the content is usually stored in a database. This offers a very efficient and well-structured system for data storage. A database language is used to set up a database, to store individual data in it or to call it up. The SQL language is particularly popular for web applications. The contents of countless web applications are stored in SQL databases. PHP is characterized, among other things, by a broad database support. Countless functions are available, especially for SQL databases. It is precisely this good interaction between programming language and database connection that makes PHP a very powerful tool for many web applications. As a result, PHP and SQL play a prominent role in the design of Internet pages. This book provides an introduction to the use of PHP and SQL. The readers first learn to write simple programs in PHP. These are gradually expanded to include important functions. Later, the functionality of SQL databases is explained and the readers learn how they can be integrated into a PHP program. At the end there is an application example that shows the interaction of PHP and SQL makes clear.

PHP - what is it anyway?

The first version of PHP was introduced in 1995. The Danish programmer Rasmus Lerdorf had created some scripts for this in the Perl programming language. The starting point for this work inventory is that he wanted to log the hits on his online resume. This eventually resulted in an extensive script collection that Lerdorf used to design his personal website. So he gave it the name Personal Home Page Tools –PHP. Of these origins, however, little remains apart from the name. For the second version, which appeared only a short time later, Lerdorf revised the concept considerably. Instead of programming this version in Perl, he now programmed it in C. To this day, all other versions have also been developed in C. In 1997 the development team changed. Lerdorf was still involved in this project, but the two Israeli programmers Andi Gutmans and Zeev Suraski now took on the main responsibility. The new development team also gave the acronym PHP a new meaning, which has since stood for PHP: Hypertext Preprocessor. The programmers wanted to avoid creating the impression that the programming language was only suitable for personal projects. The big breakthrough came with the PHP 4 version. At that time, the World Wide Web was growing at a remarkable pace and a suitable programming language was required for the creation of dynamic content. Since PHP specializes in precisely this task, it has established itself as the standard in this area and has been the basis of countless websites since then.

With the following version, the developers made PHP an object-oriented programming language. What this means exactly is explained in more detail later in the book. At this point it should only be mentioned that this is one of the foundations of modern programming techniques. This innovation also contributed to the fact that PHP was able to expand its pioneering position in the area of programming web applications. After development of version 6 was discontinued, PHP 7 was released in December 2016. This version brought numerous technical optimizations with it, so that it works much faster and also less Requires disk space. PHP is preinstalled on almost all web servers. This language is therefore ideally suited for dynamic websites programmed on the server side. A study came to the result that around 83 percent of all Internet sites used this technology in 2017 (W3Techs: retrieved from https://w3techs.com/technologies/overview/programming_language/all am14. 03.2018). This value clearly shows the importance of PHP for the development of the Internet.

Static and dynamic websites

In the previous sections it was mentioned that PHP is used to create dynamic websites. To get a better idea of what is possible with PHP, let's explain what it is all about. When the World Wide Web came into being in 1989, simple HTML pages were used for this. This is a very simple markup language that gives the content of a website a structure. However, as long as no manual adjustments are made, these contents always remain the same. For this reason, pure HTML pages are called static.

However, as the Internet evolved, these static pages were no longer enough. In many applications it was necessary to repeatedly adapt the content that is output to the user or to the current conditions. A simple example of this is the playback of e-mails in a web browser. If only the same pages are displayed here, no meaningful communication is possible. In an online shop, it is also necessary to adapt the display to the articles selected by the customer. Even for news portals that update their reports every few minutes, it would be very time-consuming to create a new HTML page for it each time. Therefore, programs are used here with which the administrators can change the content quickly and easily. With PHP it is possible to program all these functions. When designing dynamic websites it is possible to program them on the client or server side. In the case of client-side applications, the server delivers the complete program code. The browser then executes it. This makes it possible, for example, to insert the current time on the page, to provide the user with a computer for a specific task, or to check that the entries in a form are correct. For example, JavaScript is suitable for these tasks. However, PHP is a server-side language. The server is already executing the program. He then delivers the finished result to the user. This can then no longer be changed. This method is particularly useful when the information from a database is necessary for the content. Since this is usually also saved on the server, a mutual implementation is advisable in these cases.

1.3 Applications for PHP

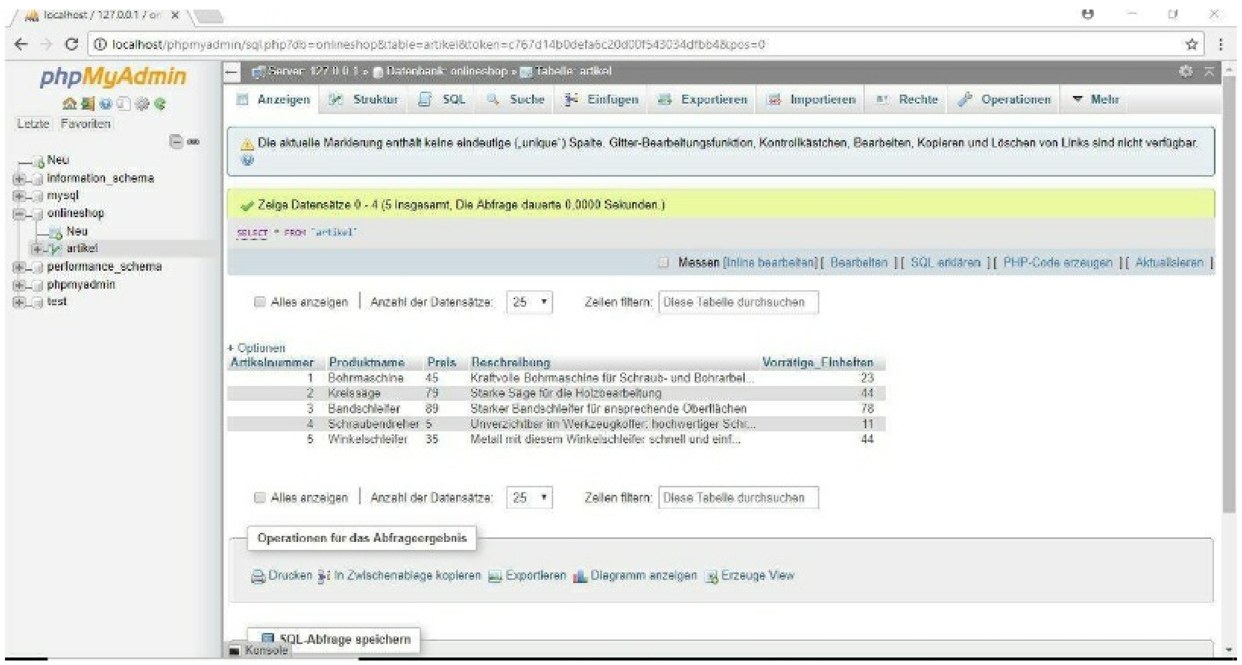
Since PHP is the programming language most commonly used for server-side programming of Internet applications

comes, there are innumerable programs that take advantage of this. This is not just about individual software that has been programmed for a very special web application and is only used for this. There are also countless other programs that provide functions that are useful for many websites. Users can download these and install and run them on their web server. Much of it is available free of charge because it is free software. Other products, however, require a fee. PHP is often used for blog software, for example. Popular blog systems such as Wordpress and Serendipity are programmed in this language. PHP is also a leader in the area of content management systems (CMS). Examples are Joomla, TYPO3 and Drupal. There are also shop systems such as Magento, forum software such as phpBB, customer relationship management systems such as SugarCRM and many other programs. This small list not only makes it clear how many applications were written in PHP. In addition, they show an important field of activity for PHP programmers. In most cases you are allowed to adapt the source code to your own needs. A frequent task is therefore to modify small details of these programs so that they correspond exactly to the wishes of the client. In addition, it is usually possible to expand the functionality of the software with small additional programs - so-called plug-ins. This is also an important task for PHP programmers.

1.4 SQL databases for internet applications

In addition to the PHP programming language, this book also covers SQL databases. These play a very important role in storing data for Internet applications. For this reason, it makes sense to briefly go into what a database is and which one

Possible applications it offers and what role the database language SQL plays in this. A database always consists of two different parts. On the one hand, there is the actual data. These are located on a corresponding memory and the users can call up, change or delete them. This is where the database management system (DBMS) comes into play. This is software that takes on all tasks related to the administration of the relevant data. The DBMS has to structure the data. There are different models for this. SQL databases are always relational databases. To put it simply, these are shown in tables. If, for example, an online shop stores product data in a database, then several entries are necessary for each individual article - the article number, the price, a description, the number of units in stock and some other information. Since all of this data relates to the same product, there is a link between them. In the table this is clear from the fact that they are on the same line. The relational database model is not the only way to structure the entries. In practice, however, it is used in almost all applications.



Die aktuelle Markierung enthält keine eindeutige (‚unique') Spalte. Gitter-Bearbeitungsfunktion, Kontrollkästchen, Bearbeiten, Kopieren und Löschen von Links sind nicht verfügbar.

Zeige Datensätze 0 - 4 (5 insgesamt, Die Abfrage dauerte 0,0000 Sekunden.)

```
SELECT * FROM `artikel`
```

Artikelnummer	Produktname	Preis	Beschreibung	Vorrätige_Einheiten
1	Bohrmaschine	45	Kraftvolle Bohrmaschine für Schraub- und Bohrarbei...	23
2	Kreissäge	79	Starke Säge für die Holzbearbeitung	44
3	Bandschleifer	89	Starker Bandschleifer für ansprechende Oberflächen	78
4	Schraubendreher	5	Unverzichtbar im Werkzeugkoffer: hochwertiger Schr...	11
5	Winkelschleifer	35	Metall mit diesem Winkelschleifer schnell und einf...	44

The DBMS also regulates access to the data. It has to create new tables, add, delete or change entries. Also processed queries to inform the user about the contents of the individual fields. It also has to control permissions. With every request it is necessary to check whether the user is allowed to carry out the corresponding action. The database language SQL is used to communicate with the DBMS. Officially, this is a proper name without any further meaning. Nevertheless, the abbreviation can be traced back to the term StructuredQuery Language - structured query language in German. This shows what the main task of SQL is: to query data from the database. This language contains all the necessary commands to request, insert, delete or change data. It is possible to enter the SQL commands directly via the DBMS user interface. In most cases, however, they are integrated into a PHP program in order to enable an automatic query.

Chapter 2

Preparatory measures for programming with PHP

The previous chapter identified the most important topics that this book will deal with. Now is the time to start the practical work and write your first programs in PHP. However, some preparatory measures are necessary first. Without this it is not possible to start programming.

2.1 HTML

an important basis for programs in PHP In the introduction it was shown that PHP is used for the creation of dynamic Internet pages. Static pages, on the other hand, use HTML. The following section deals with exactly this markup language. However, this only represents a contradiction at first glance. Because almost every website that is created with PHP also uses HTML code. The connection is that the output of a PHP program is normally in HTML. When the server executes the appropriate program, it generates an ordinary HTML page. It then forwards this to the user's browser, which then displays it like a static page. The dynamic element here lies in the execution on the server, in which individually adapted pages are created. All of the following steps are exactly the same as for displaying a traditional Internet page. It follows that every PHP programmer must have a very good knowledge of HTML . In order to write a program whose output is to take place in HTML, it is of course necessary to know exactly the structures and functions of this markup language. In addition, it is common to only program individual elements of a website in PHP. This means that a normal HTML page is created first. PHP scripts are only inserted in those places where dynamic content is required in order to insert the corresponding information. Readers who have no previous knowledge of HTML should therefore acquire these prior to further reading. In this series, a book has also appeared on this topic, which is excellently suited for this purpose. This explains the design of Internet pages with HTML from the ground up. A detailed explanation on this topic is not possible at this point. Nevertheless, the most important basics should be briefly presented for repetition. A website actually consists of ordinary text modules. With the help of tags, however, various functions are assigned to them. This makes it possible, for example, to mark headings or paragraphs or to design certain text areas in bold or italic. In addition, these tags are suitable for lists and tables. Tags are always in angle brackets: <> . This contains a combination of letters that makes the function clear - for example, h1 for the main heading, p for a paragraph or i for italics. An example of an HTML tag would be <h1> . Most HTML tags need to be opened and closed. The slash is used for the final day. In order to close the above example, it would be necessary to insert the tag </h1> . The following example illustrates the use of HTML tags:

```
<h1> This is the heading </h1>
<p> Now follows a small paragraph. This one also uses
<i> Italics </i>. At the end it is necessary to close the paragraph with a corresponding day.

</p>
```

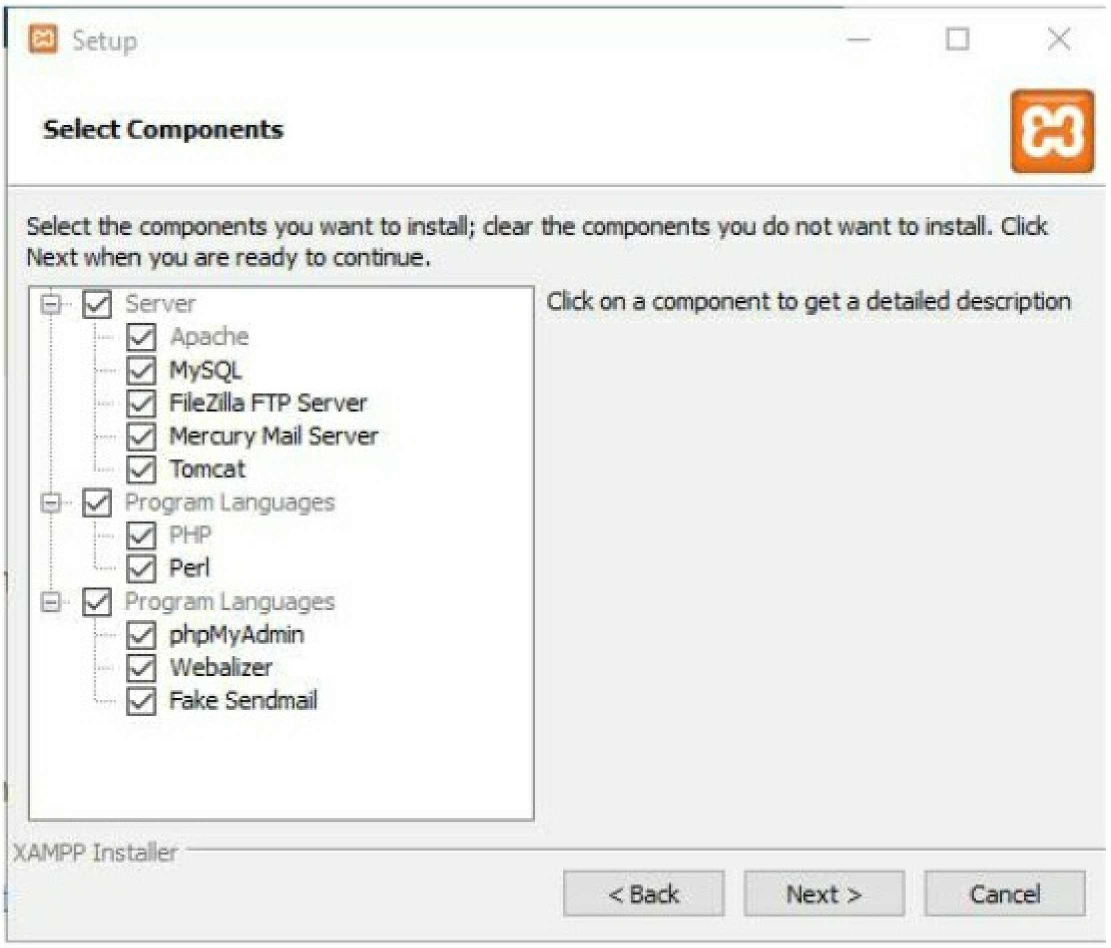
The actual page content is in the body tag (<body>). Before that, a head tag (<head>) is usually used. This contains further information on the HTML version used, the page title, a brief description and other important information. The entire document is in HTML tags (<html>). The structure of the HTML document looks like this:

```
<html>
```

```
   <head> Here you can find further information about the document.
    </head>

<body> This is where the actual page content is located.
</body>
</html>
```

2.2 Web server software for running a PHP program

PHP programs are scripts. This means that the code is not compiled and is therefore not available as an executable file. Instead, an interpreter is required. This reads in the program, which is available as pure text, and implements the functions. As mentioned several times in the previous sections, PHP programs are executed on a web server. Common server software has a PHP interpreter as standard, so using these programs is not a problem. However, many people who are just getting started with PHP do not have access to a web server. For this reason, another solution is necessary here. There are freely available programs that install a web server on the home PC. These not only contain an interpreter for PHP. It is also possible to run Perl with it.
It also includes MySQL - one of the most widely used database management systems for Internet applications. If database applications are later inserted into the programs, the execution is therefore also possible without any problems. The best-known software in this area is called XAMPP. This is free software, so there are no license fees for its use. There are versions for Windows, Linux, MacOS, and a few more operating systems. So it shouldn't be a problem to find a suitable version and install it. Current versions for various operating systems are available for download from the official website at https://www.apachefriends.org/de/index.html. The installation of the software is very easy. The installation assistant automatically guides the user through the set-up. In one of the installation steps, the wizard asks the user which functions should be installed.


Setup

Select Components

Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

- [✓] Server
  - [✓] Apache
  - [✓] MySQL
  - [✓] FileZilla FTP Server
  - [✓] Mercury Mail Server
  - [✓] Tomcat
- [✓] Program Languages
  - [✓] PHP
  - [✓] Perl
- [✓] Program Languages
  - [✓] phpMyAdmin
  - [✓] Webalizer
  - [✓] Fake Sendmail

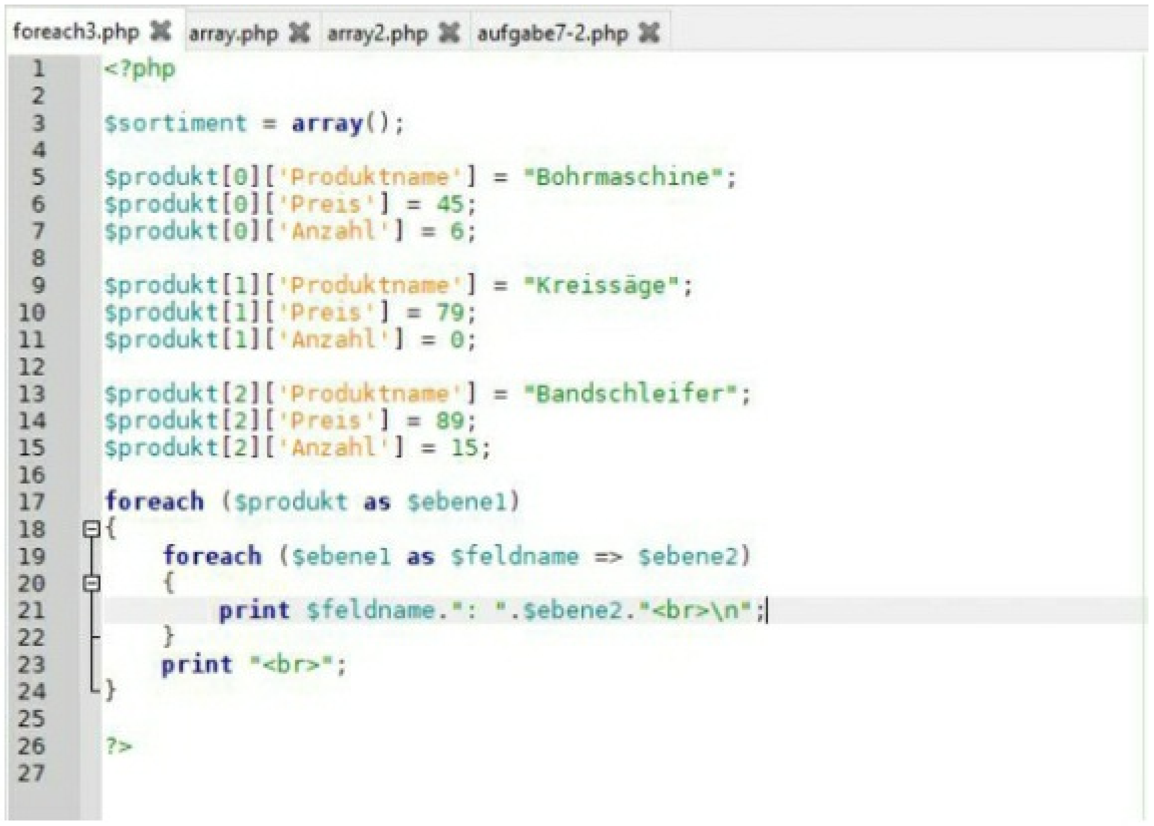
Click on a component to get a detailed description

XAMPP Installer

In principle it is possible to use the standard settings. For this book, however, only the functions Apache web server, PHP, MySQL and phpMyAdmin are required. It is therefore also possible to omit the installation of the other functions so that the program takes up less memory space. As soon as XAMPP is installed on the computer, the necessary software is available to run the PHP programs that are created in the course of this manual.

2.3 Install a suitable text editor

A PHP program consists of pure text. A suitable program is required to write and save this. It is important to note that common word processing programs such as Word or LibreOffice Writer are not suitable for this. In addition to the actual text, these save a lot more information about the layout and formatting of the text. These additional components mean that they cannot be used to create PHP programs. A text editor is required for this task. This saves the characters as pure ASCII code. In this way, no additional information is contained that would interfere with the execution of the program. Windows is already equipped with a text editor. The Microsoft Editor - also known as Notepad - offers the necessary functions. However, this is an extremely simple implementation that is only suitable for the first few programs. There are also many other editors that support users in programming with numerous additional functions. These mark, for example, certain functional elements of the program with different colors or ensure that they are automatically indented when functions are created. This makes the code much clearer and makes programming easier - especially with very complex programs. Some editors also point out syntax errors. In this way you prevent errors as they arise. It is therefore advisable to use a text editor with a slightly higher range of functions. Linux users usually don't have to worry about it. Most Linux distributions are already equipped with suitable software. For example, Ubuntu uses edit and KDE uses the text editor Kate. If no suitable program has yet been installed, there is a large selection of free software to choose from. The page https://de.wikipedia.org/wiki/Liste_von_Texteditoreng provides an overview of the various options. In principle, it is possible to use any of the text editors listed - provided that it is available for the operating system used and that the Programming in PHP supported. At this point the Geany program will be presented as an example. This Editor is ideal for PHP and versions for all common operating systems are available. This program is available for download at https://www.geany.org/Download/Releases. All you have to do is download the version for the operating system you are using and then run the installation wizard.



```php
<?php

$sortiment = array();

$produkt[0]['Produktname'] = "Bohrmaschine";
$produkt[0]['Preis'] = 45;
$produkt[0]['Anzahl'] = 6;

$produkt[1]['Produktname'] = "Kreissäge";
$produkt[1]['Preis'] = 79;
$produkt[1]['Anzahl'] = 0;

$produkt[2]['Produktname'] = "Bandschleifer";
$produkt[2]['Preis'] = 89;
$produkt[2]['Anzahl'] = 15;

foreach ($produkt as $ebene1)
{
    foreach ($ebene1 as $feldname => $ebene2)
    {
        print $feldname.": ".$ebene2."<br>\n";
    }
    print "<br>";
}

?>
```

Chapter 3

The first program with PHP design

After all the preparatory measures have been completed, it is now time to write your first own program. With this first application, of course, very simple structures are used to make it easier to get started. This chapter is used to get to know the basic design of a PHP program and to get a feel for the creation. In addition, the readers get to know the essential components of the syntax of a PHP program.

3.1 Make PHP scripts recognizable in the text

When a user calls up a PHP document with a browser, the web server must first execute the appropriate PHP program in order to generate the desired output. However, this requires that the text be recognized as a PHP program. This is the only way to ensure proper implementation. In order to convey this to the web server, a suitable file extension is required. PHP programs usually end in .php, but there are other options, such as .php3 and .phtml. Many web servers also support these endings. Since they are no longer in use, however, it is advisable to stop using them. In the previous chapter it was already stated that websites created with PHP often consist largely of ordinary HTML. PHP scripts are only used in small areas where dynamic functions are required. This mixture means that the web server cannot use the entire code of the document as a PHP program
has to interpret, but only parts of it. To enable an exact separation between the individual areas, it is necessary to mark the PHP scripts in the text. So the interpreter knows where to use it. The server

reproduces the remaining sections unchanged. Tags are used for this task. The <? Php tag must appear at the point where the relevant section begins . Then the script follows. At the end there is the final day ?>. In addition to this standard method, there are two other options. It is not advisable for beginners to use this, as the server configuration often has to be adjusted. This can easily lead to errors. However, because PHP programmers often come into contact with code that other people have created, they may encounter other tags in the process. It is therefore important to know these as well. As alternative awards, it is possible to use the combinations <? ?> and <%%> .

3.2 Write the first program

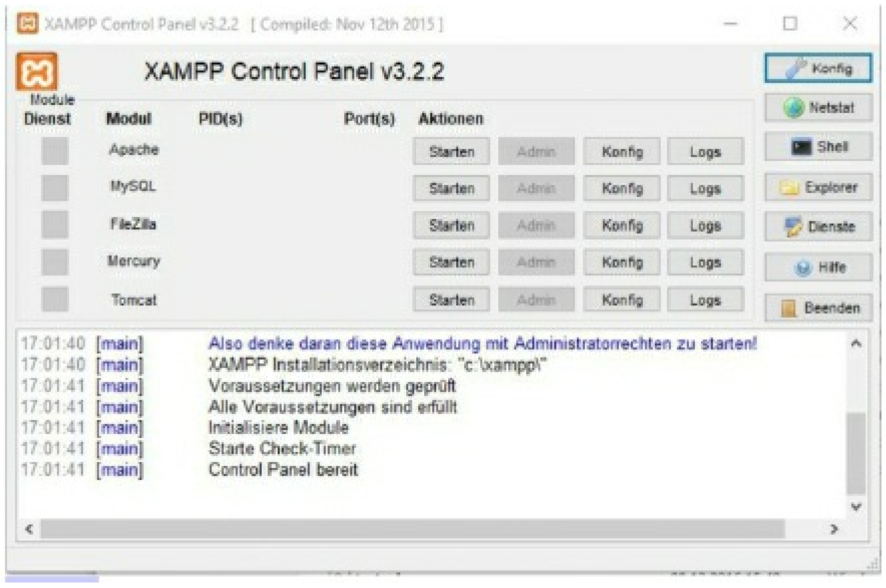
Now is the time to write your first program. For this purpose it is necessary to first open the text editor in order to enter the corresponding command lines. Since this will be a small PHP script, it is necessary to identify it in the code. The tags described in the previous section are used for this purpose: <? Php to open the script and ?> At the end. It makes sense to insert both tags at the beginning. This is helpful because many beginners forget the ending. This common mistake sometimes leads to the whole program not working. In the first step the code looks like this: <? Php?>
The actual functions should now be inserted in the space between these two tags. To start with, it makes sense to choose a simple command - for example, print . This generates an output on the screen. It is possible to use any words. For the first program, the output should read “My first program ” . The text that should appear must always be in quotation marks. This is particularly important when numbers are to be output. For example, if the print command is followed by “2 + 3 ” , the program outputs this value as a character string. Exactly the same characters appear on the screen: 2 + 3. However, if this expression is inserted without quotation marks, the program treats it as a mathematical operation. This means that it first calculates the value of this term: 2 + 3 = 5. Only the number 5 appears as output. In order to avoid errors, it is very important to always pay attention to the correct placement of the quotation marks. A semicolon must be placed after each PHP command. This ends the function and allows the program to continue with the next task. A simple line break is not sufficient for this. If the semicolon is missing, the program also interprets the following line as part of the command. This can lead to syntax errors and prevent the program from running. If the complete command is now inserted between the two existing tags, the result is the following code:

```
<? phpprint "My first program";?>
```

The first program is now ready. The way it works is extremely simple. When the server executes this code, only the title “My first program ” appears on the screen.
Now it is only necessary to save the written code as a file. The file name can be freely chosen. It is only important to use the ending .php. This is how the server executing the program recognizes that this is a PHP file. With some text editors it is necessary to remove the standard option - saving as a txt. Document - when saving. In order to keep an overview at all times, it is advisable to select self-explanatory names, for example: first_program.php.

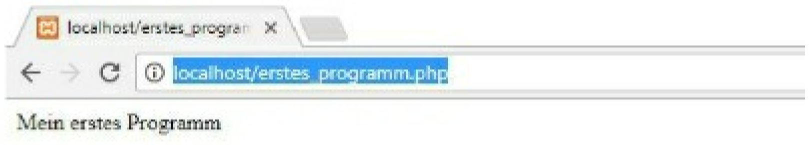
3.3 Getting the program up and running

Now the first program is already finished, but the result is not yet visible on the screen. The users can only view the text that they have written themselves in the text editor. However, to check the functionality, it is necessary to run the program. The XAMPP software is used for this. Since the installation has already been explained in one of the previous chapters, it should already be on the reader's PC. In order to start the software, it is necessary to execute the xampp-control file. This is located in the folder that was selected when installing XAMPP. After double-clicking on this file, the following window appears:



```
17:01:40  [main]   Also denke daran diese Anwendung mit Administratorrechten zu starten!
17:01:40  [main]   XAMPP Installationsverzeichnis: "c:\xampp\"
17:01:41  [main]   Voraussetzungen werden geprüft
17:01:41  [main]   Alle Voraussetzungen sind erfüllt
17:01:41  [main]   Initialisiere Module
17:01:41  [main]   Starte Check-Timer
17:01:41  [main]   Control Panel bereit
```

Now it is necessary to click on the “Start ” button in the line marked “Apache ” . This results in the web server running. It is now possible to close or minimize the corresponding window. The process continues to take place in the background. In the next step it is necessary to move the PHP program to the root folder of XAMPP. This is the “htdocs ” folder . If further programs are added later, it makes sense to add subdirectories. This improves the overview considerably. If the program has been placed in the correct directory, it is necessary to call it up via a web browser. For this it is necessary to enter the following path in the address bar: http: //localhost/erstes_programm.php. If the reader has saved the file in a subdirectory, the corresponding folder must be in the
Path to be inserted. The result of the first program then appears in the browser.



```
Mein erstes Programm
```

It often happens that the programmer has a first draft carried out, then looks at the result in the browser and then makes some changes. If he now wants to check the new version, he must first save it in the text editor. This is possible as usual via the menu bar. However, since this process has to be repeated countless times when programming with PHP, it is important to the reader to use the key combination Ctrl + S for this. A lot of time can be saved in this way. Then it is necessary to refresh the page in the browser. This is possible with the circular arrow next to the address bar or by pressing the F5 key.

3.4 Exercise: Writing a simple program in PHP

In the following section, what you have learned is to be put into practice, so there are two exercises here that you should solve on your own. The necessary knowledge was imparted in the previous sections. After the two exercises are the solutions
inserted to control your results. However, there are several possible solutions for many tasks. It is therefore always advisable to call up the control program in the browser in order to check for yourself whether it meets all requirements. The suggested solutions shown here are for comparison and are also useful if you are stuck with the programs on your own.

1. Create a program that welcomes visitors to your home page.

Show three different ways to represent the number 8 as output from a PHP program and explain the differences. Solutions:

1.

<? phpprint "Welcome to my homepage!";?>

This program is almost identical to the example that was described in the text. The only difference is that it uses different text for rendering. In addition to the two tags for identifying the PHP script, it is important to pay attention to the quotation marks and the semicolon. <? phpprint "8";?> In this case the output of the print function uses the number 8 as a string. That's because it's in quotation marks. No mathematical operations are possible with this. <? php
print 8;?> In the second example, the output is completely identical. However, the program does not treat the value as a character string, but as a number. Therefore, in a further step here it would be possible to carry out further calculations. <? phpprint 2 * 4;?> In the last example, the number 8 is not inserted directly into the print function. It is the result of the term 2 * 4. Since this expression is not in quotes, PHP treats it as a mathematical operation and automatically calculates the result.

Chapter 4

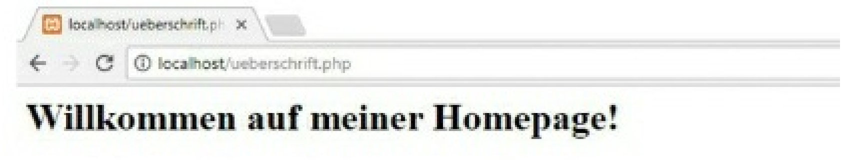
Combine PHP and HTML

When designing websites with PHP, there is a very close connection between the programming language and the HTML code. The goal in this application area is always to use PHP to output correct HTML pages. Therefore, before creating a program with PHP, it is always a good idea to consider what the output should look like. It is therefore important to design the relevant page in HTML. Then it is necessary to mark the places that use dynamic content. Then these areas have to be replaced by a PHP script. These small program pieces should now generate content that is adapted to the respective context. However, they are intended to perform exactly the same function as the corresponding elements in the original HTML page. It is therefore necessary for PHP programmers to take this close connection between PHP and HTML into account. On the one hand, it is important to integrate the PHP scripts sensibly into the HTML page. On the other hand, it is necessary that every single one of these scripts outputs valid HTML code. The following sections present the way in which it is possible to combine PHP and HTML.

4.1 Outputting HTML code with PHP

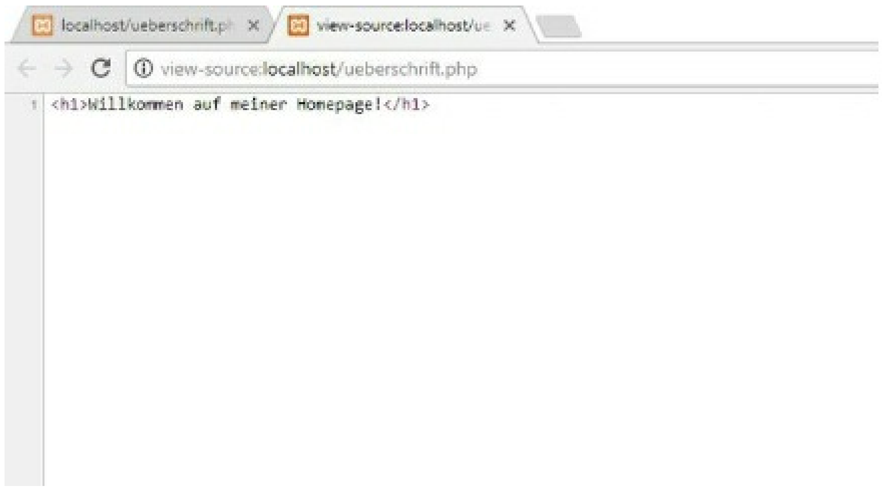
The first step is to show how it is possible to generate HTML code with PHP. This task is very simple in principle. The usual print function is used for this. However, no simple text is used here. This must be with HTML
Tags. For example, if a heading is to be marked as such, it is only necessary to include the corresponding <h1> tags in the print function:

```
<? phpprint "<h1> Welcome to my homepage! </h1>";?>
```



Willkommen auf meiner Homepage!

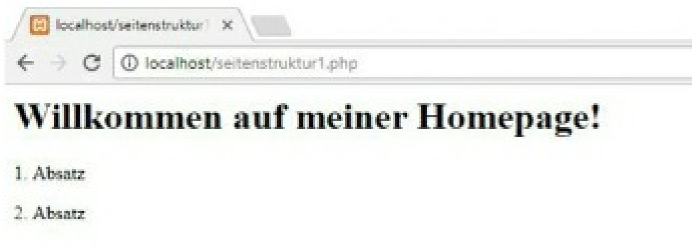
Since this is a character string, it is of course necessary to use quotation marks. This is particularly important here, since the angle brackets also have special functions in PHP. If these are not clearly identified as part of a character string, this leads to syntax errors so that the server cannot execute the program. In this example, the server is running the program and returns the output <h1> Welcome to my homepage </h1> and transmits this code to the browser. This then only shows the desired heading: Welcome to my homepage. However, it is large and bold, which shows that the browser has recognized the text as a heading and is displaying it accordingly. This can also be done through the
Check source code display. Therefore it is necessary to click with derrechten click on the appropriate page and "View source code then dieOption " select. The text then appears with the appropriate tags.


localhost/ueberschrift.ph
view-source:localhost/ue
view-source:localhost/ueberschrift.php

```
<h1>Willkommen auf meiner Homepage!</h1>
```

Usually, a website doesn't just consist of a single heading. For this reason, it is usually necessary to have several elements output with PHP. In principle it is possible to put all different elements in the same print command. That way, however, the code becomes very cluttered. It is therefore advisable to use a separate print command for each individual area :

<? phpprint "<h1> Welcome to my homepage! </h1>"; print "<p> 1st paragraph </p>"; print "<p> 2nd paragraph </p>";?>

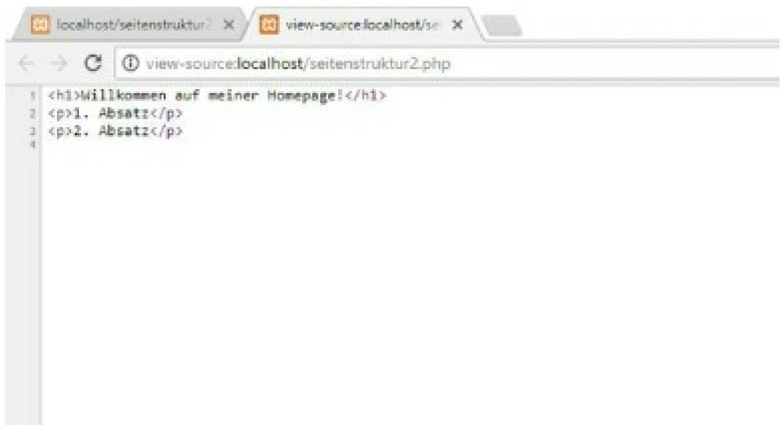


Willkommen auf meiner Homepage!

1. Absatz

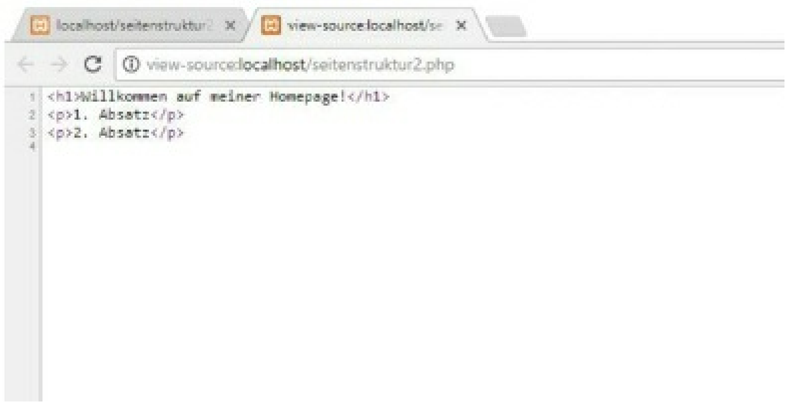
2. Absatz

When the program is now executed, the heading will first appear on the page in large characters. Below that, the first and second paragraphs follow with some space and in much smaller font. The output is therefore exactly as desired. However, it makes sense to take another look at the source code of the relevant page. It is noticeable that all the components listed on the page are on the same line. With a small program like in this example this is certainly not a big problem. With large pages, however, the source code becomes extremely confusing. This has extreme disadvantages when programming in PHP. Often times, when the programmer makes a small mistake and the page doesn't display correctly, it is very helpful to examine the source code to see what the problem is. However, if all of the code appears on a single line, it is almost impossible to find the bug.


localhost/seitenstruktur2
view-source:localhost/se
view-source:localhost/seitenstruktur2.php

```
<h1>Willkommen auf meiner Homepage!</h1>
<p>1. Absatz</p>
<p>2. Absatz</p>
```

For this reason, it is recommended to insert the command \n after each HTML element. This has no effect on the presentation of the page. However, it does cause a line break in the source code. The finished program looks like this: <? Phpprint "<h1> Welcome to my homepage! </h1> \ n"; print "<p> 1st paragraph </p> \ n"; print "<p> 2 . Paragraph </p> \ n ";?>


localhost/seitenstruktur2
view-source:localhost/se
view-source:localhost/seitenstruktur2.php

```
<h1>Willkommen auf meiner Homepage!</h1>
<p>1. Absatz</p>
<p>2. Absatz</p>
```

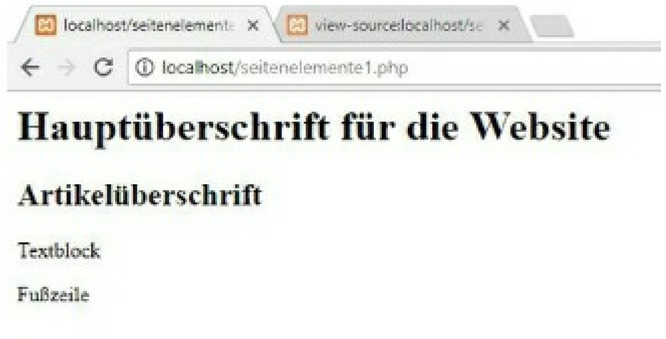
If the source text is called up again after this small change, it appears much better ordered, as each element is on its own line.

4.2 Integrating PHP scripts into an HTML page

The previous section showed that generating HTML with a PHP function is not a problem. It is actually possible to design the entire website in this way. To do this, it would be necessary to create a separate print function for each individual HTML element . However, this would involve a great deal of effort,

which is actually completely unnecessary. This is due to the fact that even with dynamic pages there are many elements across the entire website - or at least for thematically related pages - remain the same. It is therefore possible to make these areas static. All that is necessary is to encode them in HTML. PHP scripts are then only used in those areas in which dynamic content is actually generated.
A small example should clarify this. A website uses the same design for each individual page. This consists of a main heading that remains the same across the entire site. In addition, a footer with the imprint is inserted. Here, too, there are no differences between the individual pages. Only the actual articles that appear on the individual pages should differ. These consist of an article heading and a text block. These elements are to be retrieved from a database. Therefore it is possible to mark up the fixed elements with HTML only and the variable contents with PHP:

```
<h1> Main heading for the website </h1>
<? phpprint "<h2> Article heading </h2> \ n"; print "<p> Text block </p> \ n";?>
<p> Footer </p>
```

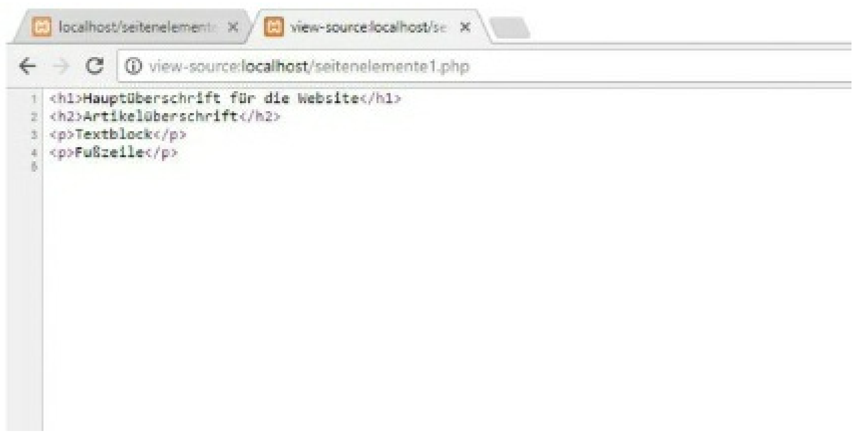


Hauptüberschrift für die Website

Artikelüberschrift

Textblock

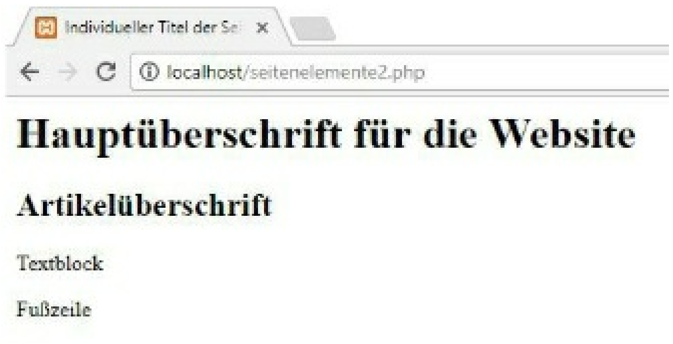
Fußzeile



```
<h1>Hauptüberschrift für die Website</h1>
<h2>Artikelüberschrift</h2>
<p>Textblock</p>
<p>Fußzeile</p>
```

In this example, it should be noted that the article heading and text block areas are only used as placeholders for a database query. How PHP can get this data will be explained in a later chapter. This code example shows that it is very easy to mix HTML and PHP together. Since the PHP code must always be marked with the appropriate tags, the interpreter recognizes exactly when it must execute a corresponding program. It is also possible to integrate several PHP scripts into the document. This is often necessary since static and dynamic elements alternate with larger content. This does not only apply to the body area, in which the actual content is located. Also in the head- area there are many requirements that are common to the entire site. Other parts such as the title and description, however, are individually adapted for each page. The use of PHP scripts for dynamic generation is therefore useful here. A complete page - albeit greatly simplified in the example - could look like this:

```
<html>
<head> <? php print "<title> Individual title of the page </title> \ n"; ?> <? php print "<meta name = \" description \ "content = \" Individual description of the page \ "/> \ n"; ?> <link rel = "stylesheet" href = "Link to stylesheet is the same for everyone" /> </head> <body> <h1> Main heading for the website </h1> <? phpprint "<h2> Article heading </ h2> \ n "; print" <p> Text block </p> \ n ";?> <p> Footer </p> </body> </html>
```

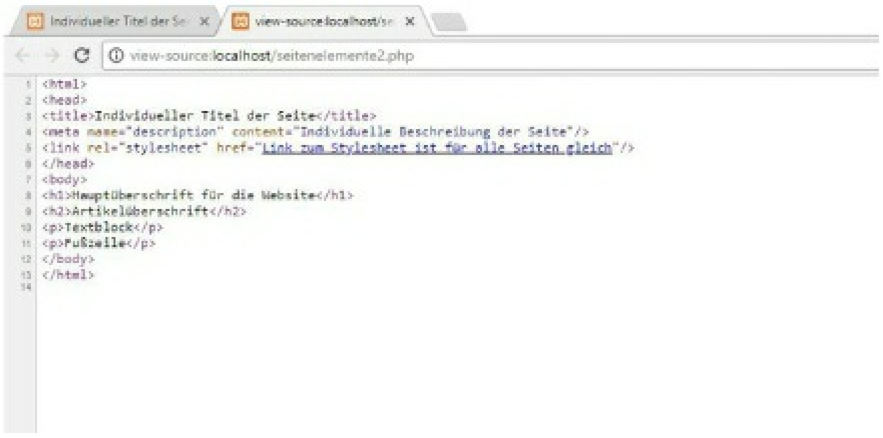


Hauptüberschrift für die Website

Artikelüberschrift

Textblock

Fußzeile


Individueller Titel der Se
view-source:localhost/se
view-source:localhost/seitenelemente2.php

```
<html>
<head>
<title>Individueller Titel der Seite</title>
<meta name="description" content="Individuelle Beschreibung der Seite"/>
<link rel="stylesheet" href="Link zum Stylesheet ist für alle Seiten gleich"/>
</head>
<body>
<h1>Hauptüberschrift für die Website</h1>
<h2>Artikelüberschrift</h2>
<p>Textblock</p>
<p>Fußzeile</p>
</body>
</html>
```

Note: The quotation marks in the meta description are part of the HTML source code. If, however, these are inserted into the PHP script without any further addition, the interpreter assumes that they terminate the input of the `print` function - which also begins with a quotation mark. This leads to syntax errors. Therefore it is necessary to insert a backslash (`\`) before the quotation mark. The interpreter outputs this unchanged in the source code.

Chapter 5

Variables: an important element of PHP programming

With the programs presented so far, some readers may have asked themselves why it was actually necessary to program the code with PHP. The same effect would have been possible with pure HTML code. That would be easier and clearer. This objection is certainly correct. The first program examples were so simple that no real programming language would be required for them. But that changes in this chapter. This introduces variables. These are placeholders that can have different values. Each variable has a unique name that enables access . As an illustrative example, it is possible to imagine the variables as drawers in a dresser. Each of them is labeled with its own name. The variable always remains the same, but the content can change. These drawers can contain very different things - for example, documents, pens or coins. It is similar with variables. There are also different types here. Mostly these are strings or numbers. Various operations can be carried out with the contents - both in the example with the chest of drawers and with variables. Users can move pens from one drawer to another that already has some pens in it. With the documents it is possible to combine them and create a longer text. The possibilities always depend on the type of content. That also applies to the variables. With variables, the numbers
completely different operations are possible than if there are strings in them. The contents of the drawers may change occasionally. At a certain point in time, however, it is always precisely defined. This also applies to the variables. A query of the value gives a clear result at any given point in time. A special case occurs when a variable has not yet been defined. This can be compared to an empty drawer. However, the return value clearly indicates that no content is available yet. Variables represent an important basis for the design of dynamic web pages. Often, only a variable is inserted into the print function. This can result in very different expenses - depending on its value.

5.1 Capture text with one variable

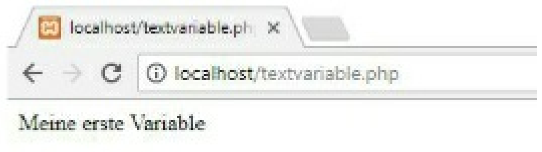
A large part of web pages consists of simple text. With dynamic websites, it is primarily important to insert changing text modules into the pages. Of course, there is also the HTML code. But even this is just text. It is also important to include graphics and multimedia content. However, since the integration takes place via the HTML code, these components must also be

inserted in text form. This property also affects the use of the variables. PHP programmers who devote themselves to the design of Internet pages use variables that contain text particularly often. Mathematical operations are comparatively rare. Since text variables play a particularly important role in programming with PHP, they should be treated first. In the previous section it was stated that variables like a
Drawer are to be understood. This can be shown particularly clearly with text variables. In this case the piece of furniture is in a publishing house. The individual employees write documents and put them in the appropriate drawer. It is possible to remove the previous content and replace it with new texts. The employees can also simply add their documents to the existing ones. Occasionally the editor comes by and puts the relevant content to the press for publication. With text variables it is also possible to exchange, add to or change content. When the desired text has been created, it is possible to output it on the screen using the print function. The following code is required to create a text variable and fill it with content:

```
<? php $ textbaustein = "My first variable";?>
```

A variable is always introduced in PHP with a dollar sign. This is followed by the name of the variable. In principle, the user can choose this freely. He just has to be careful not to assign a name twice. Furthermore, it must not contain any special characters that have a special function in PHP. Once the variable has been introduced, it is immediately filled with content. The equal sign is also used. This is followed by the text that the variable should contain. Unlike some other programming languages, PHP does not need to specify the type of variable. The interpreter automatically recognizes that this is a text variable from the fact that the content is in quotation marks. Variables can even change their type within the program. To display the content of the variable on the screen, it is only necessary to insert it into the `print` function. There is also
The dollar sign is required in front of the name of the variable. In this case, quotation marks must not be used, otherwise the function would display the variable name on the screen. The finished program therefore looks like this:

```
<? php $ textbaustein = "My first variable"; print $ textbaustein;?>
```

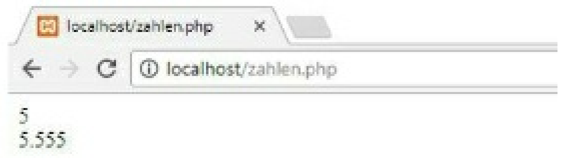


```
Meine erste Variable
```

5.2 Save numbers as variables

In addition to text, variables can also contain numbers. In many computer programs this is the most common use of the variables. Due to the special feature that PHP is mostly used for the design of Internet pages where the text is in the foreground, this application is of somewhat less importance. Nevertheless, it happens again and again in PHP programs that a variable must take on a numerical value. This function is particularly important for the control of loops, which will be explained in a later chapter.
Computer programs often distinguish between many different types of numbers. These can be whole numbers that are referred to as integers. There are also point numbers. Each type of number is limited in its size, so that the memory space required for it is fixed. In many programming languages, it is necessary to specify exactly what kind of number a variable is. The program reserves a memory space of the appropriate size. PHP programmers don't have to worry about that, however. The interpreter takes on this task independently. To assign a numeric value to a variable, it is only necessary to declare it accordingly and to fill it with content. This works almost exactly the same as with text variables:

```
<? php $ integer = 5; $ comma number = 5.555; print $ integer; print "<br>"; print $ comma number;?>
```



```
5
5.555
```

Computer programs often distinguish between many different types of numbers. These can be whole numbers that are referred to as integers. There are also point numbers. Each type of number is limited in its size, so that the memory space required for it is fixed. In many programming languages, it is necessary to specify exactly what kind of number a variable is. The program reserves a memory space of the appropriate size. PHP programmers don't have to worry about that, however. The interpreter takes on this task independently. To assign a numeric value to a variable, it is only necessary to declare it accordingly and to fill it with content. This works almost exactly the same as with text variables:

```
<? php $ integer = 5; $ comma number = 5.555; print $ integer; print "<br>"; print $ comma number;?>
```

The command `print "<br>";` generates the HTML tag `<br>` in the source code. This in turn leads to a line break on the screen. Without this command, both numbers would be without spaces or
Line breaks are output one after the other.

5.3 Boolean variables

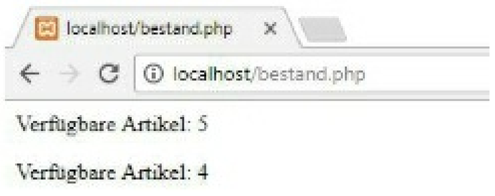
In addition to text and numbers, there is a third type of variable. These are so-called Boolean variables. These are only mentioned briefly here, as they only play a subordinate role in the remainder of this book. For later tasks, however, it is important that the reader knows that this type of variable exists and what it is used for. These variables are named after the English mathematician and logician George Boole. Actually, these are variables that can take on a finite number of values. In PHP - and in all other programming languages - exactly two different states are possible. A Boolean variable can take the values `true` and `false` - in German true and false. This already shows the use of these variables: They serve as a truth value. They are primarily used to specify a condition for functions. Only if this condition is fulfilled - i.e. if the value of the corresponding variable is true - the corresponding function is executed.

5.4 Operations with variables

The previous example programs that use variables would actually get by without these placeholders. If the appropriate text or numbers are inserted directly into the print function, the output is exactly the same. However, the use of variables is of great importance. Many functions are impossible without them, because they can be used to perform many different operations. These change the value of the variable and thus offer many new options for designing the program.
A very simple alternative is to change the content of a variable by assigning a new value:

```
<? php $ inventory = 5; print "<p> Available articles:"; print $ inventory; print "</p> \ n"; $ inventory
= 4; print "<p> Available articles:"; print $ inventory ; print "</p> \ n";?>
```

This program indicates the inventory for a specific article and displays it on the screen. After that, one of these items will be sold, so only four units are available. The new value is then assigned to the variable and then displayed again to make the results clear.

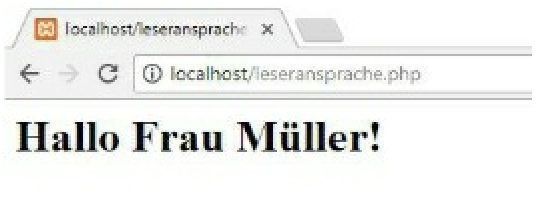


```
Verfügbare Artikel: 5

Verfügbare Artikel: 4
```

This program shows that a variable can easily be assigned a new value. In practice, however, it is not very smart to use specific values for this. Because if another item is sold, this function cannot be used again because
the new value would have to be adjusted to 3 for this. Therefore, it makes more sense to perform a math operation. This can be used with any starting value and can therefore be used for every sale, regardless of how many items are currently available: `<? Php $ stock = 5; print "<p> Available items:"; print $ stock; print " </p> \ n "; $ stock = $ stock-1; print" <p> Available articles: "; print $ stock; print" </p> \ n ";?>`

The display on the screen is exact the same as the previous program. However, it is now possible to change the output value of the variable and set it to 6, for example. In this case the program adjusts the values automatically. The variable refers to itself and performs an arithmetic operation with its original value. Many other calculations can be performed in exactly the same way - for example additions, multiplications, or divisions. The following list shows some important mathematical operators : + Additions - Subtraction * Multiplications / Division ** Powers (only from PHP version 5.6) Since many programs need to increase or decrease the value of a variable by one unit, this is why a short form is also available. Instead of the line $ stock = $ stock-1; it is quite simply possible $ consisted--; to insert. The command $ consisted ++; however, increases the value of this variable. This saves a lot of programming time. The point operator is particularly important for editing text. This is used to connect two text variables with one another. This is very useful for composing a text from different variables. If, for example, a personalized address of the visitor is to appear on the page, this can be combined from the salutation and the name:

<? php $ salutation = "woman"; $ name = "Miller"; $ reader address = $ salutation. "". $ name; print " <h1> Hello"; print $ reader address; print "! </h1> \ n ";?>


localhost/leseransprache.php

Hallo Frau Müller!

The variable $ reader address arises from a link between the variables $ address and $ name . It is important that
Please note that there is a space between the two variables in quotation marks and is also connected with the period operator. Without this addition, the program would combine the two words with each other and output them without the space. This example shows another possible application of the dot operator. Three different print functions are required for the simple greeting of the reader . This is relatively cumbersome. It is therefore advisable to combine the various outputs with the dot operator and to summarize them in a single print command: print "<h1> Hello". $ Reader address. "! </h1> \ n"; This command is significantly shorter, but the functionality does not

change.

5.5 Arrays: composite variables

In many cases it happens that variables appear in related groups. An example of this is the assortment of an online shop, which was already used in the introductory explanation of the SQL databases. The item number, product name, price, a brief description and the number of units available are required for each product. Of course, it is possible to use a separate variable for each of these values - for example `$ item number` , `$ product name` , `$ price` , `$ description` and `$ number` . This method, however, makes the processing complicated and confusing. Hence it is prone to failure. Since all of this information relates to the same item, it makes sense to combine them into one unit. This ensures a clear structure. Arrays are used for this, and an array can be thought of as a toolbox. This contains
several different subjects. A different tool - or in a figurative sense a variable - can be stored in each of these compartments. The following code is required to create an array:

```
<? php $ product = array (); $ product [0] = 1; $ product [1] = "drilling machine"; $ product [2] = 45; $
product [3] = "powerful drilling machine for drilling and screwing work" ; $ product [4] = 23;?>
```

In principle, the array is treated like a variable. It is therefore necessary to put the dollar sign in front of the array name. The expression `array ()` tells the interpreter that this is an array. In PHP it is not necessary to enter the number of individual fields. The programmer can therefore add any number of entries. This is done in the following lines. A specific value is assigned to the individual fields. PHP allows you to start with any index number. In addition, these do not have to be continuous. However, it is common practice to assign the index 0 to the first field and then to number the individual fields consecutively. There is a much shorter and more practical method of introducing an array. Instead of the entire expression in the previous code example, a single line is sufficient:

```
<? php $ product = array (1, "Drilling machine", 45, "Powerful drilling machine for drilling and
screwing work", 23);?>
```

In this case, the index number is assigned consecutively and begins with 0. This creates exactly the same array as in the previous example. To check the

results, it is useful to refer to the array
on the screen. For the time being, it is necessary to create a separate print command for each individual field for this purpose :

```
<? php $ product = array (1, "Drilling machine", 45, "Powerful drilling machine for drilling and screwing work", 23); print $ product [0]. "<br> \ n"; print $ product [1]. "<br> \ n"; print $ product [2]. "<br> \ n"; print $ product [3]. "<br> \ n"; print $ product [4]. "<br> \ n ";?>
```

In this example, the individual fields are accessed via an index number. This has numerous advantages. For example, functions can be created in this way that access the individual fields using mathematical operations. The problem, however, is that these values are not self-explanatory. Of course, when creating the array, the programmer knows that field 0 stands for the item number, field 1 for the product name and field 2 for the price. However, it may be that he forgets this after a while. If he then wants to make changes to the program, he has to
Check carefully which index number stands for which content. To avoid this problem, it is also possible to use self-explanatory terms for it. This shape is called an associative array.

```
<? php $ product = array (); $ product ['item number'] = 1; $ product ['product name'] = "drill"; $ product ['price'] = 45; $ product ['description'] = "Powerful drilling machine for drilling and screwing work"; $ product ['number'] = 23;?>
```

This command can also be summarized in one line:

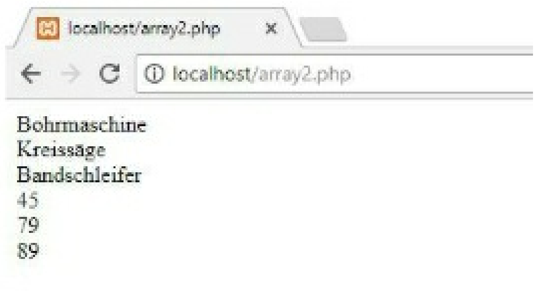
```
$ product = array ('Item number' => 1, 'Product name' => "Drilling machine", 'Price' => 45, 'Description' => "Powerful drilling machine for drilling and screwing work", 'Quantity' => 23);
```

To reflect this array, the output must also be adjusted. The full program is therefore:

```
<? php $ product = array ('Article number' => 1, 'Product name' => "Drilling machine", 'Price' => 45, 'Description' => "Powerful drilling machine for drilling and screwing work", 'Quantity' => 23); print $ product ['item number']. "<br> \ n"; print $ product ['product name']. "<br> \ n"; print $ product ['price']. "<br> \ n "; print $ product ['description']." <br> \ n "; print $ product ['quantity']." <br> \ n ";?
>
```

The text that appears on the screen is exactly the same as in the previous screenshot. In addition, it is possible to create composite arrays. In the example described here, it would be useful to use the to display the entire range of goods. However, since this would be very extensive, this is to be limited in the example to three products, each with three properties - article number, product name and price. To access the individual contents, it is now necessary to enter both indices:

```
<? php $ assortment = array (); $ assortment [0] ['item number'] = 1; $ assortment [0] ['product name'] = "drill"; $ assortment [0] ['price'] = 45 ; $ range [1] ['item number'] = 2; $ range [1] ['product name'] = "circular saw"; $ range [1] ['price'] = 79; $ range [2] ['item number '] = 3; $ assortment [2] [' product name '] = "Belt sander"; $ assortment [2] [' price '] = 89; print $ assortment [0] [' product name ']. "<br> \ n "; print $ assortment [1] ['product name']." <br> \ n "; print $ assortment [2] ['product name']." <br> \ n "; print $ assortment [0] [ 'Price']. "<br> \ n"; print $ range [1] ['Price']. "<br> \ n"; print $ range [2] ['Price']. "<br> \ n ";?>
```



```
Bohrmaschine
Kreissäge
Bandschleifer
45
79
89
```

5.6 Exercise: dealing with variables

Write a program that uses two numeric variables to first output the two values individually. Then multiply both variables and output the result. 2. Write a program that outputs the following lyrics: “Joy, beautiful spark of the gods, daughter of Elysium, we drowned in fire, heavenly ones, your sanctuary.” Each line should be recorded in a variable. Use only a single `print` command for output . 3. Create an array that contains the customer number, first name, and last name of three customers. Then output the corresponding values on the screen. The associated values for each customer should be in one line. Solutions:

1.

```
<? php $ number1 = 5; $ number2 = 7; print "<p> Variable 1:". $ number1. "</p> \ n"; print "<p> Variable 2:". $ number1. "< / p> \ n "; $ result = $ number1 * $ number2; print" <p> The result from ".
```

$ number1." * ". $ number2." is: ". $ result.". </p> \ n ";?> This is just one of several possible solutions. For example , it would also be possible to omit the $ result variable and insert the operation directly into the print command.

```
<? php $ part1 = "Joy, beautiful spark of the gods,";
$ part2 = "Daughter from Elysium,"; $ part3 = "we're entering, drunk on fire,"; $ part4 = "Heavenly,
your sanctuary."; print "<p>". $ part1. "<br> \ n". $ part2. "<br> \ n". $ part3. "<br> \ n". $ part4. "</p>
\ n";?>
```

The individual variables can easily be combined with the dot operator. This is also how you insert the HTML tags for paragraphs and line breaks.

3.

```
<? php $ customer data = array (); $ customer data [0] ['customer number'] = 10001; $ customer data
[0] ['first name'] = "Heinz"; $ customer data [0] ['last name'] = " Mahler "; $ customer data [1]
['customer number'] = 10002; $ customer data [1] ['first name'] =" Eva "; $ customer data [1] ['last
name'] =" Müller "; $ customer data [2 ] ['Customer number'] = 10003; $ customer data [2] ['First
name'] = "Michael"; $ customer data [2] ['Last name'] = "Mayer"; print "Customer number:". $
Customer data [0] ['Customer number']. ", First name:". $ Customer data [0] ['first name']. ", Last
name:". $ Customer data [0] ['last name']. "<br> \ n"; print " Customer number: ". $ Customer data [1]
['customer number'].", First name: ". $ Customer data [1] ['first name'].", Last name: ". $ Customer data
[1] ['last name']." <br> \ n "; print" Customer number: ". $ customer data [2] ['customer number'].", first
name: ". $ customer data [2] ['first name'].", last name: ". $ customer data [ 2] ['Last Name']. "<br> \
n";?>
```

Chapter 6

Decisions through if queries

Computer programs usually contain many areas that should only be executed under certain conditions. There are also numerous examples of this that can be used in a PHP program for an online shop. If the customer has already placed an item in the shopping cart here, it makes sense to display it.

However, if there is no product here, this is not necessary. In this case, the program must carry out a query. Only if the result has a certain value does it carry out the corresponding function. In the example, the program must therefore first query the number of items in the shopping cart. Only if this is greater than zero will it display the shopping cart on the screen. If such a query should be formulated in ordinary language, it could read something like this: “If the shopping cart contains at least one item, it should be displayed on the screen. " The key term" Wenn " - in English" if " - is usually used. Almost all computer programs also use this expression for this. For this reason, they are referred to as an if query.

6.1 The structure of an if-query

An if query always begins with the keyword i f. The interpreter uses this to recognize that it should only execute the next part if a certain condition is met. This condition immediately follows the key phrase. Often it is a mathematical one
Comparison. For example, it is possible to specify the condition that a certain variable must be larger or smaller than a specified value. Only if this is the case will the following part be carried out. Another possibility is to use the Boolean variables presented in the previous chapter. In this case, the part of the program that is in the query is only executed if the variable has assumed the value true . Regardless of which of these two options is used, it is necessary that the condition is in round brackets. The condition is now followed by a curly bracket. It is then possible to insert any number of commands. At the end it is necessary to close this part of the program with a curly bracket. The entire area within the curly brackets is only executed if the condition is met. If this is not the case, the program continues with the first command that comes after the closing curly bracket. The scheme of an if-query looks like this: if (condition) {In this area it is possible to insert any commands.} It is also possible to nest if- queries. In this case, there is another if query in the area between the curly brackets , which is structured according to the same pattern. In this way, highly branched and very detailed structures can be specified. With sophisticated programs it almost always happens that several of these queries are nested within one another.

6.2 Use different comparison operators

The previous section showed that for every if query it is necessary to specify a condition. Sometimes come for this purpose
Boolean variables are used. In this case it is only necessary to put the variable name in the brackets. If the value of the variable is true , the following part is performed. On the other hand , if it is false , the program continues with the next step, but a comparison is often specified with the condition. These can be mathematical comparisons or the content of a specific character string. If you want to check whether the value of a variable corresponds to a given value, it is necessary to insert a double equal sign: if ($ var == 5) {print "The value of this variable is 5.";} It is very important that the Note the difference between the double and the single equal sign. The single equal sign is used for an assignment. This means that the program assigns a new value to the variable. Queries, however, are not allowed to change the value of the variable. This is why the double equal sign is always used. It is not only possible to check a numerical value. It can also be used to check strings: if ($ product name == "Drill") {print "This is the product about a drill ";} For many variables it is not important whether they correspond exactly to a certain value, but rather whether they are larger or smaller than this. Upper and lower characters are used for this (> and <). For example, the following function could be used in an online shop:

```
if ($ order value <20)
{
print "You have not yet reached the minimum order value";
}
```

There are also a few other comparison operators :! = Executes the function if the values are not equal. <= Executes the function if the first value is less than or equal to the second value. > = Executes the function if the first value is greater than or equal to the second value.

6.3 Integrating logical operators into the query

In most cases the comparison operators presented in the previous section are sufficient to adequately formulate the condition. However, there are always cases in which the corresponding specifications are somewhat more complicated. For example, it is possible that a function should only be carried out if two different conditions are met at the same time. In other cases it is sufficient if one of several possible conditions is met. Logical operators are used to formulate such expressions. If two conditions are to be met at the same time, the logical and is used for this. For this it is possible to simply use the English word " and" . Alternatively, the programmer can insert a double ampersand (&&):

```
if ($ var1 <5 and $ var2 == 7) {print "Variable 1 is less than 5 and variable 2 is 7.";
}
```

Or

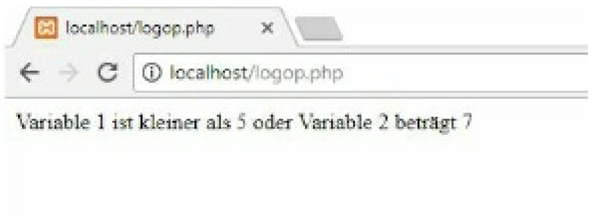
```
if ($ var1 <5 && $ var2 == 7) {print "Variable 1 is less than 5 and variable 2 is 7.";}
```

It is not only possible to specify two conditions, but any number:

```
if ($ var1 <5 && $ var2 == 7 && $ var3> = 6) {print "Variable 1 is less than 5, variable 2 is 7 and
variable 3 is greater than or equal to 6.";
}
```

In other cases it is only necessary that one of two conditions is met. It makes sense to use the logical or here. This can be inserted into the program with the keyword " or" or with two vertical bars (||).

```
<? php $ var1 = 2; $ var2 = 7; if ($ var1 <5 || $ var2 == 7) {print "Variable 1 is less than 5 or variable 2
is 7.";}?>
```



```
Variable 1 ist kleiner als 5 oder Variable 2 beträgt 7
```

The reader may have noticed that, in contrast to the previous examples, a complete program - with the opening and closing tags and with the declaration of the variables - is given. The reason for this is that it makes sense at this point to insert the program once into the text editor and run it in the browser. In the next step it is possible to change the values of the variables so that the conditions are met or not. It is noticeable that the browser not only displays the corresponding text when exactly one of the two conditions is met, but also when this applies to both specifications. However,

in some programs this is not desirable. In these cases the exclusive or is used. This is inserted with the keyword " xor" . If the program is changed accordingly, the browser will only display the text if one of the two conditions is met and the other is not. Another logical operator is the logical not, which is represented by an exclamation mark. This reverses the truth value of the
Condition to. This means that the program only executes the function if the corresponding condition is not met:

```
if (! ($ var <5)) {print "The variable 1 is not smaller than 5.";
}
```

It is important to put the expression in brackets after the exclamation mark. This is necessary because the logical not does not refer to a single value, but to the entire expression. Very different comparisons can be summarized with brackets and logical operators can be combined with further queries. However, the complexity of these expressions is very high, so this topic is not covered further in this entry-level textbook.

6.4 else and elseif

The simple if queries presented in the previous sections had one thing in common: they perform an action when a certain condition is met. However, if it does not apply, the program simply continues with the commands that follow the if query. However, it often happens that another action should be taken if the condition is not met. Of course, it is possible to insert another if query after the first block and to negate the corresponding condition:

```
if ($ gender == "m") {print "<h1> Hello Mr.". $ surname. "! </h1> \ n";
}
if (! ($ gender == "m")) {print "<h1> Hello Mrs.". $ surname. "! </h1> \ n";
}
```

However, it is much easier to use the term " else" . The action that is carried out afterwards must also be in curly brackets. The following program therefore has exactly the same function as the example described above:

```
if ($ gender == "m") {print "<h1> Hello Mr.". $ surname. "! </h1> \ n";} else {print "<h1> Hello
Mrs.". $ last name. "! </h1> \ n ";
}
```

These programs assume that there are only two different options for specifying gender. This means that if the letter m for male has not been entered as gender , it is automatically a woman. This assumption is usually correct, but it can sometimes lead to errors. For example, it is possible that a user did not provide this information, so that the variable remains empty. In this case it would not be advisable to automatically address the reader as “woman ” . Therefore it makes sense to further differentiate the query. A branched query is used for this purpose:

```
if ($ gender == "m") {print "<h1> Hello Mr.". $ surname. "! </h1> \ n";

    }
else {
  if ($ gender == "w") {print "<h1> Hello Mrs.". $ surname. "! </h1> \ n";} else {print "<h1> Hello!
</h1> \ n" ;
}
}
```

This program ensures that the salutation with “woman ” is only used if the gender is given as female. If there is no entry or a completely different value, the program decides to address it neutrally with “Hello ” . However, this type of branching is relatively complicated and is only intended to serve as an explanation. In practice, it is common to use the “ elseif” command instead . The following program works in exactly the same way, but is a bit simpler.

```
if ($ gender == "m") {print "<h1> Hello Mr.". $ surname. "! </h1> \ n";} elseif ($ gender == "w") {

print "<h1> Hello Mrs.". $ surname. "! </h1> \ n";} else {print "<h1> Hello! </h1> \ n";}
```

6.5 Exercise: Create queries yourself

Create a program that determines whether the inventory for a certain product has fallen to 0 and, if necessary, issues a message that the item is no longer available. First, create an array containing a product with its price and number of items available, then write a program that: Informs the buyer that the item is not available when the inventory is 0; issues that the product will be delivered free of charge if at least one item is in stock and if the price is at least 20 euros.
indicates that there is a shipping cost of 5 euros for delivery if the item is available but the price is less than 20 euros.

1. <? Php $ inventory = 4; if ($ inventory == 0) {print "The requested article is unfortunately no longer available.";}?>

For this program it is necessary to introduce a variable for the inventory. The `if` query then checks whether this has dropped to 0 and outputs the corresponding message if necessary. By changing the values of the `$` inventory variable (and also setting them to 0), you can check the functionality of the program.

2.

```
<? php $ product = array ('Product name' => "Drill", 'Price' => 15, 'Quantity' => 3); if ($ product
['Quantity'] == 0) {print "The product ". $ product ['product name']." is unfortunately no longer
available. ";} elseif ($ product ['price']> = 20) {print" The product ". $ product ['product name']." is
available and is delivered free of shipping costs. ";} else {print" The product ". $ product ['product
name']." is available. Shipping costs are 5 euros for the delivery. ";}
?>
```

This is just a suggested solution. There are many other correct solutions as well. If you have chosen a different method, you should change the corresponding values in the array and thus try out whether your program outputs the correct information for all combinations.

Chapter 7

Extending the functionality of a program through loops

Repeating an identical activity over and over is a great torment for most people. The monotonous occupation leads to boredom and inhibits performance. However, it is precisely this task that is the great strength of a computer program. It often repeats a simple function hundreds or even thousands of times - in a matter of seconds and without any errors. Now one could assume that this monotonous activity falls back on the programmer, since he performs the same functions countless times in your program. But there is a simple trick to avoid this: the use of loops. It is only necessary to specify the condition under which the program should execute the loop. It is then only necessary to enter the corresponding commands once. The program then executes these until the condition is no longer met - regardless of whether it takes ten or a million runs. This chapter introduces the use of loops . These play a very important role in almost all computer programs - regardless of whether it is a PHP program for designing websites or other software in a different programming language. Loops are therefore of enormous importance for all programmers and are indispensable tools.

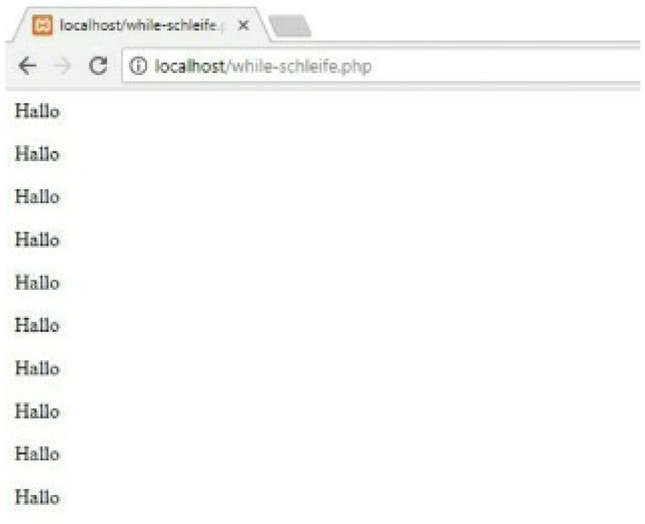
7.1 Head-controlled loops: while and for

In order to explain the function of a loop, a very simple - if not necessarily useful - example should serve. The program should only display the word hello on the screen ten times in a row. Such a task rarely occurs in practical work, but it clearly illustrates how a loop works. First, it is necessary to specify a running condition. This is introduced with the keyword `while` . It stands in front of the actual content of the loop. For this reason these are called head controlled. The same operators are used for the conditions as for the `if` queries. If the loop is to be run through with a fixed number of times, as in this example, an index variable is used in the condition. In computer science it is common to denote this as `$ i` and to set the value to 0 at the beginning of the loop:

```
$ i = 0; while ($ i <10)
```

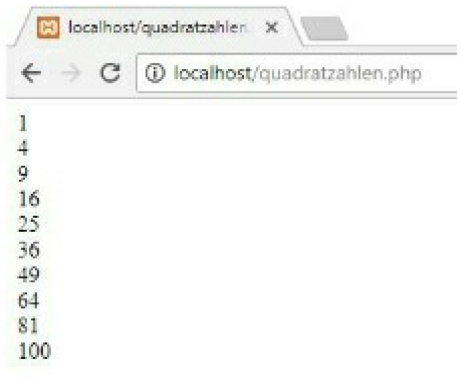
The run condition specifies that the loop should be executed as long as the value of the index variable is less than 10. In order for the loop to end at the desired point in time, it is therefore necessary to increase the value of this index variable in each pass. This happens in the loop body. This follows the run condition in curly brackets. In addition, the body of the loop contains the function that is to be carried out for each pass.

```
<? php $ i = 0; while ($ i <10) {print "<p> Hello </p> \ n"; $ i ++;}
?>
```



```
Hallo
Hallo
Hallo
Hallo
Hallo
Hallo
Hallo
Hallo
Hallo
Hallo
```

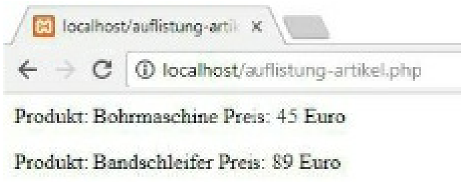
This program already shows that it is possible to use loops to perform a large number of functions with just a few commands. It is also possible to refer to the index variable for each individual run. In this way, for example, the square numbers can be easily displayed: `<? Php $ i = 0; while ($ i <10) {$ i ++; print $ i * $ i. "<br> \ n";}?>` This program multiplies the index variable by in each run itself and in this way creates the square of that number. Since it makes sense for this series to begin with the number 1, the index number must be increased before the command for the output. Alternatively, it would be possible to prefix the print command, as in the previous program, but to add the value 1 to the calculation: `print ($ i + 1) * ($ i + 1). "<br> \ n";` The functionality remains the same for both options and generates the following output on the screen:



```
1
4
9
16
25
36
49
64
81
100
```

It often happens that the loop body is only to be executed under a certain condition. In these cases it is necessary to integrate if- queries into the loop body. The following example should serve as an explanation. There is a composite array that contains the product name, price and number of available units for three different products. The loop should then list all products with the associated price - but only if at least one item is in stock. In this case it is possible to click on the individual fields
of the array with a variable. The index variable is ideal for this.

```
<? php $ assortment = array (); $ product [0] ['product name'] = "drill"; $ product [0] ['price'] = 45; $ product [0] ['quantity'] = 6 ; $ product [1] ['product name'] = "circular saw"; $ product [1] ['price'] = 79; $ product [1] ['quantity'] = 0; $ product [2] ['product name '] = "Belt sander"; $ product [2] [' Price '] = 89; $ product [2] [' Quantity '] = 11; $ i = 0; while ($ i <3) {if ($ product [$ i] ['Quantity']> 0) {print " <p> Product:". $ product [$ i] ['Product name']. "Price:". $ product [$ i] ['Price'] . "Euro </p> \ n";} $ i ++;}?>
```



```
Produkt: Bohrmaschine Preis: 45 Euro

Produkt: Bandschleifer Preis: 89 Euro
```

Several program lines are required for a while loop. First it is necessary to introduce the index variable and fill it with a value.
Then the programmer has to specify the condition. Finally, it is necessary to increase the value of the index variable in the loop body. To reduce the effort,

it makes sense to use a for loop instead. Here , all these commands are summarized in one line. This alternative is always useful if the loop should complete a fixed number of runs or if a certain value should be counted in it. The for loop is introduced by the term for . This is followed by a bracket in which the value of the variable is introduced first. This is followed by the condition and finally the specification in which way the index variable is to be changed in each pass. The individual parts are separated from one another with a semicolon. The first program in this section serves as an example. This can be represented as follows without changing the functionality using the for loop:

```
<? phpfor ($ i = 0; $ i <10; $ i ++) {print "<p> Hello </p> \ n";}?>
```

7.2 Foot-controlled loop: do while

In both the while and for loops, the condition precedes the loop body. There is also another option where the order is reversed. This is introduced by the key term do . The body of the loop then follows in curly brackets. Then the expression while and the condition must be present. In contrast to the previous loops, in this case it is necessary to set a semicolon at the end. The first example from the previous chapter looks like this with a do-while loop:

```
<? php
$ i = 0; do {print "<p> Hello </p> \ n"; $ i ++;} while ($ i <10);?>
```

In this example, the functionality is exactly the same as the while or for loop. This also applies to all other programs in which the loop is to be executed with a fixed number of runs. Which method is used depends solely on the preferences of the programmer. However, there is one important difference that can affect the way it works in some programs: With the for and while loops, the condition is checked at the beginning. The loop body is only executed if the condition is met. With the do-while loop, however, the commands in the loop body are always executed at least once. Only then is the condition checked. If it is fulfilled, another cycle follows. If it is not fulfilled, the loop is aborted. The following two are intended to distinguish this

Program examples clarify : <? Php $ number = 50; $ condition = true; while ($ condition) {print "<p> Current number:". $ number. "</p> \ n"; $ number -; $ condition = ($

```
number> 20);}?>
```

And

```
<? php $ number = 50; $ condition = true; do {print "<p> Current number:". $ number. "</p> \ n"; $
number -; $ condition = ($ number> 20 );} while ($ condition);?>
```

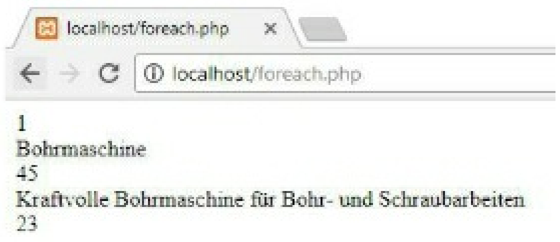
These programs output the number of a specific article. With each run - for example when selling a unit - the number is reduced. When the inventory drops to 20, the program halts the loop because the current inventory is too low. The functionality of the two alternatives is exactly the same. The difference becomes clear, however, if the value is set to `false` when introducing the variable `$ condition` . In this case, the `while` loop is not executed because the condition is not met at the beginning. The `do-while` loop executes the loop body anyway. The condition is set to `true` so that the loop is run through as often as before until the number has assumed the value 20.

7.3 foreach loops for working with arrays

As shown in the penultimate program example in Chapter 7.1, loops are often used to access the various contents of an array. Using the index number, it is very easy to navigate to the individual fields. The sample program worked fine, but it still has a problem. In many online shops it often happens that the range is expanded or reduced. In this program, however, the number of passes was set to 3 from the beginning. If the number of articles changes, the program will no longer work. In this case it would be necessary to do a manual one every time the scope of the range of goods changes
Even with associative arrays that work with terms for assignment instead of index numbers, access in this way is not possible. To avoid these problems, it makes sense to use a `foreach` loop. This is specially designed for handling arrays. It automatically adapts the number of runs to the number of entries available. Before the mentioned example from Section 7.1 is executed with a `foreach` loop, a somewhat simpler program should clarify its functionality:

```
<? php $ product = array (1, "Drilling machine", 45, "Powerful drilling machine for drilling and
screwing work", 23); foreach ($ product as $ content) {print $ content. "<br> \ n";} ?>
```

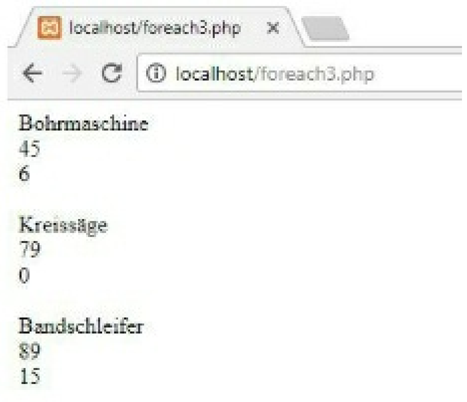
This program first determines an array with various details about a product. The keyword `foreach` must be at the beginning of the loop . This is followed by a round bracket and the name of the array. Then there is another key term: `as.` After that it is necessary to name a new variable. This serves as a copy of the contents of the array inside the loop. If the variable just introduced is now used in the loop, it reproduces one field of the array in each run - until all entries have been listed. This results in the following output in the browser:


localhost/foreach.php

```
1
Bohrmaschine
45
Kraftvolle Bohrmaschine für Bohr- und Schraubarbeiten
23
```

With this knowledge it is now possible to modify the program from section 7.1. Readers who have already understood how the foreach loop works are welcome to try this conversion themselves. <? php $ assortment = array (); $ product [0] ['product name'] = "drill"; $ product [0] ['price'] = 45; $ product [0] ['quantity'] = 6 ; $ product [1] ['product name'] = "circular saw"; $ product [1] ['price'] = 79; $ product [1] ['quantity'] = 0; $ product [2] ['product name '] = "Belt sander"; $ product [2] [' price '] = 89; $ product [2] [' number '] = 11; foreach ($ product as $ content) {if ($ content [' number ']> 0) {print "<p> Product:". $ Content ['Product Name']. "Price:". $ Content ['Price']. "Euro </p> \ n";}}?>
For the conversion it is first necessary to remove the introduction and the increase of the index variables, since these are no longer necessary when using a foreach loop. Then the foreach loop must be introduced. Inside the brackets are the name of the array and, after the term as, the variable that is to display the contents in the loop. After that it is only necessary to adapt the print command. In this way it is possible to add another product to the array. This is now automatically taken into account during playback. A foreach loop also allows a composite array to be output. In contrast to the previous example, it is not necessary to address the individual fields of the second level individually. The following program lists the contents of the entire array: <? Php $ assortment = array (); $ product [0] ['product name'] = "Drill"; $ product [0] ['Price'] = 45; $ product [0] ['Quantity'] = 6; $ product [1] ['Product name'] = "Circular saw"; $ product [1] ['Price'] = 79; $ product [1] ['Quantity'] = 0; $ product [2] ['Product name'] = "Belt sander"; $ product [2] ['Price'] = 89; $ product [2] ['number'] = 15; foreach ($ product as $ level1) {foreach ($ level1 as $ level2) {print $ level2. "<br> \ n";} print "<br>" ;}?> A nested foreach loop is required for this. The outer loop refers to the first level. The entries of the second level are like another array for each individual field of the first level
understand. Therefore, another foreach loop is necessary for this . For this, the individual entries of these small arrays are reproduced in the variable $ level2 according to the same scheme . The additional print command with the HTML

tag <br> ensures an additional line break after all information on a particular article has been output. This creates groups that belong together when outputting to the screen:



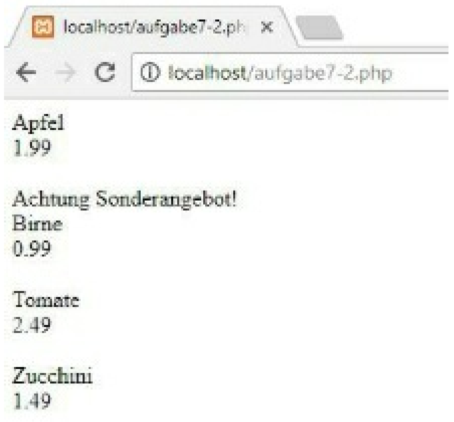
```
Bohrmaschine
45
6

Kreissäge
79
0

Bandschleifer
89
15
```

7.4 Exercise: Designing programs with loops

Create three programs that count from 1 to 10, using a while , a for and a do-while loop for each. 2 . Create a composite array for a greengrocer's assortment of apples, pears, tomatoes, and zucchini. The first level should have a numerical index. The second, on the other hand, is supposed to be associative (with the names product, price and special offer) .The product information is a character string with the
corresponding fruit or vegetable type, the price around a number and the specification Special offer should be a Boolean variable. Create a foreach loop that specifies the product name and the price. If this is a special offer, you should read "Attention special offer! " Appear.3. If you create a foreach loop, you can also access the name of the corresponding field for associative arrays. To do this, you have to insert another variable in the round brackets after the key term as , which accepts this value. This is followed by the symbol " =>" and then the variable that reproduces the contents of the array within the loop: ($ level1 as $ field name => $ level2) Change the last program from chapter 7.3 so that it can output the contents prefixes the corresponding field names.

1.

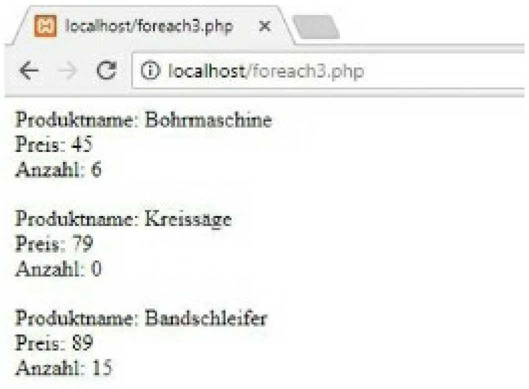
<? php $ i = 0; while ($ i <10) {$ i ++; print $ i. "<br> \ n";}?> <? phpfor ($ i = 0; $ i <10; $ i ++) {print ($ i + 1). "<br> \ n";}?> <? php $ i = 0; do {print ($ i + 1). "<br> \ n"; $ i ++;} while ($ i <10);?> 2. <? php $ assortment = array (); $ assortment [0] ['product'] = "apple"; $ assortment [0] ['price'] = 1.99; $ range [0] ['special offer'] = false; $ range [1] ['product'] = "pear"; $ range [1] ['price'] = 0.99; $ range [1] ['Special offer'] = true; $ range [2] ['Product'] = "Tomato"; $ range [2] ['Price'] = 2.49; $ range [2] ['Special offer'] = false; $ assortment [3] ['product'] = "Zucchini"; $ assortment [3] ['price'] = 1.49; $ assortment [3] ['special offer'] = false; foreach ($ assortment as $ content) {if ($ content ['special offer']) {print "Warning special offer! <br> \ n";} print $ content ['product']. "<br> \ n"; print $ content ['price']. " <br> <br> \ n ";}?>



```
Apfel
1.99

Achtung Sonderangebot!
Birne
0.99

Tomate
2.49

Zucchini
1.49
```

3.

<? php $ assortment = array (); $ product [0] ['product name'] = "drill"; $ product [0] ['price'] = 45; $ product [0] ['quantity'] = 6 ; $ product [1] ['product name'] = "Circular saw"; $ product [1] ['price'] = 79; $ product [1] ['quantity'] = 0; $ product [2] ['product name '] = "Belt sander"; $ product [2] [' Price '] = 89; $ product [2] [' Quantity '] = 15; foreach ($ product as $ level1) {foreach ($ level1 as $ field name = > $ level2) {print $ fieldname. ":". $ level2. "<br> \ n";} print "<br>";}?>



```
Produktname: Bohrmaschine
Preis: 45
Anzahl: 6

Produktname: Kreissäge
Preis: 79
Anzahl: 0

Produktname: Bandschleifer
Preis: 89
Anzahl: 15
```

Chapter 8

Functions in PHP

A computer program often contains many hundreds of lines of program code, and it is often difficult to keep track of them. For this reason it makes sense to give the whole a structure. This is done by functions that allow a certain part of the program to be outsourced. The main program can easily access these functions by calling their names. Sometimes it even happens that the actual main program only consists of a few lines. These call functions that contain the fundamental components of the program. Each of them in turn uses many other functions that give the software a clear structure. Functions not only offer the advantage that they make the program clearer. In addition, they can save the programmer a lot of work. In many cases it happens that a certain sequence of instructions occurs very frequently in a program. Programming them over and over again would take a lot of work. It also increases the risk of errors. However, if the corresponding commands are swapped out into a function, it is possible to call them up simply by the function name. This significantly reduces the work. Functions are therefore of great importance for complex programs. Like almost all programming languages, PHP offers the option of creating functions and integrating them into a program.

8.1 The structure of a function in PHP

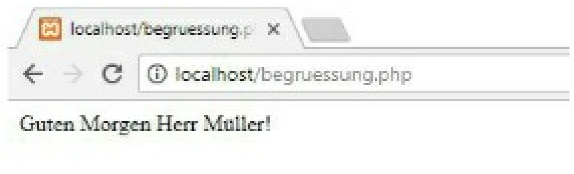
To create a function, it is first necessary to mark it as such in the program. The key term function is used for this .
This is followed by the name of the function. In principle, the programmer can choose the name freely. But it makes sense to use self-explanatory names. If the programmer no longer knows by heart which function fulfills which task when making a correction later, the name is a valuable aid. This is followed by a round bracket that must open and close. The commands that the function is supposed to carry out then follow in curly brackets. A simple function could look like this:

```
function greetings () {print "Good morning!";}
```

To call this function, it is only necessary to insert the function name -

followed by the round bracket and a semicolon - into the program. This results in the following code:

```
<? phpfunction greetings () {print "Good morning!";} greetings ();?>
```



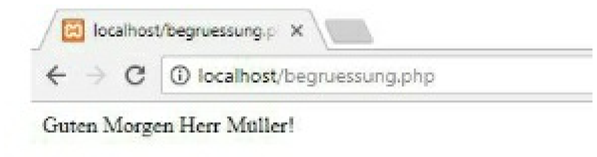
```
Guten Morgen Herr Müller!
```

The use of round brackets after the function name is noticeable in this function. At first glance, these seem to be completely useless. However, they represent an important part of the function syntax and must be used in any case. In this simple example, the brackets are empty and do not affect the result. For many functions, however, they are filled with content. The content of the brackets is used to transfer variables to the function. They only remain empty if no transfer of values from the main program is necessary. In all other cases there must be a variable name here. This takes on the value that the main program transfers. It is important to only use this variable within the function. In this example it would be possible to control the content of the output through a variable. This would make it possible in a later step, for example, to adapt the greeting to the time of day and to address the reader with “Good morning ” , “Good day ” or “Good evening ” depending on the time . If a variable is to be used, it is also necessary to specify its content in brackets when the function is called:

```
<? php
function greeting ($ text) {print $ text;} greeting ("Good morning!");?>
```

In this example, the value of the variable was inserted directly into the brackets when the function was called. However, it is common to use a variable for this. This makes it possible to adapt the content, whereby the programmer can transmit not just a single variable, but any number. In this case it is necessary to separate the individual values with a comma - both when defining the function and when calling it.

```
<? phpfunction greeting ($ text, $ address) {print $ text. "". $ address. "!";} $ greeting = "Good
morning"; $ reader = "Mr. Miller"; greeting ($ greeting, $ reader );?>
```



```
Guten Morgen Herr Müller!
```

8.2 Return values of the functions

In the previous examples, the functions were used to perform a certain action. After they were executed, their task was complete and the program made no further reference to them. However, there are also many applications where the function should calculate a certain value and return this to the main program. This can then save it and use it for further tasks. Return values are used for this purpose. The return value is determined by the `return` command . This has to be integrated into the body of the function. It is possible to simply specify a variable after this key term that contains the return value: `function doubling ($ value) {$ value = $ value * 2; return $ value;}` Alternatively, the programmer can insert the mathematical operation directly after the `return` command. In this case it must be in round brackets: `function doubling ($ value) {return ($ value * 2);}` Different variable types can be used for the return value. In this way it is therefore not only possible to transmit numbers to the main program, but also character strings and truth values.
In order to include the return value in the main program, it is possible to assign the value of the corresponding function to a variable. This is done just like assigning ordinary values with an equal sign.

```
<? phpfunction doubling ($ value) {return ($ value * 2);} $ result = doubling (5); print $ result;?>
```

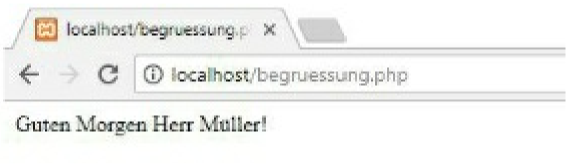
Another alternative is to integrate the return value directly into the `print` function - or into many other PHP commands:

```
<? phpfunction duplication ($ value) {return ($ value * 2);} print duplication (5);?>
```

It is important to note that each function can return a maximum of one value. However, it can also be an array. If you want the function to return multiple values, you need to pack them into an array. It is then possible to resolve this again and reproduce it in individual variables.

```
<? phpfunction doubling_square ($ value) {$ double = $ value * 2; $ square = $ value * $ value; $
```

result = array ('doubling' => $ double, 'square' => $ square); return $ result;} $ return value = doubling_square (3);
$ doubled_value = $ return value ['doubling']; $ value_to_square = $ return_value ['square']; print "The double value of this number is". $ doubled_value. ". <br> \ n"; print "The square of this number is " . $ value_to_square.". <br> \ n ";?>



Guten Morgen Herr Muller!

8.3 Integrating a function in the PHP program

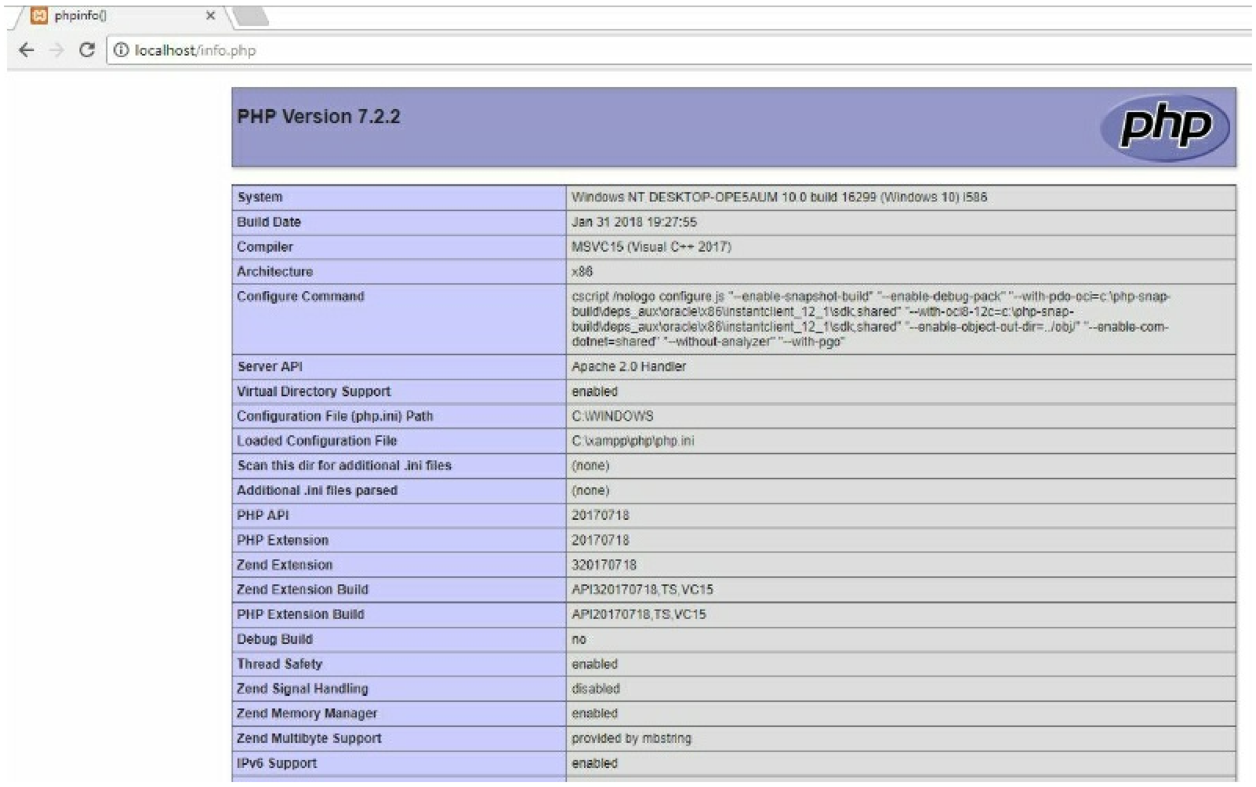
In order to use a function in a PHP program, it is necessary to include it in the main program. There are various possibilities. An alternative has already been used in the previous sections. The function was placed in front of the actual program. The call was made quite simply via the function name. Although this procedure is very simple, it does mean that one of the main advantages of using functions is lost: the clear layout of the program. If more than a hundred functions are used in extensive software, this leads to very long program code in which it is difficult to recognize the individual elements.
For this reason, this method is only used for very simple programs. For more extensive applications, it is common to outsource the functions to separate files. This means that a new file must be created for this. In order to access it, it is necessary to include the relevant files in the main program before calling the function. How this works will be described using the function duplication that was created in the previous section. In order to swap these out, it is necessary to first create a new file. In order to facilitate the assignment, the file name should correspond to the function name; the ending .php is also used here. This file only contains the area in which the function was defined - as always in the corresponding PHP tags: <? Phpfunction doubling ($ value) {return ($ value * 2);}?> Before the function is used in the main program it is now necessary to include the corresponding file. Two commands are possible for this: include and require . The difference between the two commands, however, is minimal and only shows up when a file cannot be loaded. When using include , the program continues in this case and processes all further

commands - even without the contents of the file. If require is used, however, execution aborts immediately. With both options, it is possible to add the addition _once (include_once or require_once). This causes the interpreter to check whether the corresponding file has already been called up in a previous part. If so, it does not load the file
again. This is particularly useful with extensive software, if the programmer no longer knows exactly (or if this depends on the specific execution of the program) whether he has already loaded the corresponding file. By not loading the files twice, it is possible to increase the execution speed. After the appropriate command, it is necessary to insert the file name. This is in round brackets and in quotation marks. If the file is in a different folder than the main program, it is important to ensure that you also specify the path. The main program therefore looks like this: <? Phpinclude ("doubling.php"); print doubling (3);?>

8.4 Using functions from the PHP library

In addition to the functions that the programmer creates himself, there is an extensive library in PHP. This contains numerous useful functions that are used in many programs. This can make the workers much easier. An example of a pre-built function is phpinfo (). This provides a lot of information about the system and the PHP version used. It can easily be built into a program:

```
<? phpphpinfo ();?>
```



PHP Version 7.2.2

System	Windows NT DESKTOP-OPE5AUM 10.0 build 16299 (Windows 10) i586
Build Date	Jan 31 2018 19:27:55
Compiler	MSVC15 (Visual C++ 2017)
Architecture	x86
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--enable-debug-pack" "--with-pdo-oci=c:\php-snap-build\deps_aux\oracle\x86\instantclient_12_1\sdk,shared" "--with-oci8-12c=c:\php-snap-build\deps_aux\oracle\x86\instantclient_12_1\sdk,shared" "--enable-object-out-dir=../obj/" "--enable-com-dotnet=shared" "--without-analyzer" "--with-pgo"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\WINDOWS
Loaded Configuration File	C:\xampp\php\php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20170718
PHP Extension	20170718
Zend Extension	320170718
Zend Extension Build	API320170718,TS,VC15
PHP Extension Build	API20170718,TS,VC15
Debug Build	no
Thread Safety	enabled
Zend Signal Handling	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	provided by mbstring
IPv6 Support	enabled

To get an overview of the possible functions, it is useful to call up the online documentation for PHP. This is available at http://php.net/manual/de/. It contains a lot of important information about this programming language. The Function References chapter lists the functions that are contained in the library. The online documentation is also helpful in determining the exact behavior of a function. It is either possible to access the individual functions via the function references. If the exact name is already known, it makes sense to use the search function. For example, the following entry can be found under the entry phpinfo :

It not only describes which tasks the corresponding functions take on. In addition, it is specified exactly which values the main program must transmit and which return values the function generates. The online documentation is a very important tool for using pre-built functions in a program.

8.5 Exercise: Creating PHP programs with functions

1.

Create a function that calculates the first ten powers of a number. The ** operator is used for this. This is preceded by the base and followed by the exponent. The main program should transmit the initial value. It is returned by an array. The array is output in the main program. 2. Now also output the array in a separate function. Store both functions in a separate file and integrate them into the main program. 3. Find a function in the online documentation to enable the
To determine the square root of a number and another function with which you generate a random variable. (Help: Both functions can be found in the math chapter). Find out how to use these two functions. Now create a program that calculates the square root of a random value and outputs both the initial value and the result.

1.

```
<? phpfunction potency ($ value) {$ result = array (); for ($ i = 0; $ i <10; $ i ++) {$ result [$ i] = $ value ** ($ i + 1); } return $ result;} $ powers = potency (2); foreach ($ powers as $ output value) {print $ output value. "<br>";}?>
```

2. File functions.php:

```
<? phpfunction potency ($ value) {$ result = array (); for ($ i = 0; $ i <10; $ i ++) {$ result [$ i] = $ value ** ($ i + 1); } return $ result;} function output ($ powers) {foreach ($ powers as $ output value) {print $ output value. "<br>";}}?>
```

File main.php: `<? Phpinclude ("functions.php"); output (potency (2));?>` 3. For this task you need the commands `rand ()` and `sqrt (), the` functionality of which is described in the online documentation becomes. This results in the following program: `<? Php $ random number = rand (); $ root = sqrt ($ random number); print "The root of". $ Random number. "Is". $ Root. ". <br> \ n"; ?>`

Chapter 9

Object-oriented programming: classes, objects and methods

The previous chapters described the basic functions of the PHP programming language. With loops, `if` queries and functions, a wide range of programs can be created for very different applications. In the introduction, however, it was already mentioned that PHP is an object-oriented programming language. This is a concept that is particularly suitable for large and complex programs. However, it is not easy to understand the details of object-oriented programming. This topic represents an important focus of the computer science course. Experienced programmers take care of the creation of complex programs with object orientation. In this textbook, however, the topic will be briefly introduced. On the one hand, it is important that the reader gets to know the basics of this concept - for example to understand a program code that another programmer has created. On the other hand, there are numerous frameworks that simplify the creation of web projects. There are often prefabricated objects that allow the integration of special functions. In order to be able to use this tool, it is necessary to know at least the basics of object-oriented programming. However, this book can only explain the most important basics. There are numerous textbooks available that are entirely devoted to object-oriented programming and explain all the details. Object-oriented programming focuses on the object.

This is an element that has specific properties. It is possible to think of these objects as different everyday objects. Objects of the same type also have similar properties - albeit in different ways. In the case of objects of a different type, however, different properties are also important. For example, while size, color and ripeness play an important role in the case of an apple, the speed, fuel consumption and the number of seats are decisive parameters for a car. Classes are also of great importance for object-oriented programming. These specify the basic properties that must be taken into account when describing an object. There are also methods. These are used to carry out various actions with the objects.

9.1 The class: the basis of object-oriented programming

As indicated in the previous section, the properties that an object can assume play an important role in object-oriented programming. There are usually a very large number of possible properties. However, it is important to identify the critical areas. As already mentioned, speed, fuel consumption and the number of seats play an important role for an author. But every car is characterized by other parameters - for example, length, width, brand and color. At the beginning it is therefore necessary to consider which features are important for a particular application. For example, when a programmer creates a platform on which used cars are offered for sale, he has to consider what information is required for this. If he then creates an object for a certain car, he has to assign all important properties to it.
This is where the classes come into play. This is a pattern to which all objects are oriented. In this example, the template specifies exactly which properties are to be described for a car. The variables that contain the corresponding values are called members. If an object that symbolizes a car is to be created, it must follow these guidelines carefully. This means that there must be a value for all properties that are defined in the class. Since the class is the template for an object, the first step in object-oriented programming is to define a class. The keyword `class` Use. This is followed by the name of the class, which the programmer can freely choose. The properties that all objects of this class have in curly brackets.

```
<? phpclass car {private $ seats; public $ speed; public $ fuel consumption;}?>
```

The individual members are variables. Therefore it is also necessary to put a dollar sign in front of the corresponding name. Either `public` or `private is in` front of the specification of the corresponding property . These specifications assign the access rights to the corresponding properties. If the designation `public` precedes the designation, it is possible to access the corresponding values and change them from outside the class - i.e. from the main program. If, on the other hand, the term `private` comes before the property, this is not possible. That protects the variable. In this case it is only possible to access the variables from the same class. How this works is described in Section 10.3.

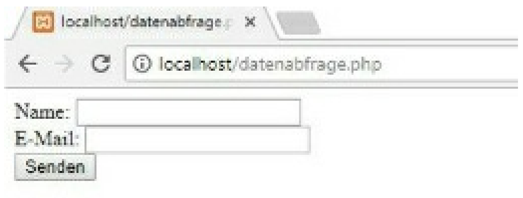
It is also possible to initialize the variables in the class. This means that every object that is created using this class is initially given the corresponding standard values. However, you can easily change them later. Since the vast majority of cars have five seats, it would make sense to set this value to 5 by default, for example. The definition of the class then looks like this:

```
<? phpclass Auto {private $ seats = 5; public $ speed; public $ fuel consumption;}?>
```

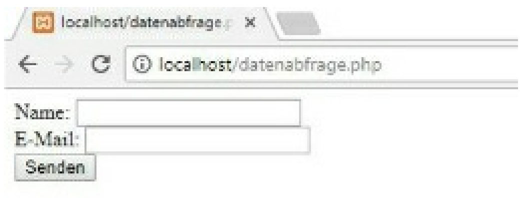
9.2 Create an object with a class

After the class has been defined, it is possible to create an object with the appropriate properties. Since it is common practice to store the description of the class in a separate file, it is first necessary to include the corresponding file. How this works has already been explained in the chapter on the functions. When using classes, the procedure is identical. If the class is defined in the same file as the main program, this step is not necessary. When the appropriate information is available, the programmer must come up with a name for the appropriate object. This must be preceded by the dollar sign. This is followed by the equal sign and finally the keyword `new` . It is then necessary to specify the class with which the new object is to be created. This is done by the name of the class. Then a pair of round brackets is necessary, which in this case remain empty. With the following command it is possible to create a new object of the class car:
`$ myCar = new Auto ();` It is then possible to access the individual members of the object. For example, the main program can enter values for the speed and fuel consumption and then display these on the screen. To access the contents it is necessary to insert the name of the object preceded by a dollar sign. This is followed by the symbol “ `->`” . Finally, the name of the member is required - but without the dollar sign in this case. The program looks like this:

```
<? phpinclude ("class_auto.php"); $ myCar = new Car (); $ myCar-> speed = 150; $ myCar-> fuel
consumption = 7; print "Speed:". $ myCar-> speed. "km / h <br> \ n "; print" Fuel consumption: ". $
meinAuto-> fuel consumption." l / 100km <br> \ n ";?>
```



The next car is a sporty two-seater. Therefore an attempt should be made to change the number of seats. For that comes
analogous to the previous example, the command $ meinAuto-> Sitzplaetze = 2; for use. However, the following error message appears:



The reason for this is that this member has been declared private. This means that access from the main program is not possible. For this, methods are used that are explained in the following section.

9.3 Methods for working with objects

Methods are often used to work with objects. These have a very similar structure to the functions already described in the previous chapter. The difference, however, is that methods can only be used on objects of the corresponding class. Methods allow access to private properties of the object. However, they also offer many advantages for manipulating public members. In the previous section, it was shown that access to the private property seat is not possible directly from the main program. The following example should show how the programmer can adapt the value using a method. The method is directly in the definition of the class. Therefore she has access to all members - even to the private ones. A method has practically the same structure as a function. It is also introduced with the same keyword. The only difference is that there is
Here, too, it is necessary to define whether the method is public or private.

The following method therefore allows the number of seats to be changed:

```
public function setSeatplaces ($ number) {$ this-> seats = $ number;
}
```

In order to access one of its members within a class, it is necessary to use the keyword $ this followed by the symbol " ->" and the name of the member (without the dollar sign). The function must be public so that the main program can access it. When the method is called, the program transfers the value that is to be inserted. This is recorded in the variable $ number . When a method changes a member, it is common to combine the function name from the component set and the name of the member (with an uppercase letter). If, on the other hand, it asks for a value, it is common to combine the name from the term get and the name of the property. If the corresponding method is integrated in the class, it can be called from the main program. It is necessary to put the object name and the symbol "->" in front of the name of the function:

```
$ meinAuto-> setSitzplaetze (2);
```

The complete class looks like this:

```
<? phpclass Auto {private $ seats = 5; public $ speed; public $ fuel consumption; public function
setSeats ($ number) {$ this-> seats = $ number;
}}?>
```

The following code is used for the main program:

```
<? phpinclude ("class_auto.php"); $ myCar = new Car (); $ myCar-> speed = 150; $ myCar-> fuel
consumption = 7; print "Speed:". $ myCar-> speed. "km / h <br> \ n "; print " Fuel consumption: ". $
meinAuto-> fuel consumption." l / 100km <br> \ n "; $ meinAuto-> setSitzplaetze (2);?>
```

9.4 Exercise: Using classes, objects and methods

1. Take another look at the array from Section 10.5. Create a class for products that allows these properties to be set. For the names of the individual members, use the designations from the associative array, which is described later in the corresponding chapter. In doing so, prevent access from outside the function. 2. Create methods that allow changes to be made to each

member. Create a main program that creates the object drill and specifies the properties as in the array in chapter 10.5. 3. The last example from this chapter makes it possible to change the number of seats. However, the value cannot be displayed yet . To do this, create the method getSitzplaetze . Use this method in the main program. Solutions:

1.

```
<? phpclass product {private $ item number; private $ product name; private $ price; private $ description; private $ number;}?>
```

2. Definition of the class:

```
<? phpclass product {private $ item number; private $ product name; private $ price; private $ description; private $ number; public function set item number ($ value) {$ this-> item number = $ value;} public function set product name ($ value) {$ this-> product name = $ value;} public function setPrice ($ value) {$ this-> item number = $ value;} public function setDescription ($ value) {$ this-> description = $ value;} public function setNumber ($ value) {$ this-> number = $ value;}}
?>
```

Main program

```
<? phpinclude ("class_product.php"); $ drill = new product (); $ drill-> setArticle number (1); $ drill-> setProductname ("drill"); $ drill-> setprice (45); $ drill -> set description ("Powerful drilling machine for drilling and screwing work"); $ drilling machine-> setNumber (23);?>
```

When you run this program, only an empty field appears in the browser, since no function has been inserted for the output. For the check, it is only important to ensure that no error messages appear. In order to display the content of the individual members, you can add methods to the program that take on this task.

3. Definition of the class:

```
<? phpclass Auto {private $ seats = 5; public $ speed; public $ fuel consumption; public function setSeats ($ number) {$ this-> seats = $ number;} public function getSeatpaces () {return $ this-> seats;
}}
?>
```

Main program:

```
<? phpinclude ("class_auto.php"); $ myCar = new Car (); $ myCar-> speed = 150; $ myCar-> fuel consumption = 7; print "Speed:". $ myCar-> speed. "km / h <br> \ n "; print" Fuel consumption: ". $ myAuto-> fuel consumption." l / 100km <br> \ n "; $ meinAuto-> setSitzplaetze (2); print" Seats: ". $ meinAuto-> getSitzplaetze (). "<br> \ n"; ">?>
```

Chapter 10

Files for storing data

When a PHP program is executed, all the necessary data is stored in variables. However, this brings with it the problem that the information is lost once the program is finished. In many cases, however, it is necessary to continue to use the data. It is therefore necessary to store it in another way . Files can be used for this purpose. These can take on many different values.

10.1 Reading in data from a file

Before it is possible to read in a file, it is necessary to create it. The text editor is also used for this. For this example we want to create a file that contains the numbers from 1 to 10 - each on a separate line. In contrast to the PHP programs, this is a text file. Therefore it should be saved under the name example.txt. It makes sense to create a separate subdirectory for the programs in this section and to store the example file in it. Now the actual program is to be created. The `fopen` command is `used` for this. This is followed by a round bracket, in which the file name is first in quotation marks. After a comma, the programmer must specify the mode - also in quotation marks -. The value `r` can be used here. This only allows the file to be read. An alternative to this is the combination `w +` . In this way, the program receives the rights to read as well as to write. The function generates a so-called handle as a return value. This
provides access to the file in the program and must be saved in a variable. It is common to use the variable name `$ handle` or `$ fh` (for file handle) for this.

The complete program line looks like this: $ handle = fopen ("example.txt", "r"); Since errors sometimes occur when opening a file, it makes sense to first include an if query. This ensures that the program is only executed if the process was successful - i.e. if the $ handle variable exists. If this is not the case, it generates an error message:

```
if ($ handle) {Here are the commands for handling the file} else {print "The file could not be opened.
<br> \ n";
}
```

Now it's time to read in the data. The fgets command is used for this purpose. It is necessary to enter the name of the handle in round brackets and optionally, separated by a comma and the desired length in square brackets. This command reads in the data until a line break, the end of the file or - if specified - the maximum length is reached. (Alternatively, the commands fread and fscan are available, but these cannot be discussed in detail at this point; however, the interested reader can find detailed information on this in the PHP online documentation.) The way in which the fgets command works shows that this reads each line individually. This means that a while loop is necessary for this , which repeats this process until the end of the document is reached. As in this example it is already known to contain 10 lines it would be possible to simply specify ten runs. However, it is simpler and more practical to use the feof function . This returns a Boolean variable. Its value is false as long as the end of the file has not yet been reached. As soon as this is the case, the value changes to true, resulting in the following program:

```
<? php $ handle = fopen ("example.txt", "r"); $ result = array (); if ($ handle) {$ i = 0; while (! feof ($
handle)) {$ content = fgets ($ handle); $ result [$ i] = $ content; $ i ++; print $ content. "<br> \ n";}
fclose ($ handle);} else {print "The file could not be opened. <br> \ n ";}?>
```

This program not only reads the corresponding numbers from the file. It also saves the individual values in the $ result array and outputs the numbers on the screen. It is also important to explain the fclose command . This closes access to the file and should always be inserted after the last use of the handle. In order to check the functionality of the program, it is possible to change the output values in the text file a little. If you run it again, the changed values should be displayed on the screen.

10.2 Save data in a file

When dealing with data, it is not only important to read it from a file. In addition, it is necessary to save them in it. The writing process is similar to reading in in many areas. At the beginning it is necessary to open the file. This happens exactly as when reading in the data with the command fopen. It is important to pay attention to the appropriate mode. The designations w for pure writing and w + for both reading and writing are suitable for this. These commands replace any existing file. If, however, the data is to be appended to the existing content, mode a (for append) must be used. The fputs command is used to write the data to the file . The name of the handle must first be in the brackets after the command. This is followed by the content that is to be written to the file. The following program creates a text file that looks exactly the same as the example file from the previous section.

```
<? php $ handle = fopen ("example2.txt", "w"); if ($ handle) {for ($ i = 0; $ i <10; $ i ++) {fputs ($
handle, ($ i + 1). "\ N");} fclose ($ handle);} else {print "The file could not be opened. <br> \ n";}?>
```

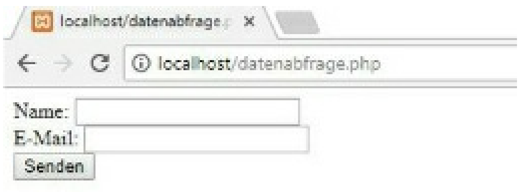
In order to put each number on its own line, it is necessary in each fputs command also insert the line break (\ n). In many Internet applications, interaction with users is very important. HTML forms are used for this. If you are not yet familiar with their use, it is advisable to first read a corresponding introduction to creating HTML Internet pages. The following only explains how these HTML forms are to be used with PHP. In this way, a program should be created that queries the name and e-mail address of the user and then saves them in a text file. The HTML form should be structured as follows be:

```
<form method = "post" action = "datenabfrage.php"> Name: <input type = "text" name = "name">
<br> E-Mail: <input type = "text" name = "e-mail "> <br> <input type =" submit "value =" Submit ">
</form>
```

It is important to always specify post as the method . The action attribute contains the file name to be used for this program. This means that the same page reappears after sending. To access the values that the user has entered, the expression $ _REQUEST ['name'] is necessary. In the square brackets is the name that was assigned to the individual form fields. This reflects the corresponding entry and can be assigned to a variable.

```
<form method = "post" action = "datenabfrage.php"> Name: <input type = "text" name = "name">
<br> E-Mail: <input type = "text" name = "e-mail "> <br> <input type =" submit "value =" Send ">
```

</form> <? php $ handle = fopen (" example3.txt "," w "); if ($ handle)
{if (! empty ($ _ REQUEST ['name']) &&! empty ($ _ REQUEST ['e-mail'])) {if ($ _REQUEST ['name']! = "") {$ name = $ _REQUEST ['name']; fputs ($ handle, $ name. "\ n");} if ($ _REQUEST ['e-mail']! = "") {$ email = $ _REQUEST ['e-mail'] ; fputs ($ handle, $ email . "\ n");}} fclose ($ handle);} else {print "The file could not be opened. <br> \ n";}?>


localhost/datenabfrage.php

Name:
E-Mail:

This program writes the values that the user enters via the input fields into a new file with the name example3.txt. This is created in the same folder in which the program is located. The if- query with the condition (! Empty ($ _ REQUEST ['name']) &&! Empty ($ _ REQUEST ['e-mail'])) needs to be explained. . This ensures that the following part is only carried out if the user has already entered a value in the form field. If this has not yet been done, the contents of $ _REQUEST ['name'] and $ _REQUEST ['e-mail'] have not yet been defined. That would lead to error messages. The query with the condition ($ _REQUEST ['name']! = "") Is also available. This checks whether a value has been entered via the form field. If this is not the case, the program does not write any data to the file.

10.3 Pay attention to the file rights

On UNIX-like systems - including Linux web servers - there are access rights to files. These are intended to prevent unauthorized persons from using, deleting or changing the files. The rights are saved in a number: 0 means that there are no rights to the file at all; the value 1 means that the file can be executed. 2 and 4 stand for read and write rights. It is possible to add the different values. The number 6 therefore means that read and write rights are available. The number 3 indicates that the user is allowed to execute and read the files. If the program is not executed on his own computer under XAMPP, but on a real web server, this can lead to problems. According to the standard settings, this only grants all rights to the creator of the file. All other users may only read it. For example, if the user sends a file to the server via FTP, he alone has the right to overwrite or change it. If the web server now

executes a program with write commands, it is not authorized to do so. It is therefore important to adjust the rights via the FTP client after uploading. This information consists of four numbers - the first always being 0. This is followed by the rights for the creator, then for his group and finally for the general public. If all users have read and write permissions
If the file is to be assigned, for example, command 0666 would have to be used. If the file is not submitted manually, but rather when it is created by a PHP program, it is also important to pay attention to the rights. These can be easily changed using the chmod () command . In the brackets the file name and then the number for the assignment of rights: chmod ("example.txt", 0666);

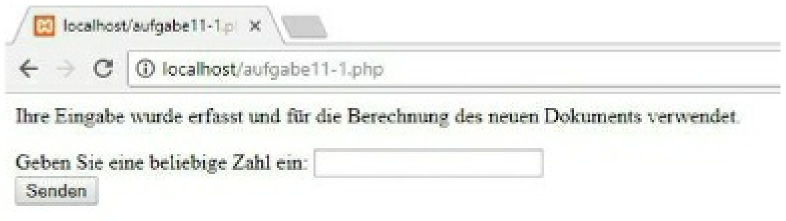
10.4 Exercise: Using files for data storage

1.

Write a program that displays a form field in the browser. This should ask the user to enter any number. Create a text document with different numbers (each on a separate line) and read this with your program. Then multiply these numbers by the value that the user entered via the browser. Then create a new text document and save the results in it. Note: PHP treats the numbers from the form field like text variables. Although it is possible to carry out arithmetic operations with it, a message then appears that this is actually not intended. To avoid this, it is necessary to convert the variable into a number beforehand. This is done with the intval () command .

Solution:

```
<? phpif (! empty ($ _ REQUEST ['number'])) {print "<p> Your input has been recorded and used to calculate the new document. </p> \ n";}?> <form method = " post "action =" task11-1.php "> Enter any number: <input type =" text "name =" number "> <br>
<input type = "submit" value = "Send"> </form> <? phpif (! empty ($ _ REQUEST ['number'])) {$ handle = fopen ("numbers.txt", "r"); $ handle2 = fopen ("result.txt", "w"); $ result = array (); if ($ handle) {$ i = 0; while (! feof ($ handle)) {$ content = fgets ($ handle); $ result [$ i] = $ content; $ i ++;} if ($ _REQUEST ['number']! = "") {$ factor = $ _REQUEST ['number']; foreach ($ result as $ value ) {$ value = intval ($ value) * $ factor; fputs ($ handle2, $ value. "\ n");}} fclose ($ handle); fclose ($ handle2);} else {print "The file could cannot be opened. <br> \ n ";}}?>
```



Ihre Eingabe wurde erfasst und für die Berechnung des neuen Dokuments verwendet.

Geben Sie eine beliebige Zahl ein:

Note: The first if query (before the input form) checks whether there is a value for $ _REQUEST ['number'] - that is, whether an entry has already been made. Therefore, the corresponding notice does not appear the first time the page is accessed. The program only outputs it when the user enters a value and the page loads again.

Chapter 11

Databases: The efficient alternative for data storage

Databases are used to retrieve the data for the execution of a PHP program or to store new values in it. However, this is not the only way to do this. In the previous chapter, files were used for precisely this purpose. For this reason it is interesting to compare both alternatives before starting to use the databases. From this comparison it quickly becomes clear why in practice databases are used almost exclusively. One of the biggest problems with files is that they are not multi-user capable. In the case of a website, however, many visitors often access the data at the same time. Therefore files are not suitable for this task. When using databases, on the other hand, it is not a problem if many users use the data at the same time. Files are also very insecure. If a user receives access rights, he can view and change all content. The structures are also openly visible and it is not a problem to manipulate values manually. In the case of databases, however, the structures are not externally recognizable. That makes it difficult to influence. In addition, there is a very differentiated access protection. Another advantage is the simpler structuring of the data. This is particularly important for extensive projects with large amounts of data

It is easier to keep track of things when using databases, and it is also much easier to access a specific value. When using files, finding a specific entry is a big problem. However, a database solves this task quickly and efficiently. For these reasons, it is advisable to always use databases for Internet applications. Files are actually only used when data is to be exchanged with the user. However, they are not recommended for internal data storage.

11.1 What is a database?

After a brief outline of what a database is in the introductory chapter, this term will be explained in more detail in the following section. As already described, a database always consists of two different levels. On the one hand there is the data level. The raw data, which represent the content of the database, are located here. It is common practice to classify the databases based on the nature of this content. For example, there are numerical databases that contain only numbers. Full text databases are ideal for storing texts, and some readers have probably come across the term image database at some point. There are image files in it. This shows that it is also possible to store complete files in the database. Direct access to this data level is not intended. For one thing, it is not known which structures were used for storage. Hence it is almost impossible to find any specific content. On the other hand, it happens that the data is only available here in encrypted form in order to increase security. Since direct access to the data level is not provided, an additional level must be used for this. This is called the database management system (DBMS) and represents an interface for communication with the user. The DBMS takes care of the

Structuring of the data and offers the possibility of calling up, inserting or changing content. A PHP programmer only communicates with the database at this level. This means that he does not have to worry about the details of structuring the data. A separate language is available for the interaction. In this textbook, the database language SQL is always used for this. It is possible to integrate SQL commands into a PHP program. This then transmits the commands to the DBMS and in this way retrieves the desired data.

11.2 The structure of databases

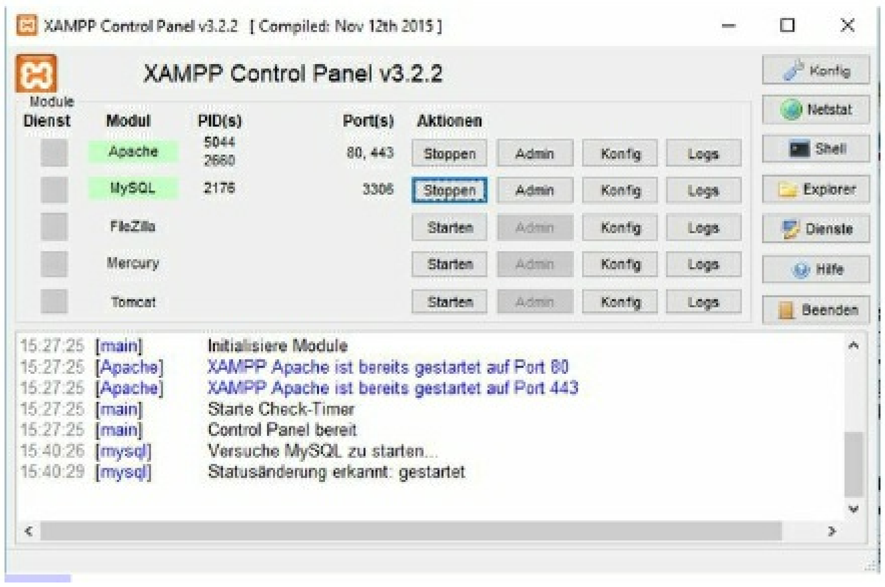
The structure - or the structuring - of databases was also briefly mentioned in the introduction. But here, too, a somewhat more extensive explanation is necessary. In order to work with databases later, it is important to understand their structures exactly. This makes it easier to access the data. The data is structured by the DBMS. Each of these systems uses a fixed, predetermined shape for the arrangement. MySQL is often used for Internet applications. This textbook also works with it. This DBMS only creates relational databases. Most other database systems for Internet applications also use this form of structuring. For this reason, the emphasis is on this model. However, other forms will also be briefly explained. This makes it easier to separate relational databases from the other alternatives. Hierarchical databases are one possibility for structuring the data. These have several levels, each containing different elements. To access an entry, it is necessary to specify the entire path name. For example, if the active athletes are to be saved in a city, it makes sense to sort them first according to the various sports clubs and
then to be divided according to the departments. In order to access a person, it is therefore necessary to enter all of these elements. Hierarchical databases have the disadvantage that they do not allow relationships between the individual entries. Their advantage is that they allow very quick access. Therefore, they are mainly used in applications where high speed is important. Networked databases are also based on the hierarchical database structure. These are structured similarly. However, they allow a particular element to have multiple ancestors. Therefore, there are different paths to get to a particular entry. That brings some new possibilities. For example, in the previous example it was not possible to insert an athlete who is active in multiple clubs or departments. For this a double mention would be necessary. In networked databases, however, it is possible to add several predecessors - i.e. several departments - to an entry. There are also object-oriented databases. These are designed for use with object-oriented programming languages. They allow objects and methods to be saved. In addition, they support the inheritance of methods and properties and they make it possible

to create cross-connections between the individual entries. Although this type of database would be very useful for programs that use objects, it has rarely been used. This is partly due to the fact that the connection to the programs is quite difficult. In addition, these databases are extremely complex, which often results in long waiting times for a query. Relational databases are used most frequently. This form uses many individual tables to store the data. Each table contains a data record that belongs together. The columns indicate what kind of data it is. One line contains
Entries that all belong to the same element. The example of an online shop for power tools has already been given several times in the previous chapters. It is possible to save several products in one table. The individual columns contain information that is important for the characterization of the individual devices - for example the price, the item number and the product description. The individual lines contain the information that applies to a very specific article. This arrangement also shows which information is necessary to access an entry. First of all, it is necessary to specify the table. This is followed by the column and finally the line in which the desired entry is located.

11.3 MySQL: important management system for databases

MySQL is a very popular database management system. The importance of MySQL is already clear from the fact that well-known applications such as Facebook, YouTube or Twitter use this technology. MySQL is preinstalled on most web servers and is therefore almost always available. The XAMPP software that is already in use also contains MySQL. This system is therefore ideal for learning how to use databases and for later creating your own database-based web applications. The Swedish company MySQL AB developed the DBMS in the early 1990s. The first version was presented in 1994. SunMicrosystems bought the provider in 2008 and MySQL has belonged to Oracle since 2010. This is open source software. This means that it is available free of charge and that the source code is freely accessible. In order to start MySQL on your own computer, it is necessary to call up the file XAMPP-control. In contrast to previous applications, it is now not only necessary to start the Apache web server, but also MySQL. There is also a corresponding one for this

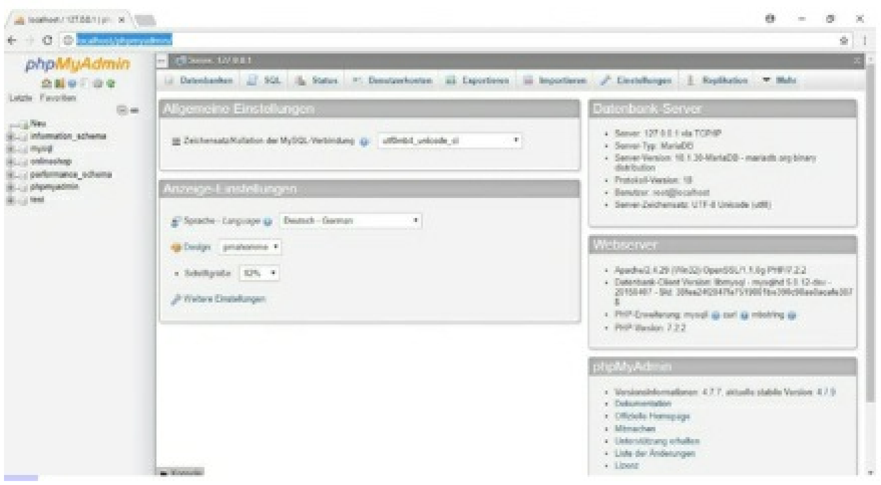
Button present.


XAMPP Control Panel v3.2.2 [Compiled: Nov 12th 2015]
XAMPP Control Panel v3.2.2
Konfig
Netstat
Shell
Explorer
Dienste
Hilfe
Beenden

Module Dienst	Modul	PID(s)	Port(s)	Aktionen			
	Apache	5044 2660	80, 443	Stoppen	Admin	Konfig	Logs
	MySQL	2176	3306	Stoppen	Admin	Konfig	Logs
	FileZilla			Starten	Admin	Konfig	Logs
	Mercury			Starten	Admin	Konfig	Logs
	Tomcat			Starten	Admin	Konfig	Logs

```
15:27:25  [main]     Initialisiere Module
15:27:25  [Apache]   XAMPP Apache ist bereits gestartet auf Port 80
15:27:25  [Apache]   XAMPP Apache ist bereits gestartet auf Port 443
15:27:25  [main]     Starte Check-Timer
15:27:25  [main]     Control Panel bereit
15:40:26  [mysql]    Versuche MySQL zu starten...
15:40:29  [mysql]    Statusänderung erkannt: gestartet
```

Another advantage of MySQL is that it works great with PHP. Among other things, the PHP Database Object (PDO) is used for this purpose. This is an object that contains many functions that are necessary for interacting with the database. This makes it particularly easy for the programmer to integrate the relevant queries. Although access is usually via the PHP program, there are also situations in which direct access to MySQL is necessary. This allows the data to be entered manually, for example change or determine the structures. It also makes it a little easier to learn how to use the databases. Therefore there is a user interface called phpMyAdmin, which enables direct access. To achieve this it is necessary to have the
Enter the URL http: // localhost / phpmyadmin / in the address line of the browser.



11.4 SQL: the database language for working with MySQL

In the previous chapters it was already stated that a PHP program communicates with the DBMS in order to query, insert or change data. In order for this communication to run smoothly, it must use uniform commands. This is where the SQL database language comes into play. When a PHP program sends a command to the DBMS, it must use this language. SQL contains all of the functions required to work with databases, but it is not a programming language. For example, SQL does not provide functions for loops or branches, which are important parts of a full-fledged programming language. SQL's history goes back well into the past century - to the 1970s. It was then that IBM developed the first relational database, which was named System R. For this novel
Database model also required a new language. That is why the company developed SQL - at that time, however, under the name SEQUEL. In 1986 the US standards institute ANSI published a corresponding standard for the first time. In the following year, the international standardization organization (ISO) followed suit. That was the basis for SQL becoming the most widely used database language.

Chapter 12

Basic operations for dealing with MySQL databases

Now that the theoretical basics of the databases have been explained, it is time to put it all into practice. This chapter introduces the basic SQL commands and enables the reader to create databases, create tables and manipulate data. To make this task as simple as it is clear, the SQL commands should be entered via the user interface phpMyAdmin, which also contains Buttons that enable the corresponding actions to be performed without the use of SQL commands. However, since the goal of this exercise is to learn how to use SQL, the actual commands are used. These are also important later to enable communication with a PHP program. However, this represents a further step and is only described in Chapter XIV.

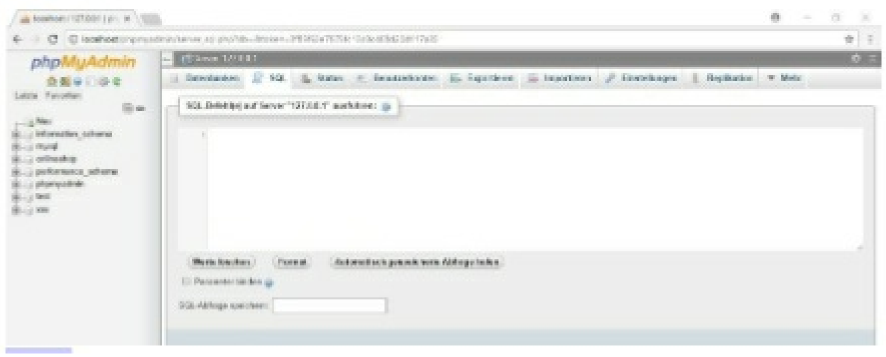
12.1 DDL, DCL and DML: three types of SQL commands

The commands in SQL are divided into three different areas, each with a different function. This ensures a good overview. The three sub-areas are DataDefinition Language (DDL), Data Control Language (DCL) and DataManipulation Language (DML). This already shows what the commands that are summarized in these groups are suitable for.
The commands from the DDL area allow the rough structures of a database to be specified. The `CREATE` command is very important and is usually the first thing to use when working with SQL. This is because it is necessary to create a database. It can also be used to create tables with very specific structures. The opposite of this command is `DROP` . It deletes databases, tables and other elements. Finally, there is the `ALTER` command , which is used to change the existing structures. The commands presented here are all written in capital letters. However, this is not absolutely necessary. SQL also accepts lowercase commands. However, it is customary to use capital letters for this, as these help to keep the layout clear. DCL commands are primarily concerned with access rights. In this way it is possible to control which user has access to the

corresponding data. The GRANT command assigns rights to a specific user . The REVOKE command , however, revokes them. Both commands are often used with the addition of ALL PRIVILEGES . This means that you assign or remove all rights. The commands from the DML area are particularly often used, which includes all functions for entering, deleting or changing data. For example, the SELECT command is very important . This is usually combined with information about the table and the exact field in which the entry is located. In this way it is possible to select a specific entry. This is the prerequisite for outputting the content. The INSERT , UPDATE and DELETE commands are used to insert, change or delete data .

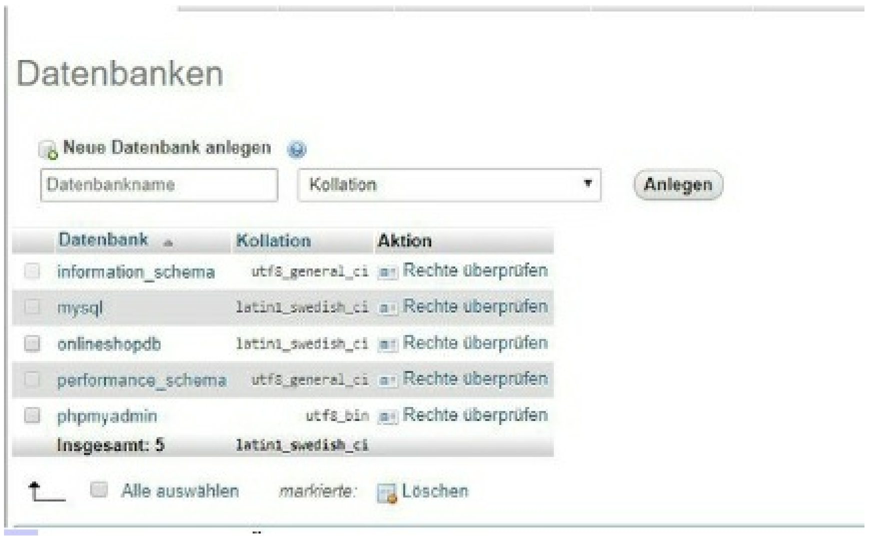
12.2 Create databases and tables

In the following section the reader learns how to create new databases and insert tables in them. The input is supposed to be via SQL commands
the user interface phpMyAdmin can be done. To do this it is necessary to call up this program via the browser (http: // localhost / phpmyadmin /). For the following tasks, it is necessary to select the field labeled SQL in the menu bar .



The corresponding commands can now be inserted in the window that then appears. First it is necessary to create a database. This forms the basis for all other applications. The CREATE command is used for this purpose, which has already been mentioned in the previous section. This is followed by the key term DATABASE, which indicates that MySQL should create a new database. A name is required to clearly define this. This can be chosen freely. Since the following examples simulate applications for an online shop, the database is

to be named onlineshopDB . The appendix DB makes it clear that this is a database. Although this form of naming is not absolutely necessary, it makes it easier to identify the individual elements later. Just like in PHP, it is also necessary in SQL to end the individual commands with a semicolon. The complete command to create the database is therefore: CREATE DATABASE onlineshopDB;
After entering it, it is necessary to click the OK button at the bottom right of the screen. The page then reloads. At first glance, no changes can be seen. Nevertheless, a new database was created as a result. This can be checked by clicking on the “Databases ” field on the far left in the menu bar. The entry onlineshopDB now appears next to the existing databases .



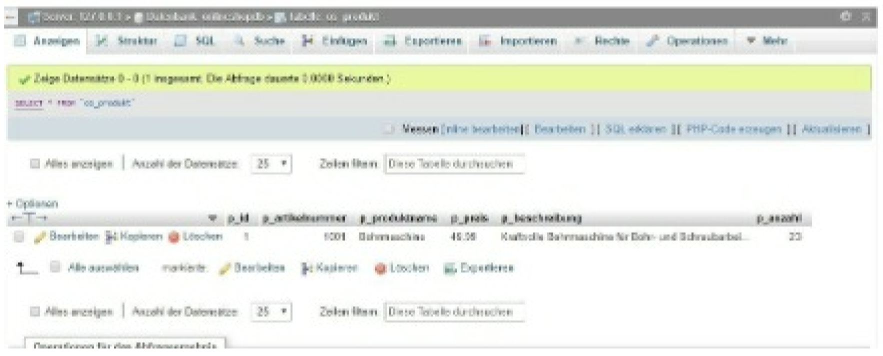
Datenbank	Kollation	Aktion
information_schema	utf8_general_ci	Rechte überprüfen
mysql	latin1_swedish_ci	Rechte überprüfen
onlineshopdb	latin1_swedish_ci	Rechte überprüfen
performance_schema	utf8_general_ci	Rechte überprüfen
phpmyadmin	utf8_bin	Rechte überprüfen
Insgesamt: 5	latin1_swedish_ci	

Many data types can be assigned additional attributes. For example, the addition unsigned to numbers is important. This means that the corresponding field can only contain positive numbers. But the maximum size doubles. Further attributes will be used in the further course of this book, but they will only be explained at the appropriate point. To create a table, it is necessary to choose a name for it. The first table should contain the products of the online shop. Hence it bears the name os_produkte . The prefix os_ indicates that the table belongs to the onlineshopDB database and thus improves clarity. While this is not absolutely necessary, it is highly recommended.
The individual columns now follow. The first column should contain a consecutive index. That makes it easier to access the individual lines. The name of this column should be called p_id . The prefix p_ makes it clear that

this column belongs to the table os_product. Whole numbers of the type integer are used as the data type . Not null , unsigned , auto_increment and primary key are used as attributes . Not null means that this value cannot be null. In computer science, however, this does not refer to the number 0, which can certainly occur. The expression zero means that there is no value. Since an index does not normally take on negative values, the unsigned attribute is used. The addition auto_increment causes the DBMS to automatically assign consecutive numbers for these cells. The primary key attribute means that the data record can be uniquely identified by the values in this column. This is important, for example, for references from other tables. Next, the column p_article number follows as an integer with the attributes not null , unsigned and unique . The first two additions are already known. The last entry means that no identical values may occur in this column. This is important because each item number can only be assigned once. A variable of the type varchar (40) with the attribute not null is used for p_productname . 40 characters should be sufficient even for very long product names. The variable type decimal (7,2) is used for p_preis . The first number in brackets indicates the total number of possible digits. The second number tells how many of them are decimal places. Not null and unsigned are used as attributes . The p_beschreibung column uses the type varchar (300) . Since the description is significantly longer than the product name, more space is required. Attributes are not necessary here. Finally, there is the p_number column - an integer variable with the unsigned attribute. Now it is possible to create the table. Here comes the CREATE command used - but with the addition TABLE . This is followed by the name of the table and finally, within brackets, the individual columns with their variable types and attributes. The individual attributes are only separated from one another by a space. A comma follows the entry for a column:

```
CREATE TABLE os_product (p_id integer unsigned not null auto_increment primary key, p_article
number integer unsigned not null unique, p_productname varchar (40) not null, p_price decimal (7,2)
unsigned not null, p_description varchar (300), p_number integer unsigned);
```

Before entering the command, it is necessary to select the “Databases ” field in the menu bar and then click on the database that has already been created. It is then necessary to switch back to the “SQL ” area and enter the appropriate command. It is then possible to view the new table under the "Structure " menu field .


Anzeigen
Struktur
SQL
Suche
Einfügen
Exportieren
Importieren
Rechte
Operationen
Mehr
Alles anzeigen
Anzahl der Datensätze: 25
Zeilen filtern: Diese Tabelle durchsuchen
+ Optionen
Alle auswählen
markierte: Bearbeiten Kopieren Löschen Exportieren

✓ Zeige Datensätze 0 - 0 (1 insgesamt, Die Abfrage dauerte 0.0000 Sekunden.)

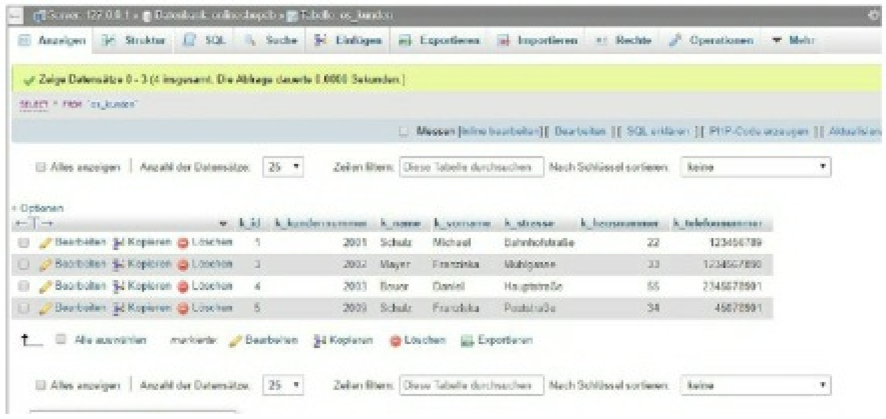
```
SELECT * FROM `os_produkt`
```

	p_id	p_artikelnummer	p_produktname	p_preis	p_beschreibung	p_anzahl
Bearbeiten Kopieren Löschen	1	1001	Bohrmaschine	45.99	Kraftvolle Bohrmaschine für Bohr- und Schraubarbei...	23

12.3 Enter data

As the last screenshot showed, the structure of the table is already

available. However, there are no entries yet. In the next step, it should now be filled with content. The INSERT command is used for this . To enter the data, it is necessary to enter the term INSERT INTO and then the name of the table. The columns in which the corresponding values are entered then follow in brackets. It is then necessary to insert the keyword VALUES , followed by a bracket with the corresponding values, strings of characters in single quotation marks. MySQL also supports double quotation marks, but since this is not the case with all database management systems, it makes sense to get used to the use of single quotation marks right from the start. This results in the following entry: INSERT INTO os_product (p_article number, p_product name, p_price, p_description, p_number) VALUES (1001, 'Drilling machine', 45.99, 'Powerful drilling machine for drilling and screwing work', 23); After this command has been executed, it makes sense to first click on the Structure field in the menu bar and then select the relevant database. You can now see the entries that have just been inserted.


Server: 127.0.0.1 » Datenbank: onlineshopdb » Tabelle: os_kunden
Anzeigen
Struktur
SQL
Suche
Einfügen
Exportieren
Importieren
Rechte
Operationen
Mehr
Messen [Inline bearbeiten] [Bearbeiten] [SQL erklären] [PHP-Code anzeigen] [Aktualisieren
Alles anzeigen
Anzahl der Datensätze: 25
Zeilen filtern: Diese Tabelle durchsuchen
Nach Schlüssel sortieren: keine
+ Optionen
Alle auswählen
markierte:
Bearbeiten
Kopieren
Löschen
Exportieren

Zeige Datensätze 0 - 3 (4 insgesamt, Die Abfrage dauerte 0.0000 Sekunden.)

```
SELECT * FROM `os_kunden`
```

	k_id	k_kundennummer	k_name	k_vorname	k_strasse	k_hausnummer	k_telefonnummer
Bearbeiten Kopieren Löschen	1	2001	Schulz	Michael	Bahnhofstraße	22	123456789
Bearbeiten Kopieren Löschen	3	2002	Mayer	Franziska	Mühlgasse	33	1234567890
Bearbeiten Kopieren Löschen	4	2003	Bauer	Daniel	Hauptstraße	55	2345678901
Bearbeiten Kopieren Löschen	5	2009	Schulz	Franziska	Poststraße	34	45678901

It is striking that for the column p_id no value is entered wurde.Trotzdem the number 1. This is Andem attribute appears in the appropriate column now auto_increment . This ensures that the DBMS automatically inserts a consecutive number. The reader's task now is to insert the values for a few other electrical tools. It is also interesting to use the same item number for two different products, which will result in an error message. This is because this field was given the unique attribute . This prevents the use of the same values.

12.4 Change data

It is often necessary to change a value. The UPDATE command is used for this . This is followed by the name of the table in which the change is to be made. This is followed by the keyword SET. Now it is necessary to specify the individual columns in which a value is to be changed. This is followed by an equal sign and the new value. Now the programmer has to determine the line in which the change is to be made. The easiest way to do this is to use the index number. The following entry changes the price for the drill:

```
UPDATE os_product SET p_preis = 40.99 WHERE p_id = 1;
```

It is possible to adjust several columns with a single command, in which case it is necessary to separate the individual entries with a comma. The following entry changes the number of products available and the description:

```
UPDATE os_produkt SET p_beschreibung = 'High quality drill today on special offer', p_nummer = 11
WHERE p_produktname = 'Drill';
```

In this example, the row cannot be accessed via the index, instead of the product name. This is to make it clear that the content of any column can be used for this. However, great care should be taken here, as it is possible that the same value appears in several lines. In these cases the command would recreate the entries in all relevant lines. With the UPDATE command it is also possible to fill empty fields. If, for example, the price has not yet been fixed when entering a new product, it is possible to leave this field empty. Later, the user can easily insert a suitable value with the UPDATE command.

12.5 Delete data

When using a database it happens again and again that an entry has to be deleted. For example, if a particular item is no longer in production, it makes sense to completely remove it from the database. This can be done with the DELETE command . To delete a row in a table, the programmer must insert the term DELETE FROM followed by the name of the table. This is followed by the key word WHERE . Finally it is necessary to enter a condition. The index number is again suitable for this. With the condition p_id = 2 , for example, the row with the index 2 can be completely deleted. The complete command for this operation looks like this: DELETE FROM os_product WHERE p_id = 2; In some cases it is necessary to delete multiple lines. A separate command is not necessary every time. This can be achieved with the addition IN . Then the index numbers of all lines that are to be removed must be in brackets. For example, the following command would be required to delete lines 1, 4 and 7:
DELETE FROM os_product WHERE p_id IN (1, 4, 7); When deleting data, the term WHERE is very important. If the programmer forgets this, the command deletes all entries in the entire table. This is irreversible and can result in serious data loss. If the DELETE command is entered without a condition, all entries are deleted, but the table remains with its basic structure. If it is to be removed completely, the DROP command is necessary. With DROP TABLE os_product; the entire table can be removed. With this command, it is also possible to remove the entire database. The command DROP DATABASE onlineshopDB; necessary.

12.6 Manage access rights

One of the great advantages of using databases is that the access rights can be assigned in great detail. For example, if a PHP program is later created that works with the database, it is necessary for it to provide a user name and identify itself. By assigning rights, you can precisely define which actions the program is allowed to carry out. This leads to a high level of security. In order to assign rights, it is first necessary to add a user. This is done with the command: CREATE USER 'user' @ 'localhost' IDENTIFIED BY 'password'; The programmer has to replace the terms 'user' and 'password' with the desired username and password. In the next step, the new user can be given certain rights
assign. This is done with the term GRANT . This is followed by the description of the rights followed by the keyword ON . Now it is necessary to enter the database to which the assignment of rights relates. Finally, the term TO and the corresponding username must be inserted. This command can look like this: GRANT ALL PRIVILEGES ON onlineshopDB. * TO 'user' @ 'localhost'; If the expression ALL PRIVILEGES appears, all rights to the database are transferred. However, it is also possible to assign only certain rights - for example for the CREATE , DELETE , INSERT or UPDATE commands . All you have to do is insert the appropriate commands instead of the phrase ALL PRIVILEGES . Also the symbols. * need an explanation after the name of the database. As a result, the new rights apply to the entire database. Instead of the asterisk, however, it is also possible to insert the name of a table. Is here onlineshopDB . os_product , the user only receives the rights to handle this table . The REVOKE command is used to revoke rights . Its syntax is almost identical to that of the GRANT command . However, the expression FROM must be used instead of the term TO . For example, if it is necessary to revoke the rights for the DELETE command on the os_product table , the command is used: REVOKE DELETE ON onlineshopDB.os_product FROM 'user' @ 'localhost';

12.7 Exercise: Creating a table in MySQL

1. Create a new table within the onlineshopDB database using SQL commands . This should record the customer data. It should therefore include an index, a customer number, the first name, the last name, the street,
contain the house number and a telephone number. Think about which data types should be used for this. 2. Think of a few names and addresses and fill the table with content. 3. Change the house number of the customer in the

first line. Solutions:

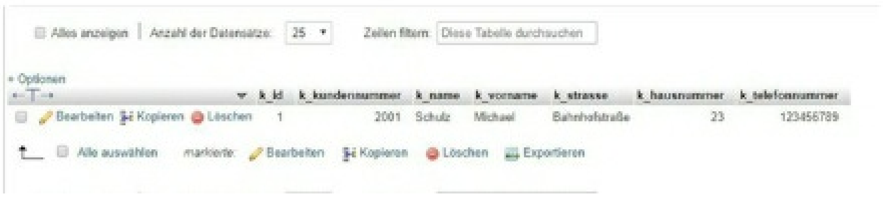
1.

CREATE TABLE os_kunden (k_id integer unsigned not null auto_increment primary key, k_kundennummer integer unsigned not null unique, k_name varchar (40), k_first name varchar (40), k_strasse varchar (60), k_hausnummer smallint unsigned, k_telefonnummer integer unsigned);

Integer variables are used for the numerical values . If no negative values occur, it is possible to define them as unsigned . The smallint type was only chosen for the house number , as this usually does not have very high values. Entries of the type varcha r with a suitable length are used for all texts , but this is only a suggested solution. It is also possible to use other types.

2.

INSERT INTO os_kunden (k_kundennummer, k_name, k_forname, k_strasse, k_hausnummer, k_telefonnummer) VALUES (2001, 'Schulz', 'Michael', 'Bahnhofstraße', 23, 0123456789);


Alles anzeigen
Anzahl der Datensätze: 25
Zeilen filtern: Diese Tabelle durchsuchen
Optionen
Alle auswählen
markierte: Bearbeiten Kopieren Löschen Exportieren

	k_id	k_kundennummer	k_name	k_vorname	k_strasse	k_hausnummer	k_telefonnummer
Bearbeiten Kopieren Löschen	1	2001	Schulz	Michael	Bahnhofstraße	23	123456789

After the same pattern, it is possible to add many more customers with different personal data.

3.

UPDATE os_kunden SET k_hausnummer = 22 WHERE k_id = 1;

Chapter 13

Other important SQL functions

The previous chapter described the essential basics of the SQL database language. Functions for creating databases and for inserting, changing or deleting entries were presented. These commands are essential for manipulating data, but there are many other functions in SQL that go far beyond these capabilities. The following chapter introduces some commands that allow many other applications when using SQL databases. The following explanations use the table `os_kunden` , which was created in the previous exercise, as an example . If this does not yet exist, it makes sense to create it using the solutions given and to fill it with some content. In this way the reader can try out the commands described for himself.

13.1 Select subsets of the data records

When dealing with databases, it is not only necessary to enter data in the tables, to change or delete them. It is also very important to query the values. For example, when a website gets its content from a database, it has to query many different entries. The `SELECT` command is used for this. This selects certain fields and displays their values. The following command is used to display the contents of an entire table:

```
SELECT * FROM os_customers;
```

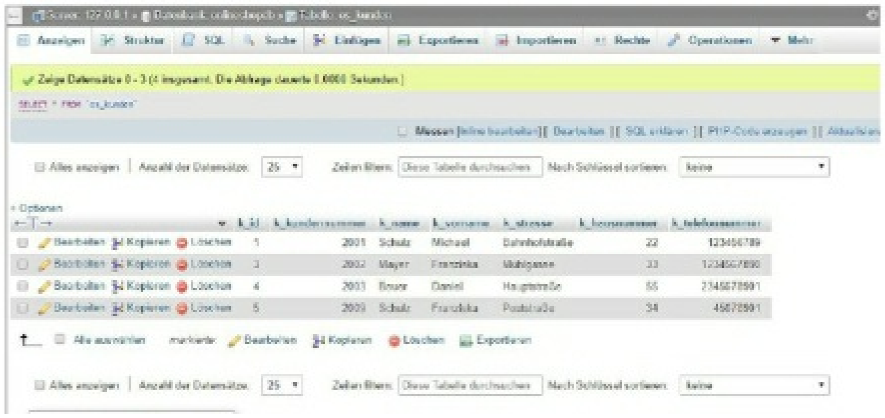
The asterisk indicates that all entries are to be displayed in the corresponding table. After the keyword `FROM` the name derTabelle follows. However, it is not always desirable to get all of the existing values. Often only a small fraction of them are required. The `SELECT` command therefore offers the option of selecting very specific subsets. For this purpose, it is possible to replace the asterisk with the name of a column. The command `SELECT k_name FROM os_customers;` displays all entries in the Name column, for example. It is not only possible to provide an indication. The programmer can also specify multiple columns. These must be separated by a comma: `SELECT k_customer number, k_name, k_first name FROM os_customers;` With the commands presented so far, it is possible to call up the values in one or more columns. However, it is often necessary to use the contents of a particular cell. For this purpose it is necessary to also specify the appropriate line. As a rule, the index variable is used again for this. Before this, the keyword WHERE must be :

```
SELECT k_name FROM os_customers WHERE k_id = 2;
```

This command returns the name of the customer. This is stored in the second row of the table. It is also possible to use logical operators for these queries. Often, for example, the logical or is used. This is represented in SQL by the term OR . This makes it very easy to query entries from several lines. For example, the following command returns the values of the third and fourth lines:

```
SELECT k_name FROM os_ customers WHERE k_id = 3 OR k_id = 4;
```

It is also possible to use the logical and. An example using the following table should illustrate when this is useful:



✓ Zeige Datensätze 0 - 3 (4 insgesamt, Die Abfrage dauerte 0.0000 Sekunden.)

```
SELECT * FROM `os_kunden`
```

k_id	k_kundennummer	k_name	k_vorname	k_strasse	k_hausnummer	k_telefonnummer
1	2001	Schulz	Michael	Bahnhofstraße	22	123456789
3	2002	Mayer	Franziska	Mühlgasse	33	1234567890
4	2003	Bauer	Daniel	Hauptstraße	55	2345678901
5	2009	Schulz	Franziska	Poststraße	34	45678901

In this example, a salesperson has the name of the customer FranziskaSchulz available and he needs her customer number. If he only retrieves this entry based on the surname, this does not lead to a unique result, as there is also an entry for Michael Schulz. The same result occurs when searching using the first name, as the customer Franziska Mayer is also included in the database. To get to the desired customer number, the following command is therefore necessary: SELECT k_kundenennummer FROM os_kunden WHERE k_name = 'Schulz' ANDk_forname = 'Franziska'; This uses the logical operator AND . This means that only entries are displayed for which both the first and last name match the corresponding values.

13.2 Determine the number of entries

For many programs it is not only necessary to fetch the contents; it is also often necessary to find out how many entries there are in a certain area. This not only enables different statistical values to be determined for the data. In addition, these values are also useful for performing various other operations within the programs. The SELECT function is also used for this function . However, the addition COUNT must be followed by the name of the corresponding column. Finally, insert the FROM command and the name of the table: SELECT COUNT (k_name) FROM os_customers; This returns the number of all entries contained in the table os_kunden in the column k_name . But even here it is possible to define certain subsets. The WHERE command is used again for this purpose . Various conditions can be specified here. For example, to determine all customers with the surname Schulz, the following command is required: SELECT COUNT (k_name) FROM os_kunden WHERE k_name = 'Schulz'; This command not only makes it possible to use the entries from the column named in brackets after the COUNT command. For example, it is also possible to use the command SELECT COUNT (k_name) FROMos_kunden WHERE k_first name = 'Michael'; to use. This also leads to the correct result, although it names two different columns. Here, too, it is possible to insert logical operators. In this way, not only entries with different values can be added up. These operators can also be used to determine how often a certain combination - for example of first and last names - is present.

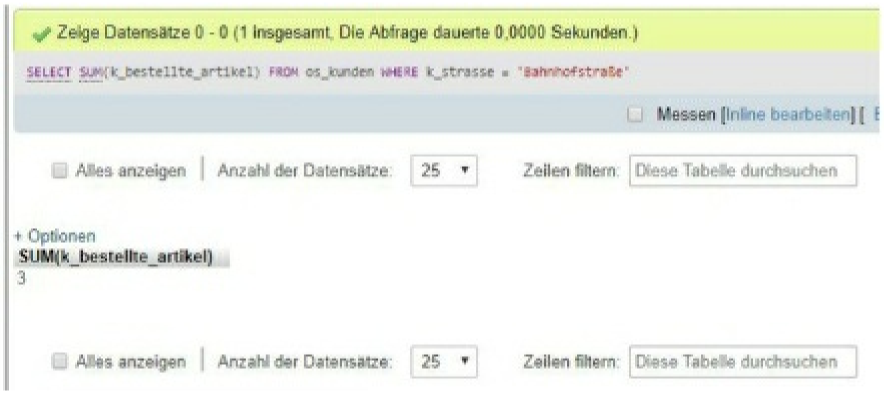
13.3 Adding up values stored in cells

In many cases it is not only necessary to know the number of fields that meet a particular condition. If there are numbers in it, it is often necessary to add up the contents. The database os_kunden there are indeed some columns with numeric values, it does not dochist it makes sense to add the values. The sum of the house, telephone or customer numbers does not usually result in any gain in knowledge. In order to be able to explain this function with a meaningful example, another important command should first be introduced at this point, which can change the structure of a table or another element of the database. The following command line is used for this:

```
ALTER TABLE os_customers ADD COLUMN k_ordered_articles integerunsigned;
```

The term ALTER is always used when the basic structure of an element of the database is to be changed. This is followed by the type of element concerned. Since in this case a table is to be extended by an additional column, the key term TABLE is used here, followed by the name of the table and finally the command ADD COLUMN, which adds another column to the table. As with the original definition, the name of the column, the variable type and, if required, the corresponding attributes must now be inserted. With the above command, the table receives an additional column with the name k_ordered_article. As an exercise for the reader, it remains at this point to fill this column with content using the UPDATE command. After the corresponding column has been added, it is possible to add the contents of the entire column. In this way, the online shop operator can check how many articles the corresponding customers have ordered. The following command is required for this: SELECT SUM (k_ordered_article) FROM os_customers;

The SELECT SUM command creates the sum of the contents of the selected fields. This is followed by the column to which the command relates in brackets. After the keyword FROM is the table from which the values are to be used. This command also allows the results to be further restricted. For the delivery of the articles it can be important, for example, to know the number of products ordered for which the recipient lives in a certain city or on a certain street. This selection is also very easy with SQL. To do this, the already known key term WHERE and a condition must be behind the function : SELECT SUM (k_ordered_articles) FROM os_kunden WHERE k_strasse = 'Bahnhofstrasse';



```
✔ Zeige Datensätze 0 - 0 (1 insgesamt, Die Abfrage dauerte 0,0000 Sekunden.)
SELECT SUM(k_bestellte_artikel) FROM os_kunden WHERE k_strasse = 'Bahnhofstraße'
```

Messen [Inline bearbeiten] [
Alles anzeigen
Anzahl der Datensätze: 25
Zeilen filtern: Diese Tabelle durchsuchen
+ Optionen

```
SUM(k_bestellte_artikel)
3
```

it receives the result 5 as the return value. Subtractions, multiplications and divisions can be carried out in the same way. The operators are the same as in

PHP. Mathematical operations can also be easily incorporated into the various commands. To address a particular line, it is not absolutely necessary to enter the index number directly. This can also be the result of a mathematical operation:

```
UPDATE os_kunden SET k_hausnummer = 22 WHERE k_id = 1 + 2;
```

This command changes the house number in the column with the index number 3, since this is the result of the term 1 + 2. Mathematical operations are also very often used to increase or decrease the value of a certain cell. In an online shop, for example, it often happens that a customer orders another item. In this case it is necessary to increase the value of the column k_ordered_article by one unit. For this purpose it is possible to use the following command:

```
UPDATE os_kunden SET k_ordered_artikel = k_bestelte_artikel + 1WHERE k_id = 3;
```

This command shows that SQL uses the names of fields in a similar way to variables in a programming language. To change the content, it is possible to use an arithmetic expression in which the field name refers to its previous value.

13.4 Exercise: Retrieving information from an inventory

1. For the following tasks, switch back to the os_product table that was created in Chapter 12. If this no longer exists or is empty, create it according to the specifications made for this and fill the individual fields with content. In the following task, the shop operator wants to determine which items he needs to reorder for. To do this, you should write an SQL command that determines the article number of the products with fewer than ten units. In addition, the retailer wants to know how many products the stock has dropped to 0 in order to remove the corresponding offers from the range. Provide an SQL command that will allow you to determine the number of items that are out of stock. 3. Since the capacities in his warehouse are limited, the retailer wants to keep products that are available in very high numbers in another location. In order to find out how many items are affected by this, an SQL command is to be generated which returns the sum of all

units, of which at least 30 pieces are in stock. 4. A new delivery of goods has now arrived that contains 25 drilling machines. Generate an SQL command that adjusts the number of units in stock for this product accordingly. Solutions: 1. SELECT p_article number FROM `os_product` WHERE p_number <10; 2. SELECT COUNT (p_number) FROM os_product WHERE p_number = 0; 3. SELECT SUM (p_number) FROM os_product WHERE p_number> = 30; 4. UPDATE os_produkt SET p_nummer = p_nummer + 25 WHEREp_produktname = 'Drill';

Chapter 14

Integrate SQL database in PHP

In the previous chapters, the SQL commands for creating and manipulating the databases were entered directly via phpMyAdmin. However, this method is rarely used in practice. The user interface also has options for performing the corresponding actions without SQL. This is much easier and usually faster. The method used so far was only used to learn about and practice using various SQL commands. Although there are practical alternatives to using SQL queries when accessing the database directly, it is very important to learn and internalize the corresponding terms. This is due to the fact that in many applications the database is not accessed via phpMyAdmin, but via a PHP program. This makes it possible to automate the corresponding processes so that the software independently queries the data from the database or enters it in it. SQL is used for the communication between the PHP program and the database. For this reason, the commands presented in the previous chapters are indispensable tools for PHP programmers. The following sections show how a database can be connected to a PHP program. They also explain how the SQL commands are integrated into the software in order to enable smooth communication with the database.

14.1 Connect the PHP program to a database

In order to access a database via a PHP program, it is necessary to authenticate - that is, to log in with a username and password. To do this, it is first necessary to create a user name via phpMyAdmin and to assign all rights to it. How this works was explained in Chapter 12.6. Using this procedure, a user with the name user1 and the password abc should be created for the following programs. In real applications that are not only used for training purposes, it is of course sensible not to choose such a simple password in order to ensure security. In the next step, it is necessary to establish a connection between the PHP program and the database. There are several ways to do this. A very simple method is presented in the first example. This is for the purpose of illustration, although this method is generally not used in practice. Then another option is presented, which is a bit more abstract, but which offers many advantages in terms of editing. For the first example, mysqli functions are used. This makes it possible to create a connection to a database and to enter various commands. First it is necessary to establish the connection using the mysqli_connect () command . Just like opening a file, this returns a handle. This must be captured in a variable. This example uses the name $ dbh - as an abbreviation for Data Base Handle. First the host and then the user name, the password and the name of the database concerned are shown in brackets. This results in the following program:

```
<? php $ dbh = mysqli_connect ("localhost", "user1", "abc", "onlineshopDB"); if
(mysqli_connect_errno ()) {print "No connection to the database possible:" .mysqli_connect_error ();
} else {print "Connection successfully established.";} mysqli_close ($ dbh);?>
```

After the command for the connection has been created, it is a good idea to verify that this operation was successful. The following if query is used for this. Should an error occur, the function mysqli_connect_errno () returns the value true . In this case it is important to generate an error message. If the corresponding user was created correctly via phpMyAdmin and if all data were entered correctly, the message “Connection successfully established. ” Appear. For practice purposes, it makes sense to change the username or password to an invalid value. If the program is now executed, a corresponding error message appears. After the program has ended, it is important to close the connection again with the command mysqli_close () . This method is rarely used in practice. The main reason for this is that the individual commands often require a great deal of program code that is often repeated. However, there are methods that have a slightly higher level of abstraction and therefore make many tasks much easier. An example of this is PDO. This abbreviation stands for PHP Database Object. To generate a handle for a database with PDO, only the following command line is necessary:

```
$ dbh = new PDO ("mysql: dbname = onlineshopDB; host = localhost", "user1", "abc");
```

Since this is an object, it must be created with the keyword “ new” . The required class PDO is
already stored in the PHP library so that the user does not have to worry about it. The key term mysql appears in brackets: dbname = followed by the name of the database. The term host = localhost follows, separated by a semicolon . This is followed by the user name and password - each separated by a comma. The complete program for connecting to the database looks like this:

```
<? phptry {$ dbh = new PDO ("mysql: dbname = onlineshopDB; host = localhost", "user1", "abc");
print "Connection successfully established."; $ dbh = null;} catch (PDOException $ e ) {print $ e->
getMessage ();}?>
```

When using PDO, too, it makes sense to check whether the connection was successful. However, this does not happen with an if query, but with exception handling. The keyword try is used for this. If the commands in the following curly braces do not cause problems, the interpreter will carry them out as normal. However, if this is not the case - because a connection to the

database was not possible - it creates an exception. This must then be caught by the term `catch` . The commands used for this cannot be explained further at this point. All that needs to be said is that they cause an appropriate error message to appear if the connection was not possible. In addition, the command `$ dbh = null;` to explain. As already explained in one of the previous chapters, the term `zero` in computer science means
that there is no value at all. Assigning the value `null` to the handle destroys it. The PDO then automatically terminates the connection.

14.2 Fill the database

In the following section, a database is to be filled with content. At the beginning, the `mysqli` functions are used again because they are particularly clear. After that, however, the same steps should also be carried out with PDO in order to show how such a program could look in practice. First, it is necessary to create a new table. This should be given the name `os_bestellungen` and contain all open orders. The following `SQL` command must be used for this, analogous to the method presented in Section 12.2:

```
CREATE TABLE os_bestellungen (b_id integer unsigned not null auto_increment primary key,
b_article number integer unsigned not null, b_customer number integer unsigned not null, b_preis
decimal (7,2) unsigned not null, b_nummer integer unsigned);
```

Now it is necessary to send this command to the database. This is done with the `mysqli_query ()` command . In the brackets you will find the name of the handle and then the corresponding SQL command in quotation marks. Since this is very long in this example, it makes sense to use the dot operator to distribute it over several program lines.

```
<? php $ dbh = mysqli_connect ("localhost", "user1", "abc", "onlineshopDB"); if
(mysqli_connect_errno ()) {print "No connection to the database possible:" .mysqli_connect_error ();}
else
{print "Connection successfully established."; mysqli_query ($ dbh, "CREATE TABLE
os_bestellungen". "(b_id integer unsigned not null auto_increment primary key,". "b_article number
integer unsigned not null,". "b_customer number integer unsigned not null," . "b_preis decimal (7,2)
unsigned not null,". "b_ number integer unsigned);");} mysqli_close ($ dbh);?>
```

After running the program, it is recommended to call up phpMyAdmin again to check. When listing dervorhandenen tables in the database `onlineshopDB` the new table should now os_bestellungen appear.

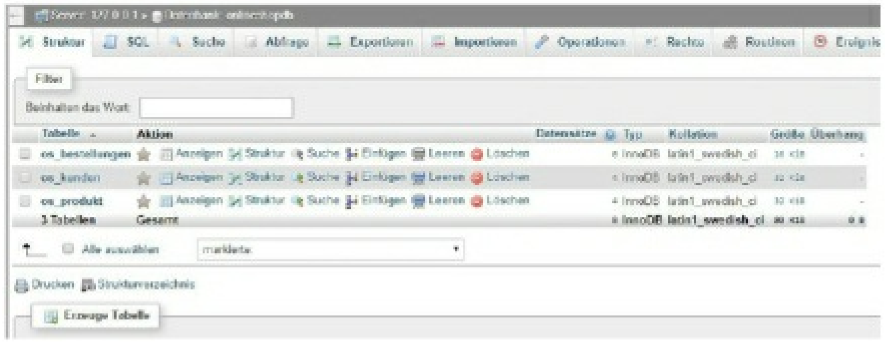


Tabelle	Aktion	Datensätze	Typ	Kollation	Größe	Überhang
os_bestellungen	Anzeigen Struktur Suche Einfügen Leeren Löschen	0	InnoDB	latin1_swedish_ci	16 KiB	-
os_kunden	Anzeigen Struktur Suche Einfügen Leeren Löschen	0	InnoDB	latin1_swedish_ci	32 KiB	-
os_produkt	Anzeigen Struktur Suche Einfügen Leeren Löschen	4	InnoDB	latin1_swedish_ci	32 KiB	-
3 Tabellen	Gesamt	4	InnoDB	latin1_swedish_ci	80 KiB	0 B

Now it is necessary to fill the individual cells. The mysqli_query command is also used for this . The only thing that changes is the SQL command used:

```
<? php $ dbh = mysqli_connect ("localhost", "user1", "abc", "onlineshopDB"); if
(mysqli_connect_errno ()) {print "No connection to the database possible:" .mysqli_connect_error ();
} else {print "Connection successfully established."; mysqli_query ($ dbh, "INSERT INTO
os_bestellungen". "(b_article number, b_customer number, b_price, b_ number)". "VALUES (1011,
2001, 45.99, 1);");}


mysqli_close ($ dbh);?>
```

Even after running this program, it is helpful to call up the corresponding table in phpMyAdmin and to check whether the entries have been added. In order to insert several lines into the table, it is also possible to insert the mysql_query command several times into the program and to adapt the values accordingly. In the following, the actions just described are to be carried out with PDO. In order to check the results, it is advisable to delete the table that has just been created manually via phpMyAdmin. When using PDO, it makes sense to always save the corresponding SQL command in a variable. In this way, only minimal adjustments are necessary to change the functions of the program. Execution takes place via the query method, which is already defined as standard in PDO. To execute this, the usual command for objects is used ($ dbh-> query). It is only necessary to transfer the corresponding SQL command as a value. Since this has already been saved in a variable, the programmer only needs to name the variable:

```
<? php $ sql = "CREATE TABLE os_bestellungen". "(b_id integer unsigned not null auto_increment
primary key,". "b_article number integer unsigned not null,". "b_customer number integer unsigned not
null,". "b_price decimal (7.2 ) unsigned not null, ".
"b_ number integer unsigned);"; try {$ dbh = new PDO ("mysql: dbname = onlineshopDB; host =
localhost", "user1", "abc"); print "Connection successfully established."; $ dbh-> query ($ sql); $ dbh =
null;} catch (PDOException $ e) {print $ e-> getMessage ();}?>
```

With PDO it is also relatively easy to check whether an error has occurred during the transmission of the data. This is done using the errorInfo () function , which is also a method of PDO and returns an array. The first field contains an SQL error code. If no problems occurred, this has the value 0. Therefore, the content of the if query is only executed if there is an error. The second field is output, which contains another error code. Finally, the third field is displayed, which contains a description. The corresponding check should be integrated in the following program. To fill the table with content, it is only necessary to change the value of the variable $ sql. All other components of the program remain the same:

```
<? php $ sql = "INSERT INTO os_bestellungen". "(b_article number, b_customer number, b_price, b_
number)". "VALUES (1011, 2001, 45.99, 1);"; try {$ dbh = new PDO ("mysql: dbname =
onlineshopDB; host = localhost "," user1 "," abc "); print" Connection successfully established. "; $
dbh-> query ($ sql); $ error = $ dbh-> errorInfo (); if ($ error [0]> 0) {
print "Error code:". $ error [1]. "<br>". $ error [2];} $ dbh = null;} catch (PDOException $ e) {print $
e-> getMessage ();}?>
```

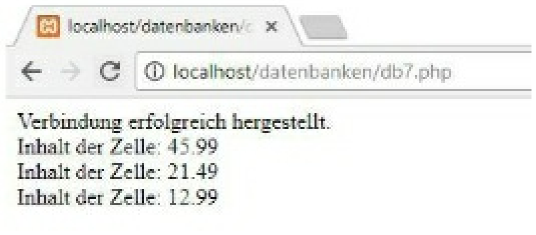
14.3 Delete or change entries

The previous section showed that the query method is always used to execute various SQL commands, the only difference being in the content of the $ sql variable . It is therefore also very easy to adapt the corresponding programs for other tasks. All that is necessary is to copy the program code from the last example. The programmer then has to fill the content of the variable $ sql with a new SQL command. How this must be designed for changing or deleting entries has already been explained in Chapter 12. In order to carry out the desired actions, it is only necessary to adapt these commands to the table that is used in this example and then to assign them to the variable $ sql . To delete the first line, the following assignment would be necessary: $ sql = "DELETE FROM os_bestellungen WHERE p_id = 1;"; To change the content of a cell, it is only necessary to adapt the content of this variable. If the customer has ordered an additional product, the programmer has to save the following value in the variable: $ sql = "UPDATE os_bestellungen SET b_nummer = b_nummer + 1 WHERE b_id = 1;";

14.4 Query data

Internet sites often get their content from databases. For this reason it is very often necessary to call the corresponding entries. This is also possible with the query () method. However, it is not sufficient to simply execute the corresponding SQL command. In addition, it is necessary to save the contents of the cells in variables or to output them directly on the screen. One possibility is to use a foreach loop for this purpose . The database query is treated like an array and inserted directly into the condition of the loop. It is then possible to access the individual contents. This is done by the name of the corresponding column:

```
<? php $ sql = "SELECT b_preis FROM os_bestellungen;"; try {$ dbh = new PDO ("mysql: dbname = onlineshopDB; host = localhost", "user1", "abc"); print "Connection successfully established. < br> \ n "; $ result = array (); $ i = 0; foreach ($ dbh-> query ($ sql) as $ content) {print" content of the cell: ". $ content ['b_price']. "<br> \ n"; $ result [$ i] = $ content ['b_preis']; $ i ++;} $ error = $ dbh-> errorInfo (); if ($ error [0]> 0) {print "Error code:". $ Error [1]. "<br>". $ Error [2];} $ dbh = null;} catch (PDOException $ e) {print $ e-> getMessage ();}?>
```



```
Verbindung erfolgreich hergestellt.
Inhalt der Zelle: 45.99
Inhalt der Zelle: 21.49
Inhalt der Zelle: 12.99
```

In addition, there is a somewhat easier way of saving and outputting the values. The PDOstatement method fetchAll () is used for this . This creates an array that contains all the results of the query. To do this, it is first necessary to execute the query with the query () command. In contrast to the previous example, however, this does not take place within a foreach loop. It is important to save the return result in a variable. The command required for this therefore looks as follows: $ return = $ dbh-> query ($ sql); The method fetchAll () is then applied to the return value, which is saved in the $ rueckgabe variable : $ result = $ rueckgabe-> fetchAll (PDO :: FETCH_ASSOC); The result is then stored in an array named $ result . It is always a composite array with the first index numerically structured. In this case the second index is associative and corresponds to the column name. This is ensured by the expression PDO :: FETCH_ASSOC , which is placed in brackets after the fetchAll () command.

Alternatively, there are other options - for example PDO :: FETCH_NUM for output as a numeric array. A foreach loop is again required to display the result on the screen. This now uses the $ result array directly . This creates the following program:

<? php $ sql = "SELECT b_preis FROM os_bestellungen;"; try {$ dbh = new PDO ("mysql: dbname = onlineshopDB; host = localhost", "user1", "abc"); $ rueckgabe = $ dbh-> query ($ sql); $ result = $ return-> fetchAll (PDO :: FETCH_ASSOC); foreach ($ result as $ content) {print "content of the cell:". $ content ['b_price']. "<br> \ n ";} $ error = $ dbh-> errorInfo (); if ($ error [0]> 0) {print" Error code: ". $ error [1]." <br> ". $ error [2] ;} $ dbh = null;} catch (PDOException $ e) {print $ e-> getMessage ();}?>

14.5 Exercise: Editing databases with PHP

1.

Create a PHP program that creates the new table os_reklamationen , which records the complaints. It should record an index, the customer number, the article number and the date on which the complaint was received (note: it makes sense to use the data type DATE for the date . This gives the date in the following format: YYYY-MM-DD). Use the database object PDO for this - and also for the following tasks.

2.

Fill out the table with two complaints.

3.

Design a program that queries the date of the complaint in line 1 and displays it on the screen. To do this, use the fetchAll () method.

1.
<? php $ sql = "CREATE TABLE os_reklamationen". "(r_id integer unsigned not null auto_increment primary key,". "r_customer number integer unsigned not null,". "r_article number integer unsigned not null,". "r_date date);"; try {$ dbh = new PDO ("mysql: dbname = onlineshopDB; host = localhost", "user1", "abc"); $ dbh-> query ($ sql); $ dbh = null;} catch (PDOException $ e) {print $ e-> getMessage ();}?> 2. <? php $ sql = "INSERT INTO os_reklamationen". "(r_customer number, r_article number, r_date)". "VALUES (1015, 2005, '2018-03- 25 '); "; $ sql2 =" INSERT INTO os_reklamationen "." (R_customer number, r_article number, r_date) "." VALUES (1013, 2012,' 2018-

```
03-29 '); "; try {$ dbh = new PDO ("mysql: dbname = onlineshopDB; host = localhost", "user1",
"abc"); $ dbh-> query ($ sql);
$ dbh-> query ($ sql2); $ error = $ dbh-> errorInfo (); if ($ error [0]> 0) {print "Error code:". $ error
[1]. "<br>". $ error [2];} $ dbh = null;} catch (PDOException $ e) {print $ e-> getMessage ();}?>
```

3.

```
<? php $ sql = "SELECT r_date FROM os_reklamationen WHERE r_id = 1; "; try {$ dbh = new PDO
(" mysql: dbname = onlineshopDB; host = localhost "," user1 "," abc "); $ return = $ dbh-> query ($
sql); $ result = $ return- > fetchAll (PDO :: FETCH_ASSOC); print "Date of the complaint:". $ result
[0] ['r_date']. "<br> \ n"; $ error = $ dbh-> errorInfo (); if ($ error [0]> 0) {print "Error code:". $ error
[1]. "<br>". $ error [2];} $ dbh = null;} catch (PDOException $ e) {print $ e- > getMessage ();}?>
```

Chapter 15

Application example: recording customer data via the

Internet

After the basics of PHP and SQL were presented in the previous chapters, the last chapter is devoted to a small practical example. Since the examples in this book were always geared towards the functions for creating an online shop, this topic will also be taken up in the last section. However, it would be too time-consuming to design a complete shop software with all functions. For this reason, only a small section should be recorded here. The resulting pages are intended to enable a new customer to register. For this purpose, a form field is created through which he can enter his personal information. The program then processes these and transfers them to a database. Once entered, the customer is redirected to a page that displays all the information so that they can review them. If he confirms the information, the data will remain in the database and the customer will be redirected to a new page that will notify him of the successful completion. However, if it indicates that the data is incorrect, the entry is deleted and the data input page reappears. This example focuses primarily on PHP and SQL. To make the task easier, the HTML code is kept as simple as possible. An appealing design is completely dispensed with. Readers with good HTML and CSS knowledge are welcome to design this themselves as a supplement. Since this program will consist of several files, it makes sense to create a new folder for it.

15.1 Customer data via a form field query generated with PHP

In order to query the customer's data, it is first necessary to create a form field. No PHP is actually required for this task. Simple HTML coding is sufficient for this. Nevertheless, PHP and SQL should be used at this point. The purpose of this task is to create a simple page structure that can serve as a template for the entire website. Since many elements are identical, this saves a lot of time. Only the components that change on each page should be designed individually. To do this, the program should call up the relevant content from a database. The rough structure of the page looks like this:

```
<html><head>Meta-TitleMeta-Description</head><body> main heading page content </body>
```

```
</html>
```

The four areas meta title, meta description, main heading and page content should now be replaced by a corresponding variable and output in a print function. In the first three cases, this function should be within the corresponding HTML tags so that the variable only reproduces pure text. The HTML tags should only be included in the corresponding variable for the page content, since they represent part of the individual content. This results in the following code:

```
<html><head> <title> <? php print $ meta_title;?> </title> <meta name = "description" content = <?
php print $ meta_description;? >>
</head><body> <h1> <? php print $ main heading;?> </h1> <? php print $ page content;?> </body>
</html>
```

After the basic structure of the page has been determined, it is necessary to create a table in which the texts are stored. This should be named os_inhalt and, in addition to the index, have four columns with the corresponding contents. It is important to note that the page content can be very extensive. It is therefore necessary to plan enough space for this. To create the table, the following SQL command should be entered inphpMyAdmin:

```
CREATE TABLE os_inhalt (i_id integer unsigned not null auto_increment primary key, i_title varchar
(60), i_description varchar (200), i_headline varchar (100), i_content varchar (2000));
```

In the following step it is necessary to open the database, call up the corresponding entries and save them in the corresponding variables. Since these functions have already been explained in detail in the previous chapters, only the necessary code should appear here:

```
<? php $ sql = "SELECT i_title, i_description, i_headline, i_content FROMos_inhalt WHERE i_id =
1;"; try {$ dbh = new PDO ("mysql: dbname = onlineshopDB; host = localhost", "user1", "abc" ); $
rueckgabe = $ dbh-> query ($ sql); $ result = $ rueckgabe-> fetchAll (PDO :: FETCH_ASSOC); $
meta_title = $ result [0] ['i_title']; $ meta_description = $ result [ 0] ['i_description']; $ main heading =
$ result [0] ['i_headline'];
$ page content = $ result [0] ['i_content']; $ dbh = null;} catch (PDOException $ e) {print $ e->
getMessage ();}?>
```

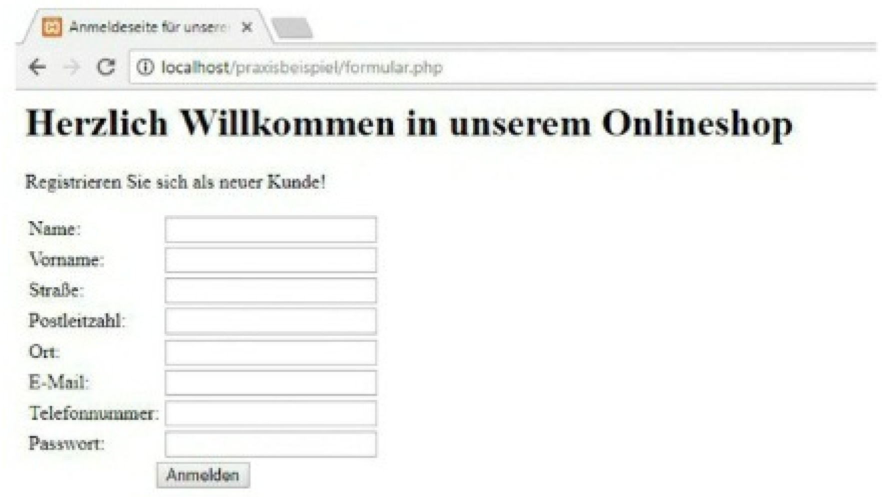
This part only needs to be placed in front of the previously created area with the HTML structure. The page for displaying the form is now ready. Only one small detail is still missing: adding the content to the database. This should again be done directly via phpMyAdmin. For the fields i_title , i_description and i_headline it is only necessary to think of a small text. For the last field, however, a complete form must be created. This should be inserted in a table with two columns for better clarity. On the left is what content should be entered. The corresponding form field follows on the right. The HTML code for the page content therefore looks like this:

```
<p> Register as a new customer! </p> <form method = "post" action = "ausgabe.php"> <table> <tr>
<td> Name: </td> <td> <input name = "name" type = "text"> </td> </tr> <tr> <td> First name: </td>
<td> <input name = "first name" type = "text"> </td> </tr><tr><td>Straße:</td> <td> <input name =
"strasse" type = "text"> </td> </tr> <tr> <td> ZIP code: </td>
<td> <input name = "plz" type = "text"> </td> </tr> <tr> <td> Place: </td> <td> <input name = "place"
type = "text "> </td> </tr> <tr> <td> E-Mail: </td> <td> <input name =" mail "type =" text "> </td>
</tr> <tr > <td> Telephone number: </td> <td> <input name = "tel" type = "text"> </td> </tr> <tr>
<td> Password: </td> <td> < input name = "password" type = "password"> </td> </tr> <tr> <td
colspan = "2" align = "center"> <input type = "submit" value = "login"> < / td> </tr> </table> </form>
```

The page content should now be saved as text in the i_content cell . Although the entry is very extensive, this is possible without any problems, since this cell was previously allocated a correspondingly large space. For an appealing display of the source code it is only necessary to replace all line breaks with the symbol \ n . This results in a very extensive INSERT command:

```
INSERT INTO os_inhalt (i_title, i_description, i_headline, i_content) VALUES ('Login page for our
online shop', 'On this page you can register as a new customer for our online shop', 'Welcome to our
online shop', '<p> Register as a new customer! </p> \ n <form method = "post" action =
"ausgabe.php"> \ n <table> \ n <tr> \ n <td> Name: < / td> \ n <td> <input name = "name" type =
"text"> </td> \ n </tr> \ n <tr> \ n <td> First name: </td> \ n
<td> <input name = "first name" type = "text"> </td> \ n </tr> \ n <tr> \ n <td> Street: </td> \ n <td> <
input name = "street" type = "text"> </td> \ n </tr> \ n <tr> \ n <td> ZIP code: </td> \ n <td> <input
name = "plz" type = "text"> </td> \ n </tr> \ n <tr> \ n <td> Location: </td> \ n <td> <input name =
"location" type = "text"> </td> \ n </tr> \ n <tr> \ n <td> E-Mail: </td> \ n <td> <input name = "mail"
type = "text"> </td> \ n </tr> \ n <tr> \ n <td> Telephone number: </td> \ n <td> <input name = "tel"
type = "text"> </td> \ n </tr> \ n <tr> \ n <td> Password: </td> \ n <td> <input name = "password" type
= "password"> </td> \ n </tr> \ n <tr> \ n <td colspan = "2" align = "center"> <input type = "submit"
value = "Register"> </td> \ n </tr> \ n </table> \ n </form> \ n ');
```

As soon as this command has been entered via phpMyAdmin, the display of the form field is possible:


Anmeldeseite für unsere
localhost/praxisbeispiel/formular.php

Herzlich Willkommen in unserem Onlineshop

Registrieren Sie sich als neuer Kunde!

Name:	
Vorname:	
Straße:	
Postleitzahl:	
Ort:	
E-Mail:	
Telefonnummer:	
Passwort:	

Finally, this page is supposed to transfer the values that are now stored in the object to the database. This is done with a single INSERT command. This should be saved again in the $ sql variable. To find the correct values, it calls the get methods on each individual member.

$ sql = "INSERT INTO os_kunden". "(k_name, k_first name, k_strasse, k_plz, k_ort, k_mail, k_tel, k_passwort, k_ customer number)". "VALUES ('". $ new customer-> getName (). "', '" . $ new customer-> getFirst name (). "','". $ new customer-> getStrasse (). "',". $ new customer-> getPlz (). ",'". $ new customer-> getOrt (). " ',' ". $ new customer-> getMail ()." ', ". $ new customer-> getTel ().",' ". $ new customer-> getPassword ()." ', ". $ new customer → get customer number () . ");";

Note: In the print version, the sequence of single and double quotes required for this command is difficult to recognize. In order to avoid confusion, it should be mentioned that the single quotation marks must always be placed within the double quotation marks. After this, it is necessary to execute the command with the query method . The previous program saves all values in the database and now looks like this:

<? phpinclude ("class_kunde.php"); $ new customer = new customer (); if (! empty ($ _ REQUEST ['name']))) {if ($ _REQUEST ['name']! = "") {$ new customer-> setName ($ _ REQUEST ['name']);}} if (! empty ($ _ REQUEST ['first name'])) {if ($ _REQUEST ['first name']! = "") {$ new customer-> setFirst name ($ _ REQUEST ['first name']);}} if (! empty ($ _ REQUEST ['street'])) {if ($ _REQUEST ['street']! = "") {$ new customer-> setStrasse ($ _ REQUEST ['street']);}} if (! empty ($ _REQUEST ['plz'])) {if ($ _REQUEST ['plz']! = "") {$ New customer-> setPlz ($ _ REQUEST ['plz']);}} if (! Empty ($ _ REQUEST [' location '])) {if ($ _REQUEST [' location ']! = "") {$ new customer-> setOrt ($ _ REQUEST [' location ']);}} if (! empty ($ _ REQUEST [' mail ']))) {if ($ _REQUEST ['mail']! = "") {$ new customer-> setMail ($ _ REQUEST ['mail']);}} if (! empty ($ _ REQUEST ['tel'])) { if ($ _REQUEST ['tel']! = "") {$ new customer-> setTel ($ _ REQUEST ['tel']);}}

```
if (! empty ($ _ REQUEST ['password'])) {if ($ _REQUEST ['password']! = "")
{$ new customer-> setPasswort ($ _ REQUEST ['password']);}} try {$ dbh = new PDO ("mysql:
dbname = onlineshopDB; host = localhost", "user1", "abc"); $ new customer- > setKundennummer ($
dbh); $ sql = "INSERT INTO os_kunden". "(k_name, k_first name, k_strasse, k_plz, k_ort, k_mail,
k_tel, k_passwort, k_ customer number)". "VALUES ('". $ new customer-> getName ( ). "','". $ new
customer-> getFirst name (). "','". $ new customer-> getStrasse (). "',". $ new customer-> getPlz (). ",'".
$ new customer -> getOrt (). "','". $ new customer-> getMail (). "',". $ new customer-> getTel (). ",'". $
new customer-> getPassword (). "',". $ new customer-> get customer number (). ");"; print $ sql. "
<br>"; $ dbh-> query ($ sql); $ dbh = null;} catch (PDOException $ e) {print $ e -> getMessage ();}?>
```

15.4 Displaying customer data with a PHP program

So far it is already possible to call the program formular.php and fill the form with content. The values are transferred to the database, but the output.php page that appears afterwards remains empty. Therefore, in the last step, it is now filled with content. For this purpose, the program should retrieve the values for checking directly from the database and display them on the screen. A `SELECT` command is required for this. The `*` indicates
that he should get the complete line. The line is identified by the customer number that is retrieved from the object. The output then takes place with a `foreach` loop:

```
$ sql = "SELECT * FROM os_customers WHERE k_customer number =". $ new customer-> get
customer number (). ";"; $ return = $ dbh-> query ($ sql); $ result = $ return-> fetchAll (PDO ::
FETCH_ASSOC); foreach ($ result as $ content) {print "<p> For the new customer with the customer
number". $ Content ['k_kundennummer']. "The following values  were recorded: <br>"; print "Name:" .
$ content ['k_name']. "<br> \ n"; print "First name:". $ content ['k_first name']. "<br> \ n"; print
"Street:". $ content [ 'k_strasse']. "<br> \ n"; print "ZIP:". $ content ['k_plz']. "<br> \ n"; print "City:". $
content ['k_ort']. " <br> \ n "; print" E-Mail: ". $ content ['k_mail']." <br> \ n "; print" Telephone
number: ". $ content ['k_tel']." <br> \ n "; print" Password: ". $ content ['k_passwort']." </p> \ n ";}
```

The customer should then be able to confirm or reject the data by inserting two small forms. The first form is very simple. It should only contain the “Confirm ” button and redirect the visitor to a page that shows that the data has been properly recorded. The code for this looks like this:

```
<form method = "post" action = "confirmation.php"> <input type = "submit" value = "confirm">
</form>
```

The following code is required for the form that is used to correct the values:

```
<form method = "post" action = "formular.php"> <input type = "hidden" name = "reference" value = "$
new customer-> get customer number ()">
<input type = "submit" value = "Correct"> </form>
```

If the customer wants to correct the data, he will be redirected to the original form at the address formular.php. The `hidden` form field needs to be explained . This is hidden and is therefore not displayed. It only serves to transmit a value to the next page. Therefore the customer number is inserted with a PHP variable. It should be noted that the code, as it stands here, is still invalid, as HTML and PHP are mixed up. This will be differentiated accordingly later when this part is entered in a `print` function. The transfer of the customer number is necessary because the page to which the users are redirected is to delete the incorrect data. To do this, it needs to know which data set it is. The customer number is therefore transmitted as a reference. To display the two form fields, these must be inserted in a print command. It is important to insert the character `\ in` front of the quotation marks, which are part of the HTML code . This is the only way to ensure correct execution. Now the frame for the page should be inserted. The pattern from Section 15.1 is used for this. It is only necessary to change the index number when querying the database. In addition, the main part should not be inserted via a database, since it is generated directly by the program. For this it is necessary to fill the database `os_inhalt` with a new line via phpMyAdmin. The reader can think of small texts for the title, description and main heading. The `i_content` column remains empty. The program for the output.php page looks like this:

```
<? php $ sql = "SELECT i_title, i_description, i_headline, i_content FROMos_inhalt WHERE i_id =
2;";

try {$ dbh = new PDO ("mysql: dbname = onlineshopDB; host = localhost", "user1", "abc"); $ return =
$ dbh-> query ($ sql); $ result = $ return-> fetchAll (PDO :: FETCH_ASSOC); $ meta_title = $ result
[0] ['i_title']; $ meta_description = $ result [0] ['i_description']; $ main heading = $ result [0]
['i_headline']; $ page content = $ result [0] ['i_content']; $ dbh = null;} catch (PDOException $ e) {print
$ e-> getMessage ();}?> <html> <head> <title> <? php print $ meta_title;?> </title> <meta name =
"description" content = <? php print $ meta_description;? >> </head> <body> <h1> <? php print $
main heading;?> </ h1> <? phpinclude ("class_kunde.php"); $ new customer = new customer (); if (!
empty ($ _ REQUEST ['name'])) {if ($ _REQUEST ['name']! = "") {$ new customer-> setName ($ _
REQUEST ['name']);}} if (! empty ($ _ REQUEST ['first name'])) {if ($ _REQUEST ['first name']! =
"") {$ new customer -> setVorname ($ _ REQUEST ['firstname']);}} if (! empty ($ _ REQUEST
['street'])) {if ($ _REQUEST ['street']! = "") {
$ new customer-> set street ($ _ REQUEST ['street']);}} if (! empty ($ _ REQUEST ['zip code'])) {if ($
_REQUEST ['zip code']! = "") {$ new customer- > setPlz ($ _ REQUEST ['plz']);}} if (! empty ($ _
REQUEST ['location'])) {if ($ _REQUEST ['location']! = "") {$ new customer-> setOrt ( $ _REQUEST
['location']);}} if (! Empty ($ _ REQUEST ['mail'])) {

if ($ _REQUEST ['mail']! = "") {$ new customer-> setMail ($ _ REQUEST ['mail']);}} if (! empty ($ _
REQUEST ['tel'])) {if ($ _REQUEST ['tel']! = "") {$ New customer-> setTel ($ _ REQUEST ['tel']);}}
if (! Empty ($ _ REQUEST ['password'])) {
if ($ _REQUEST ['password']! = "") {$ new customer-> setPasswort ($ _ REQUEST ['password']);}}
try {$ dbh = new PDO ("mysql: dbname = onlineshopDB; host = localhost "," user1 "," abc "); $ new
customer-> set customer number ($ dbh); $ sql =" INSERT INTO os_kunden "." (k_name, k_first
name, k_strasse, k_plz, k_ort, k_mail, k_tel,
k_password, k_ customer number) "." VALUES ('". $ new customer-> getName ()."', '". $ new
customer-> getFirst name ()."', '". $ new customer-> getStrasse ()."' , ". $ new customer-> getPlz ().", '".
$ new customer-> getOrt ()."', '". $ new customer-> getMail ()."', ". $ new customer-> getTel () . ", '". $
new customer-> getPassword (). "',". $ new customer-> get customer number (). ");"; $ dbh-> query ($
sql); $ sql = "SELECT * FROM os_ customers WHERE k_kundenennummer = ". $ NewKunde->
getKundennummer ()."; "; $ Rueckgabe = $ dbh-> query ($ sql); $ result = $ rueckgabe-> fetchAll
(PDO :: FETCH_ASSOC); foreach ($ result as $ inhalt) {print "<p> For the new customer with the
customer number". $ inhalt ['k_kundennummer']. " the following values  were recorded: <br> "; print"
Name: ". $ content ['k_name']." <br> \ n "; print" First name: ". $ content ['k_first name']." <br> \ n ";
print" Street: ". $ content ['k_strasse']." <br> \ n "; print" ZIP: ". $ content ['k_plz']." <br> \ n "; print
"Location:". $ content ['k_ort']. "<br> \ n"; print "E-Mail:". $ content ['k_mail']. "<br> \ n"; print
"Telephone number: ". $ content ['k_tel']." <br> \ n "; print" Password: ". $ content ['k_passwort']."
</p> \ n ";} $ error = $ dbh-> errorInfo (); $ dbh = null;} catch (PDOException $ e) {print $ e->
getMessage ();} print "<form method = \" post \ "action = \" confirmation.php \ "> \ n" . "<inputtype =
\" submit \ "value = \" Confirm \ "> \ n". "</form> \ n". "<form method = \" post \ "action = \"
formular.php \ " > \ n "." <inputtype = \ "hidden \" name = \ "reference \" value = \ "". $ new customer-
> get customer number (). "\"> \ n <input type = \ "submit \" value = \ "correct \"> ".
"</form>";?>

</body>

</html>
```

Now it is necessary to create the confirmation.php page. This uses almost exactly the same code as the form.php page. The only difference is that it is necessary to adapt the index number for the database query:

```
<? php $ sql = "SELECT i_title, i_description, i_headline, i_content FROMos_inhalt WHERE i_id =
3;"; try {$ dbh = new PDO ("mysql: dbname = onlineshopDB; host = localhost", "user1", "abc" ); $
rueckgabe = $ dbh-> query ($ sql); $ result = $ rueckgabe-> fetchAll (PDO :: FETCH_ASSOC); $
meta_title = $ result [0] ['i_title']; $ meta_description = $ result [ 0] ['i_description']; $ main heading =
$ result [0] ['i_headline']; $ page content = $ result [0] ['i_content']; $ dbh = null;
} catch (PDOException $ e) {print $ e-> getMessage ();}?> <html> <head> <title> <? php print $
meta_title;?> </title> <meta name = "description" content = <? php print $ meta_description;? >>
</head> <body> <h1> <? php print $ main heading;?> </h1> <? php print $ page content;?> </body>
</ html >
```

In addition, it is necessary to enter a page title, a description, a heading and a short confirmation message for the successful data storage in the database.

The reader can do this directly via phpMyAdmin according to the familiar pattern. The last task now is to adapt the formular.php file. If the values were not entered correctly and the customer wants to correct them, he will be forwarded to this page. So he can enter it again. However, in this case it is necessary to delete the wrong entry. The following code is used for this:

```
if (isset ($ _ REQUEST ['Reference'])) {$ sql2 = "DELETE FROM os_customers WHEREk_customer
number =". $ _ REQUEST ['Reference']. ";"; $ dbh-> query ($ sql2);}
```

This should be inserted almost at the beginning - immediately after the handle `$ dbh has been` created:

```
<? php $ sql = "SELECT i_title, i_description, i_headline, i_content FROMos_inhalt WHERE i_id =
1;";
try {$ dbh = new PDO ("mysql: dbname = onlineshopDB; host = localhost", "user1", "abc"); if (isset ($
_ REQUEST ['Reference']))) {$ sql2 = "DELETE FROM os_customers WHEREk_kundenennummer =
". $ _ REQUEST ['Reference']."; "; $ Dbh-> query ($ sql2);} $ return = $ dbh-> query ($ sql); $ result =
$ return-> fetchAll (PDO :: FETCH_ASSOC); $ meta_title = $ result [0] ['i_title']; $ meta_description
= $ result [0] ['i_description']; $ main heading = $ result [0] ['i_headline']; $ page content = $ result [0]
['i_content']; $ dbh = null;} catch (PDOException $ e) {print $ e-> getMessage ();}?> <html> <head>
<title> <? php print $ meta_title;?> <
/ title> <meta name = "description" content = <? php print $ meta_description;? >>
</head>
<body> <h1>
<? php print $ main heading;?>
</h1>
<? php print $ page content;?>
</body>
</html>
```

The `if` loop with the `isset` function determines whether a value has been passed to the program. This ensures that this part is not carried out the first time the page is called up, but only when the user wants to correct the values. The `DELETE` command should be known as far as possible. The customer number that was transferred by the hidden form field on the previous page is used as a reference. The program thus fulfills its tasks and the customer can access his data

Chapter 16

PHP and Cookies: Recognizing Visitors and Saving Information

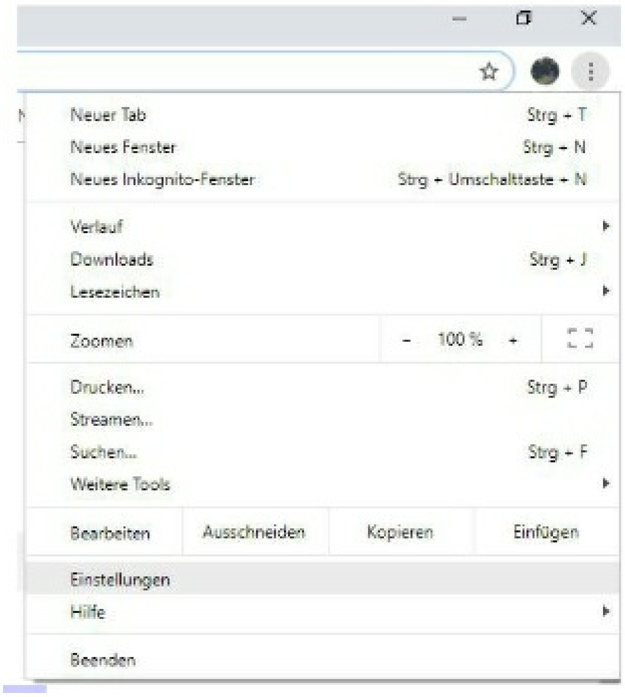
PHP is primarily used to generate dynamic Internet pages. This means that the content that is displayed on the pages is not always the same. On the other hand, they are only generated when the visitor calls up the page. This not only makes it possible to change texts and images quickly and easily. In addition, this technology allows it to be individually adapted to the visitor. If you operate a blog or a news portal, it would be possible, for example, to first present the reader with articles that he has not yet read. The texts that he already knows are usually not of great importance, so that they can be in the background. Another alternative is to include a personal address on the page. This often increases the reader's interest. The individual design is particularly important when developing a web shop. Among other things, it is necessary that the customer can call up which articles he has ordered. In order to tailor the displayed content to the visitor, it is necessary to identify him first. However, HTTP is a connection-free technology. Therefore, every time the user calls up a new page, all data is lost. It is of course possible to store the information permanently in a database. However, in order to personalize the pages, it is necessary to know who is currently visiting the page. This is the only way to call up the associated database contents.
If the user comes from one of their own pages, it is relatively easy to forward the relevant information. We already have a possibility for this in Chapter 15.4. uses: hidden forms. In this example the customer number has been passed on to the next page. It would be possible to expand this information further - for example to make the name, the contents of the shopping cart and other details available. However, it is simpler and safer to store these values in the database. Then the next page only needs the customer number as a reference to be able to access it. However, this method has the problem that it can only be used for the duration of a visit. If the user calls up another page or closes the browser, all information is lost. It makes sense to use cookies so that they are still available on your next visit. This chapter shows how these can be inserted into a PHP page.

16.1 What is a cookie?

The term cookie translates as biscuit or cookie. In computer science, however, it stands for a small data packet that is sent back and forth between different computers in order to exchange information. This is not an executable program, but only data in text form. There are several different types of cookies in computer science. Most often this term refers to HTTP cookies. These are used for data exchange between the web browser and the server. This chapter deals only with HTTP cookies. Other forms are not important for programming with PHP. When using PHP, the cookie is generated by the server and transmitted to the user's web browser. For this it is necessary to insert a corresponding command into the PHP program. It is then saved on the user's computer. The prerequisite for this is that they accept the use of cookies. Another alternative
consists in the fact that the cookie is generated directly in the browser. However, this is not possible with PHP. Client-side scripting languages such as Javascript are necessary for this purpose. Therefore, this alternative is not dealt with further here. If a cookie is stored on the computer, the web browser automatically transmits it to the server when it calls up a page from the server again. This means that the programmer does not have to request the cookie first. He receives it - if it is available - always delivered immediately. The domain whose page it was set is always stored in the cookie. The information is then available for all pages of the domain. It is therefore not necessary to use a new cookie for each subpage. However, it is not possible to access cookies from other domains. Using cookies is extremely practical, but caution is also required. On the one hand, it is important to observe the legal provisions. For the use of cookies it is almost always necessary to obtain the consent of the user. Those who do not run the risk of having to pay substantial fines. Anyone who wants to use cookies on a website should therefore find out exactly what the legal regulations for them look like. On the other hand, there are numerous users who generally block cookies. Therefore, it makes sense to use them only as additional information. A website should, however, function properly even without the use of cookies.

16.2 Set a cookie

Now it is time to generate the first cookie yourself. The setcookie () command is required for this. This is structured as follows: setcookie ("name", "content", expiry date);
The designation can be freely chosen. It is advisable to choose a name that corresponds to the content. For example, if you want to enter the customer number, it would make sense to use this word as a designation. The selected term must always be in quotation marks, followed by the content. At this point it is possible to insert any values. This is followed by the expiry date for the cookie. This parameter is not absolutely necessary. If you leave this area blank, the cookie expires as soon as the user closes the browser. Therefore, the next time you visit, no recognition would be possible and the cookie would be worthless. At this point it is necessary to generate a value according to the Unix timestamp format. The value for the current time can be obtained using the time () function . After that it is only necessary to add the desired validity period in seconds. If the cookie were to be valid for 30 days, for example, you would have to enter the value time () + 2592000 here . Alternatively you can first calculate the minutes, hours and days and then multiply this value by the desired number of days: time () + 60 * 60 * 24 * 30 . If you want to set a cookie that saves the customer number 12345 for 30 days, the following program would be necessary: <? Phpsetcookie ("customer number", "12345", time () + 2592000);?> If you do this now, then initially a blank page appears. Still something happened. You can tell when you call up the browser's cookie data. How this works is shown here using the Google Chrome browser as an example. The procedure is similar for other browsers. To do this, first click on the three points at the top right
Click the edge of the picture and then select the Settings menu item.



Chapter 16

PHP and cookies: Recognize visitors and save information

PHP is primarily used to generate dynamic Internet pages. This means that the content that is displayed on the pages is not always the same. On the other hand, they are only generated when the visitor calls up the page. This not only makes it possible to change texts and images quickly and easily. In addition, this technology allows it to be individually adapted to the visitor. If you operate a blog or a news portal, it would be possible, for example, to first present the reader with articles that he has not yet read. The texts that he already knows are usually not of great importance, so that they can be in the background. Another alternative is to include a personal address on the page. This often increases the reader's interest. The individual design is particularly important when developing a web shop. Among other things, it is necessary that the customer can call up which articles he has ordered. In order to tailor the displayed content to the visitor, it is necessary to identify him first. However, HTTP is a connection-free technology. Therefore, every time the user calls up a new page, all data is lost. It is of course possible to store the

information permanently in a database. However, in order to personalize the pages, it is necessary to know who is currently visiting the page. This is the only way to call up the associated database contents.
If the user comes from one of their own pages, it is relatively easy to forward the relevant information. We already have a possibility for this in Chapter 15.4. uses: hidden forms. In this example the customer number has been passed on to the next page. It would be possible to expand this information further - for example to make the name, the contents of the shopping cart and other details available. However, it is simpler and safer to store these values in the database. Then the next page only needs the customer number as a reference to be able to access it. However, this method has the problem that it can only be used for the duration of a visit. If the user calls up another page or closes the browser, all information is lost. It makes sense to use cookies so that they are still available on your next visit. This chapter shows how these can be inserted into a PHP page.

16.1 What is a cookie?

The term cookie translates as biscuit or cookie. In computer science, however, it stands for a small data packet that is sent back and forth between different computers in order to exchange information. This is not an executable program, but only data in text form. There are several different types of cookies in computer science. Most often this term refers to HTTP cookies. These are used for data exchange between the web browser and the server. This chapter deals only with HTTP cookies. Other forms are not important for programming with PHP. When using PHP, the cookie is generated by the server and transmitted to the user's web browser. For this it is necessary to insert a corresponding command in the PHP program. It is then saved on the user's computer. The prerequisite for this is that they accept the use of cookies. Another alternative
consists in the fact that the cookie is generated directly in the browser. However, this is not possible with PHP. Client-side scripting languages such as Javascript are required for this purpose. Therefore, this alternative is not dealt with further here. If a cookie is stored on the computer, the web browser automatically transmits it to the server when it calls up a page from the server

again. This means that the programmer does not have to request the cookie first. He receives it - if it is available - always delivered immediately. The domain whose page it was set is always stored in the cookie. The information is then available for all pages of the domain. It is therefore not necessary to use a new cookie for each subpage . However, it is not possible to access cookies from other domains. Using cookies is extremely practical, but caution is also required. On the one hand, it is important to observe the legal provisions. For the use of cookies it is almost always necessary to obtain the consent of the user. Failure to do so runs the risk of having to pay substantial fines. Anyone who wants to use cookies on a website should therefore find out exactly what the legal regulations are for. On the other hand, there are numerous users who generally block cookies. Therefore, it makes sense to use them only as additional information. A website should, however, function properly without the use of cookies.

16.2 Set a cookie

Now it is time to create the first cookie yourself. The setcookie () command is necessary for this. This is structured as follows:

```
setcookie ("name", "content", expiry date);
```

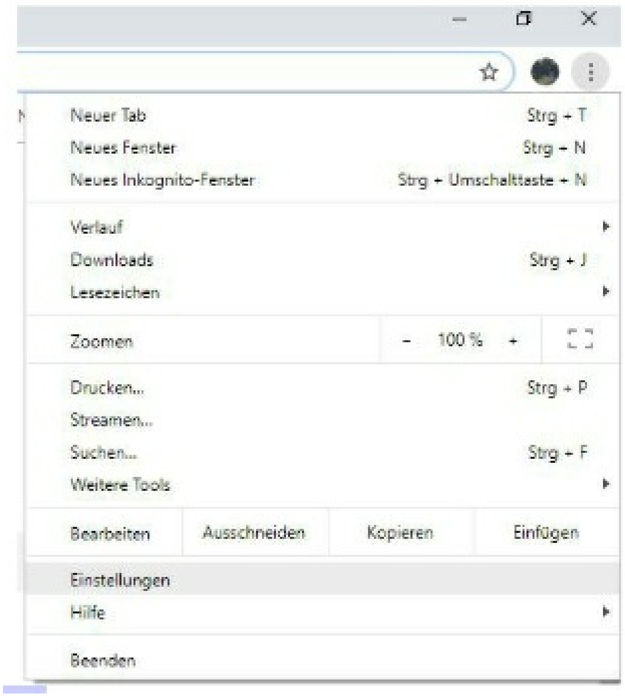
The designation can be freely chosen. It is advisable to choose a name that corresponds to the content. For example, if you want to enter the customer number, it would make sense to use this word as a designation. The selected term must always be in quotation marks, followed by the content. At this point it is possible to insert any values. This is followed by the expiry date for the cookie. This parameter is not absolutely necessary. If you leave this area blank, the cookie expires as soon as the user closes the browser. Therefore, the next time you visit, no recognition would be possible and the cookie would be worthless. At this point it is necessary to generate a value according to the Unix timestamp format. The value for the current time can be obtained using the time () function . After that it is only necessary to add the desired validity period in seconds. For example, if the cookie were to be valid for 30 days, you would have to enter the value time () + 2592000 here . Alternatively, you can first calculate the minutes, hours and days and then multiply this value by the desired number of days:

time () + 60 * 60 * 24 * 30 .

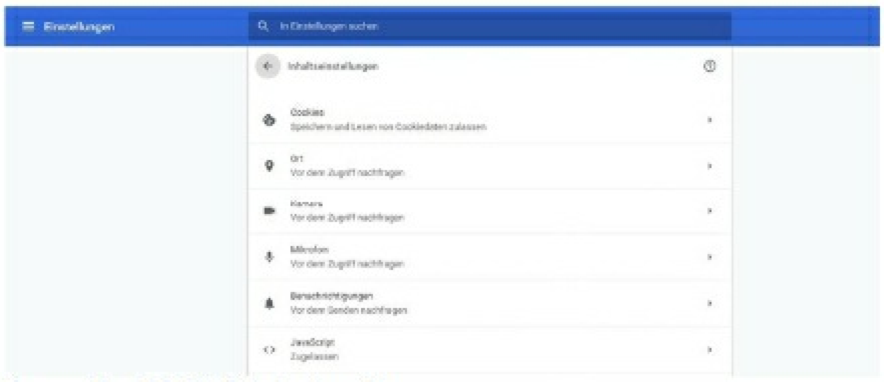
If you want to set a cookie that stores the customer number 12345 over 30 days, the following program would be necessary:

```
<? phpsetcookie ("customer number", "12345", time () + 2592000);?>
```

If you do this now, a blank page will appear. Still something happened. You can tell when you call up the browser's cookie data. How this works is shown here using the Google Chrome browser as an example. The procedure is similar for other browsers. To do this, first click on the three points at the top right
Click the edge of the picture and then select the Settings menu item.



Then a new page will open. Here you have to scroll down to the very bottom and click on “Advanced”. On the following page you then have to click on “Content Settings”



Inhaltseinstellungen

Cookies
Speichern und Lesen von Cookiedaten zulassen

Ort
Vor dem Zugriff nachfragen

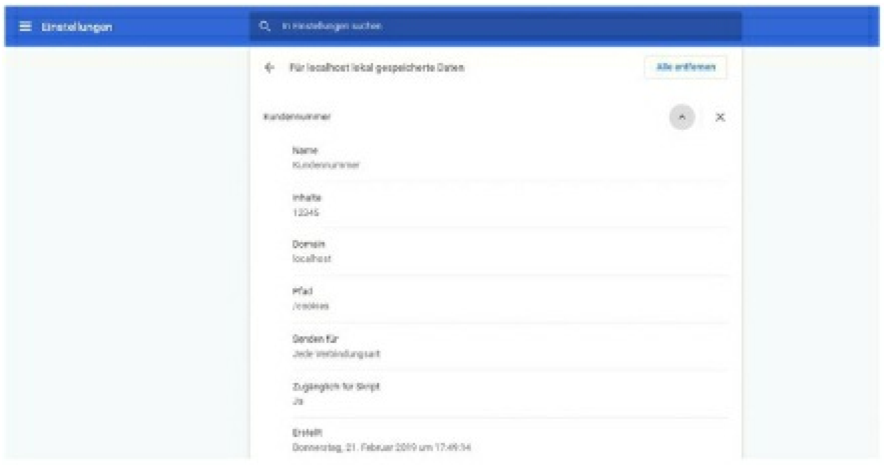
Kamera
Vor dem Zugriff nachfragen

Mikrofon
Vor dem Zugriff nachfragen

Benachrichtigungen
Vor dem Senden nachfragen

JavaScript
Zugelassen

The entry “Cookies” is located at the top. This must be clicked. You then have to select the menu item “Show all cookies and website data”. A long list of all the cookies stored on the computer will appear. In it you have to search for the entry localhost - or use the search function. The cookie with the designation customer number that we have just set will already appear. If you click on this, the corresponding entries become visible:



Für localhost lokal gespeicherte Daten

Kundennummer

Name
Kundennummer

Inhalte
12345

Domain
localhost

Pfad
/cookies

Senden für
Jede Verbindungsart

Zugänglich für Skript
Ja

Erstellt
Donnerstag, 21. Februar 2019 um 17:49:34

A lot of different information is now available here. Under the term content, for example, the text that we entered using the setcookie command can be viewed - i.e. the customer number 12345. In addition, it can be seen at what time the cookie was created and until when it is valid. The setcookie command must always be at The beginning of the document is preceded by any HTML tags, PHP print commands or other output. Even spaces are not allowed to precede it . The reason for this is that the cookie is transmitted in the HTTP header. It must therefore be in front of all content on the page. However, XAMPP does not apply this rule. This means that you can also output content here before the setcookie command. However, this is strictly not recommended. If you create such a program and later load it onto a real web server, it is very likely that it will not work. However, it is possible to use the setcookie command in an if query. This is not an output, so it does not affect the structure of the document. For example, it is common to use an if query to check whether a cookie already exists and to only set a new cookie if it does not yet exist.

16.3 Evaluate information from the cookie

In order to use the advantages of cookies, it is of course necessary to evaluate the information stored in them. As already described, the browser automatically sends the cookie to the server. So there is no need to specifically request it, which is why it is very easy to access the data. They are all stored in the $ _COOKIE variable. To use it, it is only necessary to call this variable. This is a
associative array. This means that the content has to be accessed using a special keyword. This keyword is always the name that we gave the cookie when it was set. To access the content of the cookie from the previous section, the following expression would be necessary: $ _COOKIE ['customer number'] This stores the value that was set in the second position with the setcookie command. In this case, that would be customer number 12345. If you want to call up the value of the cookie, you always have to be careful. After all, while programming, we don't know whether the user has visited the page before. If it is the first visit, there is no cookie yet. It is also possible that the user refuses to use cookies. If you call this variable but there is no corresponding cookie, the program will be aborted. For this reason, it is important to always include the call to this variable in an if query. This should first check whether the cookie is even present. The following command is used for this:

```
if (! empty ($ _ COOKIE ['customer number']))
```

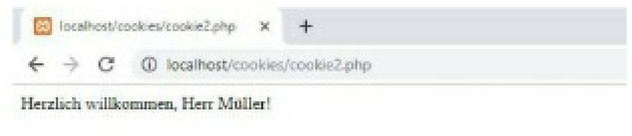
Then it makes sense to insert an else block that outputs an alternative page for visitors without a cookie. The query of the data will now be clarified using a

short example program. This contains an array with the data from three different customers. In practice it would of course be necessary to store the customer data in a database and first load it into the program. In order to keep the code simple and clear, this step is omitted in this example. The program then calls
first takes the customer number from the cookie and goes through the array with a foreach loop to check for a match. If this is found, the program records the associated salutation and name and outputs a personalized greeting (the program assumes that if a cookie is present, the associated customer data is also stored. If this is not the case, this would be the case In order to make the program more robust, it would make sense in practice to insert an alternative procedure for this as well.) If there is no cookie, the program asks the visitor to log in. For this purpose, it offers a link to the program from the previous example (which in this example is called cookie

1.php was saved). This then automatically sets the cookie.

```
<? php $ customer data = [[12345, "Herr", "Müller"], [12346, "Frau", "Mayer"], [12347, "Frau",
"Becker"]]; if (! empty ($ _COOKIE ['customer number'])) {$ customer = $ _COOKIE ['customer
number']; foreach ($ customer data as $ data set) {if ($ data set [0] == $ customer) {$ salutation = $
data set [1] ; $ name = $ record [2];}} print "Welcome,". $ salutation. "". $ name. "!";} else {print
"Please register <a href ='cookie1.php'> </a>!";}?>
```

If you run the program now (and if the cookie from the previous section is still saved), you should have a personalized salutation
appear:



```
Herzlich willkommen, Herr Müller!
```

is done again according to the familiar pattern. It is only important to make sure that the WHERE clause of the SELECT command does not refer to the form field but to the value of the cookie. To check whether the corresponding value is available, the rowCount command is used again. Here, if a match occurs, the program fills the variable name , first name and place of residence with denentsprechenden values. Since each customer number is only assigned once, it can be assumed that there is only one line in this case. You can therefore access them directly by adding the index [0] to the array result : $ sql = "SELECT * FROM customers WHERE k_id =". $ _ COOKIE ['customer number']. ";"; $ Dbh = new PDO ("mysql: dbname = customerDB; host = localhost", "user1", "abc"); $ return = $ dbh-> query ($ sql); if ($ return-> rowCount ()> 0) {$ result = $ rueckgabe- > fetchAll (PDO :: FETCH_ASSOC); print "Welcome! <br>"; print "Your data: <br>"; print "Customer number:". $ _ COOKIE ['Customer number']. "<br> "; print" Name: ". $ result [0] ['k_name']." <br> "; print" First name: ". $ result [0] ['k_first name']." <br> "; print" Residence: ". $ Result [0] ['k_wohnort']." <br> ";}

If the customer does not exist, the cookie should be invalidated. This is done with the setcookie command. However, a few seconds should be deducted from the time () command. This ensures that the expiration date is in the past and the cookie is therefore invalid. Then an error message appears and a button that reloads the page. If the customer is present, however, the program should welcome them and display their saved data. The last part looks like this:

else {try {$ sql = "SELECT * FROM customers WHERE k_id =". $ _ COOKIE ['customer number']. ";"; $ dbh = new PDO ("mysql: dbname = customerDB; host = localhost", "user1" , "abc"); $ rueckgabe = $ dbh-> query ($ sql); if ($ rueckgabe-> rowCount ()> 0) {$ result = $ rueckgabe-> fetchAll (PDO :: FETCH_ASSOC); print "Cordially Welcome! <br> "; print" Your data: <br> "; print" Customer number: ". $ _ COOKIE ['Customer number']." <br> "; print" Name: ". $ result [0] [' k_name ']. "<br>"; print "First name:". $ result [0] [' k_first name ']. "<br>"; print "City:". $ result [0] [' k_home ']. " <br>";} else {setcookie ("customer number", 0, time () - 10); print "Cookie data invalid!"; print '<form method = "post" action = "application example.php"> '; print' <input type = "submit" value = "Next"> '; print' </form> ';} $ dbh = null;} catch (PDOException $ e) {print $ e-> getMessage (); }}

The greeting of the customer and the output of the data

This completes the page. It now always outputs a suitable display - depending on whether a cookie is present or whether the user has filled out the form fields. The complete code looks like this:

<? phpif (empty ($ _ COOKIE ['customer number'])) {if (! empty ($ _ REQUEST ['customer number'])) {try {$ sql = "SELECT * FROM customers WHERE k_id =". $ _ REQUEST [' customer number ']. ";"; $ dbh = new PDO ("mysql: dbname = customerDB; host = localhost", "user1", "abc"); $ return = $ dbh-> query ($ sql); if ($ rueckgabe-> rowCount ()> 0) {setcookie ("customer number", $ _REQUEST ['customer number'], time () + 2592000); print "Successfully logged in."; print '<form method = "post" action = "application example.php"> '; print' <input type = "submit" value = "Next"> ';

print' </form> ';

} else {print "Invalid customer number. Please enter it again!"; print '<form method = "post" action = "application example.php">'; print '<input type = "submit" value = "Next">'; print '</form>';} $ result = $ return-> fetchAll (PDO :: FETCH_ASSOC); $ dbh = null;} catch (PDOException $ e) {print $ e-> getMessage ();}} elseif (! empty ($ _ REQUEST ['name']) and! empty ($ _ REQUEST ['first name']) and! empty ($ _ REQUEST ['residence'])) {$ sql = "INSERT INTO customers (k_name, k_first name, k_residence) ". "VALUES (: name,: first name,: place of residence);"; $ sql2 = "SELECT * FROM customers;"; try {$ dbh = new PDO ("mysql: dbname = customerDB; host = localhost", "user1", "abc"); $ input = $ dbh-> prepare ($ sql); $ input-> bindParam (': name', $ _ REQUEST ['name'], PDO :: PARAM_STR, 40); $ input-> bindParam (': first name', $ _ REQUEST ['first name'], PDO :: PARAM_STR, 40); $ input-> bindParam (': residence', $ _REQUEST ['residence'], PDO :: PARAM_STR, 40); $ input-> execute (); $ return = $ dbh-> query ($ sql2); $ result = $ return-> fetchAll (PDO :: FETCH_ASSOC); foreach ($ result as $ content) {$ new customer number = $ content ['k_id'];}

setcookie ("customer number", $ new customer number, time () + 2592000); print "Your data has been saved. <br>"; print "Your customer number:". $ new customer number; print '<form method = "post" action = " application example.php "> '; print' <input type =" submit "value =" Next "> '; print' </form> '; $ dbh = null;} catch (PDOException $ e) {print $ e-> getMessage ();}} else {print '<table> <tr> <td> Already registered: </td>'; print '<td> Register again: </td> </tr> <tr> <td> '; print' <form method = "post" action = "application example.php"> '; print' Customer number: <input type = "text" name = "customer number"> <br> <br> '; print' <input type = "submit" value = "Send"> '; print' </form> '; print' </td> <td> '; print' <form method = "post" action = "application example.php"> ' ; print 'Name: <input type = "text" name = "name"> <br>'; print 'First name: <input type = "text" name = "first name"> <br>'; print 'City: < input type = "text" name = "residence"> <br> <br> '; print' <input type = "submit" value = "Send"> '; print' </form> '; print' </ td > </tr> </table> ';}} else {try {$ sql = "SELECT * FROM customers WHERE k_id =" .

$ _COOKIE ['customer number']. ";"; $ Dbh = new PDO ("mysql: dbname = customerDB; host = localhost", "user1", "abc"); $ return = $ dbh-> query ($ sql); if ($ rueckgabe-> rowCount ()> 0) {$ result = $ rueckgabe-> fetchAll (PDO :: FETCH_ASSOC); print "Welcome! <br>"; print "Your data: <br>"; print "Customer number:". $ _ COOKIE ['Customer number']. "<br>"; print "Name:". $ result [0] ['k_name']. "<br>"; print "First name:". $ result [0] ['k_first name']. "<br>"; print "City:". $ result [0] ['k_wohnort']. "<br>";} else {setcookie ("customer number", 0, time () - 10); print "cookie data invalid!"; print '<form method = "post" action = "application example.php">'; print '<input type = "submit" value = "next"> '; print' </form> ';} $ dbh = null;} catch (PDOException $ e) {print $ e-> getMessage ();}}?>

16.6 Exercise: Create cookies yourself

1. Create a page that asks for the visitor's name. Then redirect the user to a new page.

2. Design this page too. The associated program should be an array

with some names and the associated age included. Check if the name is in the array. In this case, save the name and age in a session cookie and ask the visitor to confirm the values. Then forward it to a new page. If not, redirect him back to the first page. 3. Use the session ID to check the age of the visitor on this page. If he's over 18, welcome him. If this is not the case, provide a corresponding note. Solutions:

1.

```
<? phpprint '<form method = "post" action = "task1.2.php">'; print 'Name: <input type = "text" name = "name"> <br> <br>'; print ' <input type = "submit" value = "Next"> '; print' </form> ';?>
```

2.

```
<? php $ nutzer = [["Sebastian", 24], ["Marianne", 64], ["Karsten", 14]]; if (! empty ($ _ REQUEST ['name'])) {setcookie (" Visitor ", $ _REQUEST ['name'], time () + 2592000);} $ nameVorhanden = false; foreach ($ user as $ content) {if ($ content [0] == $ _REQUEST ['name']) {$ age = $ content [1]; $ nameVorhanden = true;}} if ($ nameVorhanden) {

session_start (); $ _ SESSION ['Name'] = $ _REQUEST ['name']; $ _ SESSION ['Age'] = $ age; print 'Your data: <br>'; print 'Name:'. $ _ REQUEST [ 'name']. "<br>"; print 'Age:'. $ age. "<br>"; print '<form method = "post" action = "task1.3.php">'; print '< input type = "submit" value = "confirm"> '; print' </form> ';} else {print' Name not saved. '; print' <form method = "post" action = "task1.1.php "> '; print' <input type =" submit "value =" Next "> '; print' </form> ';}?>
```

3.

```
<? phpsession_start (); if ($ _SESSION ['Age'] <18) {print 'Access under 18 not possible.';} else {print 'Welcome!';}?>
```

This is followed by the design of the email. This should first contain a personal salutation and then point out to the customer that the invoice is attached. Since this expression is a little longer, it should first be stored in the variable `content` before adding it to the `PHPMailer` object: `$ content = "Dear Sir / Madam". $ _ REQUEST ['name']; $ content. = ", \ r \ nYour invoice is attached.";` Then the mail is sent via the `PHPMailer` object. The procedure is almost the same as in section 2.2. It is only necessary to ensure that the recipient is to be determined on the basis of the input made previously by the user. The content is not entered directly, but via the variable content:

```
$ mail = new PHPMailer (); $ mail-> setFrom ('textbook.test@gmail.com', 'tool online shop'); $ mail-> addAddress ($ _ REQUEST ['address'], $ _REQUEST [' name ']);
$ mail-> Subject = 'Your Invoice'; $ mail-> Body = $ content; $ mail-> addAttachment ($ file); $ mail-> send ();
```

The creation of the PDF document and the sending of the e-mail are now completed. However, the page that opens remains empty so far. Therefore a short message should be displayed here:

```
print "Thank you for your order! <br>"; print "Your invoice will be sent to you by email.";
```

Now you have to close the curly braces of the `if` query. This is followed by a short `else` block that is executed if the customer has not ordered an item. This only outputs a corresponding message:

```
else {print "You have not ordered a product.";}
```

The program is now fully completed. The complete code for this looks like this:

```
<? php
require_once ('TCPDF-master / tcpdf.php'); use PHPMailer \ PHPMailer \ PHPMailer; use PHPMailer \ PHPMailer \ Exception; require 'C: \ xampp \ composer \ vendor \ autoload.php'; $ html = "<h1> Invoice </h1> "; $ html. =" <p> Invoice no .: 12345 </p> <br> <br> <br> "; $ price = 0; if ($ _REQUEST ['drill'] ! = 0) {$ html. = $ _REQUEST ['drilling machine']. "Drilling machine (s):"; $ html. = ($ _REQUEST ['drilling machine'] * 34.99). "EUR <br> <br>" ; $ price + = $ _REQUEST ['drilling machine'] * 34.99;} if ($ _REQUEST ['angle grinder']! = 0) {$ html. = $ _REQUEST ['angle grinder']. "Angle grinder:"; $ html. = ($ _REQUEST ['angle grinder'] * 22.99). "EUR <br> <br>"; $ price + = $ _REQUEST ['angle grinder'] * 22.99;} if ($ _REQUEST ['belt grinder']! = 0 ) {$ html. = $ _REQUEST ['belt sander']. "Belt sander:"; $ html. = ($ _REQUEST ['belt sander'] * 41.99). "EUR <br> <br>"; $ price + = $ _REQUEST ['belt sander'] * 41.99;} $ html. = "<br> <br> <br> Invoice amount:". $ Price. "EUR"; if ($ price> 0) {$ file = "Invoice.pdf "; $ pdf = new TCPDF (); $ pdf-> AddPage (); $ pdf-> writeHTML ($ html); $ pdf-> Outp ut (dirname (__ FILE __). '/'. $ file, 'F'); $ content = "Dear Sir / Madam". $ _ REQUEST ['name']; $ content. = ", \ r \ nYour invoice is attached. "; $ mail = new PHPMailer (); $ mail-> setFrom ('lehrbuch.test@gmail.com ',' Tool-Onlineshop '); $ mail-> addAddress ($ _ REQUEST [' address'], $ _REQUEST ['name']); $ mail-> Subject = 'Your invoice'; $ mail-> Body = $ content; $ mail-> addAttachment ($ file); $ mail-> send () ; print "Thank you for your order! <br>"; print "Your invoice will be sent to you by email.";
```

} else {print "You have not ordered a product.";}?> Screenshot 73 The confirmation after the order

17.5 Exercise: Sending emails and designing PDFs

1. Create an HTML form that allows the visitor to enter an email address and some text. After he has pressed the associated button, he should be redirected to a new page. This should send the entered content to the appropriate address using the mail function.

2. Use the same HTML form from the previous exercise. However, remove the field for entering the email address and redirect the visitor to another page, which should output the content in a PDF document directly via the browser. Now use the original HTML form again by entering the email address. Just change the forwarding and design a new page. This should turn the content into a PDF document

create and save on the server. It then sends this as an attachment to the specified address.

1.HTML form:

<h1> Please enter the content </h1> <form method = "post" action = "task1.php"> Content: <br> <textarea name = "content" rows = "10" cols = "30" > Your message </textarea> <br> E-Mail: <input type = "text" size = "15" name = "mail"> <br> <br> <input type = "submit" value = "send" > </form>

PHP-Script: <? Php $ receiver = $ _REQUEST ['mail']; $ subject = "Automatic E-Mail"; $ from = "From: PHP-Lehrbuch"; $ text = $ _REQUEST ['content']; mail ($ recipient, $ subject, $ text, $ from);? >

. <? phprequire_once ('TCPDF-master / tcpdf.php'); $ pdf = new TCPDF (); $ pdf-> AddPage (); $ pdf-> writeHTML ($ _ REQUEST ['content']); $ pdf-> Output ('Task2.pdf', 'I');?>

3. <? Phprequire_once ('TCPDF-master / tcpdf.php'); use PHPMailer \ PHPMailer \ PHPMailer; use PHPMailer \ PHPMailer \ Exception; require 'C: \ xampp \ composer \ vendor \ autoload.php'; $ pdfName = "Task3.pdf"; $ pdf = new TCPDF (); $ pdf-> AddPage (); $ pdf-> writeHTML ($ _ REQUEST ['content']); $ pdf-> Output (dirname (__ FILE __). '/ '. $ pdfName,' F '); $ mail = new

```
PHPMailer (); $ mail-> setFrom (' textbook.test@gmail.com '); $ mail-> addAddress ($ _ REQUEST ['
mail ']); $ mail-> Subject = 'Mail with attachment'; $ mail-> Body = 'This mail sends the entered
content as PDF.'; $ mail-> addAttachment ('Task3.pdf'); $ mail-> send (); ?>
```