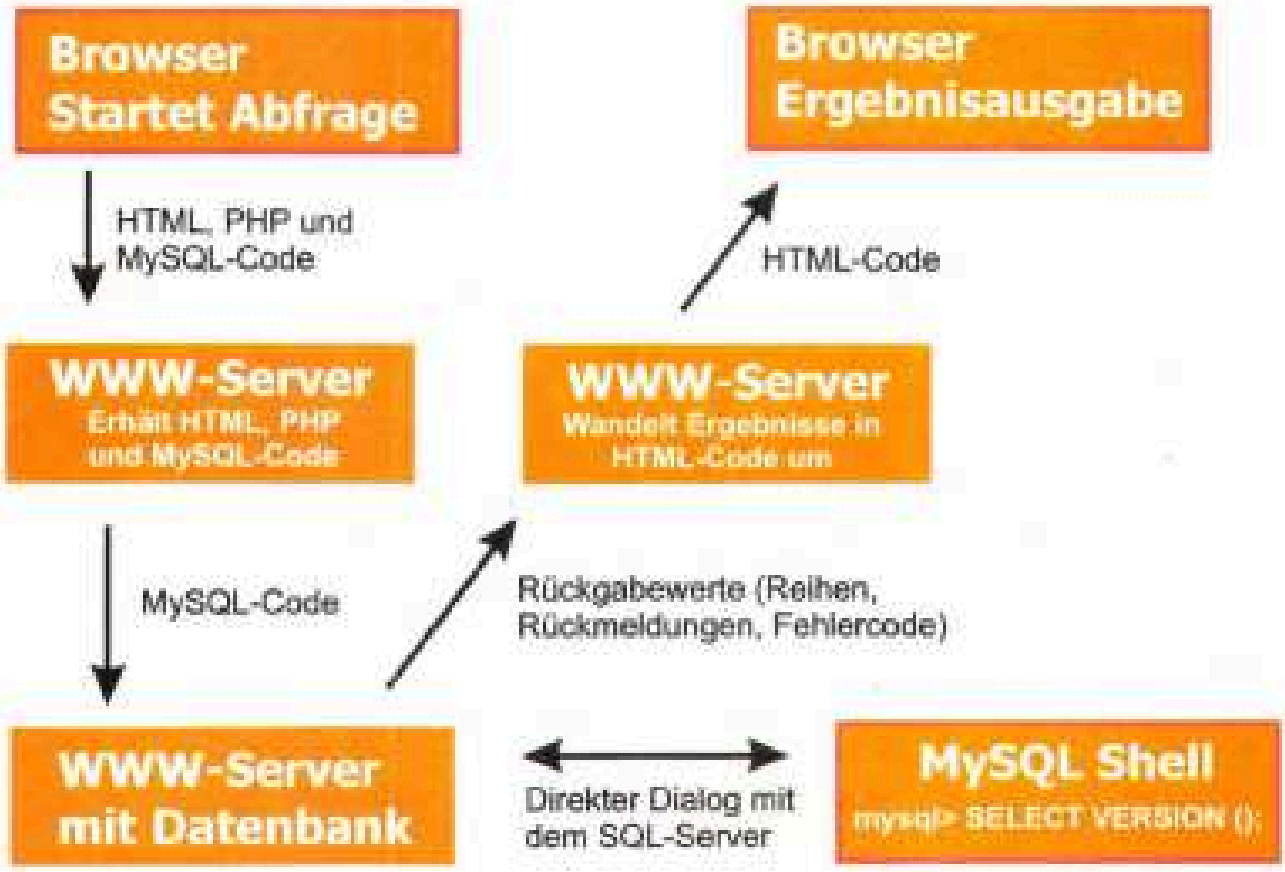
Nr. 7

€ 4,-

KnowWare EXTRA

PHP & MySQL

PHP & MySQL



Petra Bilke

www.KnowWare.de

Deutschland: 4,- EUR Österreich: 4,60 EUR
Schweiz: 8 SFR Luxemburg: 4,70 EUR Italien: 5,50 EUR

4 195087 904009 07

PHP und MySQL - dynamische Webseiten

Petra Bilke, petra@bilke.de
ISBN 87-90785-70-3, 1. Ausgabe, 2. Auflage: 2002-05

Michael Maardt, verlag@knowware.de - Karl Antz, lektorat@knowware.de
Printer: OTM Denmark, Binder: Gramo Denmark, Published by KnowWare

Bestellung für Endverbraucher und Vertrieb für den Buchhandel

Bonner Presse Vertrieb, Möserstr. 2-3
D-49074 Osnabrück
Tel.: +49 (0)541 33145-20
Fax: +49 (0)541 33145-33
knowware@bpv-online.com
Bestellformular gibt es auf www.knowware.de

Vertrieb für den Zeitschriftenhandel:

IPV Inland Presse Vertrieb, Postfach 10 32 46,
D-20022 Hamburg
Tel.: +49 (0) 40 23711-0
Fax: +49 (0)40 23711-215
www.ipv-online.de

Worum es geht

Hinter KnowWare steht der Gedanke, Wissen leichtverständlich und preisgünstig zu vermitteln. Das Projekt startete im April 1993 mit der Herausgabe des ersten Computerheftes in Dänemark. Seitdem sind in vielen Ländern zahlreiche weitere Hefte mit Themen rund um den Computer erschienen.

Wo und wann sind die Hefte erhältlich?

Die Hefte sind im allgemeinen zwei Monate im Handel, und zwar bei Kiosken, im Bahnhofsbuchhandel und im Buchhandel – bei vielen Verkaufsstellen sowie im Buchhandel auch länger. Alle beim Verlag vorrätigen Titel sind jederzeit nachbestellbar.

Bestellung

- bei deinem KnowWarehändler - Bestellformular am Ende des Heftes ausfüllen!
- beim Bonner Presse Vertrieb, siehe oben

www.knowware.de

- Beschreibungen und Bilder aller Hefte.
- Mehr als 100 kostenlose PDF-Dateien - bei jedem Heft gibt es eine kostenlose PDF-Datei von den ersten 15-20 Seiten
- Ausverkaufte Hefte: das ganze Heft als PDF ist kostenlos
- Geplante Hefte
- Online-Bestellung
- Anmeldung: kostenloser Newsletter. Viele Vorteile für dich.
- 100 Tipps & Tricks zu Windows, Word, Excel, und Internet
- Informationen für neue Autoren
- Serviceseiten zu den Heften
- Lesermeinung zu den Heften
- 800 Händleradressen in 4 Ländern.
- KnowWare Hefte in anderen sprachen

www.knowware.de

Vorwort

Nachdem ich mein erstes Heft *Start mit Datenbanken und SQL* geschrieben habe, wurde ich oft gefragt, welche Möglichkeiten es gibt, dynamische, datenbankgestützte Webseiten zu erstellen. Keine größere Webanwendung funktioniert heute mehr ohne dynamische Erzeugung von Webseiten mit Anbindung an eine Datenbank.

Um die Anzeige dieser dynamischen Webseiten in einem Webbrowser zu ermöglichen, werden sie mit Server-seitigen Programmmodulen erzeugt. Heute gibt es im Internet zahlreiche Möglichkeiten dieser Art, jede mit ihren Vor- und Nachteilen. Die Tabelle zeigt einige der Technologien, die Unterstützung durch den Internetprovider benötigen.

	Betriebssysteme, Webserver	**Kurzbeschreibung**
CGI Perl	Windows, Linux, Apache, MS-Internet-informationserver (IIS)	Eine der frühesten, immer noch benutzten Techniken ist die per CGI-Schnittstelle (Common Gateway Interface) aufgerufene Perl-Skript-Technik (Practical Extraction and Report Language). Für kleine Anwendungen ist dies der einfachste Weg, der auch von jedem Webserver unterstützt wird. Bei größeren Anwendungen können Performanzprobleme auftreten, da jede Anfrage einen eigener Prozess startet.
PHP / MySQL	Windows, Linux Apache, IIS.	Mehr Geschwindigkeit, komfortablere Bibliotheken und wesentlich bessere Möglichkeiten zur Datenbankanbindung bietet PHP (PHP Hypertext Preprocessor). PHP ist leicht zu erlernen.. MySQL eignet sich als sehr schnelle SQL-Datenbank hervorragend für den Einsatz im Internet
MS-ASP	Windows, IIS	Eine der bekanntesten Techniken zur Einbettung von serverseitig ausgeführten Skripten in HTML-Seiten ist das von Microsoft propagierte ASP (Active Server Pages). ASP lässt sich im Prinzip mit verschiedenen Skriptsprachen kombinieren. Die Microsoft-ActiveX-Library ist nur unter Windows mit dem Microsoft Internet Information Server (IIS) verfügbar.
JavaServer Pages	Windows, Linux Apache, IIS	JSP (JavaServer Pages) bieten vergleichbare Features, haben aber darüber hinaus noch weitere Vorteile. Eine JSP-Seite besteht aus normalem HTML-Code mit eingebetteten Java-Code. Der gesamte Sprachumfang samt Bibliotheken, JavaBeans- und EJB-Komponenten steht zur Verfügung. Java ist eine gut strukturierte und auf heutige Belange zugeschnittene Programmiersprache mit guter Netzwerkfähigkeit sowie komfortabler und schneller Datenbankanbindung. JSP-Anwendungen laufen unverändert unter z.B. Linux + Apache, Windows + IIS. Alle größeren Web Application Server bieten mittlerweile Java/JSP/J2EE-Unterstützung.

Das vorliegende Heft soll an Beispielen erklären, wie man anhand der zahlreichen Möglichkeiten von PHP, vor allem seiner Vielzahl von Standardfunktionen, Daten auf einem MySQL-Server ablegen und verwalten kann. Das Heft soll und kann kein Handbuch ersetzen – ich möchte nur zeigen, dass man auch mit ersten Grundkenntnissen ein Webangebot um einen dynamischen Teil ergänzen kann.

Grundkenntnisse von HTML werden vorausgesetzt. Möchtest du mehr über HTML wissen, empfehle ich die Hefte *Homepages für Einsteiger* und *Homepages für Fortgeschrittene*.

Die zahlreichen Skripts aus diesem Heft findest du im Internet auf der Homepage des KnowWare-Verlags sowie unter folgender Adresse:

`www.bilke.de/mysql`

Das erspart dir beim Ausprobieren der Skripts viel Tipparbeit. Unter den angegebenen Adressen findest du auch inhaltliche Kurzfassungen und die Beispiele gleich zum Probieren.

Voraussetzungen

Die Erstellung und Veröffentlichung von dynamischen, datenbankgestützten Webseiten mit PHP und SQL stellt gewisse Vorbedingungen:

- Auf dem Webserver deines Internetproviders muss PHP als CGI-Programm oder besser als Unterprogramm integriert sein
- Auf dem Webservers des Providers muss eine MySQL-Datenbank vorhanden sein, auf die du mit geeigneten Tools Zugriff hast.

Ich nutze in diesem Heft die MySQL-Datenbank des Internetproviders STRATO AG, der in einem PREMIUM-Paket die MySQL-Datenbank und die PHP-Sprache anbietet.

Möchtest du mit PHP arbeiten, solltest du wissen, dass das eine Skriptsprache ist, die in den HTML-Code eingebettet ist und auf einem WWW-Server abgelegte Webdokumente dynamisch aufbereitet.

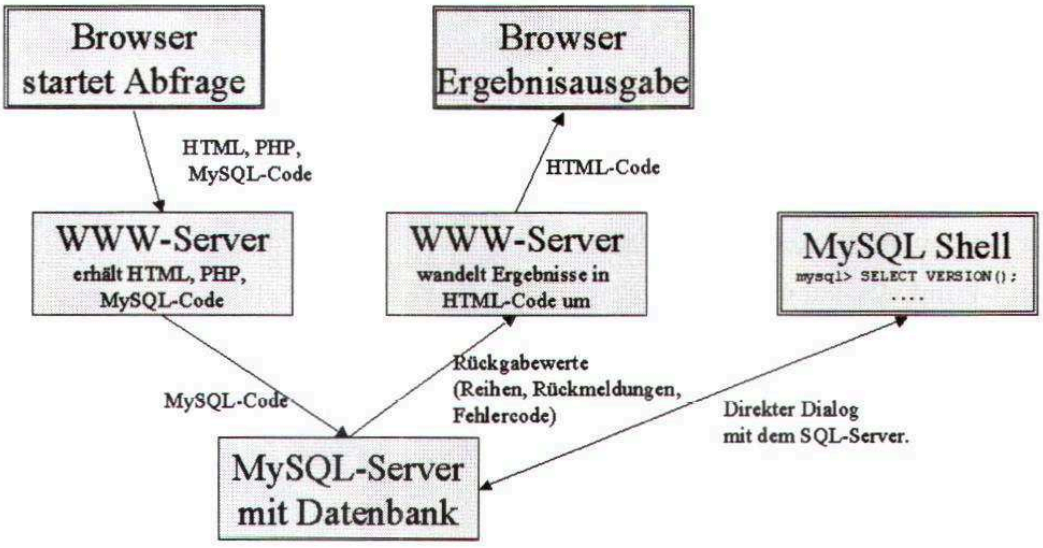
Das bedeutet, dass Fehler im Code in der Regel erst dann festgestellt werden, wenn die Seiten über einen Browser aus dem Werbserver des Providers abgerufen werden. Der PHP-Code wird auf dem Webserver verarbeitet und die SQL-Befehle an den MySQL-Server weitergeleitet.

Soll von Webseiten aus auf Datenbankinhalte zugegriffen werden, hat die Programmiersprache folgende Aufgaben:

- die Verbindung zur Datenbank auf dem Datenbankserver muss aufgebaut werden
- die Abfrage ist an die Datenbank zu senden
- die Ergebnisse müssen aufbereitet werden

Die SQL-Ergebnisse werden wieder in HTML-Code umgewandelt und zurückgegeben.

Zugriff auf Datenbankinhalte



Für deine PHP-Programme bedeutet das, dass die folgenden Schritte vorzunehmen sind:

1. Skript erstellen,
2. Skript mit `ftp` in das Veröffentlichungsverzeichnis des Providers transferieren,
3. Skript über einen Browser aufrufen,
4. im Fehlerfall editieren und bei 2. fortsetzen.

Zu Testzwecken kannst du auf deinem eigenen Rechner einen Webserver mit PHP-Modul und eine MySQL-Datenbank installieren. So kannst du in Ruhe testen, ohne laufend online zu sein. Die Softwarekomponenten können auf einer LINUX- oder Windows32- Umgebung installiert werden.

Es gibt verschiedenste Versionen von PHP und MySQL auf dem Markt. Welche Version du einsetzt, das hängt vom genutzten Betriebssystem ab. In den meisten Anleitungen wird ein LINUX-Betriebssystem empfohlen.

Möglich ist der Einsatz von PHP und MySQL auch unter Windows und unter Mac – letzteres besonders unter dem neuen Mac OS X. Wie du im nächsten Kapitel erfährst, musst du neben dem Betriebssystem auch einen Webserver zum Test betreiben. Jeder wird den Webserver einsetzen, mit dem er am meisten Erfahrungen gewonnen hat. Je nach Betriebssystem und Webserver sind Konfigurationsdateien von PHP und MySQL anyupassen. Du findest eine Beschreibung gerade der Installationsart, die zu deinem Computer passt, am besten im Internet, und zwar z.B. unter folgenden Adressen:

```
www.php-center.de
www.php-homepage.de
www.dynamic-webpages.de
```

Webserver

Als Webserver bietet sich der Apache-Webserver an. Er steht für die unterschiedlichsten UNIX-Systeme und inzwischen auch für Microsoft-Betriebssysteme, wie Windows 98 oder Windows NT zur Verfügung. Beim Mac OS X wird er sogar gleich mit geliefert. Du findest die Apache Software Foundation unter folgender Internetadresse:

```
www.apache.org/
```

Hier kannst du den Apache-Werbserver sowie entsprechende Dokumentationen herunter laden.

Arbeitest du mit Windows, kannst du auch den Microsoft Personal Web Server (PWS) oder den MS-Internetinformationserver (IIS) nutzen. Diese sind auf den jeweiligen Installations-CDs von Windows enthalten. Nach der Installation des Webservers kann er sofort eingesetzt werden. Auf die Einrichtung von Benutzern des Webservers und der damit verbunden Rechtevergabe kannst du verzichten, wenn du ihn zu Testzwecken benutzt.

Die Programmiersprache PHP

Nachdem ein Webserver erfolgreich installiert wurde, fehlt noch die Programmiersprache PHP.

Aufgabe dieser Programmiersprache ist es, auf einem WWW-Server abgelegte Web-Dokumente dynamisch aufzubereiten. Du findest sie auf der folgenden Internetseite:

```
www.php.net
```

Es gibt zwei Möglichkeiten zur Installation von PHP:

- als CGI-Programm
- als als Apache-Modul

PHP als CGI-Modul hat den Vorteil, dass es auf jedem Webserver einsetzbar ist, der CGI-Scripte ausführen kann. Sicherheitstechnisch hat diese Methode aber verschiedene Begrenzungen.

Wird PHP als Apache-Modul installiert, steht es als Unterprogramm des Apache-Webservers jederzeit zur Verfügung. Da der bei der CGI-Variante erforderliche Initialisierungsprozess eingespart wird, zeichnet sich diese Variante durch eine höhere Verarbeitungsgeschwindigkeit aus. Es gelten die Sicherheitskriterien des jeweiligen Webservers.

Datenbank MySQL

MySQL ist ein so genanntes relationales Datenbank-Managementsystem (rdbms), welches von der Firma T.c.X. DataKonsult in Schweden entwickelt wurde. Dieses System gilt als schnell, robust und einfach im Gebrauch. Die Software besteht aus dem Datenbankserver sowie unterschiedlichen Client-Programmen. Die aktuelle MySQL-Version findest du hier:

```
www.mysql.com/
```

Ein gutes Online-Handbuch zu MySQL findest du unter einer der nachfolgenden drei Adressen:

```
www.php-center.de/mirrors/
mysql/mysql.zip

www.xsl.de/mysql/mysql.zip

www.rent-a-database.de/mysql/
mysql.zip
```

Speicherung der Dateien

Für die Erfassung von PHP-Programmen reicht ein einfacher Editor wie z.B. Notepad. Die SQL-Befehle werden in den PHP-Code integriert. Beim Abspeichern ist die Erweiterung **.PHP3** (für die Version PHP3), **.PHP4** (für die Version PHP4) oder **.PHP** (für die Version PHP3 oder PHP4) zu verwenden. Bei der Eingabe der Anweisungen sind einige Regeln zu beachten:

- Eine Anweisung kann in einer Zeile oder in mehreren Zeilen eingegeben werden.
- Die Anweisung wird mit ; (Semikolon) abgeschlossen.
- Groß- und Kleinschreibung ist in den Anweisungen in der Regel möglich.

Dateinamen solltest du grundsätzlich mit Kleinbuchstaben benennen. Werden Dateien auf einem UNIX-System hinterlegt, spielt der Unterschied zwischen Groß- und Kleinbuchstaben nämlich eine große Rolle – bei den Dateien „TEST.PHP", „test.php" und „Test.php" handelt es sich also um drei verschiedene Dateien.

Zur Gestaltung der Übersichtlichkeit ist es ratsam, HTML-Code und PHP-Code optisch zu trennen. Es empfiehlt sich den PHP-Code in extra Dateien (Module) zu schreiben und diese mit der INCLUDE-Anweisung einzubinden. Wie das funktioniert, habe ich weiter hinten beschrieben.

Notationshinweise

Bevor du konkrete Anweisungen eingeben kannst, musst du die Syntax der Anweisungen kennen. Zur übersichtlicheren Darstellung der Syntax der Anweisungen werden folgende Vereinbarungen getroffen:

[ ]	Der Ausdruck innerhalb der eckigen Klammern ist optional und kann ev. entfallen. Die Klammern werden nicht eingegeben.
{}	Zusammengehörige Gruppen werden durch geschweifte Klammern umschlossen. Die Klammern werden nicht eingegeben.
()	Runde Klammern müssen dort eingegeben werden, wo sie in der Syntax angeführt sind.
< >	Spitze Klammern müssen dort eingegeben werden, wo sie in der Syntax angeführt sind.
|	Der senkrechte Strich trennt alternative Ausdrücke. Von den Alternativen ist jeweils nur eine zu nutzen.
...	Kann durch Parameter ergänzt werden

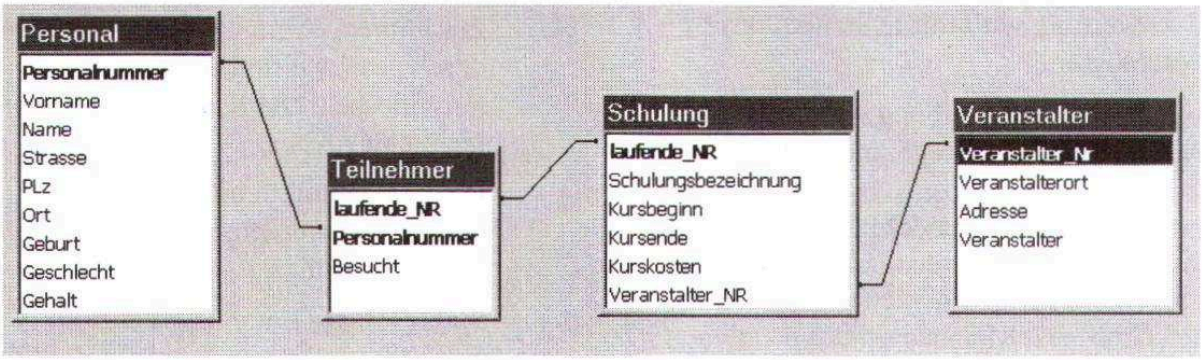
Unsere Beispieldatenbank

Ich nutze in diesem Heft wieder das Beispiel aus meinem Knowware-Heft *Start mit Datenbanken und SQL*. Hast du nie oder kaum mit Datenbanken gearbeitet, empfehle ich dir, die theoretische Einführung zu den Datenbanken nachzulesen.

In meiner Beispieldatenbank wollen Mitarbeiter der Abteilung Weiterbildung eines Unternehmens die besuchten Schulungen der Betriebsangehörigen erfassen.

Im erwähnten Heft habe ich einige Überlegungen dazu angestellt, was vor der Erstellung der Datenbank zu beachten ist. Das Ziel ist ein durchdachter Datenbankentwurf, mit dessen Hilfe die Daten in Tabellen so aufbereitet werden, dass eine redundanzfreie Speicherung möglich ist – gleiche Informationen werden nicht mehrfach erfasst.

Anhand dieser Überlegungen entstanden folgende Tabellen:



Programmskripte, in denen die Befehle zur Erstellung der Tabellen und zur Datenerfassung enthalten sind, stehen im Internet unter folgender Adresse zur Verfügung:

`www.bilke.de/mysql`

Nutzt du diese Erstellungs- und Eingabemöglichkeit, enthalten die Tabellen die nachfolgenden Daten. Im Kapitel 5 erfährst du, wie du diese Befehle mit Hilfe von PHP ausführen kannst.

personal

Personalnr	Vorname	Name	Strasse	PLz	Ort	Geburt	Geschlecht	Gehalt
1002	Kai	Müller	Hallesche Str. 16	04838	Eilenburg	28.12.51	M	3400
1005	Ede	Pfau	Nicoleiplatz 2	04232	Leipzig	02.08.52	M	4500
1008	Eike	Taro	Musterstr. 6	09999	Musterdorf	10.09.47	M	5100
1010	Jürgen	Marx	Musterstr. 12	09999	Musterdorf	22.03.56	M	3677
1015	Eerik	Wicki	Bolten-Weg 3	22587	Hamburg	09.07.50	M	4533
1016	Ulrich	Müller	Weide Str. 2 a	04838	Hohenpriessnitz	02.01.51	M	8999
1017	Klaus	Ecke	Schulze-Str. 11	04808	Wurzen	13.04.55	M	2555
1100	Kerstin	Gans	Hauptstr.10	04838	Moertitz	19.03.61	W	6000
1101	Guenter	Maus	Bergstrasse 5	06108	Halle/Saale	21.12.48	M	4500
1112	Hannes	Heik	Musterstr. 145	09999	Musterdorf	06.02.61	M	5600
1113	Karl	Seppa	Musterstr. 7	09999	Musterdorf	24.06.63	M	3400
1114	Karsten	Müller	Musterstr. 3	09999	Musterdorf	12.07.63	M	4900
1429	Torsten	Mieder	Hauptstrasse 16	04838	Laussig	15.05.71	M	5400
1430	Christine	Schwarz	Musterstr. 8	09999	Musterdorf	24.04.48	W	5420
1431	Birgit	Gemse	Bauernkoppel 39	22393	Hamburg	13.01.58	W	1200
1432	Bernd	Jach	Suelldorferstr.	22589	Hamburg	10.05.59	M	2455
1433	Silvia	Munter	Treppe 7	22587	Hamburg	28.04.52	W	4555
1434	Joerg	Schön	Dorfweg 25 a	22589	Hamburg	20.08.69	M	5600
1435	Peter	Schock	Weststrasse 2	74232	Abstatt	02.11.44	M	4700
1436	Hans-Juergen	Sieg	Gerberstrasse 10	24568	Winsen	22.05.47	M	7000
1437	Christa	Renner	Weidenstrasse 26	25469	Halstenbek	27.12.32	W	3900
1438	Manfred	Stach	Gerbergasse 7	04105	Leipzig	26.09.46	M	3500
1439	Elfi	Kaufmann	Brandstrasse 15 e	04827	Gerichshain	29.01.66	W	3900

teilnehmer

laufende_NR	Personalnummer	Besucht
1	1002	Ja
1	1017	Ja
1	1101	Ja
1	1429	Ja
2	1114	Ja
2	1439	Ja
3	1010	Ja
3	1015	Ja
3	1016	Ja
3	1017	Ja

laufende_NR	Personalnummer	Besucht
4	1432	Ja
4	1433	Ja
5	1002	Ja
5	1435	Ja
5	1439	Ja
6	1005	Ja
6	1100	Ja
7	1430	Ja
7	1431	Ja
8	1438	Ja

veranstalter

Veranstalter_NR	Veranstaltungsort	Adresse	Veranstalter
1	Mannheim	...	S & B Mannheim
2	München	...	S & B München
3	Chemnitz	...	S & B Chemnitz
4	Leipzig	...	S & B Leipzig
5	Berlin	...	S & B Berlin
6	Rostock	...	S & B Rostock
7	Essen	...	S & B Essen

schulung

laufende_NR	Schulungsbezeichnung	Kursbeginn	Kursende	Kurskosten	Veranstalter_NR
1	ABA Schulung	14.03.96	15.03.96	1600	1
2	Laborschulung 2	14.04.96	16.04.96	2000	4
3	Teamtraining	07.04.96	08.04.96	2100	5
4	Grundkurs Excel	25.11.96	26.11.96	1200	2
5	Grundlagen PC/Windows 3.1	13.12.96	13.12.96	900	3
6	Grundkurs Word für Windows	14.12.96	15.12.96	1800	4
7	Grundkurs Powerpoint	04.01.96	04.01.96	1200	4
8	Projektmanagement	07.11.95	07.11.96	1549	5
9	Telefonmarketing-Training	26.04.96	27.04.96	2034	5
10	Personalplanung in der Praxis	03.09.96	04.09.96	1220	7
11	Arbeitszeugnisse richtig formulieren & analysieren	18.09.96	19.09.96	2440	7

PHP-Einführung

PHP wurde von dem Kanadier Rasmus Lerdorf entwickelt. Es ging ihm darum, einen einfachen Datenbankzugriff zu gewährleisten. Zunächst hatte PHP einen andere Namen – und zwar PHP/FI, was für Personal Home Pages / Form Interpreter steht. Die neue Abkürzung steht für PHP – Hypertext Preprocessor. Die Syntax von PHP ähnelt teilweise der von C, Perl und Java. PHP 3 wurde Anfang 1999 beendet, PHP 4 im Jahr 2000. Aber noch heute laufen viele PHP-Server unter PHP3. PHP4 ist in weiten Teilen kompatibel zu PHP3. Die hier beschrieben Beispiele sollten auf beiden Versionen laufen.

PHP und HTML

Der PHP-Code wird unmittelbar in die HTML-Seite geschrieben. Dazu gibt es 4 Möglichkeiten:

```
<? ... ?>
<?php ... ?>
<script language="php"> ...
</script>
<% ... %>
```

Statt der Dateierweiterung HTM oder HTML erhalten die Dateien die Erweiterung PHP3 bzw. PHP4. Auch PHP ist möglich. Diese Erweiterung hat zur Folge, dass die jeweilige Datei nicht sofort zum Browser gesendet, sondern erst dem PHP-Prozesor übergeben wird. Dieser arbeitet die Datei ab. Findet er PHP-Code, wird der ausgewertet und ausgeführt.

Eine HTML-Seite kann beliebig viele solcher PHP-Blöcke enthalten. Aller Text zwischen den beiden Tags muss gültiger PHP-Code sein.

Kommentare beginnen in PHP mit `/*` und enden mit `*/`.

Auch folgende Angabe eines Kommentars ist möglich:

```
// Das ist ein Kommentar. Er endet
am Zeilenende.
```

Der PHP-Code besteht aus einer Folge von Befehlen, wie z.B. Variablenzuweisungen, Funktionsaufrufen oder Schleifen. Einzelne Befehle werden durch ein Semikolon (;) voneinander getrennt.

Probiere gleich einmal dein erstes PHP-Programm **hallo.php** aus.

```
<html><head><title>Erstes PHP-Programm</title></head>
<body>
<?          echo "Hallo";      /* Ausgabe von Texten */
?>
</body>
</html>
```

Auf deiner Bildschirmausgabeseite müsste nun folgender Text stehen:

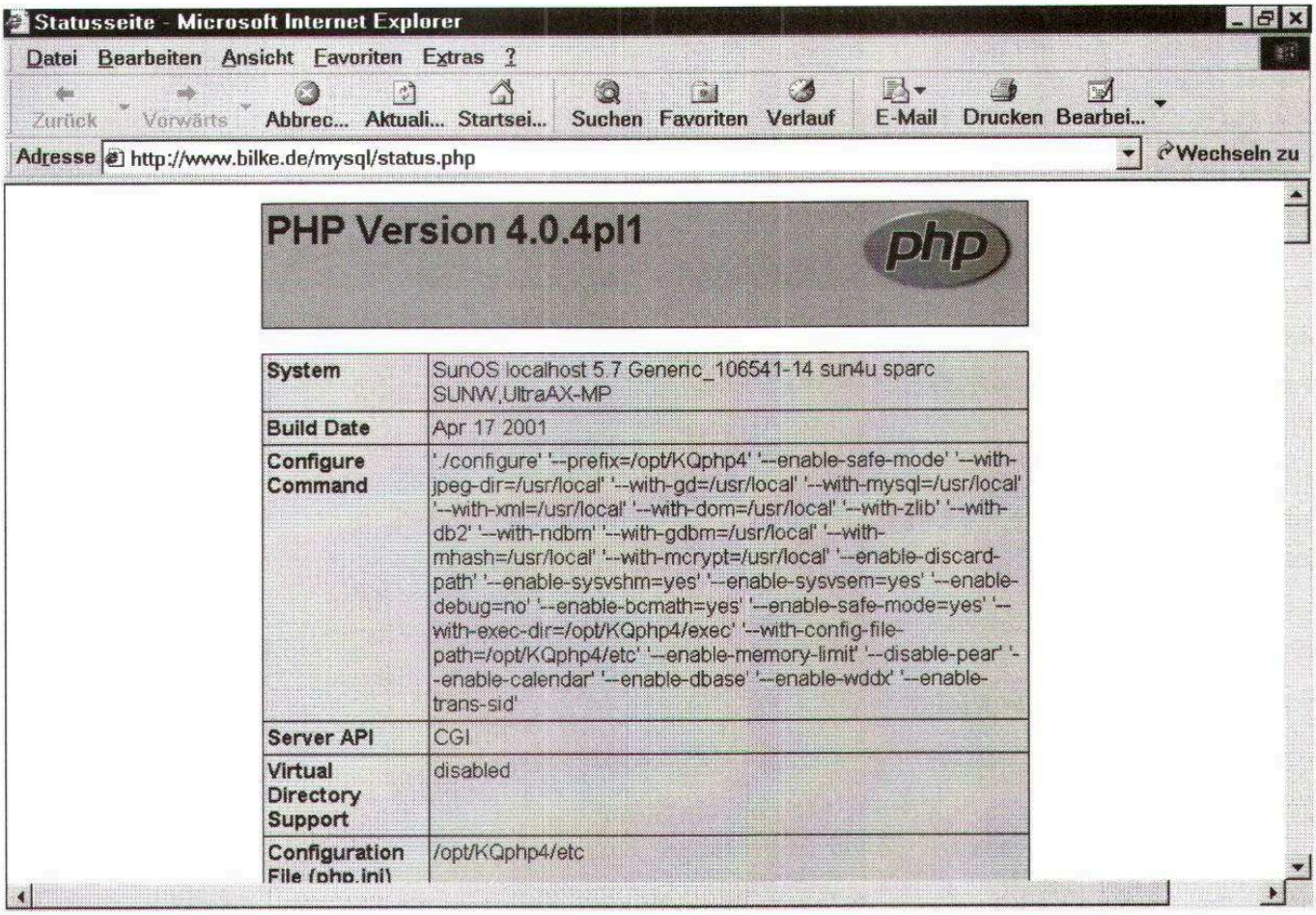
Hallo

Der **echo**-Befehl wird nachfolgend immer dann eingesetzt, wenn eine Textausgabe erforderlich ist. Die Textinformationen in diesem Befehl gehören in doppelte " " oder einfache ' ' Anführungszeichen.

In jeder Programmiersprache stehen zahlreiche Funktionen zur Verfügung. Benötigst du ausführliche Information zu deiner Version der PHP-Sprache, hilft dir die Funktion **phpinfo()**, die dir eine detaillierte Seite zur Verfügung stellt, die alle internen Konfigurationsvoreinstellungen und alle übergebenen Parameter anzeigt.

Modifiziere dein erstes Beispiel wie folgt (**status.php**).

```
<html><head><title>Statusseite</title></head>
<body>
<?                    /* Generierung einer Statusseite */
phpinfo();
?>
</body>
</html>
```



System	SunOS localhost 5.7 Generic_106541-14 sun4u sparc SUNW,UltraAX-MP
Build Date	Apr 17 2001
Configure Command	'./configure' '--prefix=/opt/KQphp4' '--enable-safe-mode' '--with-jpeg-dir=/usr/local' '--with-gd=/usr/local' '--with-mysql=/usr/local' '--with-xml=/usr/local' '--with-dom=/usr/local' '--with-zlib' '--with-db2' '--with-ndbm' '--with-gdbm=/usr/local' '--with-mhash=/usr/local' '--with-mcrypt=/usr/local' '--enable-discard-path' '--enable-sysvshm=yes' '--enable-sysvsem=yes' '--enable-debug=no' '--enable-bcmath=yes' '--enable-safe-mode=yes' '--with-exec-dir=/opt/KQphp4/exec' '--with-config-file-path=/opt/KQphp4/etc' '--enable-memory-limit' '--disable-pear' '--enable-calendar' '--enable-dbase' '--enable-wddx' '--enable-trans-sid'
Server API	CGI
Virtual Directory Support	disabled
Configuration File (php.ini)	/opt/KQphp4/etc

Variablen

Variablen ermöglichen die Speicherung von Daten. Sie beginnen in PHP mit einem Dollarzeichen, gefolgt von dem Variablenbezeichner. Für Zuweisungen wird das einfache Gleichheitszeichen verwendet. PHP verwendet untypisierte Variablen. Es muss somit nicht festlegt werden, ob in einer Variablen Zahlen, Texte oder sonstige Werte gespeichert werden sollen. PHP passt den Typ automatisch an die Verwendung an.

Hier wieder ein Beispiel **var1.php** zum Ausprobieren:

```
<html><head><title>Variablenbeispiel</title></head>
<body>
<?
$text = "Ich bin ein String !";
echo "$text <BR>";
echo "$text $text $text <BR>";
$l = "langer";
$k = "kurzer";
echo "Ich bin ein $l $l $l $l $l $l Text! <BR>" ;
echo "Ich bin ein $k Text! <BR>" ;
$i = 10;
$j = 5;
echo $i,"+", $j,"=",$i+$j;
?>
</body>
</html>
```

Das Ergebnis sieht aus wie folgt:

```
Ich bin ein String !
Ich bin ein String ! Ich bin ein String ! Ich bin ein String !
Ich bin ein langer langer langer langer langer langer Text!
Ich bin ein kurzer Text!
10+5=15
```

Neben einfachen Variablen gibt es in PHP Arrays. Der Index des Arrays wird in eckigen Klammern hinter die Bezeichnung der Arrayvariablen angehängt.

In der Arrayvariable **`$adresse`** sollen zunächst die Adressinformationen für Hans Lehmann aufgenommen werden (**var2.php**).

```
<html><head><title>Arraybeispiel</title></head>
<body>
<?   $adresse[0]= 1;
$adresse[1]= "Lehmann";
$adresse[2]= "Hans";
$adresse[3]= "Luisenweg";
$adresse[4]= "04329";
$adresse[5]= "Leipzig";
for ($i=0;$i<sizeof($adresse);$i++)
{echo "$adresse[$i] <BR>"; }
?>
</body>
</html>
```

Zur Anzeige wird das Array mit der **for**-Schleife ausgelesen. (Mehr Informationen zu Schleifen erhältst du auf der nächsten Seite.) Nachstehend siehst du das Ergebnis.

1
Lehmann
Hans
Luisenweg
04329
Leipzig

Mit Variableninhalten kann gerechnet werden. Auch stehen sie zum Vergleich zur Verfügung. Zum Einsatz in PHP-Programmen kommen folgende Rechen- und Vergleichsoperationen.

Rechenoperationen		
+	Addition	`$i+$j`
-	Subtraktion	`$i-$j`
*	Multiplikation	`$i*$j`
/	Division	`$i/$j`
%	Reste-Bildung	`$i%$j`
.	Verknüpft Strings	$l = "langer"; $k = "kurzer"; echo $l.$k; ergibt `langerkurzer`
$i++** **++$i	erhöht `$i` um 1	
$i–** **--$i	erniedrigt `$i` um 1	

Vergleichsoperationen	
==	gleich
>	größer
<	kleiner
>=	größer gleich
<=	kleiner gleich
!=	ungleich

Der Unterschied zwischen `$i++` und `++$i` ist:

$i=0; **echo $i++;**	gibt 0 aus, anschließend wird `$i` auf den Wert 1 erhöht.
$i=0;** **echo ++$i;	erhöht zuerst `$i` auf 1 und gibt den Wert 1 aus.

Programmkonstrukte

Wie in anderen Programmiersprachen gibt es auch in PHP Möglichkeiten, die Ausführung von Code von bestimmten Bedingungen abhängig zu machen.

IF-Bedingung	SWITCH-Befehl	Schleifen

`if (Bedingung) {`
`Befehl;`
`} else {`
`Befehl;`
`}`
	`switch (Ausdruck) {`
	`case 0: Befehl`
	`case 1: Befehl`
	`case n: Befehl`
	`default: Befehl`
	`}`
		`while (Ausdruck) Befehl`

		`do`
		`Befehl`
		`while(Ausdruck)`

		`for (Ausdruck1 [,...];`
		`Ausdruck2 [,...];`
		`Ausdruck3 [,...])`
		`Befehl`

Die Befehle zur Untersuchung von Bedingungen funktionieren ähnlich , wie man es von anderen Programmiersprachen gewohnt ist. Zur Illustration nachfolgend einige Beispiele.

Zur Fallunterscheidung gibt es die **if**-Bedingung:

```
if ($i<0) {
   echo "$i ist kleiner als Null\n";
}
```

Voraussetzung	$i=-12	$i=+7
Bildschirm-ausgabe	-12 ist kleiner als Null	

Bei der **if**-Bedingung kann man auch angeben, was geschehen soll, wenn die Bedingung nicht erfüllt ist. Die **if**-Bedingung ist durch den **else**-Zweig zu ersetzen.

```
if ($i<0) {
      echo "$i ist kleiner als
Null\n";
} else {
      echo "$i ist nicht kleiner als
Null\n";
}
```

Voraussetzung	$i=-12	$i=+7
Bildschirm-ausgabe	-12 ist kleiner als Null	7 ist nicht kleiner als Null

In der **if**-Bedingung darf geschachtelt werden:

```
if ($i<0) {
      echo "$i ist kleiner als
Null\n";
} else if ($i>0) {
      echo "$i ist groesser als
Null\n";
} else {
      echo "$i ist Null\n";
}
```

Voraussetzung	$i=-12	$i=+7	$i=0
Bildschirm-ausgabe	-12 ist kleiner als Null	7 ist groesser als Null	0 ist Null

Hat man mehrere Tests auf den Inhalt derselben Variable, eignet sich der SWITCH-Befehl besser.

```
switch ($name) {
 case "Susi":
  echo "Ich bin Susi Sorglos<BR>";
  break;
 case "Hans":
  echo "Ich bin Hans Heinz<BR>";
  break;
 case "Paul":
  echo "Ich bin Paul Murx<BR>";
  break;
 default:
  echo "Wir sind der Rest<BR>";
}
```

Falls die Variable **$name** den Wert **"Susi"** hat, wird als nächster Befehl ausgeführt:

```
 echo "Ich bin Susi Sorglos<BR>";
```

Normalerweise würden auch alle nachfolgenden Befehle ausgeführt. Da dies nicht erwünscht ist, verlässt man mit **break** den **switch**-Befehl.

Eine Schleife mit dem **while**-Befehl sieht so aus:

```
...
$t = "Ich soll meinen Text selbst
erstellen!<BR>";
      $i = 0;
      while ($i<9) {
            echo $t;
            $i++;
      }
...
```

Das ergibt 9-mal den Text der Variablen **$t**:

```
Ich soll meinen Text selbst erstellen!
Ich soll meinen Text selbst erstellen!
Ich soll meinen Text selbst erstellen!
Ich soll meinen Text selbst erstellen!
Ich soll meinen Text selbst erstellen!
Ich soll meinen Text selbst erstellen!
Ich soll meinen Text selbst erstellen!
Ich soll meinen Text selbst erstellen!
Ich soll meinen Text selbst erstellen!
```

Zu Beginn wird **$i** auf 0 gesetzt – und dann in jedem Schleifendurchlauf um 1 erhöht, bis **$i** den Wert 9 erreicht, wodurch die Bedingung (**$i<9**) nicht mehr wahr ist und die Schleife abbricht.

Eine weitere Möglichkeit ist der **for**-Befehl:

```
...
$t = "Ich soll meinen Text selbst
erstellen!<BR>";
 for ($i=0;$i<10;$i++) {
  echo $t;
}
...
```

Er besteht aus drei Ausdrücken: mit **ausdruck1** wird die Schleife initialisiert, **ausdruck2** gibt die Abbruchbedingung an, und in **ausdruck3** wird die Variable, die die Schleifendurchläufe zählt, erhöht bzw. erniedrigt.

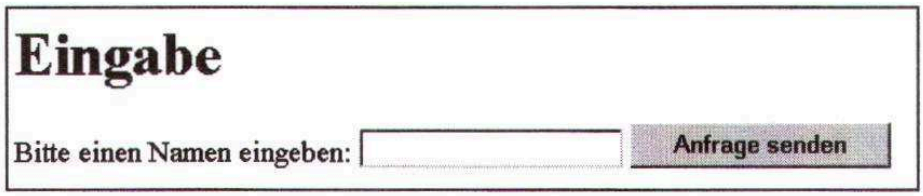
Das Ergebnis ist dasselbe wie in der letzten Aufgabenstellung.

Die vorgestellten Beispiele für Konstrukte sind in der Datei **konstrukt.php** enthalten und können durch ihren Aufruf getestet werden.

Da PHP speziell zur Erzeugung von dynamischen Webseiten geschaffen wurde, ist es sehr einfach, Eingaben aus HTML-Formularen in Variablen zu speichern und auszugeben.

Um den Einsatz von Formularen zu demonstrieren, wurde die Formulardatei **formular.htm** entwickelt. Sie enthält folgenden Code:

```
<html>
 <head>
 <title>INPUT</title>
 </head>
 <body>
 <h1>Eingabe</h1>
  <FORM ACTION="formaus.php" METHOD="POST">
    Bitte einen Namen eingeben: <INPUT NAME="einname">
      <INPUT TYPE="submit">
      </FORM>
</body>
</html>
```



Beim Klicken auf den Button **Anfrage senden** wird die Datei **formaus.php** aufgerufen, die so aussehen könnte:

```
<?
  echo "Sie haben $einname
eingegeben. ";
?>
```

Auf dem Bildschirm erscheint die Ausgabe:

Sie haben Petra eingegeben.

Der Text, der in das Formularfeld **NAME** gesetzt wurde, steht in der PHP-Variablen **$einname**. Ein umfangreicheres Beispiel für ein Formular findest du weiter hinten im Skript.

Funktionen

Ein wichtiger Bestandteil jeder Programmiersprache ist die Fähigkeit, mehrere Befehle zu einem einzigen zusammenzufassen und diesen als Funktion zu hinterlegen. Das wird erreicht mit dem Befehl **function** gefolgt vom Funktionsnamen, dem eingeklammert die Übergabeparameter folgen, die als Argumente der Funktion bezeichnet werden.

```
function funktionsname($parameter,
...)
{
// Hier werden die Befehle angegeben
return $rueckgabe
}
```

Soll die Funktion einen Wert liefern, gibst du diesen im **return** Befehl an.

Das nächste Beispiel programmiert die Funktion **tag**, die Tage in Minuten rechnet (**tagfunk.php**)

```
<?
function tag ($anz) {
$minuten = 60 * 24 * $anz;
return $minuten;
}
?>
```

Über die Variable **$anz** werden die Tage als Zahl der Funktion übergeben. Innerhalb der Funktion werden die Minuten ausgerechnet und in der Variablen **$minuten** gespeichert. Mit **return** erfolgt die Rückgabe.

Das nachfolgende Programmskript nutzt die neue Funktion (**tagfunkaus.php**). Damit das Skript die Funktion kennt, bindest du sie ein mit

```
include("tagfunk.php");
```

Die Funktion **tag** wird mit dem Parameter **7** (7 Tage) aufgerufen. Das Resultat in Minuten wird in **$x** gespeichert.

```
<html>
<head>
<title>Rechnet Tage in Minuten um </title>
</head>
<body>
<h1>Rechnet Tage in Minuten um</h1>
<?
   include("tagfunk.php");
   $x = tag(7);
   echo "7 Tage haben $x Minuten!";
?>
</body>
</html>
```

Das Ergebnis lautet wie folgt:

Rechnet Tage in Minuten um
7 Tage haben 10080 Minuten !

Neben den selbst erstellten Funktionen stellt PHP zahlreiche standardisierte Funktionen zur Verfügung. Im Beispiel **status.php** hast du bereits die Funktion **phpinfo()** zur Konfigurationsanzeige von PHP kennengelernt. Einige weitere interessante Funktionen habe ich in den nächsten Kapiteln verwendet.

Die Datumsfunktion getdate

Eine wichtige Funktion ist z.B. die Funktion **getdate**. Diese gibt Datums- und Zeitinformationen zurück. Es handelt sich um ein Array, welches in eine Variable ausgelesen werden kann.

```
$mon = getdate();
```

Die Arrayelemente sind folgende:

seconds-Sekunden	mon-Monat als Zahl
minutes-Minuten	year-Jahr als Zahl
hours-Stunden	yday-Tag des Jahres als Zahlenwert, z.B. "299"
mday-Tag des Monats	weekday-ausgeschriebener Wochentag
wday-numerischer Tag der Woche	month-ausgeschriebener Monatsname

Der Befehl

```
$cmonat = $mon['month'];
```

überträgt den Monatsnamen in die Variable **$cmonat**.

Der Einsatz der Datumsfunktion getdate() soll ausprobiert werden. Der nachstehende Bildschirminhalt:

Die Datumsfunktion getdate()

Heute ist Thursday.
Wir leben im Monat July.
Diese Aussagen betreffen den 192. Tag des Jahres 2001.

wird durch das Programmskript **getdate.php** erzeugt:

```
<html>
<head>
<title>Die Datumsfunktion getdate() </title>
</head>
<body>
<h2>Die Datumsfunktion getdate()</h2>
<?
$mon = getdate();
$cwtag=$mon['weekday'];
$cmonat = $mon['month'];
$ntag=$mon['yday'];
$yjahr=$mon['year'];
echo "Heute ist $cwtag. <BR> ";
echo "Wir leben im Monat $cmonat. <BR>";
echo "Diese Aussagen betreffen den $ntag. Tag des Jahres $yjahr. <BR>";
?>
</body>
</html>
```

Die Funktion include

Der Befehl `include("dateiname");` liest den Inhalt der Datei **dateiname** so, als ob stünde an dieser Stelle. Damit erreichst du z.B. ein einheitliches Layout bei einer größeren Anzahl von Dateien und erhöhst so die Übersichtlichkeit. Später nutzen wir diesen Befehl, um die Dateien für die Verbindungsinformationen **verbind.php** und die genutzten Funktionen **funk.php** einzubinden.

Dateien lesen und schreiben

Im nächsten Beispiel sollen Dateien, die auf dem Server liegen, gelesen bzw. editiert werden. Dies ist nützlich, um z.B. Zähler oder Gästebücher zu erzeugen. Dazu muss die Datei geöffnet, gelesen bzw. verändert und geschlossen werden. Mit Hilfe der PHP-Befehle lässt sich das wie folgt realisieren.

Du öffnest die Datei mit Hilfe der Funktion **fopen**. Das Resultat wird in der Variable **$datei** abgespeichert.

```
$datei = fopen("Datei","Parameter");
```

Dabei können die Parameter folgende Werte annehmen.

Parameter	
r	nur lesen, begonnen wird am Dateianfang
r+	lesen und schreiben, begonnen wird am Dateianfang
w	nur schreiben. Existiert die Datei bereits, wird der bisherige Inhalt gelöscht. Existiert sie nicht, wird versucht, sie zu erzeugen.
w+	lesen und schreiben, ansonsten wie w
a	nur schreiben. Begonnen wird am Ende der Datei (a wie append, anhängen). Existiert sie nicht, wird versucht, sie zu erzeugen.
a+	lesen und schreiben, ansonsten wie a.

Nun wird zeilenweise der Inhalt der Daten gelesen, bis das Ende der Datei erreicht ist:

```
while (!feof($datei)) {
$zeile = fgets($datei,1000);
echo $zeile;
}
```

Betrachten wir die einzelnen Anweisungen

`feof($datei)`	ist wahr, sobald das Dateiende erreicht wurde.
`$zeile = fgets($datei,1000);`	liest maximal die nächsten 1000 Zeichen, hört aber auf, sobald eine neue Zeile beginnt, oder das Ende der Datei erreicht ist.
`echo $zeile;`	gibt das Gelesene aus.

Mit `fwrite($datei, "Text");`

kann der String **Text** in die Datei geschrieben werden.

Zum Schluss ist die Datei wieder zu schließen.

```
fclose($datei);
```

Soll der Inhalt einer Datei im Browser ausgelesen werden, nutzt man die Funktion **readfile** :

```
readfile (Dateiname)
```

Die vorgestellten Funktionen werden in den nachfolgenden Beispielen eingesetzt.

Praxisbeispiele Dynamischer Textwechsler

Zum Schluss dieses Kaptitels sehen wir uns vier kleine Praxisbeispielen mit PHP an, die leicht zu erweitern sind. Die eben vorgestellte Funktionen **getdate()** soll eingesetzt werden, um Änderungen auf deiner Webseite automatisch vornehmen zu lassen.

Mit einem Trick kannst du bei deinen Besuchern den Eindruck erzeugen, dass sich auf deiner Seite etwas tut. Jeden Monat möchtest du einen Spruch des Monats vorstellen. Damit du die monatliche Veränderung nicht zu programmieren vergisst, überlässt du diese Aufgabe dem php-Programmskript **dyn.php**

```
<?php
$mon = getdate();
$cmonat = $mon['month'];
$monat = $mon['mon'];
echo "Spruch des Monats $cmonat <BR>";
//die Monatszahl wird in einen Dateinamen umgewandelt
$datei=$monat.".txt";
// die Datei wird gelesen und im Browserfenster ausgegeben
readfile ($datei);
?>
```

In die Variable **$cmonat** wird der Monatsname und in die Variable **$monat** die Nummer des Monat ausgelesen. Die Sprüche speicherst du in den Textdateien **1.txt** bis **12.txt**. Die Ausgabe im Browserfenster sieht im Monat Juli so aus:

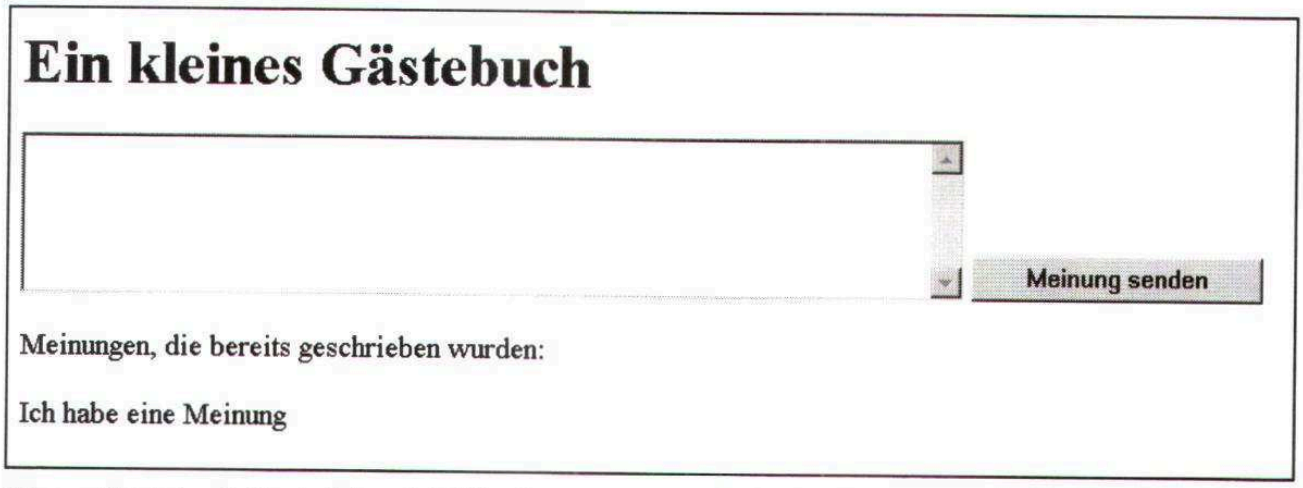
> Spruch des Monats July
> Weise Lebensführung gelingt keinem Menschen durch Zufall. Man muss, so lange man lebt, lernen, wie man leben soll. Autor: Seneca (Lucius Annaeus, der Jüngere)

Gästebuch

Als nächstes stelle ich dir das Programmskript **gast.php** vor. Es handelt sich um ein kleines Gästebuch. Zusätzlich benötigst du eine leere Textdatei **gast.txt**, in welche die Einträge gespeichert werden.

```
<html>
<head>
</head>
<h1>Ein kleines Gästebuch</h1>
<form action="<? echo $PHP_SELF?>" method="POST">
  <textarea cols=60 rows=5 name="meinung" wrap=virtual></textarea>
  <input type="submit" value=" Meinung senden ">
</form>
<?
if(isset($meinung)) {
   $fp = fopen("gast.txt","a");
   fwrite($fp,nl2br($meinung)."<p>\n");
   fclose($fp);
}
?>
<p>Meinungen, die bereits geschrieben wurden: </p>
<? readFile("gast.txt") ?>
</body>
</html>
```

In der Zeile `<form action="<? echo $PHP_SELF?>" method="POST">` wurde ein kleiner Trick angewendet. Wenn ein Formular mit der Methode POST versendet wird, speichert PHP den Inhalt in der Variable **$PHP_SELF**. Das Programmskript ruft sich somit beim Absenden selber auf. Mit der Funktion **isset** wird überprüft, ob die Variable **$meinung** belegt wurde. Ist das der Fall, wird die Datei **gast.txt** geöffnet und der Inhalt des Textfeldes **meinung** wird übertragen. Anschließend wird die Datei wieder geschlossen. Der Inhalt der Datei wird dann angezeigt durch die Befehlszeile `<? readFile("gast.txt") ?>`.



Dieses Gästebuch lässt sich mit Hilfe einer Datenbank erweitern und flexibler gestalten.

Zähler

Als nächstes habe ich ein Beispiel **zaehl.php** für einen Zähler vorbereitet. Die Speicherung des Zählerstandes erfolgt in der Textdatei **zaehl.txt**

Stellt das Programmskript fest, dass die Datei **zaehl.txt** noch nicht existiert, wird sie angelegt. Die weitere Vorgehensweise ist einfach. Die Datei **zaehl.txt** wird geöffnet, eingelesen und als Zahl interpretiert. Der Zähler wird um eins erhöht und zurückgeschrieben.

```
<?
function zaehle() {
  $zaehldatei =  "zaehl.txt";
  if (file_exists($zaehldatei))
  {
     $fp = fopen($zaehldatei, "r+");
     $zaehler = fgets($fp,6);
     $zaehler++;
     rewind($fp);
     fwrite($fp,$zaehler,6);
     fclose($fp);
  }
  else
  {
     $fp = fopen($zaehldatei, "w");
     $zaehler =  "1";
     fwrite($fp,$zaehler,6);
     fclose($fp);
```

```
  }
  return $zaehler;
}
?>
<html>
<head>
</head>
<body>
<h1>Zähler</h1>
<p>Der aktuelle Zählerstand ist: </p>
<b><? echo zaehle() ?></b>
<p>
<a href="<? echo $PHP_SELF ?>">Startet das Programmskript zur Erhöhung des
Zählers</a>
</body>
</html>
```

Deine Bildschirmseite enthält folgendes Ergebnis:

Zähler

Der aktuelle Zählerstand ist:

29

Startet das Skript erneut zur Erhöhung des Zählers

Die einzige Funktion aus dem obigen Beispiel, die noch nicht kennst, ist die rewind-Funktion, die so aussieht: `rewind($fp);`

Sie setzt den Dateizeiger auf das erste Byte der Datei. Tritt ein Fehler auf, gibt der Funktionsaufruf 0 zurück.

Cookies

Du hast bereits ein Beispiel kennengelernt, bei dem Daten in Textdateien gespeichert wurden. Die Dateien liegen dabei auf dem Webserver.

Sollen hingegen Zeichenfolgen auf der Festplatte der Clienten abgelegt werden, so spricht man von Cookies (deutsch: Kekse). Die Cookies können z.B. Informationen über den Anwender enthalten, um diesen bei seinem nächsten Besuch wieder zu erkennen. Cookies haben eine bestimmte Lebensdauer, die zwischen Sekunden und Jahren liegen kann. Die maximale Länge eines Cookies beträgt 2048 Byte.

Cookies werden über die Funktion Setcookie()gesetzt:

```
setcookie (name, [inhalt], [verfallsdatum], [pfad], [domain], [secure])
```

Abgesehen vom Argument **name** sind alle Argumente optional. Du kannst beliebige Argumente auch durch einen Leerstring ("") ersetzen, um sie zu übergehen. Der Parameter **verfallsdatum** und das Argument secure sind mit Null aufzufüllen.. Der **verfallsdatum** -Parameter ist ein normaler UNIX-Zeitwert als Integer-Zahl. Die Angabe des **secure**-Argument bedeutet, dass das Cookie nur über eine sichere HTTP-Verbindung geschickt werden soll.

Das folgende Formular fordert dich auf einen Namen einzugeben.


```
<html>
<head>
<title>Beispiel für ein Cookie</title>
</head>
<body>
<form action = "keks.php">
  <p><b>Bitte geben Sie Ihren Namen ein:</b></p>
  <p><input type="text" name="cotxt" size="50"></p>
  <p><input type="submit" value="Abschicken" name="B1"></p>
</form>
</body>
</html>
```

Beim Absenden des Formulars wird ein zweites Dokument **keks.php** angezeigt, das den eingegebenen Text darstellt und als Cookie abspeichert.

Ihr Name lautet: Petra

Die Syntax von **keks.php** lautet:

```
<?php
setcookie("cname",$cotxt, time()+
72*3600);
echo "Ihr Name lautet: $cotxt <br>";
?>
```

Dieses Cookie existiert 72 Stunden ab dem aktuellen Zeitpunkt.

So, für den Einstieg in PHP soll es an dieser Stelle genug sein. Ich hoffe, es ist dir gelungen, die Beispiele nachzuvollziehen und beim Ausprobieren ähnliche Resultate zu erzielen, wie ich. Überzeuge dich von meinen Resultaten unter der Webadresse
`//http:www.bilke.de/mysql`

Ist dein Interesse an PHP erwacht, findest du eine ausführliche PHP-Dokumentation auf der Internetseite `www.php.net`.

Datenbankprogrammierung mit PHP und MySQL

Zugriff auf Datenbanken

Ehe wir uns den konkreten Fragen der Datenbankarbeit widmen, wäre zu erklären, was eine Datenbank bzw. ein Datenbankverwaltungssystem ist.

Eine **Datenbank** ist eine Sammlung von Daten, die sich auf ein bestimmtes Thema oder einen bestimmten Zweck beziehen. Sie ermöglicht dem Benutzer den Zugriff auf gespeicherte Daten, ohne dass er wissen muss, wie die Daten im Datenbanksystem organisiert sind. Eine Datenbank gewährleistet, dass kein Benutzer ohne Zugriffsberechtigung Daten sichten oder mit ihnen manipulieren kann.

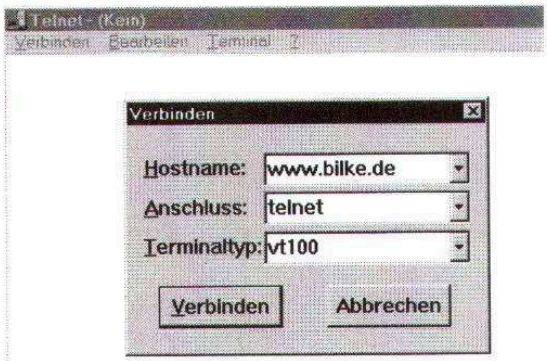
Das **Datenbankverwaltungssystem** bildet den Kern der Datenbank; es beinhaltet alle für die Datenverwaltung notwendigen Routinen. Die Datenbanksprache bildet die Schnittstelle zwischen Benutzer und Datenbankverwaltungssystem. Bei vielen Datenbanksystemen wird als Datenbanksprache die Sprache SQL (Structured Query Language) verwendet.

Das Datenbanksystem kann auf verschiedene Weise angesteuert werden:

- dialogorientiert
- im Batch-Betrieb
- durch andere Programme

Dialogorientiert bedeutet, das nach dem Start des SQL-Servers alle Befehle direkt eingeben und ausgeführt werden können.

Handelt es sich um mehrere Befehle, kann man diese zuvor in Textdateien schreiben und im Batch-Betrieb abarbeiten lassen. Anschließend sehen wir uns an, wie du dialogorientiert mit der Datenbank arbeitest, die bei deinem Provider vorhanden ist.



Zunächst verbindest du dich über eine Telnet-Sitzung zu deinem Provider. Bei Windows befindet sich ein Telnet-Programm im Lieferumfang. Du kannst es über **Start** und **Ausführen** und den Eintrag "**telnet**" in der Dialogbox starten. Das Menü Verbinden ist zu starten.

Dieses Methode ist nicht sehr komfortabel. Trotzdem solltest du zunächst dieses Weg versuchen, um sicher zu stellen, dass deine Verbindungsinformationen richtig sind.

Eine Telnet-Sitzung würde wie folgt ablaufen:

```
WebMailer Telnet Access  - Welcome to the new world order
Login: www.bilke.de
Password: ...
www.bilke.de> mysql -h rdbms -u www.domainname.de -p Dbname
Enter password: ...
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 12903 to server version: 3.22.32

Type 'help' for help.
mysql> help
```

```
MySQL commands:
help     (\h)    Display this text
?        (\h)    Synonym for `help'
clear    (\c)    Clear command
connect  (\r)    Reconnect to the server. Optional arguments are db
and host
edit     (\e)    Edit command with $EDITOR
exit     (\q)    Exit mysql. Same as quit
go       (\g)    Send command to mysql server
ego      (\G)    Send command to mysql server; Display result
vertically
print    (\p)    Print current command
quit     (\q)    Quit mysql
rehash   (\#)    Rebuild completion hash
status   (\s)    Get status information from the server
use      (\u)    Use another database. Takes database name as
argument
Connection id: 12903  (Can be used with mysqladmin kill

mysql> SELECT VERSION(), CURRENT_DATE;
+-----------+--------------+
| version() | CURRENT_DATE |
+-----------+--------------+
| 3.22.32   | 2001-06-18   |
+-----------+--------------+
1 row in set (0.02 sec)

mysql> DESCRIBE teilnehmer;
+----------------+---------+------+-----+---------+-------+
| Field          | Type    | Null | Key | Default | Extra |
+----------------+---------+------+-----+---------+-------+
| laufende_NR    | int(5)  | YES  |     | NULL    |       |
| Personalnummer | int(5)  | YES  |     | NULL    |       |
| besucht        | char(1) | YES  |     | NULL    |       |
+----------------+---------+------+-----+---------+-------+
3 rows in set (0.04 sec)
mysql> \q
Bye
www.bilke.de>
```

Die Ansteuerung über andere Programme ist wesentlich komfortabler. Geeignet ist etwa die Programmiersprache PHP, die mit Datenbanken verschiedener Hersteller umgehen kann.

Um eine SQL-Abfrage mit PHP auszuführen, müssen vergleichbare Schritte wie beim Zugriff auf eine Datei erfolgen:

- die Datenbank muss geöffnet werden,
- die SQL-Befehlszeile wird an die Datenbank geschickt,
- die Antwort der Datenbank erfolgt,
- die Datenbankverbindung wird geschlossen.

Bevor eine SQL-Abfrage gestartet werden kann, muss eine Verbindung zum Datenbankserver aufgebaut werden, der sich in der Regel bei einem Provider befindet. Dazu müssen folgende Punkte geklärt werdem:

- Name und Port des Rechners, auf dem der Datenbankserver läuft,
- eine Benutzerkennung
- das dazugehörige Passwort

Willst du auf eine Datenbank mit PHP zugreifen, benutzt du folgenden Befehl:

```
<?
$link = mysql_connect("host:port","username","password");
?>
```

War die Verbindung erfolgreich, gibt PHP einen sog. "link identifier" zurück, der bei Operationen mit der Datenbank zu übergeben ist. Damit MySQL weiß, welche Datenbank gerade gefragt ist, muss zuerst eine ausgewählt werden. Alle folgenden Befehle beziehen sich dann auf diese Datenbank, bis eine neue ausgewählt wird.

```
<?
mysql_select_db("database",$link);
?>
```

Die Befehle sind in den Programmskripten **verbind.php** und **test.php** aus dem nächsten Abschnitt enthalten.

Auch der weitere Zugriff auf die Datenbank gestaltet sich einfach, wenn man die MySQL-Funktionen kennt. Die in den weiteren Beispielen benötigten MySQL-Funktionen habe ich in nachfolgender Tabelle dargestellt.

Funktion	Beispiel	Beschreibung
mysql_affect_rows	`$anzahl= mysql_affect_rows($vid)`	Liefert die Anzahl der betroffenen Datensätze einer vorhergehenden MySQL-Operation
mysql_close	`mysql_close($vid)`	Schließt die Verbindung zum Datenbankserver.
mysql_connect	`mysql_connect($host,$user, $password)`	Stellt eine Verbindung zum MySQL-Server her.
mysql_db_query	`$res=mysql_db_query($db,$sql,$vid)`	Absetzen einer SQL-Anfrage **$sql** an die Datenbank.
mysql_error	`$errmsg= mysql_error($vid)`	Liefert den Fehlertext **$errmsg** der zuvor ausgeführten Operation
mysql_fetch_array	`$array= mysql_fetch_array($res,$type)`	liefert eine Zeile (einen Datensatz) aus einem Ergebnis einer SELECT-Abfrage und springt danach zur nächsten Zeile. Der Rückgabewert ist ein Array **$array**, das die Werte der Zeile enthält. $type gibt den Typ des Array an. MYSQL_ASSOC: assoziativ MYSQL_NUM: nummerisch MYSQL_BOTH: beides
mysql_field_name	`$name= mysql_field_name($res,$index)`	Liefert den Namen eines Feldes mit dem Index **$index** in einem Abfrageergebnis.
mysql_num_fields	`$anz= mysql_num_fields($res)`	Liefert die Anzahl **$anz** der Felder in einem Abfrageergebnis.
mysql_num_rows	`$anz= mysql_num_rows($res)`	Liefert die Anzahl **$anz** der Datensätze in einem Abfrageergebnis.
mysql_pconnect	`mysql_pconnect($host,$user, $password)`	Stellt eine persistente Verbindung zum MySQL-Server her.
mysql_query	`$succ= mysql_query($sql,$vid)`	Sendet über einen vorher geöffneten Verbindung eine SQL-Abfrage **$sql** an den Datenbankserver.
mysql_select_db	`$succ= mysql_select_db($db,$vid)`	Wählt eine Datenbank **$db** aus.

PHP-Gerüst zum Zugriff auf die Datenbank

Damit die folgenden SQL-Befehle mit einem PHP-Programmskript getestet werden können, habe ich ein allgemeines PHP-Gerüst entwickelt. Dieses PHP-Gerüst besteht hier aus drei Dateien. Die Dateien **verbind.php** und **funk.php** stellen die Verbindungsdaten und spezielle Funktionen zur Verfügung. Diese werden mit dem Include-Befehl in die Datei **test.php** eingefügt. Der jeweilige SQL-Befehl ist in der Datei **test.php** einzufügen.

Das Programmskript **verbind.php** stellt die Verbindungsdaten zur Verfügung.

hostname, **username**, **password** und **datenbankname** sind durch deine eigenen Zugangsdaten zu spezifizieren.

```
<?
/* Zugangsdaten zum Verbinden mit  dem MySQL-Server */
$link=mysql_connect("hostname","username","password");
// Auswahl der zu verwendenden Datenbank auf dem Server
$db ="datenbankname";
?>
```

Das Programmskript **funk.php** beinhaltet die eigenen Funktionen.

```
<?
/* Funktionen für eine einfache Datenbank-Schnittstelle */
function send_sql($db, $sql)
{
 if (! $res=mysql_db_query($db, $sql)){
   echo mysql_error();
   exit;
}
   return $res;
}
?>
```

Das Programmscript **test.php** muss verändert werden. Ergänze Titel, Überschrift und SQL-Befehl:

```
<html>
<head>
<title>....</title>
</head>
<body>
<h1> .... </h1>
<?
include("funk.php");
include("verbind.php");

$sql=" ... SQL-Befehl ...";

If ($res=send_sql($db,$sql))  {
echo "SQL-Kommando wurde ausgeführt";
}
?>
</body>
</html>
```

Soll der SQL-Befehl noch einmal angezeigt werden, wechselst du folgenden Programmkode:

```
If ($res=send_sql($db,$sql))  {
echo "SQL-Kommando wurde ausgeführt";
}
```

gegen diesen aus:

```
If ($res=send_sql($db,$sql))  {
   echo "Abfrage: <br> $sql";

}
```

SQL-Grundlagen

Nachfolgend gehen wir davon aus, dass die Datenbank durch dich oder deinen Provider angelegt wurde. Die beschriebenen SQL-Befehle kannst du zum Testen in das obige Programmskript **test.php** einsetzen. Die vollständigen Skripte findest du unter `www.bilke.de/mysql`. Hier kannst du die Programmskripte auch ausführen.

MySQL entspricht weitestgehend dem Entry Level der Standard ANSI SQL 92. Dennoch gibt es Erweiterungen und fehlende Funktionen.

Diese Erweiterungen werden hier angesprochen:

- Der **`select`**-Befehl ist erweitert um die Parameter **`into_outfile`** und **`straight_join`**.
- Der neue Befehl **`replace`** ersetzt die Kombination aus **`delete`** und **`insert`**.
- Die Felddatentypen **MEDIUMINT**, **SET** und **ENUM** sind zusätzlich vorhanden.
- MySQL bildet jede Datenbank als Verzeichnis ab und jede Tabelle als Datei. Ist eine Datenbank auf einem UNIX-System gespeichert, ist der Unterschied zwischen Groß- und Kleinschreibung zu beachten.
- Der Befehl SHOW ist neu.

Leider hat MySQL auch einige Einschränkungen:

- Unterabfragen sind nicht erlaubt. So ist folgender Befehl in MySQL unzulässig.

```
SELECT pe.personalnummer
FROM teilnehmer
WHERE laufende_nr IN
(SELECT  laufende_nr
FROM schulungsdatenbank );
```

- Gespeicherte Prozeduren sind nicht unterstützt.
- Trigger werden nicht unterstützt.
- Sichten (Views) werden nicht unterstützt.
- Der **`union`**-Befehl wird nicht unterstützt.
- Fremdschlüssel werden nicht unterstützt. Der Parameter **`foreign key`** im **`select`**-Befehl existiert nur aus Kompatibilitätsgründen, hat aber keine Funktion.

Bei MySQL gelten folgende Namenskonventionen für Datenbanken, Tabellen, Indizes, Spalten und Aliase:

- Es dürfen alphanumerische Zeichen und der Unterstrich verwendet werden. Der Standardzeichensatz ist ISO-8859-1 Latin 1.
- Namen für Datenbanken, Tabellen, Indizes, Spalten dürfen 64 Zeichen lang sein, Aliasnamen 256 Zeichen lang.

Feldnamen können in 3 Varianten vorkommen

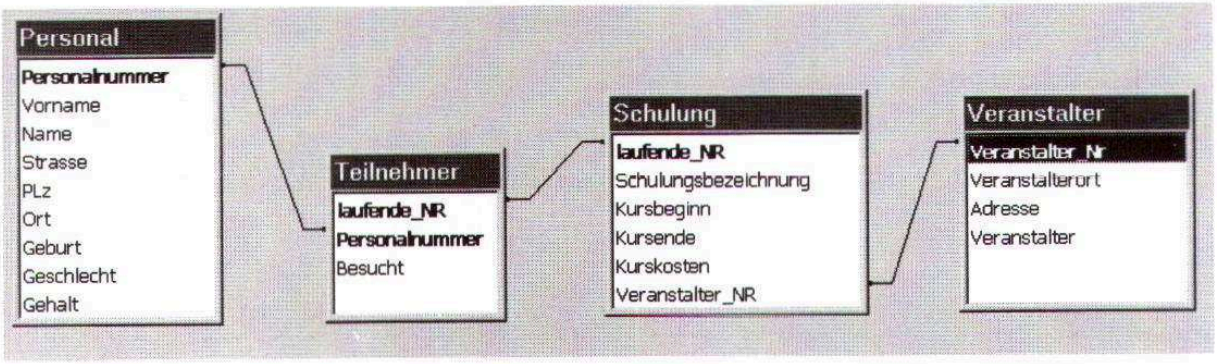
- feld
- tabelle.feld
- datenbank.tabelle.feld

Die ausführliche Schreibweise brauchst du nur zu wählen, wenn die Feldnamensauswahl nicht eindeutig ist.

Eine besondere Bedeutung besitzt der Nullwert **`null`**, der für undefinierte Felder oder nicht vorhandene Daten steht. Die Zahl 0 oder leere Zeichenketten " " werden also nicht durch den Nullwert NULL repräsentiert.

CREATE TABLE

Weiter oben habe ich unser Datenbankbeispiel schon vorgestellt. Die Mitarbeiter der Abteilung Weiterbildung eine Firma wollen die besuchten Schulungen der Betriebsangehörigen erfassen.



Der entwickelte Datenbankentwurf muss umgesetzt werden. Also werden als erstes Tabellen angelegt und Einschränkungen formuliert.

Dir steht zu diesem Zweck der Befehl CREATE TABLE mit folgender Syntax zur Verfügung.

```
CREATE TABLE Tabelle
(
Feld  Datentyp [NOT NULL | NULL] [DEFAULT Defaultwert] [AUTO_INCREMENT]
PRIMARY KEY (Indexname,...) |
KEY [Indexname, ...] |
INDEX [Indexname, ...] |
UNIQUE (Indexname,...) |
[CONSTRAINT Symbol]
FOREIGN KEY (Indexname,...)[Referenz] oder  CHECK (Feld)
)
```

Beachte bei der Namensauswahl, dass MySQL jede Tabelle als Datei abbildet. Wurde deine Datenbank auf einem UNIX-System hinterlegt, wird bei Tabellen Groß- und Kleinschreibung unterschieden. Ich wähle für die Tabellennamen die Kleinschreibung.

Beim **CREATE TABLE**-Befehl musst du einen Tabellennamen (**Tabelle**) und einen Feldnamen (**Feld**) auszuwählen. Für jedes Feld ist ein Datentyp (**Datentyp**) festzulegen. Alle anderen Parameter können, aber brauchen nicht genutzt zu werden. Folgende Datentypen stehen bei MySQL zur Verfügung:

Felddatentyp-numerisch	**Speicher-größe in Byte**	**Beschreibung**
TINYINT[(Länge)]	1	Ganzzahl-Wertebereich: -128 bis 127 bzw. 0 bis 255
SMALLINT[(Länge)]	2	Ganzzahl-Wertebereich: -32.768 bis 32.767 bzw. 0 bis 65.535
MEDIUMINT[(Länge)]	3	Ganzzahl-Wertebereich: -8.388.608 bis 8.388.607 bzw. 0 bis 4.294.967.295
INT[(Länge)] / INTEGER[(Länge)]	4	Ganzzahl-Wertebereich: -2.147.483.648 bis 2.147.483.647 bzw. 0 bis 4.294.967.295
BIGINT[(Länge)]	8	Ganzzahl-Wertebereich: -9.223.372.036.854.775.808 bis 9.223.372.036.854.775.807 bzw. 0 bis 18.446.744.073.709.551.615
REAL[(Länge, Dezimalstellen)] / DOUBLE[(Länge, Dezimalstellen)]	8	Fließkommazahl
FLOAT[(Länge, Dezimalstellen)]	4	Fließkommazahl
DECIMAL(Länge,Dezimalstellen) / NUMERIC(Länge,Dezimalstellen)		ungepackte Fließkommazahl mit Vorzeichen, Zahlen werden als Zeichenketten gespeichert

Alle Felddatentypen können mit folgenden Parametern ergänzt werden

- [UNSIGNED]-Vorzeichen wird vernachlässigt
- [ZEROFILL]-fehlende Werte werden mit Null aufgefüllt

Felddatentyp-Zeichenkette	**Speicher-größe in Byte**	**Beschreibung**
CHAR(Länge) [BINARY]	1 bis 255	Zeichenkette Länge: 1 bis 255
VARCHAR(Länge) [BINARY]	variabel	Zeichenkette variabler Länge
TINYBLOB / TINYTEXT		BLOB oder TEXT mit max. 255 Zeichen
BLOB / TEXT		BLOB oder TEXT mit max. 65.535 Zeichen
MEDIUMBLOB / MEDIUMTEXT		BLOB oder TEXT mit max. 16.777.215 Zeichen
LONGBLOB / LONGTEXT		BLOB oder TEXT mit max. 4.294.967.295 Zeichen
ENUM(Wert1,Wert2,Wert3...)		Aufzählung. Max. 65535 Einzelwerte einer Liste
SET(Wert1,Wert2,Wert3...)		wie oben, aber Gruppe von Werten-max. 64

Felddatentyp-Datum	**Speicher-größe in Byte**	**Beschreibung**
DATE	3	Datum vom Typ YYYY-MM-DD
DATETIME	8	Datum vom Typ YYYY-MM-DD HH:MM:SS
TIME	3	Zeit vom Typ HH:MM:SS
TIMESTAMP	4	UNIX-Zeitstempel
YEAR	1	Jahr vom Typ YYYY

Für unsere Beispiele könnten die **CREATE TABLE** Befehle wie folgt aussehen (**create1.php, create2.php, create3.php, create4.php**).

```
CREATE TABLE personal
(Personalnummer INT (5) NOT NULL
PRIMARY KEY,
Vorname CHAR(30),
Name CHAR(30),
Strasse CHAR(30),
PLZ CHAR(6),
Ort CHAR(30),
Geburt DATE,
Geschlecht CHAR(1),
Gehalt DECIMAL(7,2) );
```

```
CREATE TABLE schulung
(laufende_NR INT(5) NOT NULL PRIMARY
KEY,
Schulungsbezeichnung CHAR(30),
Kursbeginn DATE,
Kursende DATE,
Kurskosten DECIMAL(7,2),
Veranstalter_NR INT(5));
```

```
CREATE TABLE teilnehmer
(laufende_NR INT(5),
Personalnummer INT(5),
besucht CHAR(1));
```

```
CREATE TABLE veranstalter
(Veranstalter_NR INT(5) NOT NULL
PRIMARY KEY,
Veranstaltungsort CHAR(30),
Adresse CHAR(40),
Veranstalter CHAR(30));
```

Das vollständige Programmskript für **create1.php** sieht so aus:

```
<html>
<head>
<title>Personaltabelle </title>
</head>
<body>
<h1>Personaltabelle erstellen</h1>
<?
include("verbind.php");
include("funk.php");
$sql="CREATE TABLE personal
(Personalnummer INT (5) NOT NULL PRIMARY KEY,
Vorname CHAR(30),
Name CHAR(30),
Strasse CHAR(30),
PLZ CHAR(6),
Ort CHAR(30),
Geburt DATE,
Geschlecht CHAR(1),
Gehalt DECIMAL(6,2) )";

 If ($res=send_sql($db,$sql))  {
   echo "SQL-Kommando wurde ausgeführt";
}
?>
</body>
</html>
```

Die Ausführung des Programms ergibt das Bildschirmbild:

Personaltabelle erstellen

SQL-Kommando wurde ausgeführt

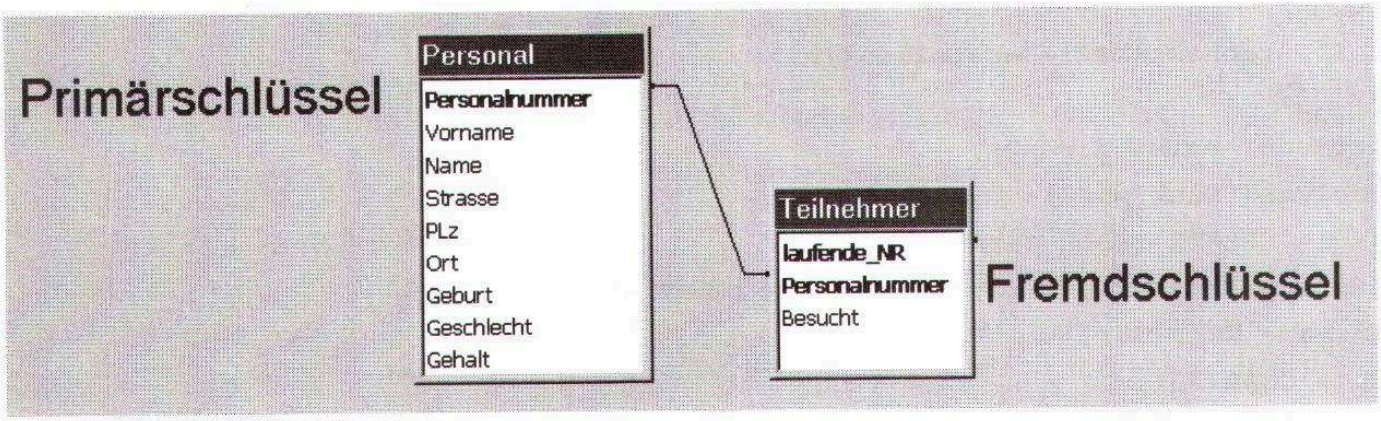
Neben den beschriebenen Parametern habe ich an einigen Stellen der **create**-Befehle den Parameter **not null** eingesetzt. Das bedeutet, dass in diesem Feld die Eingabe von Werten zwingend ist. Ich habe jeweils die Felder ausgewählt, die für den Primärschlüssel vorgesehen sind. Die Attributwerte dieser Felder dürfen nicht **null** sein. Meinst du, dass weitere Felder ebenfalls eine zwingende Eingabe benötigen, ergänze im Programmskript **not null**.

Für die Tabellen **personal, schulung** und **veranstalter** wurde jeweils 1 Feld als Primärschlüssel mit dem Parameter **PRIMARY KEY** festgelegt. Ein Primärschlüssel hat die Aufgabe zu gewährleisten, dass jeder Datensatz einer Tabelle eindeutig identifizierbar ist. Bekannte Beispiele für einen Primärschlüssel sind z.B. Seriennummern von Geldscheinen, Kraftfahrzeugnummern, Versicherungsscheinen oder Personalnummern.

Bei unseren Tabellen wurden die vorhandene Personalnummer und die zusätzlich eingeführten laufenden Nummer der Schulung, der Teilnehmer und der Veranstalter als Primärschlüssel gewählt.

Der Primärschlüssel muss nachfolgende Kriterien erfüllen:

- er muss eindeutig sein.
- er erhält für jeden neuen Datensatz sofort den entsprechenden Wert, der nicht leer bleiben darf (`not null`)



Besondere Bedeutung erlangen Fremdschlüssel bei der Definition der Beziehungen zwischen den Tabellen. Passend zu einem Primärschlüssel in der ersten Tabelle muss es einen sogenannten Fremdschlüssel in der zweiten Tabelle geben. Ein Fremdschlüssel in einer Tabelle 2 ist das Feld, das in einer Tabelle 1 den Primärschlüssel bildet.

Der Fremdschlüssel muss folgende Bedingungen erfüllen:

- Er darf nur diejenigen Feldinhalte annehmen, welche bereits in dem Primärschlüssel der Tabelle 1 existieren – also nicht leer sein.
- Die Feldnamen von Primärschlüssel und Fremdschlüssel dürfen nicht identisch sein.

In Tabelle 1 (**personal**) wurde als Primärschlüssel die `Personalnummer` vergeben. Soll eine Beziehung zwischen den beiden Tabellen hergestellt werden, wird in Tabelle 2 ein Fremdschlüssel mit entsprechenden Feldinhalten benötigt. Im obigen Beispiel ist es wiederum die `Personalnummer`. MySQL unterstützt leider keine Fremdschlüssel. Der Parameter `FOREIGN KEY` mit den damit zusammenhängenden Referenzen zwischen den Tabellen ist zwar aus Kompatibilitätsgründen vorhanden, wird aber noch nicht unterstützt. Also musst du öfters kontrollieren, ob die Daten-mäßigen Forderungen an das Feld, das den Fremdschlüssel darstellt, erfüllt sind. Klartext: du darfst in die Tabelle **teilnehmer** keine Datensätze eingetragen, die keine Entsprechung in der Tabelle **personal** haben. Für diese Kontrolle kannst du den Befehl `SELECT ... LEFT JOIN` einsetzen, den du auf Seite 53 kennen lernen wirst.

Da eine durchdachte Tabellenstruktur für die einzugeben Daten wichtig ist, sehen wir uns einige interessante Attribute des `CREATE`-Befehls an:

- Eine Spalte vom Typ INTEGER kann zusätzlich das Attribut **AUTO_INCREMENT** erhalten. In diesem Fall erhält ein solches Feld nach dem Einfügen eines neuen Datensatzes automatisch einen um 1 höheren Wert, der nicht geändert werden kann. Ein Feld dieser Art eignet sich gut als Primärschlüssel. Pro Tabelle kann jeweils nur ein Feld das Attribut **AUTO_INCREMENT** erhalten.
- Mit `[DEFAULT Defaultwert]` kann ein Vorgabewert angegeben werden. Finden fast alle Kurse in Berlin statt, könnte der Vorgabewert für den Kursort Berlin sein. Ist für ein Feld kein Defaultwert vorgegeben, wird er also als `NULL` definiert, wird der Wert **NULL** verwendet. Ansonsten legt **MySQL** automatisch je nach Datentyp einen Defaultwert fest.
  - Bei numerischen Datentypen ist dies 0.
  - Bei DATE und TIME Typen ist dies der **Startwert** dieses Typs.
  - Bei String-Typen ist dies ein leerer String.

- Neben demPrimärschlüssel können weitere Indexschlüssel vergeben werden, die dem schnellen Sortieren von Feldern und dem Wiederfinden von Daten dienen.

> Du solltest einen Indexschlüssel nur anlegen, wenn du beim Sortieren von Feldern und Wiederfinden von Daten Geschwindigkeitsprobleme hast. In diesem Falle ernennst du für einen Indexschlüssel die gewünschten Felder. Tritt der gewünschte Geschwindigkeitsgewinn nicht ein, löschst du um der Leistungsfähigkeit deiner Datenbank willen den Schlüssel wieder – und zwar mit dem Befehl **ALTER TABLE ... DROP INDEX ...**

- Nutze zum Anlegen eines Indexschlüssels die Argumente **KEY [Indexname, ...]** bzw. **INDEX [Indexname, ...]** oder **UNIQUE (Indexname,...)**
  UNIQUE legt fest, dass das Feld eindeutige Werte besitzt. Wird mehrmals derselbe Wert eingegeben, erfolgt eine Fehlermeldung.
- Die **FOREIGN KEY**, **CHECK** und **REFERENCES** Klausen haben zur Zeit keinerlei Auswirkung. Diese Syntax soll nur die Kompatibilität mit anderen SQL-Servern sichern.

Soll die Struktur der Tabellen geändert werden, ist der Befehl ALTER TABLE einzusetzen. Die Syntax ähnelt der Syntax des CREATE TABLE-Befehls, hat aber noch mehr Möglichkeiten.

```
ALTER TABLE Tabelle
ADD[COLUMN]  Feld  Datentyp [NOT NULL | NULL] [DEFAULT Defaultwert]
[AUTO_INCREMENT]|
ADD PRIMARY KEY (Indexname) |
ADD INDEX [Indexname, ...] |
ADD UNIQUE (Indexname,...) |
ALTER [COLUMN] Feld {SET DEFAULT | DROP DEFAULT}
CHANGE [COLUMN] Feld_alt Feld_definition |
MODIFY  [COLUMN] Feld_definition |
DROP [COLUMN] |
DROP PRIMARY KEY  |
DROP INDEX [Indexname, ...] |
RENAME AS Tabelle_neu
```

Die Attribute haben folgende Bedeutung:

- **ADD** fügt das jeweilige Element zur Tabelle hinzu.
- **ALTER** setzt den Standardwert neu.
- **CHANGE** ändert eine Feldelement und nennt die Spalte um.
- **MODIFY** ändert die Felddefinition (ohne Umbennung)
- **DROP** löscht das jeweilige Element
- **RENAME** benennt die Tabelle um.

Der **ALTER** Befehl wird später benötigt, um die Struktur der Tabellen **Personal** und **Schulung** um die Spalten **Euro_Gehalt** und **Euro_Kurskosten** zu ergänzen.

Einfügen von Daten in Tabellen

Daten können auf verschieden Weisen in eine Tabelle gebracht werden. Folgende Möglichkeiten sehen wir uns auf den nächsten Seiten kurz an:

- INSERT
- LOAD DATA
- Formulare

INSERT

Willst du Daten in die Tabellen einer Datenbank einfügen, benutzt du z.B. den SQL-Befehl **INSERT**, der einen neuen Datensatz in eine bestehende Tabelle einfügt. Der **INSERT ... VALUES**-Befehl fügt den Datensatz unter Verwendung der angegebenen, durch Kommata getrennten Werte ein. Für jeden neuen Datensatz ist ein neuer **INSERT**-Befehl einzugeben.

Die Syntax sieht so aus:

```
INSERT [INTO] Tabelle [(Feld,...)]
VALUES (Wert,...),...
```

Im nachfolgenden Beispiel **insert.php** wird in die Tabelle **schulung** ein Datensatz eingefügt.

```
<html>
<head>
<title>Schulungstabelle </title>
</head>
<body>
<h1>Schulungstabelle mit Daten füllen (INSERT)</h1>
<?
include("verbind.php");
include("funk.php");

$sql="INSERT INTO schulung
VALUES (1,'ABA Schulung','1996-03-14' ,'1996-03-15' ,1600,1)";

If ($res=send_sql($db,$sql))  {echo "SQL-Kommando wurde ausgeführt";}
?>
</body>
</html>
```

LOAD DATA

Handelt es sich um größere Datenmengen, ist es effizienter, mit dem Befehl **LOAD DATA INFILE** zu arbeiten. Dieser Befehl liest mit einer sehr hohen Geschwindigkeit Reihen aus einer Textdatei in eine Tabelle.

Syntax:

```
LOAD DATA [LOCAL] INFILE 'Textdatei' INTO TABLE Tabelle
[FIELDS [TERMINATED BY term]
[ENCLOSED by encl] [ESCAPED BY esc][(Feld,...)]
```

Befindet sich die Textdatei im nicht gleichen Pfad mit dem PHP-Skritp muss neben dem Dateinamen auch die relative Pfadangabe eingegeben werden.

TERMINATED	gibt die Feldtrennung an. '\t' steht für den Tabulator. Bei der vorliegenden Datei handelt es sich um eine kommagetrennte Datei.
ENCLOSED	gibt das Zeichen an, mit der die Zeichenkette eingeschlossen ist
ESCAPED	gibt das Zeichen an, mit dem Sonderzeichen getrennt werden.

Die FIELDS-Auswahl ist optional. Wird sie weggelassen, werden folgende Standardwerte verwendet:

TERMINATED	'\t'
ENCLOSED	''
ESCAPED	'\\'

Für das Einlesen der Daten in Tabelle **personal** kann der obige HTML-Code der Datei **test.php** wieder übernommen werden. Die Variable $sql muss natürlich ersetzt werden (**load1.php**).

```
$sql = "LOAD DATA LOCAL INFILE 'personal.txt' INTO TABLE personal FIELDS
TERMINATED BY ',' (Personalnummer, Vorname, Name, Strasse, PLZ, Ort,
Geburt, Geschlecht, Gehalt)";
```

Die Datei **personal.txt** hat dabei folgendes Aussehen:

```
1008,Eike,Taro,Musterstr. 6,09999,Musterdorf,1947-9-10,M,5100
1002,Kai,Müller,Hallesche Str. 16,04838,Eilenburg,1951-12-28,M,3400
1112,Hannes,Heik,Musterstr. 145,09999,Musterdorf,1961-2-6,M,5600
1429,Torsten,Mieder,Hauptstrasse 16,04838,Laussig,1971-5-15,M,5400
...
```

Für die Daten der Tabelle **teilnehmer** ist der SQL-Befehl zu modifizieren (**load2.php**).

```
$sql = "LOAD DATA LOCAL INFILE  'teilnehm.txt' INTO TABLE teilnehmer FIELDS
TERMINATED BY ',' (laufende_NR,Personalnummer,besucht)";
```

Die Datei **teilnehm.txt** hat folgendes Aussehen:

```
1,1002,1
1,1017,1
1,1429,1
1,1101,1
...
```

Formulare

Erscheinen die ersten beiden Möglichkeiten zur Dateneingabe nicht komfortabel genug, stellen eventuell HTML-Formulare eine effizientere Möglichkeit dar.

Ein einfaches Beispiel für Formulare habe ich bereits auf Seite 16 vorgestellt. Nun sollen die Veranstalterdaten mit Hilfe eines Formulars eingegeben werden, was wie folgt aussehen könnte.

Eingabeformular

Bitte geben Sie die Veranstalterdaten ein:

Veranstalternummer:

Veranstaltungsort:

Adresse:

Veranstalter:

Abschicken Löschen

Dieses Formular entsteht auf der Basis der Datei **input_form.htm** zu erfassen. Es handelt sich hier um HTML-Befehle zur Erstellung von Formularen und Tabellen.

```
<html>
<head>
<title>Eingabeformular</title>
</head>
<body>
<h1>Eingabeformular</h1>
<b>Bitte geben Sie die Veranstalterdaten ein:</b>
    <FORM ACTION="input.php" METHOD="POST">
    <TABLE BORDER=0>
       <TR>
       <TD>Veranstalternummer:
       <TD><INPUT TYPE="text" NAME="veranstalter_nr" SIZE="12">
       <TR>
       <TD>Veranstaltungsort:
       <TD><INPUT TYPE="text" NAME="veranstaltungsort" SIZE="31">
       <TR>
       <TD>Adresse:
       <TD><INPUT TYPE="text" NAME="adresse" size="60">
       <TR>
       <TD>Veranstalter:
       <TD><INPUT TYPE="text" NAME="veranstalter" size="31">
       <TR>
       <TD
       <TD><INPUT TYPE=submit VALUE="Abschicken">
       <INPUT TYPE=reset VALUE="Löschen">
    </table>
</body>
</html>
```

In dem Befehl `<FORM ACTION="input.php" METHOD="POST">` wird die Datei **input.php** aufgerufen, die wiederum den SQL- INSERT-Befehl enthält.

Die Werte werden den im Formular ausgefüllten Variablen entnommen. Sollen die Daten erneut angezeigt werden, geht das mit dem Echo-Befehl. Die relevanten Befehle aus der Datei **input.php** sind im Anschluss dargestellt.

```
...
$sql = "insert into veranstalter (veranstalter_nr,veranstaltungsort,
adresse, veranstalter)
values ('$veranstalter_nr', '$veranstaltungsort', '$adresse',
'$veranstalter')";

...
echo "<br>Veranstalter:          ", $veranstalter_nr;
echo "<br>Veranstaltungsort:     ", $veranstaltungsort;
echo "<br>Adresse:               ", $adresse;
echo "<br>Veranstalter:          ", $veranstalter;
```

Bein vorliegenden Beispiel handelt es sich um ein Eingabeformular. Zur Anzeige des Datensatzes werden die Variablen ausgelesen und mit **echo** angezeigt. Sollen die Daten aus der Tabelle angezeigt, gelöscht oder geändert werden, benötigst du weitere SQL-Befehle.

anzeigen	SELECT	vgl. Seite 40
löschen	DELETE	vgl. Seite 60
ändern	UPDATE	vgl. Seite 61

Datenbankrecherche

Tabellarische Anzeige von Rechercheergebnissen

Der `SELECT`-Befehl ist das zentrale Element im Befehlsumfang von SQL. Mit diesem Befehl werden die benötigten Daten aus einer oder mehreren Tabellen ausgelesen. Das Resultat kann eine aber auch mehrere Reihen enthalten. Soll mehr als eine Reihe angezeigt werden, empfehle ich, die Datensätze tabellarisch darstellen zu lassen. Aus diesem Grund habe ich die Datei **funk.php** um die Funktion `tab_out($result)` erweitert, die feststellt, wie viele Felder und Reihen mit dem `SELECT`-Befehl ausgelesen werden, und die benötigten Tabellenzellen erstellt, in die die Werte eingetragen werden.

Es folgt der Programmcode, mit dem du deine Datei **funk.php** erweitern solltest.

```
function tab_out($result)
{
$anz=mysql_num_fields($result);
$breit=100/$anz."%";
echo "<table width=100% border=0 cellpadding='2' cellspacing='2'>";
echo "<tr bgcolor=#D0D0D0>";
for ($i=0;$i<$anz;$i++)
{
echo "<th width='$breit'><font size='1'> ";
echo mysql_field_name($result,$i);
echo "</font> </th>";
}
echo "</tr>";
echo "</table>";
echo "<tr>";
$num = mysql_num_rows($result);
for ($j = 0; $j < $num; $j++)
{
$row = mysql_fetch_array($result);
echo "<table width=100% border=0 cellpadding='2' cellspacing='2'>";
echo "<tr bgcolor=#00FFFF>";
for ($k=0;$k<$anz;$k++)
{
$fn=mysql_field_name($result,$k);
echo " <td width='$breit'> <font size='1'> $row[$fn] </font> </td> " ;
}
echo "</tr>";
echo "</table>";
}
}
```

In die Variable **$anz** wird die Anzahl der anzuzeigenden Felder gespeichert, und zwar mit Hilfe der Funktion **mysql_num_rows**. Für alle Felder wird eine identische Spaltenbreite vereinbart, indem die volle Breite der Bildschirmanzeige durch die ermittelte Anzahl aus **$anz** dividiert wird. In einer Schleife werden die Tabellenüberschriften ausgelesen.

Die Funktion **mysql_field_name** liefert einen Datensatz aus dem SELECT-Befehl. Die Datensätze werden ihrerseits ebenfalls in einer Schleife ausgelesen.

Die Anwendung dieser Funktion sehen wir uns am Beispiel der Datei **ausgabe4.php** an. Hier werden die Daten der Tabelle **Veranstalter** angezeigt. Es wird die **SELECT**-Anweisung in der Variablen **$sql** gespeichert. Liefert die Abfrage Ergebnisse, werden diese mit dem **echo**-Befehl angezeigt, und die Resultate werden unserer Funktion **tab_out()** **übergeben**.

```
<?  // ausgabe4.php
include("verbind.php");
include("funk.php");
$sql = "select * from veranstalter ";
If ($res=send_sql($db,$sql))  {
   echo "Abfrage: <br> $sql"; }
tab_out($res);
?>
```

Abfrage:
select * from veranstalter

Veranstalter_NR	Veranstaltungsort	Adresse	Veranstalter
1	Berlin	12345 Berlin Musterstr. 5	AOK
2	Berlin	12345 Berlin Badstr. 6	Verwaltungsakademie
3	Leipzig	01234 Leipzig Eisenbahnstr. 6	WBK
4	Musterdorf	99999 Musterstr. 7	Musterveranstalter

SELECT

Bevor neue Aufgabenstellungen gelöst werden, stelle ich zunächst die Syntax des Befehls vor:

```
SELECT [DISTINCT | DISTINCTROW | ALL]
[Tabelle].Feld,...Ausdruck,...
[INTO OUTFILE 'Dateiname' Exportoptionen]
[FROM Tabelle, ...]
[WHERE Vergleichsausdruck]
[GROUP BY [Tabelle].Feld,...]
[HAVING Vergleichsausdruck]
[ORDER BY  Feld, ... [ASC | DESC] ,...]
[PROCEDURE procedure_name] ];
```

Werden mehrere Schlüsselworte genutzt, müssen diese in der genauen Reihenfolge aus der Syntax angegeben werden.

Die einzelnen Parameter des SELECT-Befehls werden im Anschluss mit Hilfe von Beispielen demonstriert.

Einfacher SELECT-Befehl

Um die einzelnen Tabellen mit ihrem Inhalt anzuzeigen, können folgende Befehle genutzt werden. Modifiziere zum Test der Befehle am besten das Programmskript **ausgabe4.php** und benenne es **ausgabe1.php** bis **ausgabe4.php**

```
SELECT * FROM personal;
SELECT * FROM teilnehmer;
SELECT * FROM schulung;
SELECT * FROM veranstalter;
```

Es handelt sich hier um den einfachsten **SELECT**-Befehl. Das Sternchen * steht für die Anzeige aller Felder. Möchtest du nur bestimmte Feldinhalte anzeigen, ist statt dem Sternchen * eine Liste der Feldnamen anzugeben. Die Feldnamen werden durch Kommata getrennt.

Die Anschriften der einzelnen Mitarbeiter lassen sich unter Angabe der einzelnen Feldnamen wie folgt anzeigen **(ausgabe5.php)**:

Abfrage:
SELECT name, vorname, strasse, plz, ort FROM personal

name	vorname	strasse	plz	ort
Taro	Eike	Musterstr. 6	09999	Musterdorf
Müller	Kai	Hallesche Str. 16	04838	Eilenburg
Heik	Hennes	Musterstr. 145	09999	Musterdorf
Mieder	Torsten	Hauptstrasse 16	04838	Laussig
Müller	Ulrich	Weide Str. 2 a	04838	Hohenpressnitz
Schwarz	Christine	Musterstr. 3	09999	Musterdorf
Wicki	Erik	Bolten-Weg 3	22587	Hamburg

Manchmal sind Datensätze oder Teile davon doppelt vorhanden. In solch einem Fall hilft der Parameter DISTINCT weiter.

Die verschiedenen Veranstaltungsorte aus der Tabelle **veranstalter** sollen ausgelistet werden. **(ausgabe5a.php)**:

Abfrage:
SELECT DISTINCT veranstaltungsort FROM veranstalter

veranstaltungsort
Berlin
Chemnitz
Essen
Leipzig
Mannheim
München
Rostock

WHERE-Klausel im SELECT-Befehl

Im nächsten Beispiel wird nur der Datensatz des Veranstalters mit der Veranstalternummer 1 benötigt. Der **SELECT**-Befehl muss in diesem Fall mit der **where**-Klausel mit einem Vergleichsausdruck ergänzt werden. Darunter versteht man ein Kriterium mit Bedingungen, die die Datensätze erfüllen müssen, damit sie weiterverarbeitet werden können.

Ein einfacher Vergleichsausdruck hat folgenden Aufbau:

Operand1	Vergleichs-operator	Operand2
veranstal-ter_nr	=	1

Als **Operanden** sind *Spaltennamen*, *Konstanten*, *Ausdrücke* und *Systemvariable* erlaubt.

Die Vergleichsoperatoren sind mit denen von PHP identisch (vgl. Seite 13)

Da für die Ausgabe die Einschränkung gemacht wurde, dass der Datensatz des Veranstalters mit der Veranstalternummer 1 angezeigt werden soll, wird wieder das Jokerzeichen * eingesetzt **(ausgabe6.php)** .

Abfrage:
SELECT * FROM veranstalter where veranstalter_nr=1

Veranstalter_NR	Veranstaltungsort	Adresse	Veranstalter
1	Mannheim	..	S & B Mannheim

Wie wir eben sahen, dürfen Operanden auch Ausdrücke sein. Das heißt, das du mathematische Ausdrücke mit den nachfolgend angezeigten Operatoren einsetzen darfst. Nutzt du mehrere Operatoren, gilt, dass die Punktrechnung (* /) vor der Strichrechnung (+-)ausgeführt wird. Ist dies nicht erwünscht, sind Klammern einzusetzen.

Operatoren	Bedeutung
* /	Multiplikation und Division
+ -	Addition und Subtraktion

In der nächsten Abfrage werden alle Kurse angezeigt, die nach Erhöhung der Kosten 20% mehr als 2000 DM kosten würden **(ausgabe7.php)** .

Abfrage:
SELECT * FROM schulung where 1.20*kurskosten>2000

laufende_NR	Schulungsbezeichnung	Kursbeginn	Kursende	Kurskosten	Veranstalter_NR
2	Laborschulung 2	1996-04-14	1996-04-16	2000.00	4
3	Teamtraining	1996-04-07	1996-04-08	2100.00	5
6	Grundkurs Word für Windows	1996-12-14	1996-12-15	1800.00	4
9	Telefonmarketing-Training	1996-04-26	1996-04-27	2034.00	5
11	Arbeitszeugnisse richtig	1996-05-18	1996-05-19	2440.00	7

Mathematische Ausdrücke sind auch in der Ausdrucksliste (gleich hinter SELECT) erlaubt. So lässt sich die jeweilige berechnete Spalte anzeigen. Der Spaltenname kann durch den Aliasnamen **`Kurskosten2`** festgelegt werden **(ausgabe7a.php)** .

Abfrage:
SELECT laufende_nr, Schulungsbezeichnung, Kursbeginn, Kursende, Kurskosten, 1.20*kurskosten AS Kurskosten2, Veranstalter_nr FROM schulung where 1.20*kurskosten>2000

laufende_nr	Schulungsbezeichnung	Kursbeginn	Kursende	Kurskosten	Kurskosten2	Veranstalter_nr
2	Laborschulung 2	1996-04-14	1996-04-16	2000.00	2400.00	4
3	Teamtraining	1996-04-07	1996-04-08	2100.00	2520.00	5
6	Grundkurs Word für Windows	1996-12-14	1996-12-15	1800.00	2160.00	4
9	Telefonmarketing-Training	1996-04-26	1996-04-27	2034.00	2440.80	5
11	Arbeitszeugnisse richtig	1996-05-18	1996-05-19	2440.00	2928.00	7

Die mathematischen Ausdrücke können relativ kompliziert werden. Ehe du beim Lösen deiner Probleme verzweifelst, solltest du dir die folgende Liste von nützlichen MySQL-Funktionen ansehen.

Funktion	Beschreibung
`abs(X)`	Gibt den absoluten Wert von x zurück.
`ascii(str)`	Zeigt den ASCII-Code-Wert des ersten (linken) Zeichens eines Zeichenausdrucks an.
`avg(ausdruck)`	Liefert den Mittelwert. AVG kann nur bei numerischen Spalten verwendet werden. NULL-Werte werden ignoriert.
`ceiling(X)`	Liefert den kleinsten Integerwert, der nicht kleiner als X ist.
`char(N,...)`	Wandelt einen ASCII-Code-Wert in ein Zeichen um. Der ASCII-Code-Wert muss zwischen 0 und 255 liegen, andernfalls wird NULL zurückgegeben.
`coalesce(liste)`	Gibt das 1. nicht-NULL Element der Liste wieder.
`concat(str1, str2,...)`	Gibt den String zurück, der durch Zusammenführen der Argumente entstanden ist.
`connection_id( )`	Gibt die Verbindungs-ID für die Verbindung zurück. Jede Verbinung hat eine eigene ID.
`count(ausdruck)`	Zählt die Reihen, deren Werte ungleich **`NULL`** sind.
`curdate( )` `current_date`	Gibt das heutige Datum als ein Wert im **`YYYY-MM-DD`**- oder **`YYYYMMDD`**-Format wieder (es kommt darauf an, ob die Funktion als String oder numerischer Wert genutzt wird).
`curtime( )` `current_time`	Gibt die aktuelle Zeit als ein Wert im **`HH:MM:SS`**- oder **`HHMMSS`**-Format wieder (es kommt darauf an, ob die Funktion als String oder numerischer Wert genutzt wird).
`database( )`	Gibt den aktuellen Datenbanknamen zurück.

`date_add(date, ausdruck type)`
`date_sub(date, ausdruck type)`
`adddate(date, ausdruck type)`
`subdate(date, ausdruck type)`
	Diese Funktion dienen der Datenarithmetik. **`Adddate()`** und **`Subdate()`** sind die entsprechende Synonyme für **`Date_Add()`** und **`Date_Sub()`**.
	
	`date` ist ein Datumswert.
	`ausdruck` ist die Anzahl, die addiert oder subtrahiert wird.
	`type` beschreibt, wie der Ausdruck interpretiert wird.

`date_ format (date, format)`	Formatiert den Datumswert. Die möglichen Formate werden in der untenstehenden Tabelle aufgeführt.
`dayname(date)`	Gibt den Namen des Wochentages für das Datum zurück.
`dayofmonth(date)`	Gibt den Tag des Monats für das Datums zurück (1-31).
`dayofweek(date)`	Gibt den Wochentag-Index eines Datums zurück (1 = Sonntag, 2 = Montag, . . . 7 = Samstag).
`dayofyear(date)`	Gibt den Tag des Jahres für das Datum zurück.
`floor(X)`	Gibt den größten Integerwert wieder, der nicht größer als X ist.

format(X,D)	Formatiert X zu einen Format wie **#,###,###.##**. **D** gibt die zu rundenden Dezimalstellen an.
greatest(X,Y,...)	Gibt den größten Wert in der Reihe zurück.
hour(time)	Gibt die Stunde (0 bis 23) für time wieder.
if(ausdruck1,ausdruck2,ausdruck3)	Wenn **ausdruck1** wahr ist, wird **ausdruck2** ausgeführt, ansonsten **ausdruck3**.
ifnull(ausdruck1, ausdruck2)	Wenn **ausdruck1** nicht null ist, wird **ausdruck1** zurückgegeben, ansonsten **ausdruck2**.
lcase(str) **lower(str)**	Gibt den String in Kleinbuchstaben aus.
least(X,Y,...)	Gibt den kleinsten Wert aus.
left(str,Länge)	Gibt von links die Anzahl der Zeichen aus.
length(str)	Diese Funktion gibt die Länge des Strings wieder.
load_file(file_name)	Liest das File aus und gibt den Dateiinhalt als ein String wieder. Die Datei muss auf dem Server liegen, und der Benutzer muss den vollen Pfadnamen angeben.
locate(substr,str)	Gibt die Position des gesuchten Strings aus. Wenn der String nicht enthalten ist, wird 0 ausgegeben.
locate(substr,str, pos)	Gibt die Position des gesuchten Strings aus, wobei ab Position gestartet wird. Wenn der String nicht enthalten ist, wird 0 ausgegeben.
log(X)	Gibt den natürlichen Logarithmus von X zurück.
log10(X)	Gibt den Logarithmus zur Basis 10 von X an.
ltrim(str)	Gibt den String wieder, wobei vorangestellten Leerzeichen gelöscht werden
min(ausdruck) **max(ausdruck)**	Liefert den kleinsten (bzw. größten) Wert im Ausdruck.
minute(time)	Gibt die Minute für **time** (0-59) wieder.
mod(N,M)	Gibt den ganzzahligen Rest bei der Division von N durch M an.
month(date)	Gibt den Monat für **date** (1-12) zurück.
monthname(date)	Gibt den Name des Monats für **date** zurück.
now( ) **sysdate( )** **current_timestamp**	Gibt aktuelles Datum und Zeit als Wert in **YYYY-MM-DD HH:MM:SS**- oder **YYYYMMDDHHMMSS**-Format (es kommt darauf an, ob die Funktion als String oder numerischer Wert genutzt wird).
nullif(ausdruck1, ausdruck2)	Wenn **ausdruck1 = ausdruck2**, wird NULL wiedergegeben, ansonsten wird **ausdruck1** wiedergegeben.
password(str)	Verschlüsselt den String.
period_add(P,N)	Addiert **N** Monate zu Periode **P** (im Format **YYMM** oder **YYYYMM**). Gibt den Wert im Format **YYYYMM** zurück.
period_diff(P1,P2)	Gibt die Nummer von Monaten zwischen den Perioden **P1** und **P2**, die im Format **YYMM** oder **YYYYMM** angegeben werden sollten.

`pi( )`	Gibt den Wert von π (pi) wieder.
`pow(X,Y)` `power(X,Y)`	Gibt den Wert von x hoch y wieder.
`quarter(date)`	Gibt das Quartal des Jahres für das Datum wieder (1-4).
`rand( )`	Liefert den Zufallswert zwischen 0 und 1.
`reverse(str)`	Gibt den String von hinten nach vorn wieder.
`round(X)`	Gibt das Argument X, gerundet als Integer, wieder.
`round(X,D)`	Gibt das Argument X, gerundet als Integer, wieder. D gibt die Anzahl der Dezimalstellen an.
`rtrim(str)`	Entfernt nachfolgende Leerzeichen.
`second(time)`	Gibt die Sekunden für die Zeit wieder (0-59).
`sign(X)`	Gibt das Vorzeichen als Argument (-1, 0 oder 1) an. Es kommt darauf an ob X negativ, 0 oder positiv ist.
`sin(X)`	Gibt den Sinus von X wieder.
`space(N)`	Liefert eine Zeichenfolge mit wiederholten Leerstellen. Die Anzahl der Leerstellen entspricht dem Ganzzahlausdruck. Ist der Ganzzahlausdruck negativ, wird NULL zurückgegeben.
`sqrt(X)`	Gibt die Wurzel von X zurück.
`substring(str,pos, len)`	Liefert einen Teil einer Zeichenfolge. Der erste Parameter kann eine Zeichenfolge, Binärkette, ein Feldname sein oder ein Ausdruck, der einen Feldnamen enthält. Der zweite Parameter legt fest, wodie Teilzeichenfolge beginnt. Der dritte Parameter gibt die Anzahl der Zeichen der Teilzeichenfolge an.
`substring(str,pos)`	Gibt einen Substring von String **`str`** wieder – Start bei der Position **`pos`**.
`sum(ausdruck)`	Liefert die Summe.
`time_ format (time, format)`	Wird wie **`Date_Format()`** genutzt. Format enthält Werte, wie Stunden, Minuten und Sekunden.
`time_to_sec(time)`	Gibt das **`time`**-Argument in Sekunden konvertiert wieder.
`to_days(date)`	Wandelt das Datum in die Anzahl der Tage von Jahr 0 um.
`ucase(str)` `upper(str)`	Wandelt den String in Großbuchstaben um.

`user( )`
`system_user( )`
`session_user( )`
	Diese Funktion gibt den aktuellen MySQL Benutzernamen wieder.

`version( )`	Gibt die MySQL-Serverversion wieder.
`week(date)` `week(date, first)`	Gibt die Kalenderwoche an (0-53). Das 2. Argument erlaubt dem Benutzer zu spezifizieren, wo die Woche startet (Sonntag (0) oder Montag (1)).
`weekday(date)`	Gibt den Wochentag-Index für das Datum wieder (0 = Montag, 1 = Dienstag, . . . 6 = Sonntag).
`year(date)`	Gibt das Jahr für Datum (1000-9999) zurück.
`yearweek(date)`	Gibt das Jahr und Woche für das Datum wieder.

Es folgt die Zusammenstellung möglicher Formate für die Funktion **`date_format`**

%W	Wochentag	%y	Jahr (2stellig)
%w	Tag in der Woche (0 = Sonntag, . . . , 6=Samstag)	%T	Uhrzeit (24 Std.) (hh:mm:ss)
%d	Tag des Monats (00-31)	%S	Sekunden (00-59)
%e	Tag des Monats (0-31)	%s	Sekunden (00-59)
%j	Tag im Jahr (001-366)	%i	Minuten (00-59)
%U	Woche, mit Sonntag als 1. Tag der Woche (00-52)	%H	Stunde (00-23)
%u	Woche, mit Montag als 1. Tag der Woche (00-52)	%k	Stunde (0-23)
%M	Monatsname	%h	Stunde (00-12)
%m	Monat, numerisch (01-12)	%I	Stunde (00-12)
%c	Monat, numerisch (1-12)	%l	Stunde (0-12)
%Y	Jahr (4stellig)		

In den Beispielen **ausgabe8.php `bis` ausgabe11.php** kommen einige der aufgeführten MySQL-Funktionen zum Einsatz.

Besonders nützlich sind die zahlreichen Datumsfunktionen.

Sieh dir das Ergebnis des folgenden **`SELECT`**-Befehls an (**ausgabe8.php**).

Abfrage:
SELECT vorname, name, geburt FROM personal

vorname	name	geburt
Eike	Taro	1947-09-10
Kai	Müller	1951-12-28
Hannes	Heik	1961-02-06
Torsten	Mieder	1971-05-15
Ulrich	Müller	1951-01-02
Christine	Schwarz	1948-04-24
Eenk	Wicki	1950-07-09
Elfi	Kaufmann	1966-01-29
Hans-Jürgen	Sieg	1947-05-22
Peter	Schoek	1944-11-02
Christa	Renner	1932-12-27
Jörg	Schön	1969-08-20
Bernd		1959-07-10

Das Datum wird im internen MySQL-Format JJJJ-MM-TT ausgegeben. In das deutsche Format **`TT.MM.JJJJ`** wird es mit folgender Funktion transformiert:

`DATE_FORMAT` (Datumswert, Formatierungsstring) . (**ausgabe9.php**)

Abfrage:
SELECT vorname, name, DATE_FORMAT(geburt, '%d.%m.%Y') geburt FROM personal

vorname	name	geburt
Eike	Taro	10.09.1947
Kai	Müller	28.12.1951
Hannes	Heik	06.02.1961
Torsten	Mieder	15.05.1971
Ulrich	Müller	02.01.1951
Christine	Schwarz	24.04.1948
Eenk	Wicki	09.07.1950
Elfi	Kaufmann	29.01.1966
Hans-Jürgen	Sieg	22.05.1947
Peter	Schock	02.11.1944
Christa	Renner	27.12.1932
Jörg	Schön	20.08.1969
Bernd	Jach	10.05.1959
Birgit	Gemse	13.01.1958
Silvia	Munter	28.04.1952
[illegible]	Marx	22.03.1956

Weiterhin sollen Vorname und Name in einer Spalte **Namen** zusammengefasst werden. Sollte ein Datensatz kein Vornamen besitzen, wird statt dessen **Firma** ausgegeben (**ausgabe10.php**).

Die **IF(Ausdruck1,Ausdruck2,Ausdruck3-**Funktion funktioniert so: ist **Ausdruck1** wahr, wird **Ausdruck2** eingesetzt, andernfalls **Ausdruck3**.

Wir benutzen die Funktion, um zu testen, ob die Vornamen- und Namen-Felder ausgefüllt sind.

Die Funktion **CONCAT(Ausdruck1[,...]** verkettet alle Strings, die als Argumente übergeben werden. Die **LENGTH(string**-Funktion liefert die Länge eines Strings.

Abfrage:
SELECT IF(LENGTH(vorname), CONCAT(vorname, ' ', name), CONCAT('Firma', name)) AS name, DATE_FORMAT (geburt, '%d.%m.%Y') AS datum FROM personal

name	datum
Eike Taro	10.09.1947
Kai Müller	28.12.1951
Hannes Heik	06.02.1961
Torsten Mieder	15.05.1971
Ulrich Müller	02.01.1951
Christine Schwarz	24.04.1948
Eenk Wicki	09.07.1950
Elfi Kaufmann	29.01.1966

Als nächstes werden die SQL-Version, der aktive Benutzer, das heutige Datum in Normal- und Langform, der Tag im Monat, der Tag in der Woche und das aktuelle Jahr angezeigt (**ausgabe11.php**).

Abfrage:
SELECT version(),user( ) , curdate(), sysdate( ), week(curdate()) ,weekday(curdate()), year(curdate())

version()	user( )	curdate()	sysdate( )	week(curdate())	weekday(curdate())	year(curdate())
3.22.32	www.bilke.de@cheery.store	2001-07-04	2001-07-04 16:58:04	27	2	2001

Mehrere Bedingungen in der WHERE-Klausel

Sollen mehrere Bedingungen erfüllt werden, benötigt man **logische Operatoren,** die in SQL wie folgt verwendet werden:

logischer Operator	**Bedeutung**
NOT	kehrt das Resultat eines logischen Ausdruckes um.
AND	kombiniert zwei oder mehr logische Ausdrücke. Das Resultat erhält den logischen Wert **WAHR**, wenn **alle Bedingungen WAHR** sind, andernfalls ist der logische Wert des Resultats **FALSCH**.
OR	kombiniert zwei oder mehr logische Ausdrücke. Das Resultat erhält den logischen Wert WAHR, wenn **eine Bedingung** WAHR ist, andernfalls wird der logische Wert des Resultats FALSCH.

Aus der Tabelle **personal** sollen alle Frauen aus Musterdorf angezeigt werden, die älter als 50 Jahre sind
(**ausgabe13.php**)

Abfrage:
SELECT * FROM personal WHERE Ort='Musterdorf' and geschlecht ='W' and year(curdate())-year(geburt)>50

Personalnummer	Vorname	Name	Strasse	PLZ	Ort	Geburt	Geschlecht	Gehalt
1430	Christine	Schwarz	Musterstr. 8	09999	Musterdorf	1948-04-24	W	5420.00

Statt bei einer Suchbedingung eine Reihe von Werten eines Feldes mit OR zu vergleichen, nutzt du besser den Operator IN. Der Operator bestimmt, ob der Wert eines Feldes einem von mehreren Werten in der angegebenen Liste entspricht.

Möchtest du alle Personen aus Hamburg, Leipzig oder Hamburg anzeigen, kann man den logischen OR Operator verwenden:

```
SELECT  * FROM personal
WHERE ort ='Hamburg'
or ort='Leipzig'
or ort='Musterdorf');
```

oder den **IN**-Operator nutzen(**ausgabe14.php**)

Abfrage:
SELECT * FROM personal WHERE ort IN('Hamburg' ,'Leipzig','Musterdorf')

Personalnummer	Vorname	Name	Strasse	PLZ	Ort	Geburt	Geschlecht	Gehalt
1008	Eike	Taro	Musterstr. 6	09999	Musterdorf	1947-09-10	M	5100.00
1112	Hannes	Heik	Musterstr. 145	09999	Musterdorf	1961-02-06	M	5600.00
1430	Christine	Schwarz	Musterstr. 8	09999	Musterdorf	1948-04-24	W	5420.00
1015	Eerik	Wicki	Bollen-Weg 3	22587	Hamburg	1950-07-09	M	4533.00
1434	Jörg	Schön	Dorfweg 25 a	22589	Hamburg	1969-08-20	M	5600.00
1432	Bernd	Jach	Suelldorferstr. 124	22589	Hamburg	1959-05-10	M	2455.05
1431	Birgit	Gemse	Bauernkoppel 39	22393	Hamburg	1958-01-13	W	1200.00
1433	Silvia	Munter	Treppe 7	22587	Hamburg	1952-04-28	W	4555.00
1010	Jürgen	Marx	Musterstr. 12	09999	Musterdorf	1956-03-22	M	3677.00
1113	Karl	Seppa	Musterstr. 7	09999	Musterdorf	1963-06-24	M	3400.00
1005	Ede	Pfau	Nicoleiplatz 2	04232	Leipzig	1952-08-02	M	4500.00
1438	Manfred	Stach	Gerbergasse 7	04105	Leipzig	1946-09-26	M	3500.00
1114	Karsten	Müller	Musterstr. 3	09999	Musterdorf	1963-07-12	M	4900.00

Platzhalterzeichen in der WHERE-Klausel

Irgendwann einmal hast du sicher keine genaue Vorstellung von einem genauen Suchbegriff. Du möchtest z.B. nach einem Feldinhalt suchen, der mit einem angegebenen Muster übereinstimmt. Dazu benutzt du den Operator **`Like`**. Für das Muster kann man einen vollständigen Wert angeben oder einen Wert mit Platzhalterzeichen.

Platzhalterzeichen	Bedeutung
_	ein beliebiges Zeichen
%	kein oder mehr Zeichen

Es sollen alle Orte angezeigt werden, die mit H beginnen (**ausgabe15.php**)

Abfrage:
SELECT * FROM personal WHERE ort LIKE 'H%'

Personalnummer	Vorname	Name	Strasse	PLZ	Ort	Geburt	Geschlecht	Gehalt
1016	Ulrich	Müller	Weide Str. 2 a	04838	Hohenpriessnitz	1951-01-02	M	8999.00
1015	Erik	Wicki	Bolten-Weg 3	22587	Hamburg	1950-07-09	M	4533.00
1437	Christa	Renner	Weidenstrasse 26	25469	Halstenbek	1932-12-27	W	3900.00
1434	Jörg	Schön	Dorfweg 25 a	22589	Hamburg	1969-08-20	M	5600.00
1432	Bernd	Jach	Suelldorferstr 124	22589	Hamburg	1959-05-10	M	2455.00
1431	Birgit	Gemse	Bauernkoppel 39	22393	Hamburg	1958-01-13	W	1200.00
1433	Silvia	Munter	Treppe 7	22587	Hamburg	1952-04-28	W	4555.00
1101	Günter	Maus	Bergstrasse 5	06108	Halle/Saale	1948-12-21	M	4500.00

Platzhalterzeichen werden auch benötigt, wenn du erfahren möchtest, wer in einer Gasse wohnt (**ausgabe15.php**)

Abfrage:
SELECT * FROM personal WHERE strasse LIKE '%gasse%'

Personalnummer	Vorname	Name	Strasse	PLZ	Ort	Geburt	Geschlecht	Gehalt
1438	Manfred	Stach	Gerbergasse 7	04105	Leipzig	1946-09-26	M	3500.00

Aggregatsfunktionen im Zusammenhang mit der GROUP BY-und HAVING-Klausel

Möchtest du statistische Auswertungen vornehmen, nutzt du die Zusammenfassungsfunktionen.

Funktion	Bedeutung
AVG	Berechnet den arithmetischen Mittelwert einer Menge von Werten in einem bestimmten Feld.
MAX	Gibt den größten Wert aus einer Menge von Werten in einem bestimmten Feld aus.
MIN	Gibt den kleinsten Wert aus einer Menge von Werten in einem bestimmten Feld aus.
SUM	Gibt die Summe einer Menge von Werten in einem bestimmten Feld aus.
COUNT	Gibt die Anzahl einer Menge von Datensätze aus.

Üben wir nun diese Funktionen an einem praktischen Beispiel. Uns interessiert etwa, wie viele Personen in der Tabelle PERSONAL enthalten sind. Weiterhin sollen das mittlere Gehalt, die Gesamtsumme des Gehaltes und das größte und kleinste Gehalt ermittelt werden (**ausgabe12.php**).

Abfrage:
SELECT COUNT(*), AVG(gehalt), SUM(gehalt), MAX(gehalt), MIN(gehalt) FROM personal

COUNT(*)	AVG(gehalt)	SUM(gehalt)	MAX(gehalt)	MIN(gehalt)
23	4556.260870	104794.00	8999.00	1200.00

Möchtest du diese Angaben für die einzelnen Orte sehen, ist die Anweisung durch die **GROUP BY**-Klausel zu ergänzen. In der Ausdrucksliste ist der Feldname **ort** zu ergänzen (**ausgabe12a.php**).

Abfrage:
SELECT ort, COUNT(*), AVG(gehalt), SUM(gehalt), MAX(gehalt), MIN(gehalt) FROM personal GROUP BY ort

ort	COUNT(*)	AVG(gehalt)	SUM(gehalt)	MAX(gehalt)	MIN(gehalt)
Abstatt	1	4700.000000	4700.00	4700.00	4700.00
Eilenburg	1	3400.000000	3400.00	3400.00	3400.00
Genichshain	1	3900.000000	3900.00	3900.00	3900.00
Halle/Saale	1	4500.000000	4500.00	4500.00	4500.00
Halstenbek	1	3900.000000	3900.00	3900.00	3900.00
Hamburg	5	3668.600000	18343.00	5600.00	1200.00
Hohenpriessnitz	1	8999.000000	8999.00	8999.00	8999.00
Laussig	1	5400.000000	5400.00	5400.00	5400.00
Leipzig	2	4000.000000	8000.00	4500.00	3500.00
Moertitz	1	6000.000000	6000.00	6000.00	6000.00
Musterdorf	6	4682.833333	28097.00	5600.00	3400.00
Winsen	1	7000.000000	7000.00	7000.00	7000.00
Wurzen	1	2555.000000	2555.00	2555.00	2555.00

Werden Aggregatsfunktionen eingesetzt, dürfen in der Feldliste nur Felder aus der **GROUP BY**-Klausel enthalten sein.

Sollen diese Angaben nur für den Ort Leipzig angezeigt werden, wird der Befehl aus dem letzten Beispiel um **HAVING** ergänzt (**ausgabe12b.php**)

Abfrage:
SELECT ort, COUNT(*), AVG(gehalt), SUM(gehalt), MAX(gehalt), MIN(gehalt) FROM personal GROUP BY ort HAVING ort='Leipzig'

ort	COUNT(*)	AVG(gehalt)	SUM(gehalt)	MAX(gehalt)	MIN(gehalt)
Leipzig	2	4000.000000	8000.00	4500.00	3500.00

Sortieren der Ausgabeergebnisse

Bei unseren Beispielen wurden die Datensätze in der eingegebenen Reihenfolge angezeigt. Ist eine andere Reihenfolge erwünscht, erreichst du das über die **ORDER BY**-Klausel des **SELECT**-Befehls mit Angabe von bestimmten Attributen. **ASC** sortiert aufsteigend. Da das Standard ist, machst du keine Eingabe. DESC sortiert absteigend.

ORDER BY ist in der Regel der letzte Eintrag in einer SELECT-Anweisung.

Die Adressen der Tabelle **personal** sollen alphabetisch nach Namen sortiert werden. Gibt es doppelte Namen, soll nach Vornamen sortiert werden. (**ausgabe12c.php**).

Abfrage:
SELECT name, vorname, strasse,plz, ort FROM personal order by name, vorname

name	vorname	strasse	plz	ort
Ecke	Klaus	Schulze-Str. 11	4838	Wurzen
Gans	Kerstin	Hauptstr.10	04668	Moertitz
Genze	Birgit	Bauernkoppel 39	22393	Hamburg
Heik	Hannes	Musterstr. 145	09999	Musterdorf
Jach	Bernd	Boelldorferstr. 124	22389	Hamburg
Kaufmann	Rik	Brandstrasse 15 a	04537	Genichshain
Matz	Jürgen	Musterstr. 12	09999	Musterdorf
Maus	Günter	Bergstrasse 5	06108	Halle/Saale
Mueller	Torsten	Hauptstrasse 16	04838	Laussig
Munter	Silvia	Treppe 7	22587	Hamburg
Müller	Kai	Hallesche Str. 16	04838	Eilenburg
Müller	Karsten	Musterstr. 3	09999	Musterdorf
Müller	Ulrich	Weide Str. 2 a	04838	Hohenpriessnitz
Pfau	Ede	Nicolaiplatz 2	04232	Leipzig
Renner	Christa	Weidenstrasse 28	25469	Halstenbek
Scheck	Peter	Weststrasse 2	74232	Abstatt

Export in eine Textdatei

Mit dem **SELECT**-Parameter **INTO OUTFILE** werden die Ergebnisse eines **SELECT**-Befehls in eine Textdatei exportiert. Die Syntax sieht so aus:

```
SELECT INTO OUTFILE 'Dateiname'
[FIELDS [TERMINATED BY term]
[ENCLOSED by encl] [ESCAPED BY
esc][(Feld,...)]
```

Die **FIELDS**-Auswahl ist identisch mit der, die beim **LOAD**-Befehl beschrieben wurde. Willst du z.B. die Adressen aller Musterdorfer in eine Textdatei auslesen, nutzt du diesen **SELECT**-Befehl:

```
SELECT CONCAT(vorname, ' ', name),
strasse, plz, ort
INTO OUTFILE 'aus.txt'
FROM personal
WHERE ort = 'Musterdorf';
```

Daten aus mehreren Tabellen

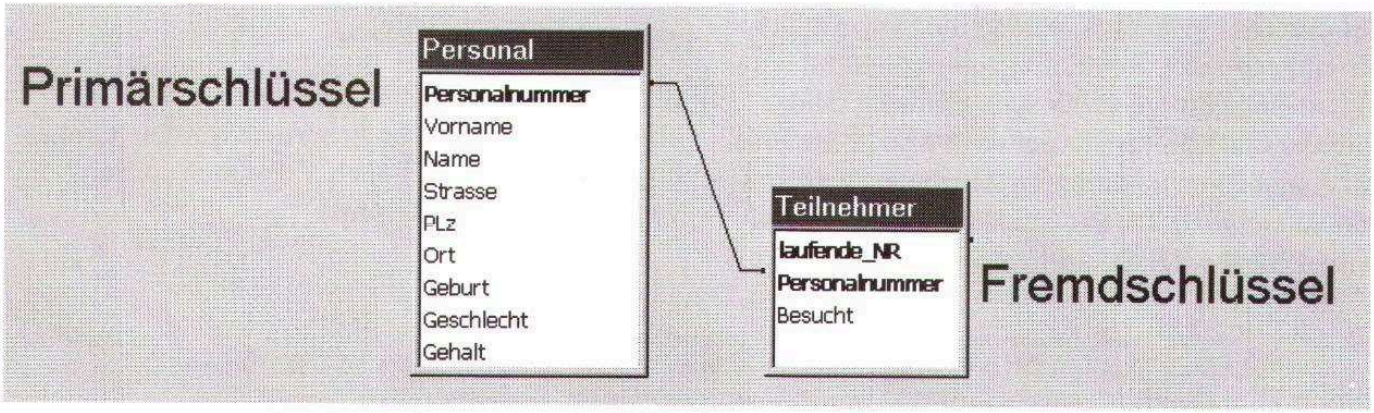
Mit der Verknüpfung von Daten aus mehreren Tabellen kannst du zwei oder mehrere Tabellen abfragen, worauf das Resultat zeilenweise ausgegeben wird. Dazu musst du die gewünschten Tabellen in der **SELECT**-Abfrage aufführen. **SELECT vorname, name, laufende_NR FROM personal, teilnehmer** bringt nun nicht das gewünschte Ergebnis. Jede Zeile der ersten Tabelle wird mit jeder Zeile der zweiten Tabelle kombiniert (**mausgabe1.php**).

Abfrage:
SELECT vorname, name, laufende_NR FROM personal , teilnehmer

vorname	name	laufende_NR
Eike	Taro	1
Kai	Müller	1
Hannes	Heik	1
Torsten	Mieder	1
Ulrich	Müller	1
Christine	Schwarz	1

Du erreichst das gewünschte Ergebnis, indem du zwischen beiden Tabellen eine Verknüpfung herstellst. Die Verknüpfung erfolgt über Spalten, die übereinstimmende Datenwerte enthalten. Die Verknüpfung erlaubt also die erneute Verbindung der durch die Normalisierung aufgeteilten Daten.

Die Verknüpfung wird ermöglicht durch eine Gleichsetzung von Primär- und Fremdschlüssel in der **WHERE**-Klausel. Man spricht in diesem Fall von einer Gleichverteilung. Darüber hinaus kann man Tabellen über einen **LEFT JOIN** abfragen.



Bei Verknüpfungen ist folgendes zu beachten:

- DieVerknüpfungen basieren auf dem Primärschlüssel der Tabelle1 und dem Fremdschlüssel in Tabelle2.
- Die zu verknüpfenden Spalten müssen von gleichem Datentyp sein.
- Die Feldnamen müssen nicht unbedingt identisch sein.
- Sind die Feldnamen aus beiden Tabellen identisch, müssen die Feldbezeichnungen um die Tabellennamen bzw. den Tabellenalias erweitert werden.
  {Tabellename | Tabellenalias}.Feld
- Halte die Anzahl der zu verknüpfenden Tabellen gering. Abfragen, in denen zahlreiche Tabellen verknüpft sind, werden langsamer ausgeführt als solche mit wenigen Tabellen.

Vollständige Verknüpfung zweier Tabellen

Der einfachste Verknüpfung ist der sogenannte Gleichverknüpfung. Eine allgemeine SELECT-Anweisung könnte so aussehen:

```
SELECT A.EineSpalte,
B.EineAndereSpalte
FROM Tabelle1 AS A, Tabelle2 AS B
WHERE A.EinWert = B.EinAndererWert;
```

Das Beispiel zeigt die Namen und Lehrgangsnummern aller Teilnehmer an (**mausgabe1.php**).

Abfrage:
SELECT vorname, name, laufende_NR FROM personal As p, teilnehmer AS t WHERE p.personalnummer=t.personalnummer

vorname	name	laufende_NR
Kai	Müller	1
Klaus	Ecke	1
Torsten	Mieder	1
Günter	Maus	1
Karsten	Müller-Schwarze	2
Elfi	Kaufmann	2
Jürgen	Marx	3
Benk	Wicki	3
Ulrich	Müller	3
	Ecke	

Aus Gründen der Kompatibilität kann auch das Schlüsselwort **JOIN** direkt in die **SELECT**-Anweisung eingeben werden.

```
SELECT vorname, name, laufende_NR
FROM personal As p JOIN teilnehmer
AS t
WHERE p.personalnummer=t.personal-
nummer
```

Man spricht hier von einer vollständigen Verknüpfung (Full-Join) zweier Tabellen.

Dieser Befehl kann durch die MySQL-eigene Erweiterung **STRAIGHT_JOIN** ergänzt werden. In diesem Fall wird hier zuerst die linke Tabelle gelesen und dann die rechte.

Abfrage:
SELECT vorname, name, laufende_NR FROM personal As p STRAIGHT_JOIN teilnehmer AS t WHERE p.personalnummer=t.personalnummer

vorname	name	laufende_NR
Kai	Müller	1
Klaus	Ecke	1
Torsten	Mieder	1
Günter	Maus	1
Karsten	Müller-Schwarze	2
Elfi	Kaufmann	2

LEFT JOIN

Bei dieser Verknüpfung wird die linke Tabelle komplett gelesen. Bei fehlenden Feldern in der linken Tabelle werden die entsprechenden Reihen mit NULL ergänzt. Anstatt **WHERE** wird **ON** verwendet.

```
SELECT A.EineSpalte,
B.EineAndereSpalte
FROM Tabelle1 AS A LEFT JOIN
Tabelle2 AS B
ON A.EinWert = B.EinAndererWert;
```

Schau dir den LEFT JOIN an einem konkreten Beispiel an. Du möchtest z.B. wissen, welche Personen aus der Tabelle **personal** noch nie bei einem Lehrgang waren. Bei der vollständigen Verknüpfung werden nur die Personen aus der Tabelle **personal** angezeigt, die bereits bei einem Lehrgang waren. Nun bist du aber an den anderen Personen interessiert. Hier hilft der **LEFT JOIN** weiter.

Zunächst lässt du alle Personen aus der Tabelle **personal** anzeigen und dann die passenden Personen aus der Tabelle **teilnehmer** (**mausgabe4.php**).

Abfrage:
SELECT vorname, name, laufende_NR FROM personal As p LEFT JOIN teilnehmer AS t ON p.personalnummer=t.personalnummer

vorname	name	laufende_NR
Eike	Tero	
Kai	Müller	1
Kai	Müller	5
Hannes	Heik	
Torsten	Mieder	1
Ulrich	Müller	3
Christine	Schwerz	7
Eerik	Wicki	3
Elfi	Kaufmann	2
Elfi	Kaufmann	5
Hens-Jürgen	Sieg	
Peter	Schock	5
Christa	Renner	
Jörg	Schön	
Bernd	Jach	4
Birgit	Gemse	7

Du siehst, dass bei der laufenden Teilnehmernummer teilweise keine Anzeige erfolgt. Diese Felder sind mit NULL belegt.

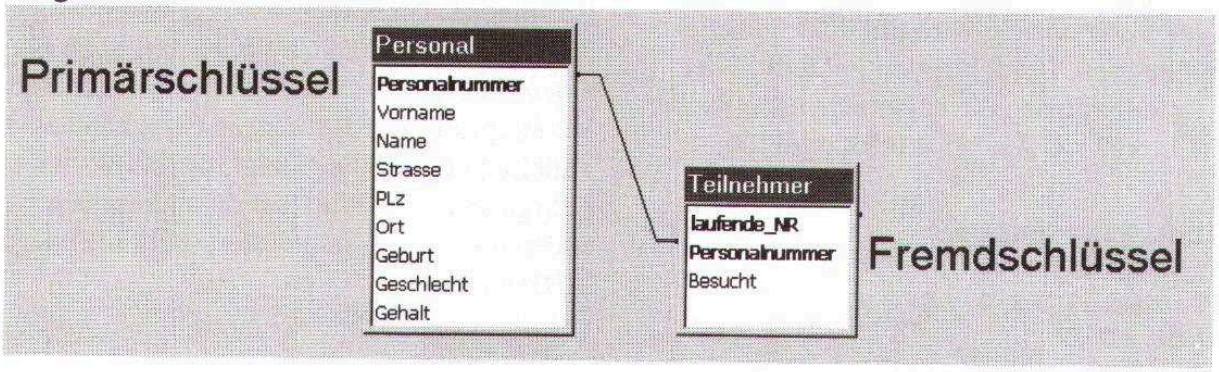
Mit diesem Wissen ergänzt du deinen **SELECT**-Befehl um eine WHERE-Klausel und weißt, wer nie bei einem Lehrgang war (**mausgabe5.php**).

Abfrage:
SELECT vorname, name, laufende_NR FROM personal As p LEFT JOIN teilnehmer AS t ON p.personalnummer=t.personalnummer WHERE laufende_NR IS NULL

vorname	name	laufende_NR
Eike	Taro	
Hannes	Heik	
Hens-Jürgen	Sieg	
Christa	Renner	
Jörg	Schön	
Karl	Seppa	

Nutzt du NULL in einem Vergleichsausdruck ist statt dem Gleichheitszeichen IS einzusetzen.

Da du bei MySQL nicht die Möglichkeit hast, die Beziehungen zwischen den Tabellen über Referenzen zu steuern, solltest du öfters kontrollieren, ob in der Tabelle mit dem Fremdschlüssel, keine Feldinhalte vorhanden sind, die in der Tabelle mit dem Primärschlüssel nicht enthalten sind.



Hier hilft die LEFT JOIN Verknüpfung weiter. Lasse dir aus der Tabelle **teilnehmer** alle Datensätze anzeigen und aus Tabelle **personal** nur die Datensätze, in denen die Inhalte der verknüpften Felder gleich sind (**mausgabe5a.php**).

Abfrage:
SELECT personal.Personalnummer, Vorname, Name, laufende_NR, teilnehmer.Personalnummer FROM teilnehmer LEFT JOIN personal ON personal.Personalnummer = teilnehmer.Personalnummer

Personalnummer	Vorname	Name	laufende_NR	Personalnummer
1002	Kai	Müller	1	1002
1017	Klaus	Ecke	1	1017
1429	Torsten	Mieder	1	1429
1101	Günter	Maus	1	1101
1114	Karsten	Müller	2	1114
1439	Elli	Kaufmann	2	1439
1010	Jürgen	Marx	3	1010
1015	Erik	Wicki	3	1015
1016	Ulrich	Müller	3	1016
1435	Peter	Schock	5	1435
1100	Kerstin	Gans	6	1100
1005	Ede	Pfau	6	1005
1430	Christine	Schwarz	7	1430
1431	Birgit	Gemse	7	1431
1438	Manfred	Stach	8	1438
			8	2222

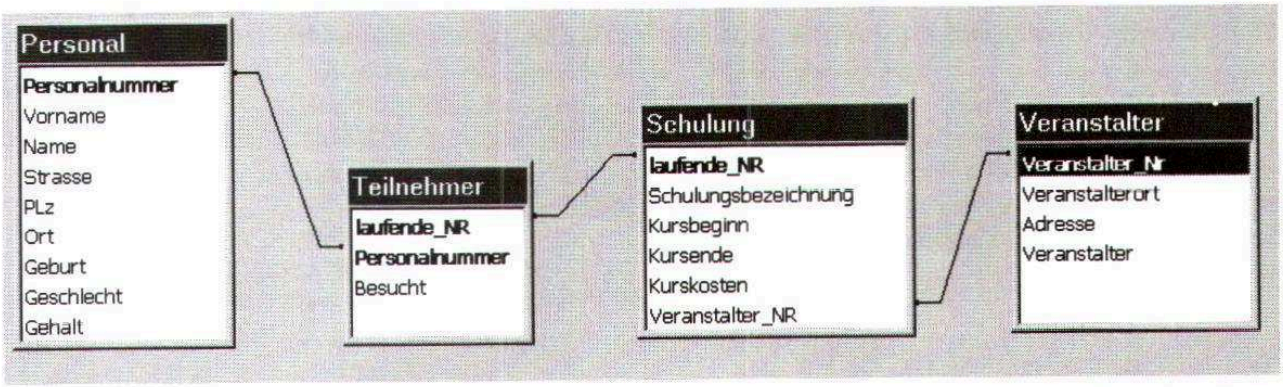
Du siehst, es gibt tatsächlich in der Tabelle **teilnehmer** die Personalnummer 2222, die keine Entsprechung in der Tabelle **personal** hat. Es reicht wenn du dir nur die betreffenden Datensätze anzeigen lässt.

```
SELECT personal.Personalnummer, Vorname, Name, laufende_NR,
teilnehmer.Personalnummer
FROM teilnehmer LEFT JOIN personal
ON personal.Personalnummer = teilnehmer.Personalnummer
WHERE personal.Personalnummer IS NULL
```

Die Überprüfungen müssen natürlich auch zwischen den Tabellen **schulung-teilnehmer** und **veranstalter-schulung** stattfinden.

Recherchen in mehreren Tabellen

Zu Erinnerung an unserer Datenstruktur noch einmal das Schema der vorhanden Tabellen.



Mit dem bisher gelernten kannst du Informationen aus allen vorhanden Tabellen zusammenstellen. Bei der Auswahl der Tabellen beachte:

> Wähle für die zu verknüpfenden Tabellen nur so viel Tabellen wie nötig, nicht wie möglich.

Zunächst interessiert, wer bei welchem Lehrgang war. Es werden nur Information aus den Tabellen **personal** und **teilnehmer** benötigt. Für diese Tabellen wurde keine direkte Beziehung erstellt. Deshalb musst du für diese Aufgabe die Tabellen **personal**, **teilnehmer** und **schulung** auswählen. (**mausgabe6.php**).

Abfrage:
SELECT vorname, name, t.laufende_NR FROM personal As p, teilnehmer AS t, schulung AS s WHERE p.personalnummer=t.personalnummer And t.laufende_NR=s.laufende_NR

vorname	name	laufende_NR
Kai	Müller	1
Klaus	Ecke	1
Torsten	Mieder	1
Günter	Maus	1
Karsten	Müller-Schwarze	2
Elfi	Kaufmann	2
Jürgen	Merx	3
Eenk	Wicki	3
Ulrich	Müller	3
Klaus	Ecke	3
Bernd	Jach	4
Silvia	Munter	4
Kai	Müller	5
Elfi	Kaufmann	5
Peter	Schock	5
Kerstin	Gans	6

Interessiert dich zusätzlich, wo diese Schulungen stattgefunden haben, musst du auch die vierte Tabelle in den **SELECT**-Befehl aufnehmen. (**mausgabe7.php**).

Abfrage:
SELECT vorname, name, t.laufende_NR, Veranstaltungsort FROM personal As p, teilnehmer AS t, schulung AS s, veranstalter v WHERE p.personalnummer=t.personalnummer AND t.laufende_NR=s.laufende_NR AND s.Veranstalter_NR=v.Veranstalter_NR

vorname	name	laufende_NR	Veranstaltungsort
Kai	Müller	1	Mannheim
Klaus	Ecke	1	Mannheim
Torsten	Mieder	1	Mannheim
Günter	Maus	1	Mannheim
Bernd	Jach	4	München
Silvia	Munter	4	München
Kai	Müller	5	Chemnitz
Elfi	Kaufmann	5	Chemnitz
Peter	Schock	5	Chemnitz
Karsten	Müller-Schwarze	2	Leipzig
Elfi	Kaufmann	2	Leipzig
Kerstin	Gans	6	Leipzig
Ede	Pfau	6	Leipzig
Christine	Schwarz	7	Leipzig
Birgit	Gemse	7	Leipzig
Jürgen	Merx	3	Berlin

Hier folgt noch einmal der vollständige **SELECT**-Befehl etwas übersichtlicher. Vergiss bei eindeutigen Feldnamen nicht den Tabellenname bzw. das Tabellenalias!

```
SELECT vorname, name, t.laufende_NR, Veranstaltungsort
FROM personal As p, teilnehmer AS t,
schulung AS s, veranstalter v
WHERE p.personalnummer=t.personalnummer
AND
t.laufende_NR=s.laufende_NR
AND
s.Veranstalter_NR=v.Veranstalter_NR
```

Möchtest du wissen, wer in Hamburg beim Lehrgang war, brauchst du wieder alle 4 Tabellen (**mausgabe8.php**).

Abfrage:
SELECT vorname, name, t.laufende_NR, Veranstaltungsort FROM personal As p, teilnehmer AS t, schulung AS s, veranstalter v WHERE p.personalnummer=t.personalnummer AND t.laufende_NR=s.laufende_NR AND s.Veranstalter_NR=v.Veranstalter_NR AND Veranstaltungsort='Berlin'

vorname	name	laufende_NR	Veranstaltungsort
Jürgen	Merz	3	Berlin
Erik	Wicki	3	Berlin
Ulrich	Müller	3	Berlin
Klaus	Ecke	3	Berlin
Manfred	Stach	3	Berlin

Auch hier stelle ich den SELECT-Befehl noch einmal übersichtlich dar:

```
SELECT vorname, name, t.laufende_NR, Veranstaltungsort
FROM personal As p, teilnehmer AS t,
schulung AS s, veranstalter v
WHERE p.personalnummer=t.personalnummer
AND
t.laufende_NR=s.laufende_NR
AND
s.Veranstalter_NR=v.Veranstalter_NR
AND
Veranstaltungsort='Berlin'
```

Als nächstes interessiert dich, wieviel Kurskosten ortsweise entstanden sind. Das ist wieder ein Beispiel mit der GROUP BY-Klausel. Diesmal werden aber 3 Tabellen benötigt. Außerdem lässt du dir auch anzeigen, um wieviel Kurse es sich jeweils handelt. Die Ausgabe soll sortiert erfolgen (**mausgabe9.php**).

Abfrage:
SELECT Ort, Sum(Kurskosten) AS Kurskostensumme, Count(Kurskosten) AS Kursanzahl FROM personal As p, teilnehmer AS t, schulung AS s WHERE p.personalnummer=t.personalnummer AND t.laufende_NR=s.laufende_NR GROUP BY Ort ORDER BY Ort

Ort	Kurskostensumme	Kursanzahl
Abstatt	900.00	1
Eilenburg	2500.00	2
Genichshain	2900.00	2
Halle/Saale	1600.00	1
Hamburg	5700.00	4
Hohenpriessnitz	2100.00	1
Laussig	1600.00	1
Leipzig	3349.00	2
Moeritz	1800.00	1
Musterdorf	5300.00	3
Wurzen	3700.00	2

Der SELECT-Befehl sieht aus wie folgt:

```
SELECT Ort, Sum(Kurskosten) AS Kurskostensumme, Count(Kurskosten) AS
Kursanzahl
FROM personal As p, teilnehmer AS t, schulung AS s
WHERE p.personalnummer=t.personalnummer
AND
t.laufende_NR=s.laufende_NR
GROUP BY Ort
ORDER BY Ort
```

Datenmanipulation

Mit den Befehlen der Datenmanipulation änderst du Daten, was du in der Regel nicht mehr rückgängig machen kannst. Erstelle so oft wie möglich Sicherungskopien der Daten. Wenn du versehentlich die falschen Datensätze löschst, kannst du diese dann aus den Sicherungskopien wiederherstellen.

Aus diesem Grund erstelle ich mir ein Doppel der vorhandenen Tabellen und nenne sie **personal1, teilnehmer1, schulung1, veranstalter1**.

Zunächst lege ich die Strukturen mit den bekannten CREATE-Befehlen an. (**ncreate1.php,.., ncreate4.php**)

```
CREATE TABLE personal1
(Personalnummer INT (5) NOT NULL
PRIMARY KEY,
Vorname CHAR(30),
Name CHAR(30),
Strasse CHAR(30),
PLZ CHAR(6),
Ort CHAR(30),
Geburt DATE,
Geschlecht CHAR(1),
Gehalt DECIMAL(7,2) );
```

```
CREATE TABLE schulung1
(laufende_NR INT(5) NOT NULL PRIMARY
KEY,
Schulungsbezeichnung CHAR(30),
Kursbeginn DATE,
Kursende DATE,
Kurskosten DECIMAL(7,2),
Veranstalter_NR INT(5));
```

```
CREATE TABLE teilnehmer1
(laufende_NR INT(5),
Personalnummer INT(5),
besucht CHAR(1));
```

```
CREATE TABLE veranstalter1
(Veranstalter_NR INT(5) NOT NULL
PRIMARY KEY,
Veranstaltungsort CHAR(30),
Adresse CHAR(40),
Veranstalter CHAR(30));
```

Anschließend benutzt du den **INSERT**- oder den **REPLACE**-Befehl und liest die aktuellen Daten aus den Originaltabellen **personal, teilnehmer, schulung** und **veranstalter** ein.

REPLACE

Der **INSERT**- sowie der **REPLACE**-Befehl lassen sich beide mit dem **SELECT**-Befehl kombinieren.

Die Syntax der Befehle lautet:

```
INSERT [INTO] Tabelle [(Feld1,...)]
  SELECT ...

REPLACE [INTO] Tabelle [(Feld1,...)]
  SELECT ...
```

Der Unterschied zwischen diesen Befehlen liegt darin, dass **REPLACE** alte Datensätze löscht, da dieser Befehl eine Kombination von **DELETE** und **INSERT** ist.

Folgende REPLACE-Befehle werden eingesetzt, um die aktuellen Daten aus den Orginaltabellen einzulesen (**replace1.php, ..., replace4.php**)

```
REPLACE  INTO personal1
  SELECT * FROM personal;

REPLACE  INTO teilnehmer1
  SELECT * FROM teilnehmer;

REPLACE  INTO schulung1
  SELECT * FROM schulung;

REPLACE  INTO veranstalter1
  SELECT * FROM veranstalter;
```

Anschließend werden die Daten mit Hilfe des SELECT-Befehls angezeigt:

personal1 (Kopie) mit Werten aus personal füllen

SQL-Kommando wurde ausgeführt

Abfrage:
select * from personal1

Personalnummer	Vorname	Name	Strasse	PLZ	Ort	Geburt	Geschlecht	Gehalt
1008	Eike	Taro	Musterstr. 6	09999	Musterdorf	1947-09-10	M	5100.00
1002	Kai	Müller	Halleache Str. 16	04838	Eilenburg	1951-12-28	M	3400.00
1112	Hennes	Heik	Musterstr. 145	09999	Musterdorf	1961-02-06	M	5600.00
1429	Torsten	Mieder	Hauptstrasse 16	04838	Laussig	1971-05-15	M	5400.00
1016	Ulrich	Müller	Weide Str. 2 a	04838	Hohenpriessnitz	1951-01-02	M	8999.00
1430	Christine	Schwarz	Musterstr. 8	09999	Musterdorf	1948-04-24	W	5420.00
1015	Erik	Wicki	Bulten-Weg 3	22587	Hamburg	1950-07-09	M	4533.00
1439	Elfi	Kaufmann	Brandetrasse 15 e	04827	Gerichshain	1966-01-29	W	3900.00
1436	Hans-Jürgen	Sieg	Gerberstrasse 10	24568	Winsen	1947-05-22	M	7000.00
1435	Peter	Schock	Weststrasse 2	74232	Abstatt	1944-11-02	M	4700.00
1437	Christa	Renner	Weidenstrasse 26	25469	Halstenbek	1932-12-27	W	3900.00
1434	Jörg	Schön	Dorfweg 25 a	22589	Hamburg	1969-08-20	M	5600.00
1432	Bernd	Jach	Suelldorfersir. 124	22589	Hamburg	1959-05-10	M	2455.00

DELETE

Mit der Anweisung **DELETE** werden Datensätze aus einer oder mehreren in der **FROM**-Klausel aufgeführten Tabellen gelöscht, wenn diese die in der **WHERE**-Klausel angegebene Bedingungen erfüllen. Ist keine **WHERE**-Klausel angegeben, werden alle Datensätze der Tabelle gelöscht.

Syntax:

```
DELETE FROM Tabelle
[WHERE Vergleichsausdruck]
```

> Das Löschen von Datensätzen lässt sich nicht rückgängig machen!

Der Mitarbeiter mit der Personalnummer 1112 ist aus dem Unternehmen ausgeschieden. Der entsprechende Datensatz soll gelöscht werden. **(del1112.php)**

```
DELETE FROM personal1
WHERE personalnummer=1112;
```

Folgende Rückmeldung informiert dich, dass der Befehl ausgeführt wurde:

Personaltabellendaten (Kopie) löschen

SQL-Kommando wurde ausgeführt

Abfrage:
select * from personal where personalnummer=1112

Personalnummer	Vorname	Name	Strasse	PLZ	Ort	Geburt	Geschlecht	Gehalt

... und das Ergebnis eines **SELECT**-Befehl zeigt, dass dieser Datensatz gelöscht wurde.

Sollen alle Datensätze für einen Neubeginn aus den einzelnen Tabellen gelöscht werden, erfolgt das mit folgenden Befehlen (**del1.php, ..., del4.php**).

```
DELETE FROM personal1;
DELETE FROM teilnehmer1;
DELETE FROM veranstalter1;
DELETE FROM schulung1;
```

Hast du die Befehle ausgeführt, liest du für die nächsten Übungen die Daten wieder ein über die Skripte **replace1.php** bis **replace4.php**.

UPDATE

Mit dem **UPDATE**-Befehl kannst du Werte in schon existierenden Tabellen verändern.

Syntax:

UPDATE Tabelle

SET Feld1 = Wert1,

[Feld2 = Wert2]...

[WHERE Vergleichsausdruck];

Willst du den UPDATE-Befehl benutzen, gibst du den Tabellennamen und die zu ändernden Feldnamen an. Mit **Wert** ist ein Feldwert oder eine Berechnung gemeint. Über den Vergleichsausdruck **WHERE** kann angegeben werden, welche Datensätze in der Tabelle zu ändern sind. Wird keine Bedingung angegeben, werden alle Datensätze der Tabelle aktualisiert. Es folgen zwei Beispiele. Im ersten wird nur ein Datensatz verändert, im zweiten werden noch leere Spalten durch Berechnung gefüllt.

Der Mitarbeiter Müller (Personalnummer=1114) hat geheiratet und trägt nun den Namen Müller-Schwarze. Diese Namensänderung wird mit dem **UPDATE**-Befehle vorgenommen. **(update1.php)**

```
UPDATE personal
SET name = "Müller-Schwarze"
WHERE  personalnummer =1114
```

Überzeuge dich vor dem **UPDATE**-Befehl mit einem **SELECT**-Befehl, ob es die Person gibt und ob die Umbenennung des Namens funktioniert.

Abfrage:
select * from personal1 where personalnummer =1114

Personalnummer	Vorname	Name	Strasse	PLZ	Ort	Geburt	Geschlecht	Gehalt
1114	Karsten	Müller	Musterstr. 3	09999	Musterdorf	1963-07-12	M	4900.00

Abfrage:
UPDATE personal1 SET name = "Müller-Schwarze" where personalnummer =1114

Abfrage:
select * from personal1 where personalnummer =1114

Personalnummer	Vorname	Name	Strasse	PLZ	Ort	Geburt	Geschlecht	Gehalt
1114	Karsten	Müller-Schwarze	Musterstr. 3	09999	Musterdorf	1963-07-12	M	4900.00

Die Kurskosten in der Tabellen **schulung** und die Gehälter in der Tabelle **personal** wurden im DM-Währungsformat erfasst, werden aber im Währungsformat Euro benötigt. Der Eurobetrag ergibt sich aus dem Markbetrag geteilt durch 1,95583.

Es wird vorgeschlagen, mit dem **ALTER TABLE**-Befehl die Strukturen der Tabellen zu verändern, um in die Tabelle **Schulung** die Spalte **Euro_Kurskosten** und in die Tabelle Personal die Spalte **Euro_Gehalt** einzufügen (**alter1.php, alter2.php**).

```
ALTER TABLE personal ADD Euro_Gehalt
DECIMAL(7,2);

ALTER TABLE schulung  ADD Euro_Kurs-
kosten DECIMAL(7,2);
```

Anschließend wird die Spalte mit Hilfe des **Update**-Befehls berechnet **(update2.php, update3.php)**:

```
update personal
set Euro_Gehalt = gehalt / 1.95583;

update schulung
set Euro_Kurskosten = kurskosten /
1.95583;
```

Von dem Resultat deiner Bemühungen kannst du dich mit dem SELECT-Befehl überzeugen. **(ausgabe1.php, ausgabe3.php)**:

Abfrage:
select * from personal

Personalnummer	Vorname	Name	Strasse	PLZ	Ort	Geburt	Geschlecht	Gehalt	Euro_Gehalt
1008	Eike	Taro	Musterstr. 6	09999	Musterdorf	1947-09-10	M	5100.00	2607.59
1002	Kai	Müller	Hallesche Str. 16	04838	Eilenburg	1951-12-28	M	3400.00	1738.39
1112	Hannes	Heik	Musterstr. 145	09999	Musterdorf	1961-02-06	M	5600.00	2863.23
1429	Torsten	Mieder	Hauptstrasse 16	04838	Laussig	1971-05-15	M	5400.00	2760.98
1016	Ulrich	Müller	Weide Str. 2 a	04838	Hohenpriessnitz	1951-01-02	M	8999.00	4601.12
1430	Christine	Schwarz	Musterstr. 8	09999	Musterdorf	1948-04-24	W	5420.00	2771.20
1015	Eenk	Wicki	Bolten-Weg 3	22587	Hamburg	1950-07-09	M	4533.00	2317.69
1439	Elfi	Kaufmann	Brandstrasse 15 e	04827	Gerichshain	1966-01-29	W	3900.00	1994.04
1436	Hans-Jürgen	Sieg	Gerberstrasse 10	24568	Winsen	1947-05-22	M	7000.00	3579.04
1435	Peter	Schock	Weststrasse 2	74232	Abstatt	1944-11-02	M	4700.00	2403.07
1437	Christa	Renner	Weidenstrasse 26	25469	Halstenbek	1932-12-27	W	3900.00	1994.04
1434	Jörg	Schön	Dorfweg 25 a	22589	Hamburg	1969-08-20	M	5600.00	2863.23
1432	Bernd	Jach	Suelldorferstr. 124	22589	Hamburg	1959-05-10	M	2455.00	1255.22
1431	Birgit	Gemse	Bauernkoppel 39	22393	Hamburg	1958-01-13	W	1200.00	613.55
1433	Silvia	Munter	Treppe 7	22587	Hamburg	1952-04-28	W	4555.00	2328.93

Und so müsste die Tabelle **schulung** aussehen:

Abfrage:
select * from schulung

laufende_NR	Schulungsbezeichnung	Kursbeginn	Kursende	Kurskosten	Veranstalter_NR	Euro_Kurskosten
1	ABA Schulung	1996-03-14	1996-03-15	1600.00	1	818.07
2	Laborschulung 2	1996-04-14	1996-04-16	2000.00	4	1022.58
3	Teamtraining	1996-04-07	1996-04-08	2100.00	5	1073.71
4	Grundkurs Excel	1996-11-25	1996-11-26	1200.00	2	613.55
5	Grundlagen PC/Windows 3.1	1996-12-13	1996-12-13	900.00	3	460.16
6	Grundkurs Word für Windows	1996-12-14	1996-12-15	1800.00	4	920.33
7	Grundkurs Powerpoint	1996-01-04	1996-01-04	1200.00	4	613.55
8	Projektmanagement	1995-11-07	1996-11-07	1549.00	5	791.99
9	Telefonmarketing-Training	1996-04-26	1996-04-27	2034.00	5	1039.97
10	Personalplanung in der Praxis	1996-09-03	1996-09-26	1220.00	7	623.78
11	Arbeitszeugnisse richtig	1996-05-18	1996-05-19	2440.00	7	1247.55

Möchtest du später die DM Spalten löschen, nutzt du wieder den **ALTER TABLE**-Befehl:

```
ALTER TABLE personal DROP Gehalt;
ALTER TABLE schulung DROP Kurskosten;
```

Löschen von Tabellen

Der Befehl DROP TABLE löscht die angegebene Tabelle aus der Datenbank. Wurde dieser Befehl ausgeführt, sind Inhalt, Struktur, sowie alle abhängigen Datenbankobjekte (Indizes), die mit dieser Tabelle verbunden waren, gelöscht. Es gibt keinen aufhebenden Befehl. Die Syntax lautet:

```
DROP TABLE [IF EXISTS] Tabelle
```

Du hast nun vor, die Doppel deiner Tabellen zu löschen. Dazu wurden die Programmskripte **drop1a.php, ...drop4a.php** geschaffen, die folgende SQL-Befehle enthalten.

```
DROP TABLE [IF EXISTS] personal1
DROP TABLE [IF EXISTS] teilnehmer1
DROP TABLE [IF EXISTS] schulung1
DROP TABLE [IF EXISTS] veranstalter1
```

Systeminformation

Möchtest du dir Systeminformationen zu deiner Datenbank verschaffen, lässt sich das in MySQL auf verschiedene Weisen bewerkstelligen:

Aufgabe	Befehl
Anzeige der Datenbanken des MySQL-Servers	`SHOW DATABASES`
Anzeige der Tabellen der Datenbank	`SHOW TABLES [FROM Datenbank]`

Anzeige der Spalten der Tabelle einer Datenbank	`SHOW COLUMNS FROM Tabelle [FROM Datenbank]`
	oder
	`DESCRIBE Tabelle`

Anzeige der Indizes der Tabelle	`SHOW INDEX FROM Tabelle[FROM Datenbank]`
Anzeige der aktuellen Statusinformationen	`SHOW STATUS`
Anzeige der Systemvariablen	`SHOW VARIABLES`
Anzeige der aktuellen Prozessliste	`SHOW PROCESSLIST`
Anzeige der Rechte des Benutzers	`SHOW GRANTS FOR Benutzer`
Anzeige der aktuellen Datenbank	`SELECT DATABASE()`

Rechte

Ein relationales Datenbanksystem ist so angelegt, dass Informationen relativ leicht in ihm gespeichert und wiedergewonnen werden können. Oft sind in einem solchen System Daten gespeichert, auf die nicht jedermann Zugriff haben soll. Diese Daten müssen also geschützt werden. Häufig ist es so, dass nur ein bestimmter Personenkreis auf spezielle Daten zugreifen oder sie ändern darf. Wie umfassend der Schutz sein soll, hängt von den Daten und vom relevanten Personenkreis ab. Rechte werden je nach Anforderungen vergeben.

Solche Rechte sind verfügbar ab der **MySQL**-Version 3.22.11.

In einer geschützten Umgebung, z.B. im Webspace eines Providers, stehen die Befehle **GRANT** und **REVOKE** in der Regel nicht zur Verfügung. Der jeweils eingerichtete Benutzer hat keine Berechtigung, Rechte zu vergeben bzw. zu entziehen.

Der Administrator des MySQL-Servers kann mit **GRANT** und **REVOKE** Zugriffsrechte auf vier verschiedenen Ebenen einstellen.

Globale Ebene

Diese Rechte gelten für alle Datenbanken auf dem Server. Sie werden in der Systemtabelle **mysql.user** gespeichert.

Datenbankebene

Diese Rechte beziehen sich auf eine bestimmte Datenbank. Sie werden in den Systemtabellen **mysql.db** und **mysql.host** gespeichert.

Tabellenebene

Diese Rechte steuern den Zugriff auf einzelne Tabellen. Sie werden in den Systemtabellen **mysql.tables_priv** gespeichert.

Zeilenebene

Diese Rechte gelten für die einzelnen Spalten einer Tabelle. Sie werden in den Systemtabellen **mysql.columns_priv** gespeichert.

Der Befehl **GRANT** vergibt Rechte, während der Befehl **REVOKE** sie entzieht.

```
GRANT Rechte_Typ [(Spaltenliste)]
    ON {Tabelle | * | *.* | Datenbank.*}
    TO Benutzer [IDENTIFIED BY 'password']
    [WITH GRANT OPTION]

REVOKE Rechte_Typ [(Spaltenliste)]
    ON {Tabelle | * | *.* | Datenbank.*}
    FROM Benutzer
```

Der Parameter **Rechte_Typ** bestimmt, welche Befehle der ausgewählte **Benutzer** ausüben darf:

Rechte_Typ	**Bedeutung**
ALL PRIVILEGES	alle Rechte
USAGE	keine Rechte
ALTER, CREATE, DELETE, DROP, FILE, GRANT, INDEX, INSERT, PROCESS, RELOAD, SELECT, SHUTDOWN	erteilt dem Benutzer das Recht diese Befehle zu nutzen

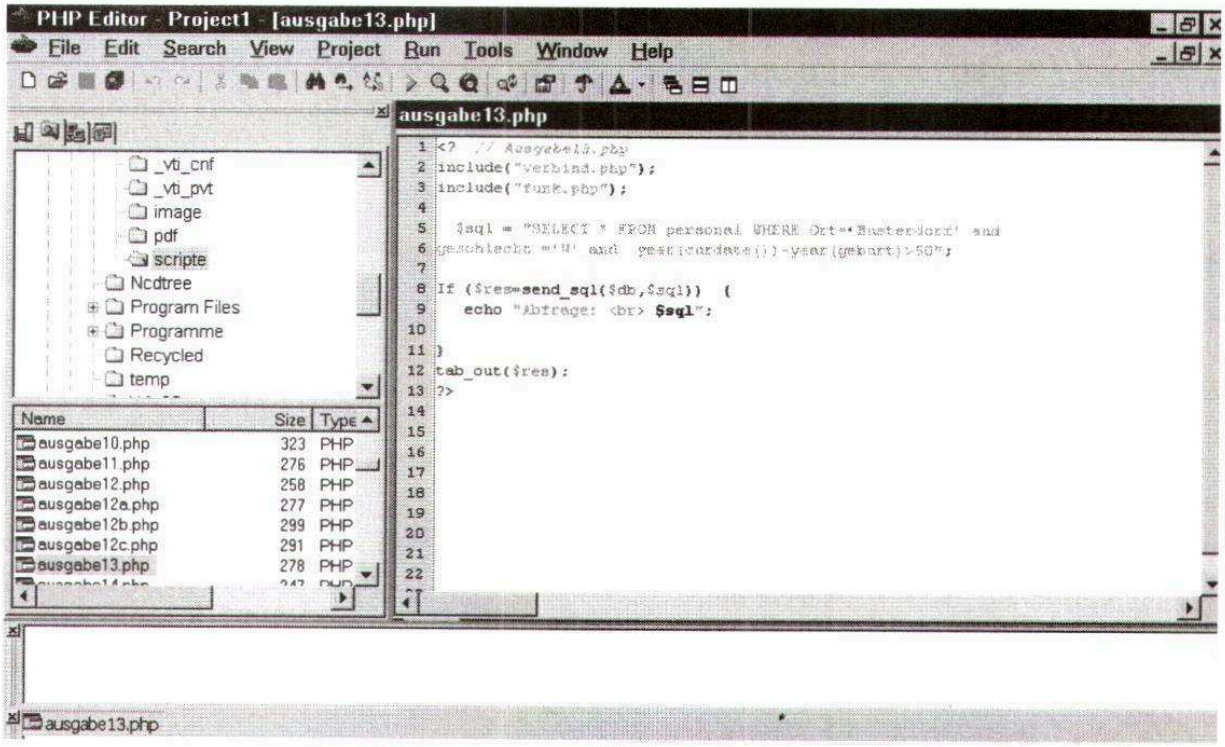
Entwicklungsumgebungen

Bisher haben wir unsere Beispiele mit Hilfe eines einfachen Editors erfasst. Programmierst du aber professionell, solltest du dich nach einer guten Entwicklungsumgebung umsehen. Für PHP stelle ich hier das kostenlose Programm **PHPEd** vor. Für MySQL gibt es die in der Sprache PHP geschriebene Oberfläche **phpMyAdmin**

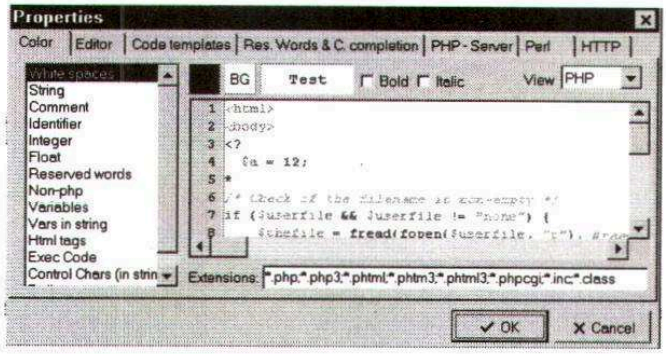
Der PHP-Editor PHPEd

... ist ein leistungsfähiger Editor, der spezielle an die PHP-Sprache angepasst ist. Du findest die neuste Version unter `www.soysal.com/PHPEd`

Das Programm geht davon aus, dass man in einem Projekt arbeitet. Diesem Projekt können dann PHP- und HTML- Dateien hinzugefügt werden. Wird eine Datei geöffnet, steht eine PHP- sowie eine HTML-Ansicht zur Verfügung.



Über die **Eigenschaften** lässt sich der Editor an die eigenen Bedürfnisse anpassen. Unter **Color** hast du die Wahl zwischen Farben für bestimmte Gruppen.



Im Register **Editor** stellst du die Funktionsweise des Editors ein. **Code Templates** enthält Code-Vorlagen.

Das Register **Reserved Words & Code completion** enthält eine Liste reservierter Wörter und automatisch zu vervollständigender Codes.

Im Register **PHP-Server** werden Einstellungen für PHPEd als FTP-Client vorgenommen.

In den Registern **Perl** und **HTTP** werden sprachspezifische Änderungen vorgenommen.

phpMyAdmin

... ist eine Verwaltungsoberfläche für MySQL, von Tobias Ratschiller in PHP geschrieben, die einfach zu installieren und einzusetzen ist. Hier kann man u.a. Datenbanken anlegen und löschen, Tabellen anlegen, kopieren, löschen und verändern sowie Daten erfassen und anzeigen.

Dieses Programm kann man im Internet unter folgender Adresse herunterladen:

`phpwizard.net/phpMyAdmin`

Die heruntergeladene ZIP- oder TGZ-Datei wird in ein Verzeichnis **phpMyAdmin** extrahiert und in das entsprechende Verzeichnis des Webservers kopiert. Eine Dokumentation findet sich in der Datei **Documentation.html**. Hier erhältst du den Hinweis, dass die Nutzung von **phpMyAdmin** die Anpassung einiger Parameter in der Konfigurationsdatei **config.inc.php3** erfordert. Ich habe für meine Arbeit folgende Parameter geändert:

```
$cfgServers[1]['host'] = '[Host]';
$cfgServers[1]['port'] = ''; // Leave blank for default port
$cfgServers[1]['adv_auth'] = false;
$cfgServers[1]['stduser'] = '[Benutzername]';
$cfgServers[1]['stdpass'] = '[Passwort]';
$cfgServers[1]['user'] = '[Benutzername]';
$cfgServers[1]['password'] = '[Passwort]';
$cfgServers[1]['only_db'] = '[Datenbankname]';
```

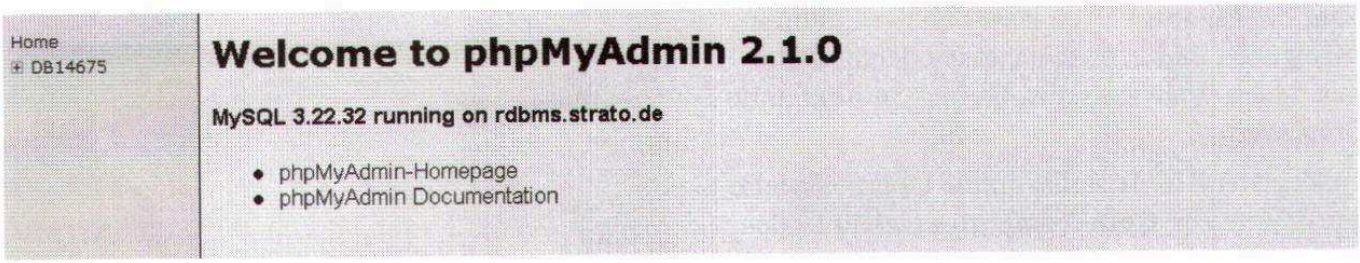
Die Einträge `[Host]`, `[Benutzername]`, `[Passwort]` und `[Datenbankname]` werden durch eigene Daten ersetzt. Willst du deutschen Text sehen, änderst du die **require**-Zeile:

```
require ("german.inc.php3")
```

Soll **PHPMyAdmin** auf dem Webserver deines Providers genutzt werden, solltest du das Verzeichnis vor der Kopie umbenennen, um Missbrauch zu verhindern.

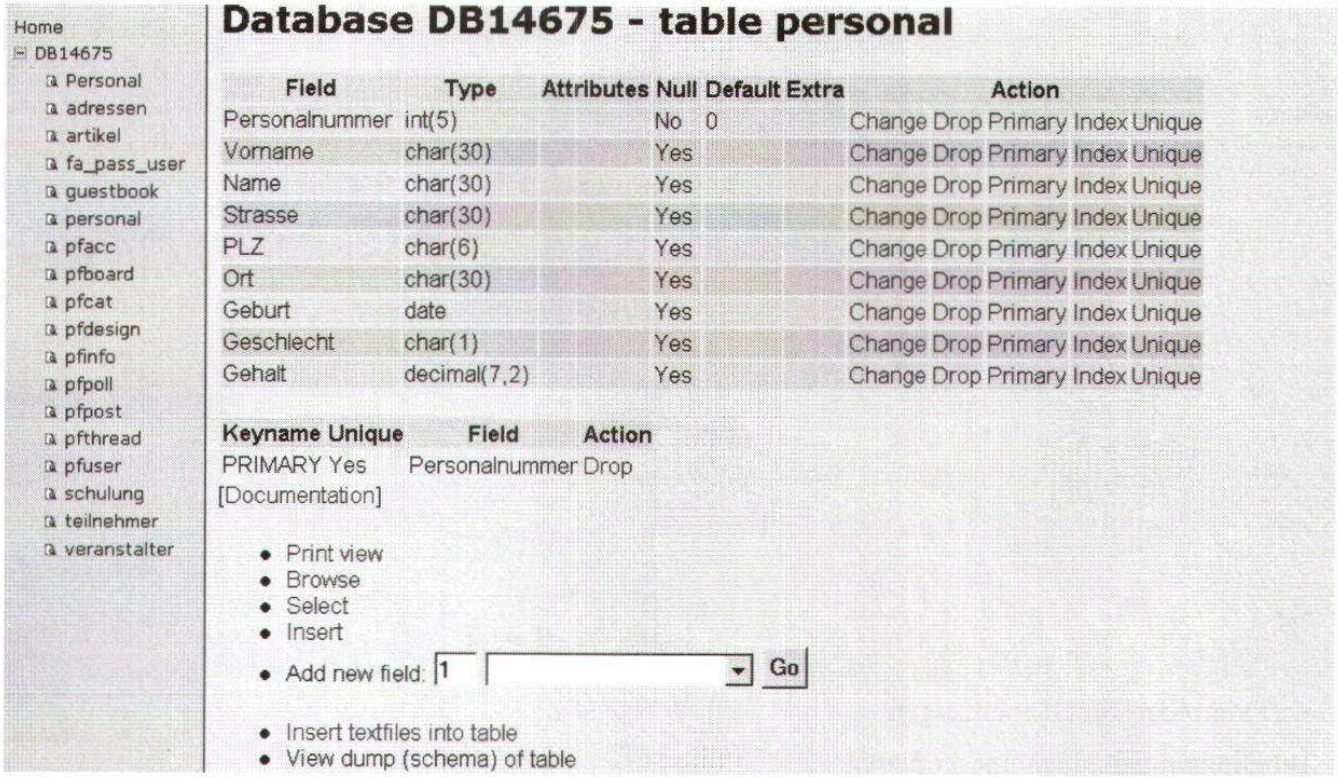
Auch werden eventuell einige Funktionen aus Sicherheitsgründen gesperrt, erwa das Löschen von Datenbanken.

Nach Aufruf des Programmes **PHPMyAdmin** über den Browser erscheint der Begrüßungsbildschirm und der Hinweis, dass MySQL gestartet ist und die Datenbank zur Verfügung steht. Bist du mit den SQL-Befehlen in etwa vertraut, ist das Programm weitestgehend selbsterklärend.



Zur weiteren Arbeit lässt sich im linken Fenster mit dem Pluszeichen die Ansicht der Datenbank erweitern. Es erscheint eine Übersicht zu allen vorhandenen Tabellen. Wählst du eine Tabelle aus, lässt sich ihre Struktur anzeigen und in weitere Menüs verzweigen.

Hier siehst du, dass es 2 Tabellen mit dem Namen Personal gibt. Der einzige Unterschied ist die Schreibung des ersten Zeichens.



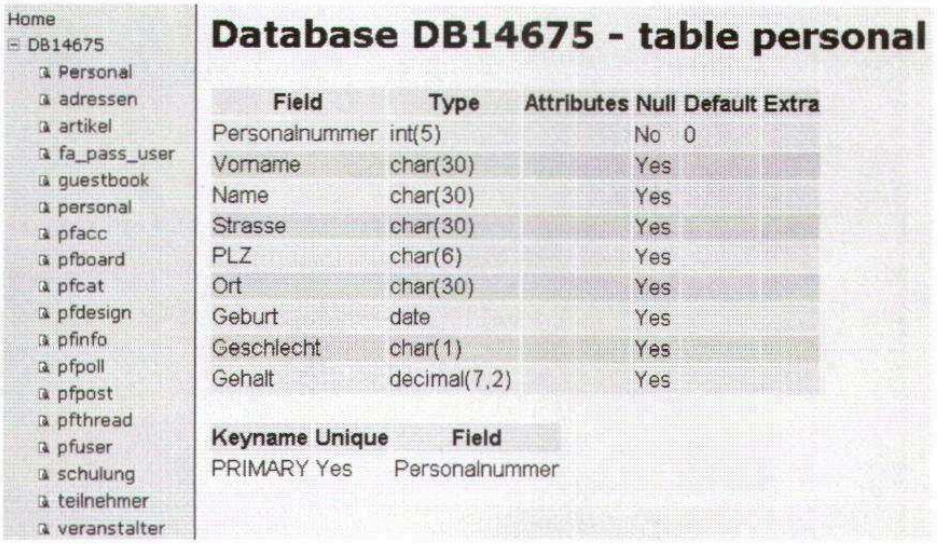
Nach der Tabellenauswahl erhältst du obiges Bild. Es wird die Struktur der gewählten Tabelle angezeigt. Weiterhin ist eine Strukturänderung möglich. Wie du im obigen Bildschirmbild sehen kannst, stehen dir die verschiedensten Aktionen zur Verfügung, die man auch zu Änderungen in der gewählten Tabelle verwenden kann.

Hier bedeuten

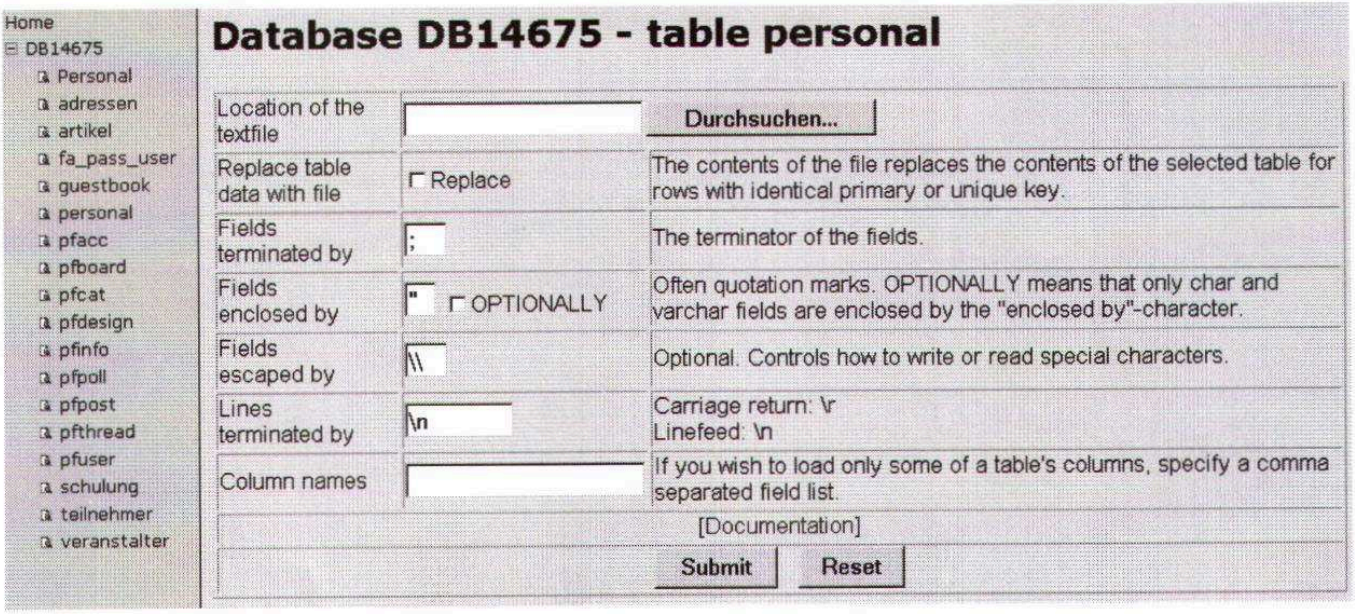
Change	Ändern der Feldeigenschaften
Drop	Löschen des Feldes
Primary	Primärschlüssel anlegen
Index	Index anlegen
Unique	eindeutig Index anlegen

Die Verzweigung in weitere Menüs ist in der unteren Hälfte des Bildschirms vorgesehen. Hier findest du mehrere Links, mit deren Hilfe du folgendes vornehmen kannst:

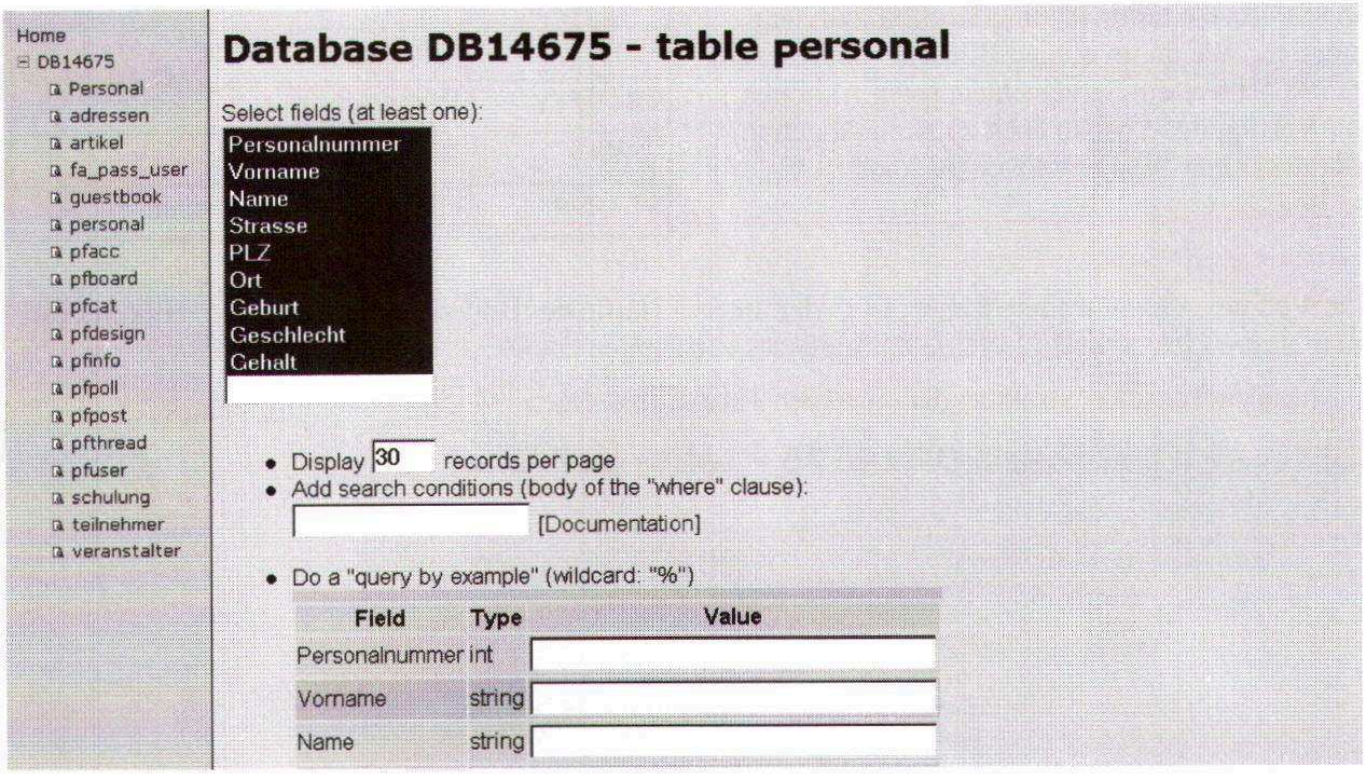
- die Druckansicht (Struktur) der einzelnen Tabellen anzeigen



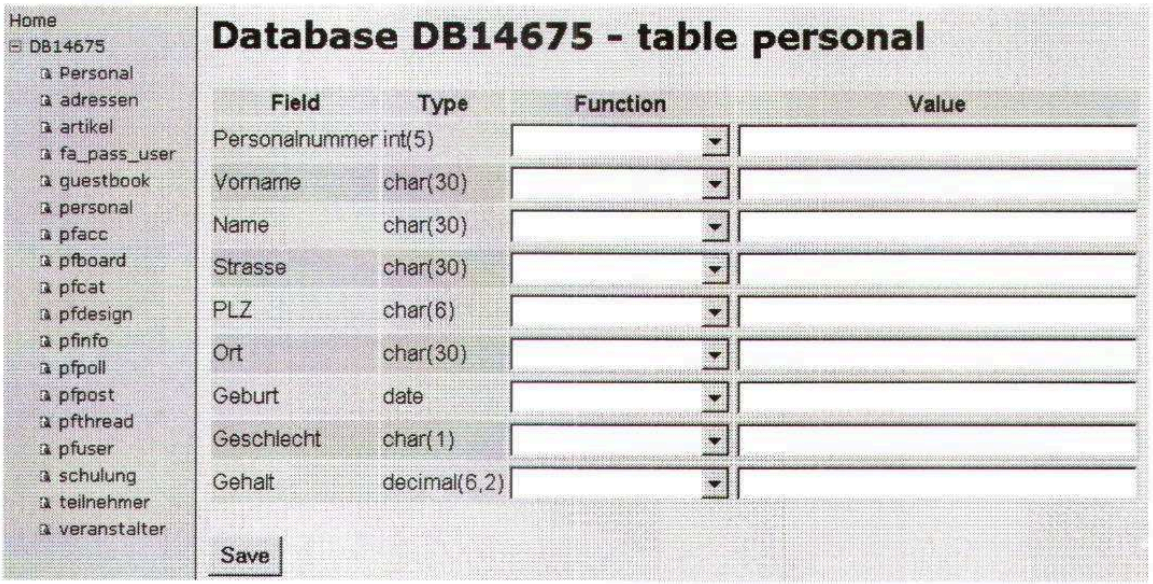
- Felder hinzufügen
- Textdateien in die Tabellen einfügen



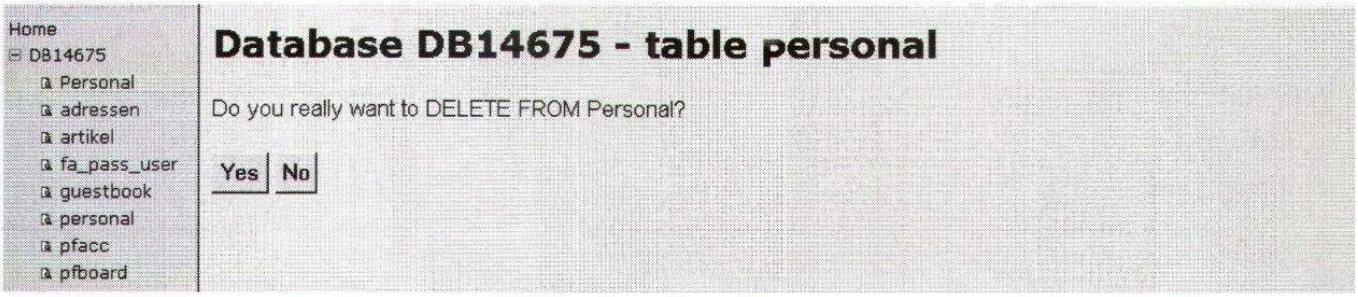
- ein Dump der Tabellen anzeigen
- Tabellen umbenennen und kopieren
- Daten anzeigen



- Daten einfügen



- Tabellen löschen



Diese Beispiele zeigen, dass es sich lohnt, dieses Werkzeug zu installieren. Außerdem kannst du dir den gesamten PHP-Code anzeigen lassen und so dein Wissen über die PHP-Sprache erweitern.

ODBC-Treiber MyODBC

Möchtest du mit einem vertrauten Datenbanksystem (z.B. MS-Access) auf eine MySQL-Datenbank zugreifen, muss der ODBC-Treiber **MyODBC** installiert werden, den du im Internet unter `www.tcx.se` findest.

Willst du mit diesem Datenbanksystem auf die MySQL-Datenbank zugreifen, erfordert das allerdings, dass dein Provider diese Technologie unterstützt. Mein Provider antwortet auf die Frage nach dem ODBC-Zugriff folgendes:

> Derzeit ist ein Zugriff auf die MySQL-Datenbank per ODBC bzw. "MySQL Fern-Administrations-Tools" aus sicherheitstechnischen Gründen nicht möglich. Wir bitten um Verständnis, dass die Sicherheit für unsere Kunden im Vordergrund steht und dadurch zwangsläufig Abstriche bei der Konfigurierbarkeit und Administration der Komponenten stattfinden.

Preis: 4,- EUR pro Heft

Dein KnowWare-Händler kann alle Hefte bei seinem Lieferanten ohne Risiko bestellen.

Du kannst auch bei Bonner Presse Vertrieb bestellen, entweder via www.knowware.de oder per Telefon, Fax oder Brief:

Bonner Pressevertrieb, Moeserstr. 2-3,
49074 Osnabrück, knowware@bpv-online.com
Tel: +49 (0)541 33145-20
Fax:+49 (0)541 33145-33

Versand und Verpackung in EUR - Deutschland
2,00 bis 3 Hefte
2,60 bis 7 Hefte
5,00 bis 10 Hefte
mehr als 10 Hefte: kostenfrei
Auslandsporto: siehe www.knowware.de

Name
Anschrift

E-Mail

Tel.

Geplante Hefte:
Internet für Fortg., XML, Corel Draw, Virus
Javascript Fortg., Office XP

* Bestseller

St	**Windows**	
	Start mit Windows 3.11	105
	Start mit Windows 95	139
	Windows 95 für Einsteiger	148
	Windows 2000 für Einsteiger	E05
	Windows 2000 für Fortg.	P17
	Windows ME/98 für Einsteiger	166
	Windows-Netzwerke für Einsteiger	E04*
	Windows Tips und Tricks	P02
	Windows-Tuning mit der Registry	P01
	Windows XP für Einsteiger	E11
	Word	
	Word 7 für Anfänger	129
	Word 97 für Anfänger (=147)	E03*
	Word 7 für Fortgeschrittene	132
	Weiter mit Word 97/2000	160
	Word f Studenten Ver. 7/97/2000	138
	Word Tips & Tricks	P05
	Word 2000 für Einsteiger	164*
	Xtra - Diverse	
	Acrobat und PDF für Einsteiger	E10
	ISDN für Einsteiger	P13
	Mac für Einsteiger - OS 8.5-9	1
	Rund um den PC (für Anfänger)	143
	Staroffice 5.x für Einsteiger	P09

St	**Datenbank: Access, SQL**	
	Start mit Access 2	107
	Start mit Access 7/97	146*
	Access 2000 für Einsteiger	162*
	Access 97/2000 für Fortg.	154*
	Access: Formulare und Berichte	P18
	Start mit Datenbanken und SQL	131*
	DOS	**Nr**
	Nutze Deinen PC optimal	100
	Was ist denn DOS?	104
	Excel	
	Excel 7 für Anfänger	135
	Start mit Excel (7, auch 5 und 97)	145
	Weiter mit Excel (Ver. 5/7)	112
	Excel VBA Makro-Programmierung	126
	Excel 97 für Einsteiger	156*
	Excel 97 für Fortgeschrittene	155*
	Excel 2000 für Einsteiger	E02*
	Excel 2000 für Fortg.	P20
	Grafik	
	Bildbearbeitung für Einsteiger	P16
	Paint Shop Pro 5/6 für Einsteiger	P10
	Hardware	
	CD-Brennen für Einsteiger	S02
	CD-Brennen und MP3 für Einsteiger	P22
	Homepages	
	Barrierefreies Webdesign	E08
	Dreamweaver 3/4 für Einsteiger	P14
	Flash5 für Einsteiger	E09
	Frontpage 2000 für Einsteiger	159*
	GoLive für Einsteiger	P21
	HomePages für Einsteiger	161*
	WWW - Homepages selbst erstellen	122*
	HomePages mit HTML und CSS	168
	HomePages für Fortg.	P12*
	Intranet, HTML und Java, 2. Ausg.	133
	JavaScript für Einsteiger, 2. Ausg.	P06*
	PHP und MySQL Einsteiger	E07
	Internet	
	Start ins Internet, 4. Ausg.	157*
	Internet für Einsteiger, 2. Ausg.	167*
	Internet Explorer 4 für Einsteiger	152
	E-Mail mit Outlook Express 5	P08*
	Outlook 98/2000 für Einsteiger	S03*
	Outlook 98/2000/2002 für Einsteiger	165*
	Linux	
	Linux für Einsteiger	153
	Linux im Netzwerk	P11
	PowerPoint	
	Start mit PowerPoint 7	140*
	PowerPoint 2000 für Einsteiger	S01*
	Programmierung	
	Batchprogrammierung DOS	125
	C++ für Einsteiger	E06
	CGI & Perl für Einsteiger	P15
	Java2 für Einsteiger	P19
	Visual Basic für Einsteiger	S04*

Bestseller im KnowWare Verlag