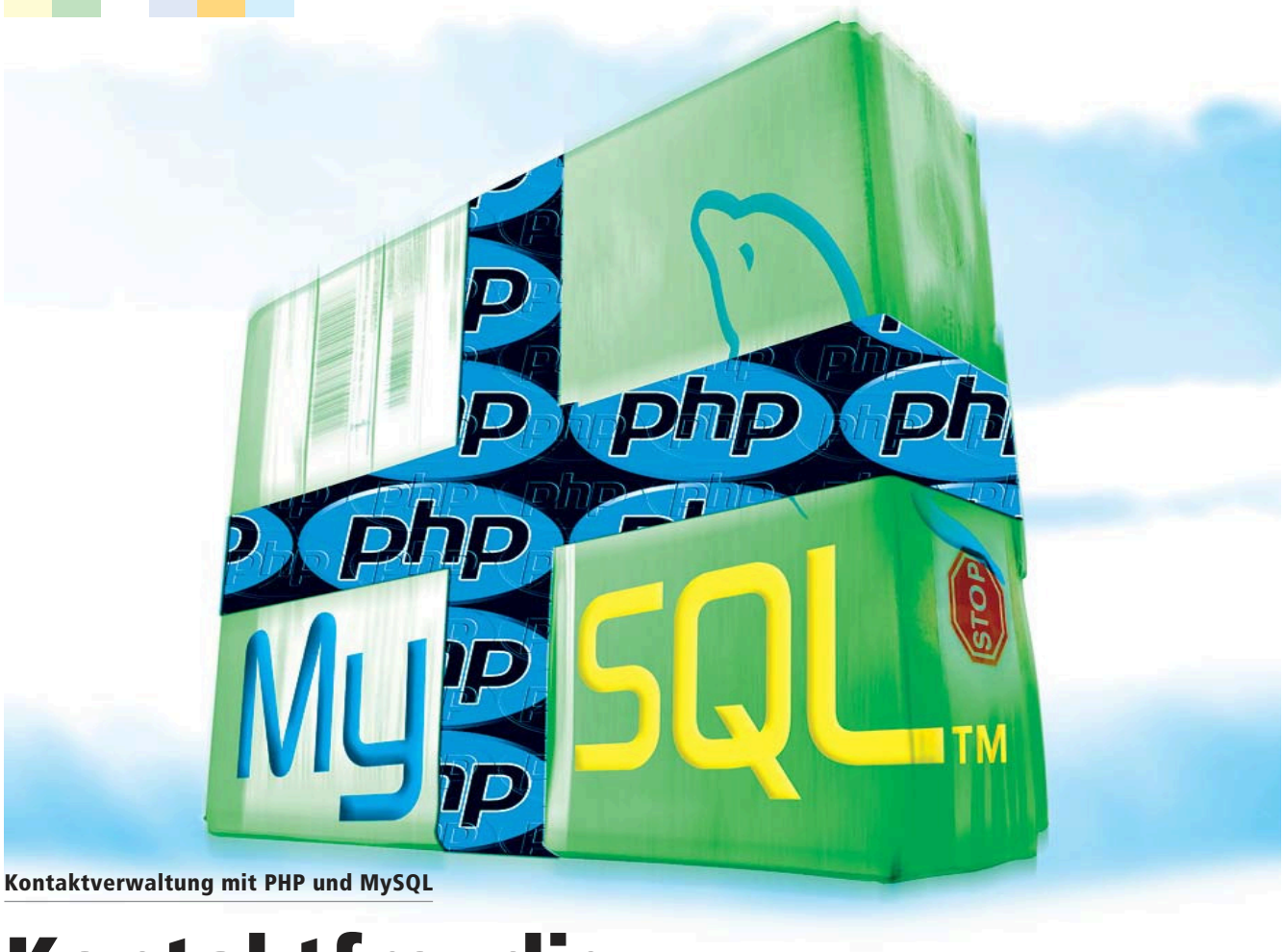


Kontaktverwaltung mit PHP und MySQL

Kontaktfreudig

Datenbankanwendungen im Web funktionieren nach einem immer wiederkehrenden Muster: Daten einfügen, anzeigen, bearbeiten und löschen. Dieser Workshop setzt all dies exemplarisch am Beispiel einer Kontaktverwaltung um. **Von Christian Wenz**

Info

Auf einen Blick

»Der Workshop zeigt eine klassische Web-Anwendung, realisiert mit PHP 5 und MySQL im Backend. Dabei wird ein Schwerpunkt auf den OOP-Zugriff sowie auf sichere Programmierung gelegt.

Das brauchen Sie

»PHP 5 mit MySQLi-Erweiterung und deaktivierten Magic Quotes
»MySQL
»Listings auf Heft-CD

Viele Entwickler sind in Bezug auf die Datenbank, die sie im Web einsetzen, festgefahren. Es gibt einen speziellen Favoriten, der anscheinend konkurrenzlos ist. Und in der Tat hat jede relevante Datenbank irgendwelche Besonderheiten, die sie von der Konkurrenz abgrenzen. Die Praxis zeigt jedoch, dass der Großteil der Datenbankanwendungen immer nach demselben Strickmuster verfährt: Daten werden eingefügt und dann angezeigt. Unter Umständen können diese Daten dann auch noch bearbeitet oder gelöscht werden. Es dreht sich also alles um die vier SQL-Kommandos *INSERT*, *SELECT*, *UPDATE* und *DELETE*.

PHP macht es zudem den Entwicklern einfach, sich bei der Wahl der Datenbank an der Datenbank selbst zu orientieren und nicht an der Skriptsprache, denn PHP unterstützt fast jedes einigermaßen bekannte und verbreitete RDBMS und auch einige Exoten. Dennoch ist MySQL meist die erste Wahl, vor allem wegen der Hoster-Situation. Dieser Workshop implementiert eine Standard-Web-Anwendung mit PHP und MySQL. Das Augenmerk liegt hierbei auf PHP 5.x und dem damit möglichen OOP-Zugriff auf die neue MySQL-Erweiterung dieser PHP-Version, *ext/mysqli*. Diese muss dazu im System bereits installiert sein.

Außerdem liegt ein weiterer Schwerpunkt des Workshops auf sicherer Programmierung: Daten werden also vor der Verarbeitung geprüft. Auch wenn das Anwendungsszenario, eine Kontaktdatenbank, primär für den Eigenbedarf geeignet ist und somit Sicherheit ein klein wenig in den Hintergrund rückt, ist der Einsatz von Best Practices generell bei der Programmierung eine gute Grundlage. Im nebenstehenden Kasten finden Sie die Informationen zur Datenbankstruktur, die von dem Workshop verwendet wird. Dann aber können Sie bereits loslegen.

»Schritt 1: Schreiben«

Zunächst soll es die Anwendung ermöglichen, Kontakte in der Webdatenbank abzulegen. Dazu bedarf es als Erstes einer simplen HTML-Eingabemaske:

```
<form method="post">
<table>
<thead>
<tr><th>Information</th><th>Wert</th>
</tr>
</thead>
<tbody>
<tr><td>Vorname</td><td><input type=
```

```
"text" name="vorname" /></td></tr>
<tr><td>Nachname</td><td>
<input type="text" name="nachname"
/></td></tr>
<tr><td>E-Mail</td><td>
<input type="text" name="email"
/></td></tr>
</tbody>
</table>
<input type="submit" value="Kontakt
anlegen" />
</form>
```

Wird dieses Formular verschickt, prüft das PHP-Skript zunächst, ob überhaupt alle Daten übermittelt worden sind. Dabei könnten Sie diese Überprüfung auch ausführlicher gestalten, etwa darauf, ob in allen Feldern auch etwas Sinnvolles steht:

```
<?php
if (isset($_POST['vorname']) &&
is_string($_POST['vorname']) &&
isset($_POST['nachname']) &&
is_string($_POST['nachname']) &&
isset($_POST['email']) &&
is_string($_POST['email'])) {
```

Nur dann dürfen die Daten überhaupt eingetragen werden. Dazu wird zunächst die Verbindung zum MySQL-Server aufgebaut und die richtige Datenbank ausgewählt. Server, Benutzernamen und Passwort müssen Sie hier und in allen anderen Listings an Ihr System anpassen. Im Beispiel verwenden die Autoren den Standardnutzer von MySQL, *root* ohne Passwort, was aber auf einem Produktivsystem nichts zu suchen hat:

```
try {
$db = new mysqli('localhost',
'root', '');
$db->select_db('kontakte');
```

Dann schickt das Skript die Kontaktdaten an die Datenbank. Dazu werden Prepared Statements verwendet, eine sehr performante und sichere Möglichkeit, Daten an den Server zu senden. Ein Vorteil: Um die Beseitigung gefährlicher Sonderzeichen in den Eingaben kümmert sich MySQL. Im PHP-Code bereiten Sie eine SQL-Anweisung mit der Methode *prepare()* vor. Dynamische Daten werden durch einen Fragezeichen-Platzhalter ersetzt.

```
$sql = $db->prepare('INSERT INTO
kontakt (vorname, nachname, email)
VALUES (?, ?, ?)');
```

Im zweiten Schritt binden Sie die Werte an die Platzhalter. Die Syntax der Methode *bind_param()* ist ungewöhnlich: zunächst ein String mit den einzelnen Datentypen (*s* steht für String, *i* für Integer und so weiter), dann die Werte als Referenzen:

```
$sql->bind_param('sss', $_POST
['vorname'], $_POST['nachname'],
$_POST['email']);
```

Die Methode *execute()* schickt dann die Anfrage an den Datenbankserver. Da jeder Eintrag in der Datenbank einen Autowert hat, ist es natürlich interessant zu wissen, welcher dies ist. Dafür gibt es die Eigenschaft *insert_id*, die diese Infos liefert:

```
if ($sql->execute()) {
$id = $db->insert_id;
echo 'Kontakt eingetragen (ID: ' .
$id . ')!';
}
```

Etwas Fehlerbehandlung, und die Seite ist fertig.

»Schritt 2: Auslesen«

Das Auslesen von Daten mit PHP funktioniert bei allen Datenbanktypen ähnlich: Ein *SELECT*-Kommando wird an die Datenbank geschickt, die Ergebnisliste per *while*-Schleife durchschritten und Stück für Stück ausgegeben. Bei MySQL ist das nicht anders. Ziel der Ausgabe soll eine Tabelle sein, die alle Daten enthält. Diese Tabelle hat eine zusätzliche Spalte, in der Links zum Bearbeiten und Löschen der Einträge untergebracht werden. Die Verbindung zur Datenbank wird aufgebaut und das Datenbank-Handle in der Variablen *$db* abgespeichert. Die *query()*-Methode schickt eine Anfrage an MySQL und liefert ein Handle auf das Ergebnis zurück:

```
if ($ergebnis = $db->query('SELECT *
FROM kontakt')) {
```

Jetzt wird es etwas knifflig. Die Methode *fetch_object()* liefert die aktuelle Zeile der Ergebnisliste als Objekt zurück – die Objekteigenschaften sind die Spalten der Ergebnisliste. Außerdem wird der Zeiger auf die Ergebnisliste eine Zeile weiterbewegt. Sind nun keine weiteren Zeilen mehr da, liefert *fetch_object()* den Wert *false* zurück.

Der Ausdruck *($zeile = $ergebnis->fetch_object())* hat also dann und nur dann den Wert *false*, wenn *fetch_object()* den Wert *false* zurückliefert. Damit eignet er sich exzellent als Abbruchkriterium für eine *while*-Schleife.

In jedem Schleifendurchlauf wird der gerade aktuelle Datensatz ausgegeben. Etwaige HTML-Sonderzeichen werden dabei selbstverständlich speziell kodiert:

```
while ($zeile = $ergebnis->
fetch_object()) {
printf('<tr><td>%s</td><td>%s</td>
<td>%s</td><td><a
href="mailto:%s">%s</a></td><td>
<a href="bearbeiten.php?id=%s">
Bearbeiten</a> 
<a href="loeschen.php?id=%s">
L&ouml;schen</a></td></tr>',
(int)$zeile->id,
htmlspecialchars($zeile->vorname),
htmlspecialchars($zeile->nachname),
htmlspecialchars($zeile->email),
htmlspecialchars($zeile->email),
```

▸ weiter auf Seite 30

Einen neuen Kontakt legen Sie über das webbasierte Formular von überall aus an.

Die Anzeige aller Kontakte lässt Ihnen auch die Möglichkeit zur Bearbeitung.

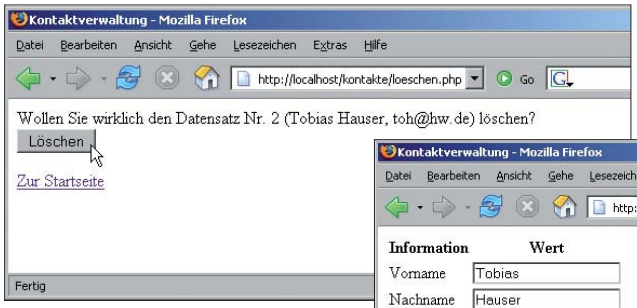
Info

Die Datenbank

Um das Beispiel übersichtlich zu halten, wird eine eher einfache Datenbankstruktur gewählt.

Am besten legen Sie diese mit PhpMyAdmin oder einem der anderen auf Seite 38 gezeigten Tools an.
Die Datenbank heißt *kontakte*. Darin gibt es eine Tabelle namens *kontakt*, die alle Daten enthält. Für den Vor- und Nachnamen sowie die E-Mail-Adresse gibt es jeweils ein *VARCHAR*-Feld. Außerdem stellt das Feld *id*, ein Autowert, den Primärschlüssel dar, um später die Bearbeitung der Daten zu vereinfachen. Der zugehörige SQL-Befehl zur Erstellung der Tabelle sieht dann wie folgt aus:

```
CREATE TABLE 'kontakt' (
'id' INT NOT NULL AUTO_INCREMENT
PRIMARY KEY ,
'vorname' VARCHAR( 50 ) NOT NULL ,
'nachname' VARCHAR( 50 ) NOT NULL ,
'email' VARCHAR( 50 ) NOT NULL
) ENGINE = MYISAM ;
```



Sicherheitsabfrage: Soll der Kontakt wirklich verschwinden?

Tippfehler können Sie im Nachhinein jederzeit beseitigen.

```
(int)$zeile->id,
(int)$zeile->id);
}
```

Beachten Sie, wie die Bearbeiten- und Löschen-Links jeweils den Aufbau *bearbeiten.php?id=XX* und *loeschen.php?id=XX* haben. Der Platzhalter *XX* wird dabei durch den jeweiligen Primärschlüssel ersetzt.

»Schritt 3: Löschen«

Das Löschen von Kontakten aus der Datenbank ist ein zweistufiger Prozess. Um Irrtümer zu vermeiden, muss jeder Löschvorgang bestätigt werden. Diese Bestätigung erfolgt nicht, wie häufig im Web zu sehen, per Javascript, denn dann kann ein Unglück passieren, wie amüsant in *www.thedailywtf.com/forums/65974/ShowPost.aspx* nachzulesen ist. Stattdessen werden die Daten des zu löschenden Kontakts noch einmal angezeigt – nebst Formular, das zum Löschen auffordert. Die Datenabfrage erfolgt dabei analog zum Auslesen aller Daten, nur dass diesmal keine *while*-Schleife notwendig ist. Schließlich dürfte es maximal einen entsprechenden Datensatz geben:

```
$id = (int)$_GET['id'];
$db = new mysqli('localhost',
'root', '');
$db->select_db('kontakte');
if (($ergebnis = $db->query('SELECT
* FROM kontakt WHERE id=' . $id)) &&
$zeile = $ergebnis->fetch_object())
{
printf('Wollen Sie wirklich den
Datensatz Nr. %s (%s %s, %s)
l&ouml;schen?<br />' .
'<form method="post"><input
type="hidden" name="ok" value="true"
/><input type="submit"
value="L&ouml;schen" /></form>',
$id,
html specialchars($zeile->vorname),
html specialchars($zeile->nachname),
html specialchars($zeile->email));
}
```

Wird dieses Formular verschickt, sorgt ein *DELETE*-Kommando dafür, dass der Kontakt aus der Datenbank verschwindet:

```
if ($ergebnis = $db->query('DELETE
FROM kontakt WHERE id=' . $id)) {
echo 'Datensatz gel&ouml;scht!';
}
```

Einige Datenbankadministratoren vertreten die Auffassung, dass man Daten gar nicht löschen dürfe, sondern sie per Flag inaktiv machen sollte. In diesem Fall würde also statt *DELETE* das SQL-Kommando *UPDATE* zum Einsatz kommen und eine zusätzliche Spalte auf einen bestimmten Wert setzen. Bei der Ausgabe der Daten müsste dann diese Spalte explizit abgefragt werden.

»Schritt 4: Bearbeiten«

Die Königsdisziplin ist das Bearbeiten bestehender Daten. Die Programmierung ist ein klein wenig anspruchsvoller. Aufwendiger wird es in Hinblick auf das Vorausfüllen von Formularfeldern. Wenn Sie nicht aufpassen, handeln Sie sich schnell eine Sicherheitslücke ein. Zunächst müssen die Daten überhaupt erst einmal aus der Datenbank ausgelesen werden. Welcher Kontakt bearbeitet werden muss, wird im *GET*-Parameter *id* angegeben:

```
$id = (int)$_GET['id'];
try {
$db = new mysqli('localhost',
'root', '');
$db->select_db('kontakte');
if (($ergebnis = $db->query('SELECT
* FROM kontakt WHERE id=' . $id)) &&
$zeile = $ergebnis->fetch_object())
{
$vorname = $zeile->vorname;
$nachname = $zeile->nachname;
$email = $zeile->email;
```

Die Daten liegen nun in den Variablen *$vorname*, *$nachname* und *$email*. Damit lassen sich Formularfelder vorausfüllen, wobei ein Aufruf von *html specialchars()* wie immer Pflicht ist:

```
<form method="post">
<table>
<thead>
<tr><th>Information</th>
<th>Wert</th></tr>
</thead>
<tbody>
<tr><td>Vorname</td><td>
<input type="text" name="vorname"
value="<?php
echo html specialchars($vorname);
?>" /></td></tr>
<tr><td>Nachname</td><td>
<input type="text" name="nachname"
value="<?php
echo html specialchars($nachname);
?>" /></td></tr>
<tr><td>E-Mail</td><td>
<input type="text" name="email"
value="<?php
echo html specialchars($email);
?>" /></td></tr>
</tbody>
</table>
<input type="submit" value="Kontakt
aktualisieren" />
</form>
```

Nach dem Formularversand werden die alten Daten aktualisiert. Auch hier ist es wieder vorteilhaft, auf Prepared Statements zu setzen. Das entsprechende SQL-Kommando heißt *UPDATE:*

```
$sql = $db->prepare('UPDATE kontakt
SET vorname=?, nachname=?, email=?
WHERE id=?');
$sql->bind_param('sssi',
$_POST['vorname'], $_POST
['nachname'], $_POST['email'], $id);
$id = (int)$_GET['id'];
if ($sql->execute()) {
echo 'Kontakt aktualisiert!';
}
```

Damit ist die Kontaktverwaltung fertig. Sie haben nun die Möglichkeit, von überall – also auch von unterwegs im Internet-Café – diese Informationen abzurufen.

Ein nächster Ausbauschritt könnte darin bestehen, die Seite mit einem Zugriffsschutz zu versehen (etwa per HTTP-Authentifizierung) sowie das ganze System mehrbenutzerfähig zu machen.

»Fazit«

Dieser Workshop zeigt, wie Sie einen Großteil der im Alltagsgeschäft der Web-Entwicklung anfallenden Aufgaben mit PHP und MySQL lösen können. Und auch wenn Sie keine Kontakte verwalten, sondern ein Weblog erstellen, ein Gästebuch betreiben oder ein Newsportal erstellen möchten – die prinzipiellen Arbeitsschritte sind dieselben. **[jp]**