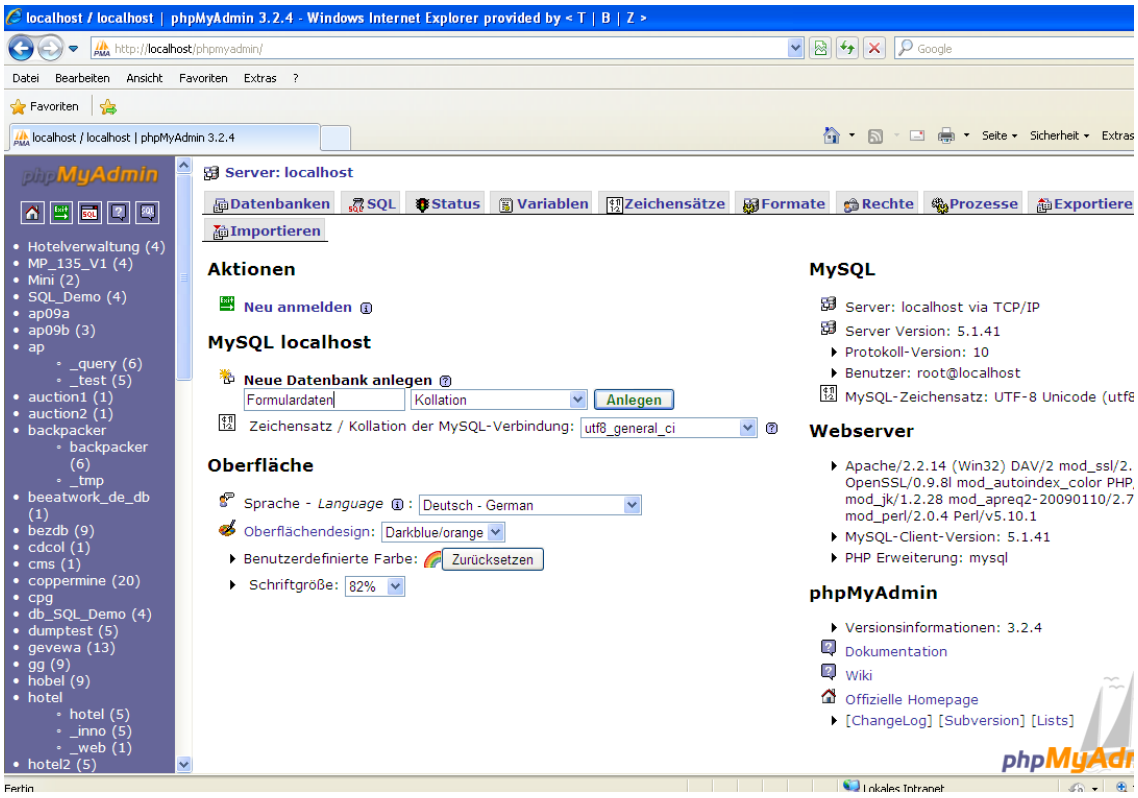
Ziel

Dies ist eine Anleitung, um eine einfache Webapplikation mit Datenbankanbindung zu erstellen. Es sollen in einem ersten Schritt Daten aus einem Formular in eine Datenbank abgespeichert werden. Diese Daten sollen auch als Tabelle im Browser angezeigt werden.

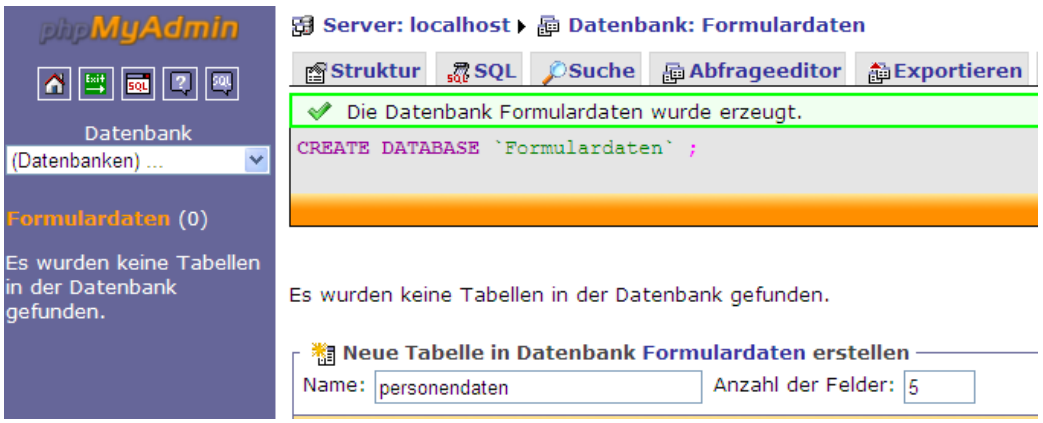
Datenbank anlegen

Datenbankname : Formulardaten

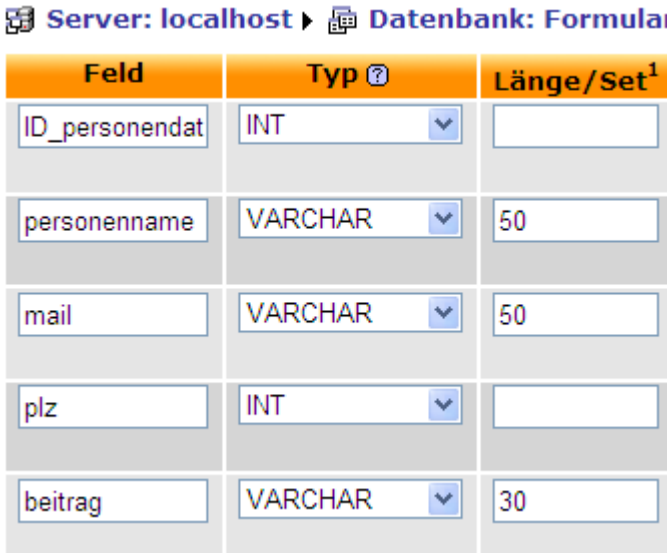
Tabelle: personendaten



Erstellen Sie zu Beginn eine Datenbank mit Hilfe des MySQL-Frontends (z. B. PHPMyAdmin). Wir nennen die neue Datenbank „Formulardaten“ und die Tabelle „personendaten“ mit 5 Feldern (Attributen).



Anschliessend werden die Attribute definiert. Wir haben mit 5 Attributen eines mehr als im Formular. Wir verwenden das erste Attribut (ID_personendaten) als Primärschlüssel (Index – Primary) und lassen die Datensätze automatisch nummerieren (auto-increment AI– yes). Die anderen Attribute definieren wir in Anlehnung an unser Formular. Der Datentyp Integer für die Postleitzahl ist genaugenommen nicht ok, da Postleitzahlen in Deutschland führende Nullen enthalten können und in anderen Ländern auch Buchstaben. Für unser Beispiel ist es aber mal akzeptabel.



Sie können die Tabelle jetzt auch über phpMyAdmin mit Daten füllen, um die Datentypen zu testen.

DB-Verbindungsaufbau aus PHP

```
$linkID = mysql_connect('localhost', 'root', '');

mysql_select_db('Formulardaten', $linkID)
      or die (mysql_errno()."Select_DB Problem:".mysql_error());
```

Im ersten Statement wird die Verbindung mit der Datenbank aufgebaut. Die Authentifizierung erfolgt mit den drei Angaben Host, Benutzername und Passwort.

Das zweite Statement wählt die Datenbank aus, mit der wir in der Folge arbeiten werden - entspricht dem SQL-Befehl *use.*

Die Erweiterung „or die…“ im 2. Statement wird dann ausgeführt, wenn die Funktion einen Fehler verursacht (z. B. falsches Passwort, DB nicht vorhanden). Es wird dann die MySQL-Fehlernummer zurückgegeben und die Textmeldung. Für Testzwecke bei der Entwicklung ist das ok, aber in produktiven Applikationen hat diese Funktion nichts zu suchen. Sie beendet das Skript direkt und gibt mit der MySQL-Errornummer Informationen bekannt, die nicht Enduser-geeignet sind.

Da diese Zeilen in vielen Code-Fragmenten benötigt werden, empfiehlt es sich, sie in eine separate Datei zu schreiben und dann mit dem include-Befehl in den PHP-Code einzubetten: *include ("connect_db.php");*

DB-Insert – einfügen von Datensätzen

Die Formulareinträge werden bei der Methode post in einem Array abgelegt und sind über den Namen der Formularelemente ansprechbar. Es ist möglich direkt auf diese Array-Variablen zuzugreifen, aber durch die folgenden Zeilen speichern wir die Inhalte in lokalen Variablen ab:

```
$mail  = $_POST["mail"];
$plzet = $_POST["plzet"];
$beitrag = $_POST["beitrag"];
$name = $_POST["FName"];
```

Jetzt benötigen wir nur noch das SQL-Statement für den Eintrag in die Datenbank:

```
$query = "INSERT INTO personendaten (personenname, mail, plz, beitrag)
      values ('$name', '$mail', '$plzet', '$beitrag')";

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

Für das „or die…“ gilt das Gleiche, wie bereits oben beschrieben.

Die Kontrolle, ob das Statement korrekt durchgeführt wurde, können Sie über phpMyAdmin vornehmen.

DB-Select – auslesen von Datensätzen

Natürlich soll der Benutzer der Applikation nicht über phpMyAdmin arbeiten. Daher bauen wir jetzt noch eine Funktion ein, die uns den Inhalt der Datenbank nach dem erfolgreichen Eintrag anzeigt.

```
      $result = mysql_query("SELECT * FROM personendaten", $linkID);

      print "<table border>";
      print
      "<th>Nummer</th><th>Name</th><th>Mailadresse</th><th>Postleitzahl</t
      h><th>Beitrag</th>";

      while ($zeile =mysql_fetch_array($result))
          {
             print "<tr>";
             print
      "<td>$zeile[ID_personendaten]</td><td>$zeile[personenname]</td><td>$
      zeile[mail]</td><td>$zeile[plz]</td><td>$zeile[beitrag]</td>";
             print "</tr>";
          }

       print "</table>";
```

Die wichtigen Codestellen im obigen Programmausschnitt sind die erste Zeile (das SQL-Select-Statement), das das Ergebnis der Abfrage in der Variablen *$result* abspeichert und die Funktion *mysql_fetch_array (Param)*, die jeweils einen Datensatz aus diesem Ergebnis-Array ausliest.

Formularvalidierung serverseitig

Also es gibt folgende Möglichkeiten:

- 1. Javascript und PHP Überprüfung (Benutzer kann Formular erst gar nicht abschicken, wenn nicht alle Felder den gewünschten Wert haben; zusätzliche PHP Überprüfung, falls jemand versucht diese Sperre zu umgehen)
- 2. Formular mittels Ajax an PHP senden (Überprüfung wird ausschließlich von PHP durchgeführt; im Fehlerfall wird ein entsprechendes Javascript zurückgegeben, dass die fehlerhaften Felder in einem anderen Style formatiert; im Erfolgsfall wird der komplette Inhalt des Elternelementes durch die Erfolgsseite ersetzt)
- 3. Formular traditionell an PHP senden (Überprüfung ebenfalls ausschließlich durch PHP; im Fehlerfall muss Formular neu generiert werden mit entsprechender Formatierung der einzelnen Felder; im Erfolgsfall wird die Erfolgsseite zurückgegeben)

Von HTML zu PHP: Schreibe Formularverarbeitungen in Normalform

Geschrieben von Kristian Köhntopp,
zuletzt geändert von Niels Braczek am 12. August 2008

Da jede HTML-Datei auch ein gültiges, bedeutungsgleiches PHP-Programm ist, existiert ein einfacher und systematischer Weg, um von einem HTML-Formular zu einem PHP-Programm zu kommen, das dieses Formular bearbeitet. Hält man sich an diesen Weg, wird das resultierende Formular zugleich ein bestimmtes Format haben.

Der erste Schritt ist die Entwicklung eines reinen HTML-Formulares, das die zu verarbeitenden Daten abfrägt.

In einem zweiten Schritt wird man aus diesem Formular ein sogenanntes Affenformular machen, indem man dafür sorgt, dass das Formular sich selbst aufruft und seine alten Werte immer wieder einsetzt. Es heißt Affenformular, weil eine Million Affen dieses Formular eine Million mal aufrufen können, ohne etwas zu bewirken.

```
<form action="<?php echo $_SERVER['PHP_SELF']; ?>">
<input type="text"
       name="textfeld"
       value="<?php if (isset($_REQUEST['textfeld'])) echo htmlspecial-
chars($_REQUEST['textfeld']); ?>">
<br>
<input type="submit"
       name="do_form_x"
       value="Ausführen">
</form>
```

Das Affenformular enthält an zwei Stellen PHP-Code: Bei 'action' wird der Dateiname des Skripts eingetragen. Durch Einsetzen von `$_SERVER['PHP_SELF']` (oder vor PHP 4.1.0 `$HTTP_SERVER_VARS['PHP_SELF']`) an Stelle des Dateinamens kann das Formular seine eigene Adresse bestimmen und sich somit selbst aufrufen. Und außerdem werden die Defaultwerte der verschiedenen Formularfelder durch PHP wieder mit den Ausgangswerten belegt. Wenn das Formular also abgesendet wird und die Eingabewerte nicht korrekt sind, wird das Formular wieder dargestellt und die alten Werte werden als Standardwerte wieder eingesetzt. Damit kann der Anwender sie korrigieren, ohne sie noch einmal eingeben zu müssen.

Der dritte Schritt besteht dann darin, eine Reihe von Funktionen zu codieren, die die Werte aus diesem Formular validieren (siehe "Wie erkenne ich fehlerhafte/fehlende Eingaben?") und ggf. die passenden Fehlermeldungen erzeugen, die dann an den geeigneten Stellen wieder in das Formular eingesetzt werden. Das Endresultat muss dann eine Variable haben, an der entscheidbar ist, ob der Formularinhalt gültig ist oder nicht.

```
<?php

// Funktion zum Drucken von Fehlermeldungen
function errmsg($msg) {
   ?>
   <font color="#ff0000"><b><?php print nl2br($msg)
?></b></font>
   <?php
}

// Überprüft Eingabewerte für $textfeld auf Korrektheit.
function validate_textfeld($val) {
   $msg = "";
   if (strlen($val) < 3)
      $msg .= "Die Eingabe muss mindestens 3 Zeichen lang
sein.\n";

   if (preg_match("/\s/", $val))
      $msg .= "Die Eingabe darf keine Leerzeichen "
            ."oder Tabulatoren enthalten.\n";

   return $msg;
}

// Für jedes Formularfeld werden nun ein oder mehrere
// Validatoren aufgerufen und das Ergebnis der Überprüfung
// gemerkt.
$valid = true;
if (isset($_REQUEST["textfeld"])) {
   $error["textfeld"] = vali-
date_textfeld($_REQUEST["textfeld"]);
   if ($error["textfeld"] != "")
      $valid = false;
}

?>
<form action="<?php print $_SERVER["PHP_SELF"]; ?>">
<input type="text"
       name="textfeld"
       value="<?php print htmlspecial-
chars($_REQUEST["textfeld"]); ?>"><br>
<?php
// Ggf. Fehlermeldung ausdrucken.
if ($error["textfeld"] != "")
   print errmsg($error["textfeld"]);
?>
```

```
<input type="submit"
       name="do_form_x"
       value="Ausführen">
</form>
<hr>
<?php if ($valid and isset($_REQUEST["do_form_x"])) { ?>
<!-- Nutzlast -->
<?php } ?>
```

Schließlich kann man im vierten Schritt daran gehen, die validierten Formulardaten einer Bearbeitungsfunktion zu übergeben und diese Funktion die eigentliche Arbeit machen zu lassen. Das Ergebnis dieser Verarbeitung wird in den meisten Fällen unterhalb des Formulars dargestellt werden, so dass man mit den Eingabedaten oben gleich die nächste Abfrage starten kann.

An dieser Stelle hat man ein funktionierendes Formular und zwei Baustellen, an denen man weiterarbeiten kann: Zum einen muss man sich Gedanken darüber machen, welche Funktionalität aus diesem Formular an anderer Stelle so auch wieder verwendet werden könnte und sollte sich überlegen, ob man nicht eine Klasse schreiben möchte, in die man diese Funktionalität anwendungsunabhängig kapseln könnte. Auf diese Weise wird man sich Schritt für Schritt eine kleine Bibliothek an Funktionen zulegen, die in vielen anderen Projekten ebenfalls Anwendung finden kann.

Zum anderen muss man sich überlegen, ob man die Verkettung von Bildschirmen, Knöpfen und Aktionen nicht ein wenig globaler lösen kann und welche Struktur man seinem Programm dafür geben wird.