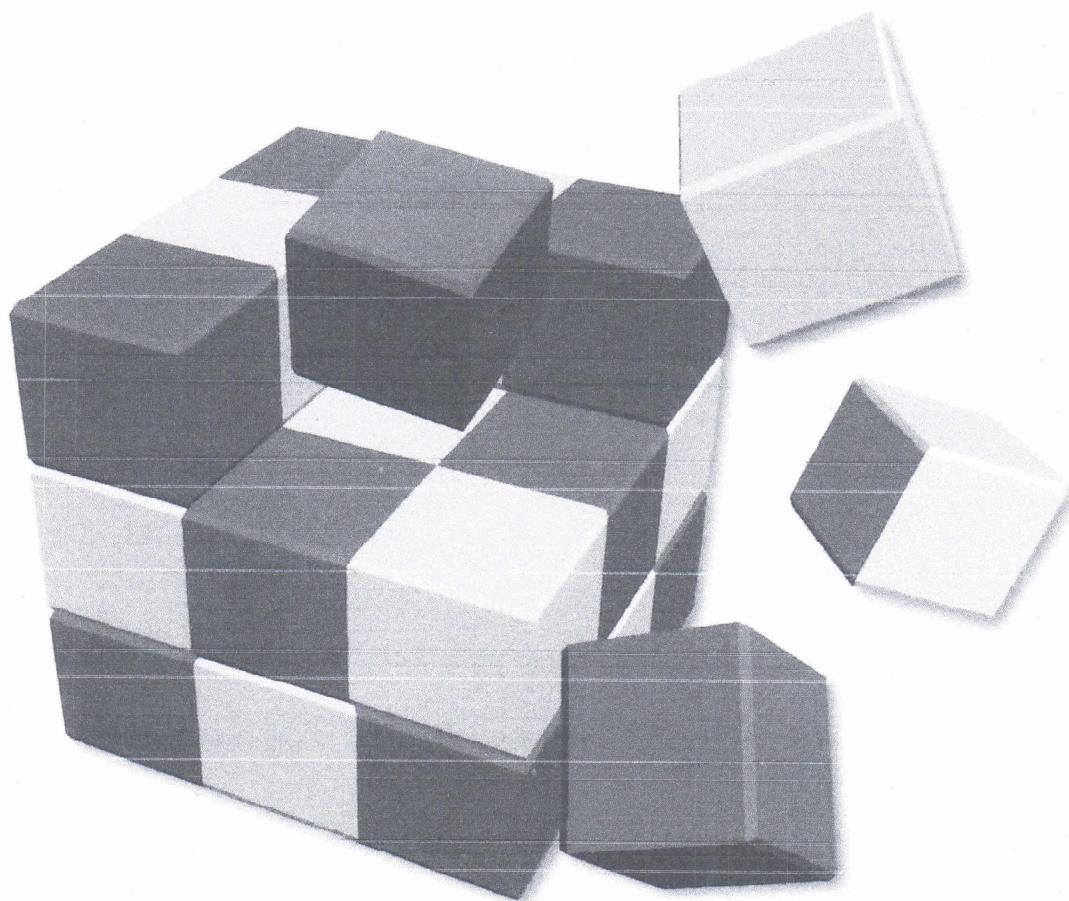


WE CERTIFIED WEB DEVELOPER I

PHP und MySQL

mit PDO

.....



Webmasters Europe
European Webmasters Association

Offizielles Curriculum des Europäischen Webmasterverbandes

Inhaltsverzeichnis

1	Einführung	7
1.1	Ziele und Aufbau dieses Lernhefts	7
1.1.1	Inhalte	7
1.1.2	Aufgaben zur Selbstkontrolle	7
1.1.3	Voraussetzungen	8
1.2	Objektorientierte Programmierung	8
1.2.1	Konzept	8
1.2.2	Objekte	8
1.2.3	Methoden	9
2	Einführung in PDO	10
2.1	Das Problem	10
2.2	Einführung in PDO	10
2.3	Datenbank-Verbindung aufbauen	11
2.3.1	Das PDO-Objekt	11
2.3.2	Der DSN	11
2.3.3	Datenbank mit Login	12
2.4	SQL-Anweisungen ausführen	13
2.4.1	query()	13
2.4.2	errorInfo()	13
2.5	PDOStatement	13
2.5.1	fetch()	14
2.5.2	fetchAll()	14
2.5.3	Ergebnisse zählen	15
2.5.4	Ergebnisse verarbeiten	15
2.5.5	Eine Abfrage schließen	16
2.6	Mit SQL arbeiten	17
2.6.1	SELECT	17
2.6.2	INSERT	17
2.6.3	UPDATE	18
2.6.4	DELETE	18
2.6.5	CREATE TABLE	19
2.6.6	ALTER TABLE	19
2.6.7	TRUNCATE TABLE	20
2.6.8	DROP TABLE	20
3	Fortgeschrittene PDO-Funktionen	22
3.1	PDO-Attribute	22
3.1.1	PDO-Attribute setzen	22
3.1.2	Auswahl von PDO-Attributen	23
3.2	MySQL und Unicode	26
3.2.1	Hintergrund	26
3.2.2	Unicode-Tabellen erzeugen	26
3.2.3	Unicode-Daten aus MySQL auslesen	27
3.3	Prepared Statements	27
3.3.1	Einführung in prepared statements	28
3.3.2	Mehrfaches Ausführen von prepared statements	28
3.3.3	Limitierungen von Platzhaltern in prepared statements	29
3.3.4	Prepared statements mit benannten Platzhaltern	29

3.4	PDO und Sicherheit	31
3.4.1	SQL-Injections	31
3.4.2	PDO und SQL-Injections	32
4	PDO in der Praxis	34
4.1	CRUD	34
4.1.1	Konzept	34
4.1.2	Vorbereitungen	34
4.2	CREATE	36
4.2.1	Code	36
4.2.2	Erklärung	37
4.3	READ	37
4.3.1	Code	37
4.3.2	Erklärung	38
4.4	UPDATE	39
4.4.1	Code	39
4.4.2	Erklärung	40
4.5	DELETE	40
4.5.1	Code	40
4.5.2	Erklärung	41
Lösungen	42
Index	44

1

Einführung

In dieser Lektion lernen Sie:

- die Ziele dieses Lernhefts kennen.
- was Sie an Vorwissen mitbringen sollten.
- wie Sie in PHP mit Objekten arbeiten.

1.1 Ziele und Aufbau dieses Lernhefts

1.1.1 Inhalte

Dieses Lernheft wird Sie in die moderne Art einführen, datenbankbasierte Webapplikationen mit PHP zu programmieren. Schon seit Jahren beherrscht PHP zwar den Umgang mit dem verschiedensten Datenbank-Systemen, aber erst in jüngerer Vergangenheit hat PHP in diesem Bereich einen Status erreicht, der sich hinter den anderen großen Programmiersprachen nicht mehr zu verstecken braucht.

Dank *PDO*, dem Standard-Datenbank-Zugriffsverfahren für PHP, bietet PHP endlich die Funktionalität, die für größere, komplexe Projekte notwendig ist.

1.1.2 Aufgaben zur Selbstkontrolle

Fragen

Gegen Ende jeder Lektion finden Sie einen Abschnitt mit Fragen. Dort werden Ihnen zu den wichtigen Themen dieser Lektion Fragen gestellt, um zu testen, wie viel Sie bereits von dem Lehrstoff behalten und verstanden haben. Gehen Sie die Fragen gewissenhaft durch und beginnen Sie erst mit den nächsten Lektionen, wenn Sie alle Fragen korrekt beantworten können.

Am Ende dieses Lernhefts gibt es zu allen Fragen, die am Ende jeder Lektion gestellt werden, die Antworten. Machen Sie sich bitte zuerst Gedanken über die Fragen, bevor Sie die Antworten nachschlagen.

Übungen

In dem Abschnitt »Übungen« am Ende jeder Lektion finden Sie kleinere Aufgaben, die überprüfen sollen, ob Sie den gelernten Stoff wirklich anwenden können. Sie sollten erst dann mit einer neuen Lektion beginnen, wenn Sie alle Aufgaben gelöst und auch verstanden haben.

Die Lösungen zu den Übungsaufgaben jeder Lektion finden Sie im Begleitmaterial zu diesem Lernheft (Download über Online Campus). Auch hier gilt, nicht spicken!

Optionale Übungen

Die optionalen Übungen testen ebenfalls den Stoff dieser Lektion, aber die Aufgaben sind umfassender und meistens auch schwieriger. Die Aufgaben verbinden auch oft den Stoff dieser Lektion mit den Techniken der vorherigen Lektionen. Wenn Sie bisher wenig Erfahrung in der Programmierung haben, werden Sie eventuell nicht alle optionalen Übungen selbständig lösen können.

Ist das der Fall, machen Sie sich keine Sorgen und gehen Sie zur nächsten Lektion über. Die Pflichtübungen prüfen alle wichtigen Grundlagen ab. Kehren Sie, nachdem Sie das ganze Skript durchgearbeitet haben, noch einmal zu den optionalen Übungen zurück. Sie werden sehen, dass Sie diese nun lösen können.

1.1.3 Voraussetzungen

Um mit diesem Lernheft erfolgreich arbeiten zu können, sollten Sie folgende Voraussetzungen mitbringen:

- Sie sollten mit der grundlegenden Programmierung in PHP keine Probleme mehr haben. Die üblichen Sprachkonstrukte (Funktionen, if-else, Schleifen ...) werden als bekannt vorausgesetzt.
- Sie sollten Kenntnisse in einem relationalen Datenbanksystem, vorzugsweise MySQL haben. Das Arbeiten mit der Datenbank sollte Ihnen flüssig von der Hand gehen.
- Sie sollten mit den gängigen SQL-Anweisungen (SELECT, INSERT, UPDATE, DELETE, CREATE TABLE ...) umgehen können. Es existiert zwar eine kurze Wiederholung zur Syntax von SQL in diesem Lernheft, diese soll aber kein Ersatz für eine Datenbank/SQL-Schulung sein.
- Vorteile haben Sie, wenn Sie schon grundlegende Kenntnisse in der objektorientierten Programmierung mitbringen, da *PDO* die Syntax der *OOP*¹ verwendet. Dies ist allerdings keine feste Voraussetzung.

Auf Ihrem System sollte PHP mindestens in der Version 5.2 inklusive der Erweiterung PDO installiert sein. Die meisten Code-Beispiele wurden unter der Version PHP 5.2.4 entwickelt und getestet.

1.2 Objektorientierte Programmierung

1.2.1 Konzept

Um mit diesem Schulungsskript erfolgreich arbeiten zu können, müssen Sie zumindest minimale Grundkenntnisse in der *objektorientierten Programmierung*, kurz *OOP* besitzen. Sie müssen nicht selbst auf diese Weise programmieren können, sondern nur den fertigen Code anderer Personen verwenden können.

Hierfür brauchen Sie ein gewisses Verständnis der Konzepte, die ich Ihnen nun kurz näherbringen möchte.

1.2.2 Objekte

Die OOP führt einen neuen Datentyp ein, der ihr den Namen gegeben hat, nämlich *Objekte*. Diese sind Behälter für Informationen, ähnlich wie Arrays es sind. Der Unterschied zu Arrays besteht allerdings darin, dass nicht nur Variablen, sondern auch Funktionen in dem Objekt enthalten sein können. Diese an Objekte gebundenen Funktionen, nennt man *Methoden*.

Ebenfalls ein wenig anders ist die Art, wie Objekte erzeugt werden. Für jedes Objekt gibt es eine Art Bauplan, genannt *Klasse*, aus der Sie ein neues Objekt erzeugen. Zu diesem Zweck gibt es das PHP-Schlüsselwort new.

Beispiel

```
1 <?php
2   $bmw = new Auto();
3 ?>
```

Listing 1-1: neues_objekt.php

In der Variablen \$bmw wird das neue Objekt abgelegt, genau so, wie Sie es bisher mit Strings, Integern oder Arrays getan haben. Das Auto() ist die Klasse, also der Bauplan des Objekts. Damit ist sichergestellt, dass das Objekt \$bmw ein Auto ist. Die Beispiele können Sie leider nicht selbst nachprogrammieren, da PHP leider keine Klasse Auto enthält.

1.2.3 Methoden

Eine Methode verhält sich im Prinzip so, wie Sie es von einer Funktion gewohnt sind. Sie können ihr Parameter übergeben und erhalten, je nach Methode, einen Rückgabewert.

Der eigentliche Unterschied ist die etwas andere Schreibweise. Da eine Methode zu einem Objekt gehört, werden beide auch zusammen geschrieben. Der Aufruf einer Methode sieht folgendermaßen aus:

Beispiel

```
1 <?php
2   $bmw = new Auto();
3   $bmw->fahreLos();
4 ?>
```

Listing 1-2: methoden1.php

Es wird die Methode `fahreLos()` aufgerufen, die im Objekt `$bmw` enthalten ist. Der Pfeiloperator (`->`) trennt das Objekt von dem Methodennamen.

Wenn Sie der Methode Parameter übergeben wollen, sieht das so aus:

Beispiel

```
1 <?php
2   $bmw = new Auto();
3   $bmw->fahreLos();
4   $bmw->biegeAb('rechts');
5 ?>
```

Listing 1-3: methoden2.php

Es sieht exakt so aus, wie Sie es von Funktionen her kennen. Methoden können wie Funktionen auch Rückgabewerte liefern:

Beispiel

```
1 <?php
2   $bmw = new Auto();
3   $liter = $bmw->zeigeTankFuellung();
4   echo $liter . ' sind noch im Tank';
5 ?>
```

Listing 1-4: methoden3.php

Der Rückgabewert wird wie üblich per Zuweisung in einer Variable abgelegt.

2

Einführung in PDO

In dieser Lektion lernen Sie:

- wie Sie eine Datenbank mit PHP ansprechen.
- wie Sie SQL in PHP einsetzen.
- welche Vorteile PDO gegenüber den alten Zugriffsmethoden bietet.

2.1 Das Problem

Wahrscheinlich haben Sie bisher Daten Ihrer Webanwendungen in Textdateien gespeichert. Eventuell haben Sie zu diesem Zweck auch schon die Funktion `serialize()` verwendet, die auch komplexe Datenstrukturen wie Arrays oder Objekte als Text speicherbar macht.

Diese Methode, Daten persistent zu halten, stößt bei komplexeren Datenstrukturen oder großen Datenmengen aber sehr bald an ihre Grenzen. Große Datenmengen machen die Zugriffe sehr langsam, das Suchen nach bestimmten Datensätzen gestaltet sich zunehmend als schwierig und der gleichzeitige Zugriff von zwei Clients (Webseiten) auf die selben Daten kann im schlimmsten Fall Teile der Daten oder den kompletten Datenbestand unbrauchbar machen. Um diese Probleme zu umgehen, wurden Datenbanken entwickelt. Sie sind Experten im Speichern von riesigen Datenmengen und komplexen Strukturen.

PHP hat schon seit Jahren eine breite Unterstützung für fast alle auf dem Markt erhältlichen Datenbanksysteme, seien es freie oder kommerzielle. Im Zusammenhang mit PHP hat sich aber besonders die freie Datenbank MySQL als Favorit herauskristallisiert. Sie werden heutzutage kaum einen Webhoster finden, der nicht zumindest PHP und eine MySQL-Datenbank als Paket anbietet.

Doch auch die anderen Datenbanken haben ihre Existenzberechtigung. Gerade im kommerziellen Umfeld wird eher auf die "großen" Datenbank-Systeme wie *PostgreSQL*, *Microsoft SQL-Server* oder *Oracle* gesetzt. Auf der anderen Seite gibt es Datenbanken wie *SQLite*, die ideal für sehr kleine, einfache Applikationen geeignet sind. Alle diese Systeme werden von PHP natürlich hervorragend unterstützt.

Bisher war es allerdings so, dass jedes Datenbank-System in PHP eigene Funktionen hatte, um mit der Datenbank zu kommunizieren. Die Funktion, um sich per MySQL mit einer Datenbank zu verbinden, heißt `mysql_connect()` oder `mysqli_connect()`², die für PostgreSQL `pg_connect()` und die für Oracle `oci_connect()`. Wenn also Ihre PHP-Applikation mehrere Datenbanken unterstützen sollte, mussten Sie den kompletten Datenbank-Code mehrfach schreiben.

Diese Funktionen existieren immer noch, doch seit PHP 5.1³ gibt es einen einheitlichen Standard, um auf Datenbanken zuzugreifen: *PDO*.

2.2 Einführung in PDO

PDO steht für *PHP Data Objects* und stellt für alle von PHP unterstützten Datenbanken eine einheitliche Schnittstelle zur Verfügung. Das bedeutet, dass es nicht mehr für eine Aufgabe eine Funktion pro Datenbank gibt, sondern nur noch eine einzige Funktion, besser gesagt Methode, für alle Datenbanken. Das vereinfacht die Programmierung mit mehreren Datenbanken ungemein. Daher ist PDO inzwischen die von den PHP-Entwicklern offiziell empfohlene Methode in PHP auf Datenbanken zuzugreifen⁴.

2. Für MySQL bis Version 4.0 `mysql_connect()`, ab 4.1 `mysqli_connect()`.

3. PDO kann ab PHP 5.0 installiert werden, erst seit Version 5.1 ist es standardmäßig dabei.

Ein weiterer Bonus ist, dass PDO eine komplette Neuentwicklung ist und die PHP-Programmierer bei der Gelegenheit eine Menge alten Ballast abgeworfen haben. PDO verfügt über ein objektorientiertes, modernes Interface und unterstützt sogar konsequent PHP-Ausnahmen zur Fehlerbehandlung.

Ein Problem löst PDO allerdings nicht. Die verschiedenen Datenbanken verwenden zwar alle grundsätzlich SQL, haben aber alle eigene Veränderungen und Erweiterungen eingebaut. Wenn Sie für mehrere Datenbanken entwickeln, dann kann es Ihnen also trotzdem passieren, dass Sie Code mehrfach schreiben müssen.

In diesem Lernheft wird zwar PDO, aber nur in Verbindung mit der MySQL-SQL-Syntax behandelt. Auf andere Datenbanken werde ich nicht eingehen. Kenntnisse in SQL sollten Sie bereits besitzen. Es befindet sich zwar eine kleine Wiederholung im Skript, was aber kein Ersatz für eine komplette Schulung in SQL sein kann.

2.3 Datenbank-Verbindung aufbauen

2.3.1 Das PDO-Objekt

PDO stellt eine objektorientierte Schnittstelle zu Datenbanken zur Verfügung. Dementsprechend wird eine Datenbank auch durch ein Objekt von der Klasse PDO repräsentiert, das durch new erzeugt wird.

Beispiel

```
1 <?php
2     $db = new PDO();
3 ?>
```

Listing 2-1: pdo.php (Version 1)

Beachten Sie, dass die Klasse PDO entgegen dem sonst üblichen Standard komplett groß geschrieben wird, da es sich um eine Abkürzung handelt.

2.3.2 Der DSN

Wenn Sie diesen Code ausführen, erhalten Sie von PHP eine Warnung, dass dem Konstruktor mindestens ein Parameter übergeben werden muss. Bei diesem handelt es sich um den sogenannten **DSN** (*Data Source Name*), der beschreibt, wo die Datenbank zu finden ist. Dieser wird in Form eines *URI*⁴ angegeben. Die Form ist für jede der Datenbanken spezifisch. Für MySQL besteht er aus folgenden Elementen:

- **Datenbank-Typ:** Der Typ der Datenbank, mit der sich PDO verbinden soll. Für MySQL ist das immer der String mysql.
- **host:** Hier können Sie den Hostnamen des Servers angeben, mit dem Sie sich verbinden wollen. Wenn sich die Datenbank auf dem gleichen Rechner, wie der Webserver befindet, können Sie hier localhost verwenden.
- **port:** Der TCP/IP-Port, auf dem die Datenbank lauscht. Der Standard-Port von MySQL ist 3306. Dieser Parameter kann weggelassen werden.
- **dbname:** Der Name der Datenbank, mit der Sie sich verbinden wollen.

4. Leider haben immer noch nicht alle Webhoster auf PHP > 5.1 umgestellt. Ich habe sogar Hoster entdeckt (über Teilnehmerfeedback), die PDO manuell deaktiviert hatten, obwohl zum Beispiel PHP Version 5.2 installiert war. Nach einigen bösen E-Mails war aber auch dieses Problem gelöst. ;-)

Falls Ihr Hoster ebenfalls noch zu dieser antiken Sorte gehört, beschweren Sie sich ruhig! Die alten PHP-Versionen sollten heutzutage nicht mehr verwendet werden und PDO gehört in jede moderne PHP-Installation.

5. Uniform Resource Identifier

Beispiel

Um sich also mit einer Datenbank namens pdotest auf dem lokalen Rechner zu verbinden, würde folgender Code funktionieren:

```
1 <?php
2   $db = new PDO('mysql:host=localhost;dbname=pdotest;port=3306');
3 ?>
```

Listing 2-2: pdo.php (Version 2)

Wie gesagt, port dürfen Sie weglassen. Wenn Sie diesen Code ausführen und die Datenbank pdotest noch nicht existiert, wird ein Fehler gemeldet:

Beispiel

```
1 <b>Fatal error</b>: Uncaught exception 'PDOException' with message
  'SQLSTATE[HY000] [2005] Unknown MySQL server host 'localhost;dbname=pdotest'
  (11004)' in pdo.php:2
2 Stack trace:
3 #0 pdo.php(2): PDO->__construct('mysql:host=loca...')
4 #1 {main}
```

Listing 2-3: pdo.php (Ausgabe)

Wenn Sie die Datenbank pdotest anlegen, werden Sie nun einen anderen Fehler auslösen, da zwar die Datenbank gefunden wurde, Sie diese aber nicht sehen dürfen.

Lassen Sie uns stattdessen die bereits vorhandene Datenbank mysql verwenden, in der MySQL unter anderem Benutzerinformationen abspeichert.

```
1 <?php
2   $db = new PDO('mysql:host=localhost;dbname=mysql');
3 ?>
```

Listing 2-4: pdo.php (Version 3)

2.3.3 Datenbank mit Login

MySQL ist eine Datenbank, die einen Login erfordert. Daher müssen Sie dem Konstruktor von PDO noch einen Benutzernamen und ein Passwort als weitere, optionale Parameter übergeben. In den meisten Entwicklungsumgebungen ist das der Benutzer root mit leerem Passwort.

Beispiel

```
1 <?php
2   $user = 'root';
3   $pass = '';
4   $db = new PDO('mysql:host=localhost;dbname=mysql', $user, $pass);
5 ?>
```

Listing 2-5: pdo.php (Version 4)

Jetzt sollte das Skript laufen, ohne einen Fehler zu erzeugen. Die folgenden Beispiele werden den Code zur Datenbank-Verbindung nicht mehr immer enthalten, um sie kürzer zu halten.

2.4 SQL-Anweisungen ausführen

2.4.1 query()

Eine SQL-Anweisung können Sie ausführen, indem Sie die Anweisung als String an die Methode query() des PDO-Objekts übergeben.

Beispiel

```
1 <?php
2   $db->query('SELECT * FROM user');
3 ?>
```

Listing 2-6: query.php (Version 1)

2.4.2 errorInfo()

Um zu sehen, ob die SQL-Anweisung erfolgreich war, können Sie die Methode errorInfo() aufrufen. Diese gibt ein Array mit den von MySQL gelieferten Fehlermeldungen zurück.

Beispiel

```
1 <?php
2   $db->query('SELECT * FROM user');
3   var_dump($db->errorInfo()); //gibt "Array ( [0] => 00000 )" aus
4
5   $db->query('SELECT * FROM users;');
6   var_dump($db->errorInfo()); //gibt "Array ( [0] => 42S02 [1] => 1146 [2]
7 => Table 'mysql.users' doesn't exist )" aus
7 ?>
```

Listing 2-7: query.php (Version 2)

Im ersten Fall war die Abfrage erfolgreich, das Array besteht nur aus dem Fehlercode 00000, was bedeutet, das alles in Ordnung ist. Im zweiten Fall haben wir uns leider verschrieben und so liefert uns errorInfo() den Fehlercode und eine Fehlermeldung. Hier besteht der Fehler darin, dass eine Tabelle namens users nicht existiert.

Wie Sie sehen ist das Standardverhalten von PDO, keine Fehlermeldungen auszugeben. Das mag für den produktiven Einsatz ideal sein, denn die Besucher Ihrer Webseiten möchten keine hässlichen Fehlermeldungen sehen. Während Sie aber die Webseite entwickeln, wäre es besser, sofort zu sehen, ob und was schief gelaufen ist.

An dieser Stelle möchte ich Sie auf den Abschnitt »Fehlermodus einstellen (PDO::ATTR_ERRMODE)« verweisen, in dem auf genau dieses Problem eingegangen wird.

2.5 PDOStatement

Wenn eine Abfrage ohne Fehler durchgeführt wird, erhalten Sie als Rückgabewert der query()-Methode ein neues Objekt, das die Klasse **PDOStatement** hat. Mit diesem Objekt können Sie das Ergebnis der Datenbank-Abfrage auslesen, also die Informationen, die die Datenbank uns zurückgeliefert hat.

Beispiel

```
1 <?php
2   $abfrage = $db->query('SHOW DATABASES;');
3 ?>
```

Listing 2-8: query.php (Version 3)

2.5.1 fetch()

Um die Ergebnisse aus dem Objekt zu erhalten, müssen Sie die Methode `fetch()` auf das `PDOStatement`-Objekt aufrufen. Jeder Aufruf liefert Ihnen den nächsten Datensatz des Ergebnisses (also eine Zeile der befragten Tabelle) oder `false`, wenn keine weiteren existieren.

Beispiel

```
1 <?php
2   $abfrage = $db->query('SHOW DATABASES;');
3   $ergebnis = $abfrage->fetch(); //liefert eine Ergebniszeile
4 ?>
```

Listing 2-9: query.php (Version 4)

Um nacheinander alle Datensätze auszugeben, verwenden Sie am Besten eine `while`-Schleife:

Beispiel

```
1 <?php
2   $abfrage = $db->query('SHOW DATABASES;');
3   while ($ergebnis = $abfrage->fetch()) {
4     var_dump($ergebnis); // tue etwas mit den Daten dieser Tabellen-Zeile
5   }
6 ?>
```

Listing 2-10: fetch-mit-while.php

Da `false` zurückgegeben wird, wenn keine Zeilen mehr übrig sind, bricht `while` auch im richtigen Moment ab.

2.5.2 fetchAll()

Die Methode `fetchAll()` liefert Ihnen alle Ergebniszeilen auf einmal. Sie haben also die Möglichkeit, die Methode `fetch()` so lange aufzurufen, bis keine Ergebnisse mehr kommen, oder mit `fetchAll()` alle Ergebnisse auf einmal zu erhalten. Die Methode `fetch()` wird normalerweise in Verbindung mit einer `while`-Schleife verwendet, `fetchAll()` mit einer `foreach`-Schleife, da ein Array aller Ergebniszellen zurückgeliefert wird.

Beispiel

```
1 <?php
2   $abfrage = $db->query('SHOW DATABASES;');
3   $ergebnisse = $abfrage->fetchAll(); //liefert alle Ergebniszeilen
4
5   foreach ($ergebnisse as $zeile) {
6     var_dump($zeile);
7   }
8 ?>
```

Listing 2-11: query.php (Version 5)

Normalerweise sollten Sie `fetchAll()` bevorzugen, wenn Sie ohnehin alle Datensätze brauchen, da der Code kürzer ist. Die Methode verursacht allerdings unter bestimmten Umständen Probleme. Wenn das Ergebnis Ihrer Suche sehr groß ist und Sie `fetchAll()` aufrufen, werden wie üblich alle Datensätze auf einmal geladen. Bei extrem vielen Datensätzen kann das den Speicher Ihres Servers füllen. Wenn Sie immer nur einen Datensatz holen, wird auch immer nur einer in den Speicher geladen. Dafür dauern viele kleine Ladevorgänge länger als ein einzelner großer.

So lange Sie allerdings keine Datenbanken mit hunderttausenden von Einträgen haben, müssen Sie sich über derartige Probleme keine Gedanken machen.

2.5.3 Ergebnisse zählen

Wenn Sie wissen wollen, wie viele Ergebnisse Ihre Abfrage enthält, können Sie das mit der PHP-Funktion `count()` erreichen. Da die Methode `fetchAll()` ein Array mit Ergebniszeilen zurückliefert, können Sie diese einfach zählen.

Beispiel

```
1 <?php
2   $abfrage = $db->query('SELECT * FROM user;');
3   $ergebnisse = $abfrage->fetchAll();
4   echo count($ergebnisse);
5 ?>
```

Listing 2-12: query.php (Version 5)

Noch besser ist es allerdings, wenn Sie den SQL-Operator `COUNT` direkt verwenden, da auf diese Weise nicht alle Datensätze ausgelesen werden müssen, nur um sie zu zählen.

Beispiel

```
1 <?php
2   $abfrage = $db->query('SELECT COUNT(*) FROM user');
3   $ergebnis = $abfrage->fetch();
4   echo $ergebnis[0];
5 ?>
```

Listing 2-13: query.php (Version 6)

Diese Version wird bei Tabellen mit vielen Datensätzen wesentlich schneller verarbeitet werden.

2.5.4 Ergebnisse verarbeiten

Eine Ergebniszeile, die `fetch()` standardmäßig zurückliefert, besteht aus einem Array mit allen Spalten des Ergebnisses der Datenbank. Es werden alle Spalten sowohl als numerisches Array als auch als assoziatives Array zurückgegeben. Das bedeutet, jeder Wert kommt zweimal im Array vor.

Beispiel

```
1 <?php
2   $abfrage = $db->query('SHOW DATABASES;');
3   $ergebnisse = $abfrage->fetchAll();
4   var_dump($ergebnisse);
5 ?>
```

Listing 2-14: query.php (Version 5)

Sie erhalten als Ergebnis ein Array mit folgendem oder ähnlichem Inhalt. Dies hängt natürlich davon ab, wie die von Ihnen angelegten Datenbanken heißen.

```
1 Array
2 (
3   [0] => Array
4     (
5       [Database] => information_schema
6       [0] => information_schema
7     )
8
9   [1] => Array
10    (
11      [Database] => cdc01
12      [0] => cdc01
```

```

13      )
14
15  [2] => Array
16    (
17      [Database] => mysql
18      [0] => mysql
19    )
20
21  [3] => Array
22    (
23      [Database] => phpmyadmin
24      [0] => phpmyadmin
25    )
26 ...
27 )

```

Listing 2-15: query.php (Ausgabe)

Sie sehen, dass jeder Wert zweimal abgelegt ist, einmal mit dem numerischen Index 0 und einmal über den assoziativen Index Database. Dieser entspricht dem Spaltennamen des Ergebnisses.

Diese Ergebnisse können Sie statt mit var_dump auch mit einer foreach-Schleife durchlaufen:

Beispiel

```

1 <?php
2   $abfrage = $db->query('SHOW DATABASES;');
3   $ergebnisse = $abfrage->fetchAll();
4
5   foreach ($ergebnisse as $zeile) {
6     echo $zeile['Database'] . "<br />";
7   }
8 ?>

```

Listing 2-16: query.php (Version 6)

2.5.5 Eine Abfrage schließen

Wenn Sie mit einer Abfrage fertig sind und die Daten nicht weiter benötigen, sollten Sie das PDO-Statement-Objekt mit unset() löschen, da ansonsten unnötig Ressourcen verschwendet werden und PDO sogar in einigen Fällen Schwierigkeiten hat, mehrere Abfragen gleichzeitig offen zu halten.⁶

Beispiel

```

1 <?php
2   $abfrage = $db->query('SHOW DATABASES;');
3   $ergebnisse = $abfrage->fetchAll();
4   unset($abfrage);
5
6   foreach ($ergebnisse as $zeile) {
7     echo $zeile['Database'] . "<br />";
8   }
9 ?>

```

Listing 2-17: query.php (Version 7)

Falls Sie den Code in einer Funktion oder Methode aufrufen, die ohnehin kurz darauf endet, ist das ausdrückliche Löschen der Abfrage natürlich unnötig. Mit dem Ende der Funktion/Methode wird automatisch auch die Abfrage gelöscht.

6. Siehe Abschnitt »Gepufferte Abfragen (PDO::MYSQL_ATTR_USE_BUFFERED_QUERY)«.

2.6 Mit SQL arbeiten

Der nächste Abschnitt enthält eine Wiederholung der wichtigsten SQL-Anweisungen und Anleitungen, wie diese in PHP verwendet werden.

2.6.1 SELECT

Mit der SELECT-Anweisung können Sie aus einer Tabelle Datensätze auslesen.

Beispiel

```

1 <?php
2 // liefert alle Spalten der Tabelle zurück
3 $abfrage1 = $db->query('SELECT * FROM benutzer');
4
5 // liefert nur die Spalten benutzername und email zurück
6 $abfrage2 = $db->query('SELECT benutzername, email FROM benutzer');
7
8 // liefert nur die Einträge aus der Tabelle, wo das Feld id mit 5 belegt ist
9 $sql = 'SELECT benutzername, email FROM benutzer WHERE id = 5';
10 $abfrage3 = $db->query($sql);
11 ?>

```

Listing 2-18: select.php (Version 1)

Diese Abfragen können Sie dann wie üblich mit `fetch()` oder `fetchAll()` weiterverarbeiten.

Beispiel

```

1 <?php
2 $abfrage = $db->query('SELECT benutzername, email FROM benutzer');
3 $ergebnisse = $abfrage->fetchAll();
4 unset($abfrage);
5
6 echo '<p>Es existieren ' . count($ergebnisse) . ' Benutzer!</p>';
7 foreach ($ergebnisse as $e) {
8     echo 'Der Benutzer' . $e['benutzername'];
9     echo 'hat die E-Mail-Adresse ' . $e['email'] . '<br />';
10 }
11 ?>

```

Listing 2-19: select.php (Version 2)

2.6.2 INSERT

Mit einer INSERT-Anweisung können Sie einen neuen Datensatz in eine Tabelle einfügen.

Beispiel

```

1 <?php
2 $sql = 'INSERT INTO benutzer (benutzername, passwort, email) ';
3 $sql .= 'VALUES ("schneider", "geheim", "schneider@example.com")';
4 $abfrage = $db->query($sql);
5 ?>

```

Listing 2-20: insert.php (Version 1)

Wenn Sie in der Tabelle einen Primärschlüssel mit `autoincrement` haben und den eben erzeugten Schlüssel wissen wollen, können Sie sich diesen bequem von der Methode `lastInsertId()` des PDO-Objekts geben lassen.

Beispiel

```

1 <?php
2   $sql = 'INSERT INTO benutzer (benutzername, passwort, email) ';
3   $sql .= 'VALUES ("schneider", "geheim", "schneider@example.com")';
4   $abfrage = $db->query($sql);
5   echo $db->lastInsertId();
6 ?>

```

Listing 2-21: insert.php (Version 2)

Zeile 5 wird Ihnen den Wert des Primärschlüssels von dem Datensatz zurückgeben, den Sie in Zeile 4 eingefügt haben.

2.6.3 UPDATE

Mit UPDATE können Sie einen oder mehrere vorhandene Datensätze ändern. Welche Datensätze bearbeitet werden, hängt von der WHERE-Bedingung ab, die Sie verwenden. Sollten Sie das WHERE weglassen, werden **alle** Datensätze geändert.

Beispiel

```

1 <?php
2   $sql = 'UPDATE benutzer SET email="t.schneid@example.com",
3   benutzername="boss" WHERE benutzername="schneider"';
4   $abfrage = $db->query($sql);
5 ?>

```

Listing 2-22: update.php (Version 1)

Wenn Sie wissen wollen, wie viele Datensätze Ihre UPDATE-Abfrage verändert hat, können Sie die Methode `rowCount()` des PDOStatement-Objekts verwenden.

Beispiel

```

1 <?php
2   $sql = 'UPDATE benutzer SET email="t.schneid@example.com",
3   benutzername="boss" WHERE benutzername="schneider"';
4   $abfrage = $db->query($sql);
5   echo $abfrage->rowCount();
5 ?>

```

Listing 2-23: update.php (Version 2)**2.6.4 DELETE**

Mit der DELETE-Anweisung können Sie Datensätze aus einer Tabelle löschen. Wie viele es erwischt, hängt wieder von der WHERE-Bedingung ab.

Beispiel

```

1 <?php
2   $abfrage = $db->query('DELETE FROM benutzer');
3 ?>

```

Listing 2-24: delete.php (Version 1)

Das Beispiel aus Listing 2-24 löscht tatsächlich alle Datensätze der Tabelle benutzer. Das folgende Beispiel ist vielleicht etwas sicherer:

Beispiel

```
1 <?php
2 $abfrage = $db->query('DELETE FROM benutzer WHERE id=5');
3 ?>
```

Listing 2-25: delete.php (Version 2)

Jetzt wird nur noch der Datensatz gelöscht, dessen Feld `id` den Wert 5 enthält.

2.6.5 CREATE TABLE

Mit der Anweisung `CREATE TABLE` können Sie eine komplett neue Tabelle anlegen.

Beispiel

```
1 <?php
2 $sql = 'CREATE TABLE benutzer (id INT(5) AUTO_INCREMENT, benutzername
3 VARCHAR(25), passwort VARCHAR(25), email VARCHAR(25), PRIMARY KEY (id))';
4 $abfrage = $db->query($sql);
5 ?>
```

Listing 2-26: create_table.php (Version 1)

Da gerade diese Anweisungen recht lang werden, versuchen Sie, die Abfrage etwas lesbarer zu schreiben.

Beispiel

```
1 <?php
2 $sql = 'CREATE TABLE benutzer (id INT(5) AUTO_INCREMENT,
3 benutzername VARCHAR(25),
4 passwort VARCHAR(25),
5 email VARCHAR(25),
6 PRIMARY KEY (id))';
7 $abfrage = $db->query($sql);
8 ?>
```

Listing 2-27: create_table.php (Version 2)

Es ist in PHP ohne Probleme möglich, einen String auf mehrere Zeilen zu verteilen, was das vorherige Beispiel wesentlich übersichtlicher, aber auch länger macht.

2.6.6 ALTER TABLE

Mit `ALTER TABLE` können Sie die Struktur einer Tabelle verändern, z.B. Felder umbenennen, erlaubte Längen ändern oder ein Feld von `NUL` auf `NOT NULL` setzen.

Beispiel

```
1 <?php
2 $sql = 'ALTER TABLE benutzer CHANGE emaill email VARCHAR(50)';
3 $abfrage = $db->query($sql);
4 ?>
```

Listing 2-28: alter_table.php

Hier wurde der kleine Rechtschreibfehler `emaill` auf `email` korrigiert.

2.6.7 TRUNCATE TABLE

Mit der Anweisung `TRUNCATE TABLE` können Sie eine Tabelle komplett leeren. Alle Datensätze werden gelöscht. Der Unterschied zu `DELETE FROM` ohne WHERE ist, dass auch Dinge wie der Autoinkrement-Wert zurückgesetzt werden. Die Tabelle wird wirklich wieder in einen jungfräulichen Zustand versetzt.

Beispiel

```
1 <?php
2   $abfrage = $db->query('TRUNCATE TABLE benutzer');
3 ?>
```

Listing 2-29: truncate_table.php

2.6.8 DROP TABLE

Mit `DROP TABLE` schließlich können Sie eine Tabelle komplett löschen. Sollten sich in der Tabelle noch Datensätze befinden, sind diese ebenfalls verloren⁷. Wenn Sie nicht sicher sind, ob die Tabelle schon existiert und Sie diese nur löschen wollen, wenn Sie existiert, können Sie die SQL-Anweisung um `IF EXISTS` erweitern.

Beispiel

```
1 <?php
2   $abfrage = $db->query('DROP TABLE IF EXISTS benutzer');
3 ?>
```

Listing 2-30: drop_table.php

Aufgaben zur Selbstkontrolle

Fragen

1. Welchen Datentyp hat in PHP eine Datenbank-Verbindung mit PDO?
2. Wie können Sie mit PDO eine SQL-Anweisung ausführen?
3. Auf welche Arten können Sie Datensätze aus einem PDO-Ergebnis auslesen?
4. Wann müssen Sie bei SQL `INSERT`, wann `UPDATE` verwenden?
5. Wie können Sie alle Datensätze einer Tabelle löschen?
6. Wie können Sie eine Tabelle löschen?

Übungen

1. Schreiben Sie ein PHP-Skript `eintragen.php`, das eine Tabelle namens `filme` mit den Spalten `id`, `titel`, `beschreibung` und `dauer` anlegt. Die Spalte `id` ist der Primärschlüssel.
2. Erweitern Sie das Skript, so dass es zu Beginn immer die Tabelle löscht.
3. Erweitern Sie das Skript, so dass es drei Datensätze in die Tabelle `filme` einträgt. Die Filme dürfen Sie frei wählen.
4. Erstellen Sie ein neues PHP-Skript `filme_anzeigen.php`, das Ihnen die vorhandenen Filme als HTML-Tabelle auflistet.

7. Habe ich schon erwähnt, dass es bei MySQL keinen Papierkorb oder Ähnliches gibt? Habe ich nicht?!

Optionale Übungen

5. Erweitern Sie das Skript eintragen.php, so dass es eine Tabelle namens regisseure mit den Spalten id, vorname und nachname anlegt. Die Tabelle filme erhält eine neue Spalte regisseur_id, das einen Fremdschlüssel zur Tabelle regisseure (Spalte id) darstellt.
6. Erweitern Sie das Skript eintragen.php, so dass es die passenden Regisseure zu den Filmen in die Datenbank schreibt. Vergessen Sie nicht, die Filme mit der passenden regisseur_id zu versehen.
7. Erweitern Sie filme_anzeigen.php, so dass der volle Name des Regisseurs (also die Inhalte der Spalten vorname und nachname) in einer Spalte Regisseur mit angezeigt wird.
8. Erweitern Sie filme_anzeigen.php, so dass der Name des Regisseurs anklickbar ist und auf eine neue Seite regisseur_anzeigen.php führt. Dort soll der Regisseur mit vollem Namen angezeigt werden.
9. Erweitern Sie eintragen.php um einige zusätzliche Filme und Regisseure, wobei einige Filme den gleichen Regisseur haben sollten.
10. Erweitern Sie regisseur_anzeigen.php, so dass die Titel aller Filme aufgelistet werden, in denen der Regisseur Regie geführt hat.

3

Fortgeschrittene PDO-Funktionen

In dieser Lektion lernen Sie:

- wie Sie das Standard-Verhalten von *PDO* an Ihre Bedürfnisse anpassen können.
- wie Sie MySQL dazu bringen, korrekt mit Unicode-Daten umzugehen.
- welche Vorteile *prepared statements* gegenüber dem klassischen Schreiben von SQL bieten.
- wie Sie *prepared statements* einsetzen können.
- wie *prepared statements* mit numerischen und assoziativen Arrays umgehen können.
- wie *prepared statements* Ihren PHP-Code sicherer machen.

3.1 PDO-Attribute

Die Datenbank-Verbindung, also das *PDO*-Objekt, kann mit verschiedenen **PDO-Attributen** an Ihre speziellen Bedürfnisse angepasst werden. Die Schreibweise der einzelnen Attribute ist zwar für Neulinge der objektorientierten Programmierung - vorsichtig ausgedrückt - gewöhnungsbedürftig, folgt dafür aber einem festen Schema.

Der Name eines Attributs wird komplett in Großbuchstaben geschrieben und besteht aus dem Prefix **PDO**, zwei Doppelpunkten **::**, und dem eigentlichen Namen, z.B. **ATTR_PERSISTENT**. Der vollständige Name lautet also **PDO::ATTR_PERSISTENT**⁸.

Es gibt eine ziemlich große Auswahl dieser Attribute, mit denen Sie so ziemlich jedes Verhalten von *PDO* beeinflussen und sehr viele Informationen auslesen können. Im Folgenden möchte ich Ihnen zuerst die beiden Möglichkeiten vorstellen, Attribute zu setzen und Ihnen danach die Attribute zeigen, die Sie - wahrscheinlich - am häufigsten benötigen werden.

3.1.1 PDO-Attribute setzen

Erzeugen des PDO-Objekts

Sie können beim Erzeugen des *PDO*-Objekts, also wenn Sie `new PDO(...)` aufrufen, ein Array mit den Attributen übergeben, die Sie ändern wollen.

Beispiel

```

1 <?php
2     $optionen = array(
3         PDO::ATTR_PERSISTENT      => true,
4         PDO::ATTR_ERRMODE        => PDO::ERRMODE_EXCEPTION
5     );
6     $db = new PDO('mysql:host=localhost;dbname=test', 'root', '', $optionen);
7 ?>

```

Listing 3-1: pdo_mit_optionen.php

8. Wenn Sie es genau wissen wollen, handelt es sich bei derartigen Gebilden um sogenannte Klassenkonstanten.

Hier werden die Attribute PDO::ATTR_PERSISTENT auf true und PDO::ATTR_ERRMODE auf den Wert PDO::ERRMODE_EXCEPTION gesetzt. Das Array \$optionen wird anschließend beim Erzeugen des PDO-Objekts mit übergeben.

setAttribute()

Sie können die Attribute auch während einer bestehenden Datenbank-Verbindung mit der PDO-Methode setAttribute() ändern. Dabei übergeben Sie als ersten Parameter den Namen des Attributs und als zweiten Parameter den gewünschten Wert.

Beispiel

```
1 <?php
2   $db = new PDO('mysql:host=localhost;dbname=pdo_projekt', 'root', '');
3   $db->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_ASSOC);
4 ?>
```

Listing 3-2: pdo_setattribute.php

Hier wird nach dem Aufbau der Verbindung ATTR_DEFAULT_FETCH_MODE mittels setAttribute() auf PDO::FETCH_ASSOC gesetzt.

3.1.2 Auswahl von PDO-Attributen

Persistente Datenbank-Verbindungen (PDO::ATTR_PERSISTENT)

Das Attribut PDO::ATTR_PERSISTENT legt fest, wie PDO mit Datenbank-Verbindungen umgeht. Normalerweise wird die Datenbank-Verbindung geschlossen, wenn ein PHP-Skript endet. Sind jedoch persistente Verbindungen aktiviert, wird die Verbindung zur Datenbank offen gehalten. Das nächste PHP-Skript, das eine Datenbank-Verbindung anfordert, muss nun keine neue aufbauen, sondern kann die bereits vorhandene nutzen.

Der Vorteil von persistenten Verbindungen besteht darin, dass der Aufbau einer Verbindung zu Datenbanken ein relativ aufwändiger Vorgang ist und Sie somit einiges an Rechenzeit sparen können, wenn Sie Verbindungen wiederverwerten. Heutzutage sind persistente Datenbank-Verbindungen für die allermeisten Hosting-Anbieter kein Problem und Sie können sie getrost einschalten.

Persistente Verbindungen werden aktiviert, indem Sie das Attribut PDO::ATTR_PERSISTENT auf true setzen. Dieses Attribut können Sie nicht mittels setAttribute() verändern. Es kann nur beim Erzeugen des Objekts eingestellt werden.

Beispiel

```
1 <?php
2   $optionen = array(PDO::ATTR_PERSISTENT => true);
3   $db = new PDO('mysql:host=localhost;dbname=test', 'root', '', $optionen);
4 ?>
```

Listing 3-3: pdo_attribute.php (Version 1)

Fehlermodus einstellen (PDO::ATTR_ERRMODE)

Wann immer Sie eine fehlerhafte SQL-Anweisung abschicken, wird MySQL einen Fehler melden. Dieser wird jedoch standardmäßig von PDO nicht angezeigt. Das ist an sich vollkommen in Ordnung, denn in einer produktiven Webseite möchten Sie nicht, dass derartige Meldungen ungefragt erscheinen und entweder Kunden verschrecken oder potentiellen Angreifern wichtige Informationen liefern.

Während Sie jedoch die Webseite entwickeln, benötigen Sie möglichst viele Informationen über alle Probleme in Ihrer Programmierung. Daher macht es Sinn, die MySQL-Fehlermeldungen anzuzeigen. Dieses Verhalten können Sie über das Attribut PDO::ATTR_ERRMODE einstellen.

Es sind drei Einstellungen möglich:

- `PDO::ERRMODE_SILENT`: Dies ist der Standardfall. PDO wird keine Fehlermeldungen von MySQL weiterreichen⁹. Wenn Sie die Meldungen sehen wollen, müssen Sie diese von Hand über die PDO-Methode `errorInfo()` ausgeben.
- `PDO::ERRMODE_WARNING`: Ist diese Einstellung aktiv, wird von PDO jedes mal eine PHP-Warnung (*warning*) erzeugt, wenn MySQL einen Fehler meldet. Das PHP-Skript selbst läuft weiter, nur die Meldung wird im Browser angezeigt.
- `PDO::ERRMODE_EXCEPTION`: Ist `PDO::ERRMODE_EXCEPTION` gesetzt, wird das Skript sofort mit einem PHP-Fehler¹⁰ (*error*) abgebrochen und die Meldung ausgegeben.

Beispiel

```

1 <?php
2 $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
3 ?>

```

Listing 3-4: pdo_attribute.php (Version 2)

Dieses Beispiel sorgt dafür, dass das PHP-Skript bei einem MySQL-Fehler abbricht und die Meldung ausgibt.

Standard-Fetch-Modus (PDO::ATTR_DEFAULT_FETCH_MODE)

Wenn Sie normalerweise mit den Methoden `fetch()` oder `fetchAll()` Datensätze auslesen, werden diese als assoziative und numerische Arrays zurückgeliefert. Für den - doch häufig vorkommenden - Fall, dass Sie nicht beide Versionen der Daten benötigen, können Sie sich auch entweder das assoziative oder das numerische Array zurückgeben lassen.

Dies können Sie über das Attribut `PDO::ATTR_DEFAULT_FETCH_MODE` einstellen. Es existiert eine recht große Anzahl an möglichen Werten, die häufigsten drei möchte ich Ihnen vorstellen:

- `PDO::FETCH_BOTH`: Dies ist die Standard-Einstellung. Sie bewirkt, dass von nun an jede Datenbank-Spalte im Ergebnis-Array sowohl mit ihrem numerischen Index als auch mit ihrem Namen auftaucht.
- `PDO::FETCH_NUM`: Mit dieser Einstellung werden nur noch die numerischen Indizes zurückgeliefert.
- `PDO::FETCH_ASSOC`: Mit dieser Einstellung wird nur noch das assoziative Array mit den Spaltennamen als Indizes zurückgeliefert. Diese Einstellung möchte ich Ihnen als sinnvoller Standard wärmstens ans Herz legen.

Beispiel

```

1 <?php
2 $db->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_NUM);
3 ?>

```

Listing 3-5: pdo_attribute.php (Version 3)

Diese Einstellung sorgt dafür, dass standardmäßig Ergebnis-Arrays nur noch mit numerischen Indizes befüllt werden.

Wenn Sie für einzelne Abfragen vom Standard-Fetch-Modus abweichen wollen, können Sie den gewünschten Modus einfach den Methoden `fetch()` und `fetchAll()` übergeben.

Beispiel

```

1 <?php

```

9. Mit Ausnahme der Meldungen, die während des Aufbaus der Datenbank-Verbindung erzeugt werden, also z.B. "falsches Passwort" oder "Datenbank existiert nicht".

10. Eigentlich wird eine PHP-Ausnahme erzeugt, die moderne (PHP 5) Version von Fehlern. In diesem Fall läuft das jedoch auf das Gleiche hinaus.

```

2 $db->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_NUM);
3 $abfrage = $db->query('SELECT * FROM user');
4 $ergebnis = $abfrage->fetch(PDO::FETCH_ASSOC);
5 // das Ergebnis enthält nur assoziative Indizes
6 var_dump($ergebnis);
7 ?>

```

Listing 3-6: pdo_fetch_modus.php (Version 1)

Der Standardfall bleibt weiterhin bei der alten Einstellung, nur für diesen einen Aufruf von `fetch()` wird `PDO::FETCH_ASSOC` als Modus eingestellt.

Gepufferte Abfragen (PDO::MYSQL_ATTR_USE_BUFFERED_QUERY)

Pro Datenbank-Verbindung kann standardmäßig immer nur eine Abfrage gleichzeitig aktiv sein. Folgender Code wird also normalerweise einen Fehler produzieren:

Beispiel

```

1 <?php
2 $db = new PDO('mysql:host=localhost;dbname=mysql', 'root', '');
3 $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_WARNING); // um die
MySQL-Meldungen zu sehen
4 $abfrage1 = $db->query('SELECT * FROM user');
5 $abfrage2 = $db->query('SHOW DATABASES');
6 ?>

```

Listing 3-7: pdo_gepuffert.php (Version 1)

Folgende Ausgabe wird erzeugt:

Beispiel

```

1 Warning: PDO::query() [function.PDO-query]: SQLSTATE[HY000]: General error:
2014 Cannot execute queries while other unbuffered queries are active.
Consider using PDOStatement::fetchAll(). Alternatively, if your code is only
ever going to run against mysql, you may enable query buffering by setting
the PDO::MYSQL_ATTR_USE_BUFFERED_QUERY attribute. in
C:\xampp\htdocs\phptest\pdo_gepuffert.php on line 5?>

```

Listing 3-8: pdo_gepuffert.php (Ausgabe)

Hier steht nicht nur eine kurze Erklärung des Problems, nämlich dass nur eine ungepufferte Anfrage gleichzeitig aktiv sein kann, sondern es werden auch zwei Lösungen präsentiert:

Zum Einen können Sie `fetchAll()` verwenden, um alle Daten aus der Abfrage zu holen und diese schließen, bevor Sie die zweite Abfrage starten.

Beispiel

```

1 <?php
2 $db = new PDO('mysql:host=localhost;dbname=mysql', 'root', '');
3 $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_WARNING);
4 $abfrage1 = $db->query('SELECT * FROM user');
5 $ergebnis1 = $abfrage1->fetchAll();
6 $abfrage1 = null;
7 $abfrage2 = $db->query('SHOW DATABASES');
8 $ergebnis2 = $abfrage2->fetchAll();
9 ?>

```

Listing 3-9: pdo_gepuffert.php (Version 2)

In Zeile 5 werden alle Datensätze aus der Abfrage geholt und in `$ergebnis1` gespeichert. Die Abfrage wird schließlich in Zeile 6 geschlossen. Damit ist der Weg für die zweite Abfrage in Zeile 7 frei.

Die zweite Variante ist, einen speziellen Modus von MySQL zu verwenden, die ***buffered queries***, also gepufferte Abfragen. Dieser, nur für MySQL erhältliche Modus sorgt dafür, dass Sie so viele Abfragen gleichzeitig stellen können, wie Sie wollen. Falls Sie die vorherigen Beispiele nachprogrammiert und keine Fehler erhalten haben, liegt das vermutlich daran, dass Ihre PHP-Installation diese Fähigkeit standardmäßig aktiviert hat¹¹.

Um diesen Modus zu aktivieren, müssen Sie das Attribut PDO::MYSQL_ATTR_USE_BUFFERED_QUERY auf true setzen.

Beispiel

```
1 <?php
2   $db->setAttribute(PDO::MYSQL_ATTR_USE_BUFFERED_QUERY, true);
3 ?>
```

Listing 3-10: pdo_attribute.php (Version 3)

Welche der beiden Varianten Sie verwenden wollen, bleibt Ihnen überlassen. Sie müssen jedoch folgendes beachten:

☞ Die Einstellung PDO::MYSQL_ATTR_USE_BUFFERED_QUERY existiert in PDO nur für MySQL. Wenn Sie mit anderen Datenbanken (Oracle, PostgreSQL, SQLite ...) arbeiten wollen, müssen Sie mit der ersten Lösung (fetchAll() und Abfrage schließen) arbeiten, um das Problem zu umgehen.

3.2 MySQL und Unicode

3.2.1 Hintergrund

Seit Version 4.1 kann MySQL mit verschiedenen Zeichensätzen umgehen. Da sich in den letzten Jahren **Unicode** in der Programmierung immer mehr durchgesetzt hat, um eines (fernen) Tages das Durcheinander mit den Zeichensätzen zu lösen, sollten auch Sie in Ihren Webseiten von Anfang an konsequent mit Unicode arbeiten.

3.2.2 Unicode-Tabellen erzeugen

Um in einer MySQL-Tabelle Unicode-Daten speichern zu können, müssen Sie beim Erzeugen¹² der Tabelle den gewünschten Zeichensatz angeben. Hinter die schließende Klammer von CREATE TABLE (...) können Sie die Anweisung DEFAULT CHARSET schreiben, um die Tabelle auf den gewünschten Zeichensatz umzustellen.

Beispiel

```
1 <?php
2   $sql = 'CREATE TABLE personen (' .
3     'id INT(5) AUTO_INCREMENT,' .
4     'vorname VARCHAR(255),' .
5     'nachname VARCHAR(255),' .
6     'PRIMARY KEY (id)) ' .
7     'DEFAULT CHARSET=utf8';
8   $db->query($sql);
9 ?>
```

Listing 3-11: create_table_unicode.php (Version 3)

11. Dies ist zum Beispiel bei aktuellen Versionen von XAMPP der Fall.

12. Sie können das auch später mit ALTER TABLE nachholen. Dann haben Sie aber unter Umständen schon Daten in anderen Zeichensätzen gespeichert und müssen diese konvertieren.

Beachten Sie, dass MySQL den String `utf8` ohne Bindestrich erwartet, die Schreibweise `utf-8` würde einen Fehler erzeugen.

Von diesem Zeitpunkt an ist die Tabelle `personen` in der Lage, Unicode-Daten aufzunehmen.

3.2.3 Unicode-Daten aus MySQL auslesen

Die bloße Tatsache, dass eine Tabelle mit `UTF-8` umgehen kann, ist leider noch nicht ausreichend. Wenn Sie MySQL nicht explizit mitteilen, dass eine Datenbank-Verbindung `UTF-8` verwenden soll, werden die Daten trotzdem als Standard-Latin-1 verarbeitet.

Um MySQL wirklich auf Unicode umzustellen, können Sie entweder die Server-Konfiguration umschreiben, oder nach dem Verbindungsauflauf explizit Unicode verlangen. Lösung 1 scheitert meistens daran, dass Ihr Webhoster Sie garantiert nicht an die MySQL-Konfigurationsdateien heranlässt, also bleibt Lösung 2.

Über die SQL-Anweisung `SET NAMES` können Sie MySQL mitteilen, dass alle Eingaben von nun an in dem gewünschten Zeichensatz kommen und auch der Server alle Daten in diesem Zeichensatz ausliefern soll.

Beispiel

```
1 <?php
2   $db = new PDO('mysql:host=localhost;dbname=pdo_projekt', 'root', '');
3   $db->query('SET NAMES utf8');
4 ?>
```

Listing 3-12: set_names.php

Ab Abfrage in Zeile 3 werden alle Daten als `UTF-8` interpretiert. In Verbindung mit den Unicode-Tabellen haben Sie nun Ihre PHP-Anwendung komplett auf Unicode umgestellt¹³.

3.3 Prepared Statements

Jeder PHP-Programmierer kennt das Problem: Es ist, vorsichtig ausgedrückt, eine Qual, SQL-Anweisungen mit PHP-Variablen zu modifizieren. Auch Sie haben wahrscheinlich schon Code wie diesen geschrieben:

Beispiel

```
1 <?php
2   $benutzername= 'emrich';
3   $passwort = 'sehrgeheim';
4   $email = 'emrich@example.com';
5
6   $sql = 'INSERT INTO benutzer (benutzername, passwort, email) VALUES ("' .
$benutzername . '", "' . $passwort . '", "' . $email . '")';
7   $db->query($sql);
8 ?>
```

Listing 3-13: klassische_abfrage.php

Die Variablen `$benutzername`, `$passwort` und `$email` müssen mit String-Verknüpfungen in das SQL eingefügt werden. Gerade bei Variablen, die Strings enthalten, haben wir besonders viel Spaß mit den unterschiedlichen Anführungszeichen. Dabei ist dieses Statement, wenn wir ehrlich sind, noch relativ simpel. Größere Tabellen oder gar SQL-JOINS über Tabellen hinweg sorgen garantiert dafür, dass wir die Übersicht verlieren.

13. Zumindest sobald PHP 6 mit vollständiger Unicode-Unterstützung veröffentlicht ist.

3.3.1 Einführung in prepared statements

Daher möchte ich Ihnen eine der besten Erfindungen der Datenbank-Programmierung vorstellen: *prepared statements*. Sie unterscheiden sich von normalen Abfragen, indem Sie nicht direkt ausgeführt, sondern im ersten Schritt mit der Methode `prepare()` vorbereitet werden. Das eigentliche Ausführen wird von einer zweiten Methode `execute()` erledigt.

Beispiel

```

1 <?php
2   $sql = 'INSERT INTO benutzer (benutzername, passwort, email) VALUES
3     ("schneider", "geheim", "schneider@example.com")';
4
5   //eine klassische Abfrage mit query()
6   $abfrage = $db->query($sql);
7
8   //dasselbe mit einem prepared statement
9   $abfrage = $db->prepare($sql);
10  $abfrage->execute();
11 ?>

```

Listing 3-14: prepared_statements.php (Version 1)

Jetzt fragen Sie sich vielleicht, worin der Vorteil liegt, eine Anweisung durch zwei zu ersetzen. Und Sie haben recht, momentan bringt Ihnen die zweite Schreibweise nichts. Der Vorteil von *prepared statements* liegt darin, dass in ihnen Platzhalter verwendet werden können, die Sie später befüllen können.

Beispiel

```

1 <?php
2   $name= 'emrich';
3   $passwort = 'sehrgeheim';
4   $email = 'emrich@example.com';
5
6   $sql = 'INSERT INTO benutzer (name, passwort, email) VALUES (?, ?, ?)';
7   $abfrage = $db->prepare($sql);
8
9   $daten = array($name, $passwort, $email);
10  $abfrage->execute($daten);
11 ?>

```

Listing 3-15: prepared_statements.php (Version 2)

Die Werte in der zweiten Klammer wurden einfach durch Fragezeichen ersetzt. Beachten Sie auch, dass um die Fragezeichen keine Anführungszeichen stehen müssen. Repräsentiert ein ? eine Datenbank-Spalte mit Text (z.B. varchar, text ...), wird dieser automatisch in Anführungszeichen gesetzt.

Da nun aber die Werte fehlen, müssen sie beim Aufruf von `execute()` in Form eines numerischen Arrays nachgereicht werden. Die Werte müssen in der Reihenfolge im Array stehen, wie sie in der SQL-Anfrage stehen würden. Sehen Sie, wie viel übersichtlicher der Code jetzt geworden ist?

3.3.2 Mehrfaches Ausführen von prepared statements

Ein weiterer Vorteil von *prepared statements*¹⁴ ist, dass sie mehrmals mit `execute()` ausgeführt werden können. Sie müssen das eigentliche SQL nur einmal schreiben und vorbereiten und können es mit verschiedenen Parametern so oft Sie wollen wiederverwenden.

Beispiel

```
1 <?php
```

14. und ihr ursprünglicher Zweck

```

2 $sql = 'UPDATE benutzer SET name=?, passwort=?, email=?';
3 $abfrage = $db->prepare($sql);
4
5 $abfrage->execute(array('emrich', 'sehrgeheim', 'emrich@example.com'));
6 $abfrage->execute(array('remolt', 'auchgeheim', 'remolt@example.com'));
7 $abfrage->execute(array('schneider', 'pssst', 'schneider@example.com'));
8 ?>

```

Listing 3-16: prepared_statements.php (Version 3)

Wir haben auf dasselbe *prepared statement* drei Mal die Methode `execute()` mit verschiedenen Werten aufgerufen. Das ist dasselbe, wie drei Mal ein `query()` mit den kompletten Anweisungen abzusetzen, nur kürzer, und wir gewinnen als Bonus sogar noch ein wenig Geschwindigkeit¹⁵.

Die erhöhte Lesbarkeit des Codes sollte jedoch für Sie der wichtigste Grund sein.

 Verwenden Sie *prepared statements* immer, wenn Sie Variablen in die Anweisung einbauen oder eine Anweisung mehrfach ausführen müssen. Für alle anderen Fälle ist ein einfaches `query()` genauso geeignet.

3.3.3 Limitierungen von Platzhaltern in prepared statements

Sie können Platzhalter nur auf der Seite der Werte verwenden, niemals als Ersatz für einen Tabellen- oder Spaltennamen. Da der Ausdruck schon während der Methode `prepare()` auf seine Korrektheit überprüft wird¹⁶, müssen diese Informationen schon vorliegen und können nicht erst später eingefügt werden.

Beispiel

```

1 <?php
2   $db = new PDO('mysql:host=localhost;dbname=mysql', 'root', '');
3   $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_WARNING);
4
5   // das ist OK
6   $abfrage = $db->prepare('SELECT * FROM user WHERE Host=?');
7   $abfrage->execute(array('localhost'));
8
9   // das funktioniert nicht
10  $abfrage = $db->prepare('SELECT * FROM ? WHERE Host=?');
11  $abfrage->execute(array('user', 'localhost'));
12 ?>

```

Listing 3-17: prepared_statements.php (Version 4)

MySQL erzeugt zwar erfahrungsgemäß nicht immer die korrekten Fehlermeldungen, aber Sie werden immer ein leeres Ergebnis erhalten, wenn Sie versuchen, Datensätze auszulesen.

3.3.4 Prepared statements mit benannten Platzhaltern

Vorteile

Statt den bereits bekannten Fragezeichen können Sie auch Namen als Parameter in *prepared statements* verwenden.

Dies hat mehrere Vorteile:

- **Benannte Platzhalter** machen die SQL-Anweisung noch lesbarer. Anstatt eines Fragezeichens steht tatsächlich da, was an dieser Stelle ersetzt werden soll.

15. *Prepared statements* mehrmals auszuführen, ist schneller als einzelne `query()`-Anweisungen. Dafür ist ein einzelner `query()` einen Tick schneller als die Kombination aus `prepare()` und `execute()`.

16. Ist die Syntax in Ordnung, existieren alle verwendeten Tabellen, existieren alle benannten Spalten ...

- Mit benannten Platzhaltern können Sie `execute()` auch assoziative Arrays übergeben. Gerade wenn die Daten schon in Form eines assoziativen Arrays vorliegen, ist es so nicht mehr notwendig, dieses Array auf ein numerisches umzubauen.
- Bei assoziativen Arrays ist natürlich auch die Reihenfolge der Einträge nicht mehr wichtig, nur noch die Namen der Schlüssel. Sie müssen also nicht mehr darauf achten, die Parameter im Array in der korrekten Reihenfolge zu halten. Das macht Ihren Code robuster gegen Fehler.

Anwendung

Benannte Platzhalter haben die Form `:platzhalter`, also z.B. `:vorname` oder `:email`. Der Doppelpunkt vor dem Namen ist für PDO das Zeichen, dass es kein gewöhnlicher String ist, sondern eben ein benannter Platzhalter. Sie dürfen ihn nicht weglassen.

Beispiel

```

1 <?php
2   $db = new PDO('mysql:host=localhost;dbname=phpschulung', 'root', '');
3   $abfrage = $db->prepare('INSERT INTO pretest (vorname, nachname, email)
4                           VALUES (:vorname, :nachname, :email)');
4 ?>

```

Listing 3-18: benannte_platzhalter1.php

Nun können Sie der Methode `execute()` ein assoziatives Array übergeben, deren Schlüssel so heißen, wie die Platzhalter ohne den Doppelpunkt¹⁷.

Beispiel

```

1 <?php
2   $db = new PDO('mysql:host=localhost;dbname=phpschulung', 'root', '');
3   $abfrage = $db->prepare('INSERT INTO pretest (vorname, nachname, email)
4                           VALUES (:vorname, :nachname, :email)');
4 ?>
5   $abfrage->execute(array('vorname' => 'Thorsten', 'nachname' =>
6                           'Schneider', 'email' => 't.schneider@example.com'));
6 ?>

```

Listing 3-19: benannte_platzhalter2.php

Falls Sie mehrere Datensätze haben und diese in Form eines mehrdimensionalen Arrays vorliegen, können Sie `execute()` in einer Schleife aufrufen.

Beispiel

```

1 <?php
2   $benutzer = array();
3   $benutzer[] = array('vorname' => 'Marc', 'nachname' => 'Remolt', 'email'
4                      => 'm.remolt@example.com');
4 ?>
5   $benutzer[] = array('vorname' => 'Marco', 'nachname' => 'Emrich', 'email'
6                      => 'm.emrich@example.com');
5 ?>
6   $benutzer[] = array('vorname' => 'Frank', 'nachname' => 'Schad', 'email'
7                      => 'f.schad@example.com');
6 ?>
7   $benutzer[] = array('vorname' => 'Thorsten', 'nachname' => 'Schneider',
8                      'email' => 't.schneider@example.com');
8 ?>
9   $db = new PDO('mysql:host=localhost;dbname=phpschulung', 'root', '');
9 ?>
10  $abfrage = $db->prepare('INSERT INTO pretest (vorname, nachname, email)
11                           VALUES (:vorname, :nachname, :email)');
10 ?>

```

17. Genau genommen würde PDO auch Schlüssel mit dem Doppelpunkt akzeptieren. So aussehende Schlüssel sind in PHP-Arrays aber unüblich.

```

11   foreach ($benutzer as $b) {
12     $abfrage->execute($b);
13   }
14 ?>

```

Listing 3-20: benannte_platzhalter3.php

Es ist Ihnen überlassen, mit welcher Art Platzhaltern Sie lieber arbeiten. Beide arbeiten zuverlässig und es existiert keine offiziell bevorzugte Variante.

3.4 PDO und Sicherheit

3.4.1 SQL-Injections

Unter den Begriff **SQL-Injection** versteht man das Einschleusen von SQL-Anweisungen in Ihren Code. Im schlimmsten Fall kann so ein Benutzer beliebige Anweisungen (z.B. `DELETE FROM ...`) auf Ihrem Server ausführen. Erschreckenderweise geht das leichter als Sie denken.

Betrachten Sie sich folgenden, harmlos aussehenden PHP-Block:

```

1 <?php
2   $db = new PDO('mysql:host=localhost;dbname=phpschulung', 'root', '');
3   $name = 'Marc';
4   $abfrage = $db->query('SELECT * FROM preptest WHERE vorname="'.$name.'"');
5   var_dump($abfrage->fetch());?>

```

Listing 3-21: sql_injection1.php

Dieser Code gibt Ihnen wie erwartet den ersten Datensatz mit dem Feld vorname gleich Marc zurück, da das erzeugt `SELECT * FROM preptest WHERE vorname="Marc"` war. Nehmen wir an, die Variable \$name haben wir nicht von Hand belegt, sondern Sie stammt aus einer Benutzereingabe. Statt seinem Namen hat der bösartige Benutzer aber folgenden Code eingegeben:

`Marc"; DELETE FROM preptest; --`

Wenn dieser Inhalt in das SQL eingefügt wird, erhalten wir also folgendes SQL-Statement:

`SELECT * FROM preptest WHERE vorname="Marc"; DELETE FROM preptest; --"`

Wenn dieses SQL ausgeführt wird, ist der komplette Inhalt der Tabelle preptest gelöscht.

Was ist hier passiert? Der böse Benutzer hat geschickt eine zweite Abfrage in das SQL eingebaut, der die Tabelle leert. Da SQL mehrere Anweisungen auf einer Zeile zulässt, so lange diese durch ein Semikolon getrennt sind, ist das an sich kein Problem. Das letzte, schließende Anführungszeichen wurde durch das doppelte Minus (das SQL-Kommentarzeichen) ungültig gemacht.

Statt einer werden nun also zwei SQL-Anfragen ausgeführt, wobei die zweite quasi beliebigen Code enthalten kann. So sind das Verändern von Admin-Passwörtern, das Anlegen von neuen Benutzern oder das Löschen bestimmter ungeliebter Datensätze prinzipiell kein Problem. Mit etwas Übung¹⁸ können Sie auf diese Weise sogar das Passwort des MySQL-Root-Users verändern.

Die einzige Möglichkeit dieses Problem zu vermeiden, ist die Benutzereingaben zu überprüfen und gegebenenfalls zu korrigieren. Hier gibt es viele Möglichkeiten, eine recht einfache Methode ist es, jede Benutzereingabe mit der PHP-Funktion `addslashes()` zu maskieren. Jedem potentiell gefährlichen Zeichen, insbesondere die Anführungszeichen, wird so automatisch ein Backslash (\) vorangestellt.

Beispiel

```

1 <?php
2   $name = 'Marc"; DELETE FROM preptest; --';
3   $name = addslashes($name);

```

18. Natürlich nur, wenn der Administrator die Berechtigungen in der Datenbank nicht korrekt gesetzt hat.

```

4 echo $name;
5 ?>
```

Listing 3-22: addslashes.php

Die Ausgabe dieses Skriptes lautet:

```
Marc\"; DELETE FROM preptest; --
```

Das gefährliche Anführungszeichen wurde also maskiert und das SQL ist nun nicht mehr gültig. Folgende Version des Beispiels von gerade ist also sicher gegen *SQL-Injections*:

Beispiel

```

1 <?php
2   $db = new PDO('mysql:host=localhost;dbname=phpschulung', 'root', '');
3   $name = 'Marc"; DELETE FROM preptest; --';
4   $abfrage = $db->query('SELECT * FROM preptest WHERE vorname="' .
5     addslashes($name) . '"');
6   var_dump($abfrage->fetch());
```

Listing 3-23: sql_injection1.php

Natürlich müssen Sie die Funktion bei jeder Benutzereingabe in Ihrem SQL einbauen. Wenn Sie `addslashes()` an einer Stelle vergessen, ist Ihr Code angreifbar. Gerade an diesem Problem kranken viele ältere PHP-Projekte, die sich um derartige Probleme erst seit relativ kurzer Zeit kümmern.

3.4.2 PDO und SQL-Injections

Jetzt kommt die gute Nachricht: So lange Sie *prepared statements* verwenden, müssen Sie sich um *SQL-Injections* keine Sorgen machen. Sämtliche potentiell gefährlichen Anweisungen werden automatisch maskiert und sind somit ungefährlich.

Daher sollten Sie immer *prepared statements* einsetzen, wenn Sie PHP-Variablen in Ihrem SQL einbauen. So ist Ihre Webanwendung von dieser Seite aus nicht angreifbar und der Aufwand für diesen erheblichen Sicherheitsgewinn ist minimal.

Aufgaben zur Selbstkontrolle

Fragen

1. Was müssen Sie in PHP tun, um standardmäßig jeden Fehler den MySQL liefert, zu sehen?
2. Welchen Vorteil haben *buffered queries*?
3. Welche Schritte sind notwendig, um von PHP aus Unicode-Daten in MySQL verwalten zu können?
4. Wie unterscheiden sich *prepared statements* von herkömmlichen SQL-Anweisungen?
5. Warum sind *prepared statements* sicherer als herkömmliche SQL-Anweisungen?
6. Was müssen Sie tun, um `execute()` ein assoziatives Array übergeben zu können?

Übungen

1. Schreiben Sie das Skript *eintragen.php* aus Lektion 2 »Einführung in PDO« um, so dass, wo sinnvoll, statt der `query()`-Methode *prepared statements* verwendet werden.

Optionale Übungen

2. Schreiben Sie ein neues Skript *regisseur_neu.php*, das ein HTML-Formular anzeigt, mit dem ein neuer Regisseur angelegt werden kann. Dieses Formular wird an eine zweite Datei *regisseur_eintragen.php* versendet.

3. Das Skript *regisseur_eintragen.php* verwendet die Daten des Formulars aus Übung 2 und trägt einen neuen Datensatz in die Tabelle *regisseure* ein. Verwenden Sie hierfür *prepared statements*, um die Benutzereingaben sicher zu halten.

4

PDO in der Praxis

In dieser Lektion lernen Sie:

- das CRUD nichts Unanständiges ist.
- wie Sie mit PDO praktisch Webanwendungen umsetzen.

4.1 CRUD

4.1.1 Konzept

In dieser letzten Lektion werden wir das bisher Gelernte praktisch anwenden, indem wir ein kleines, datenbankbasiertes Projekt entwickeln werden. Es ist an sich nichts Besonderes, eine kleine Personen-Verwaltung, zeigt aber sehr schön ein wichtiges Konzept, wie man in Programmen mit persistenten Daten umgeht.

Da in der Programmierung so ziemlich jede Technologie und Vorgehensweise mit einer Abkürzung benannt ist, wundert es Sie sicher nicht, dass dies auch hier der Fall ist. **CRUD** steht für die einzelnen Wörter *create* (erzeugen), *read* (lesen/ansehen), *update* (aktualisieren) und *delete* (löschen) und bezeichnet die einzelnen Operationen, die auf persistente Daten möglich sind.

- **CREATE** bezeichnet das erste Eintragen eines Datensatzes in dem Speichermedium, wobei es sich in unserem Fall um eine Datenbank handelt. Im SQL wird diese Aktion mit dem `INSERT`-Ausdruck ausgeführt.
- **READ** bezeichnet das Auslesen von einem oder auch mehreren Datensätzen aus der Datenbank. Im SQL wird für diese Aktion das `SELECT`-Statement verwendet.
- **UPDATE** bezeichnet das Aktualisieren von bereits bestehenden Datensätzen und wird im SQL vom `UPDATE`-Statement repräsentiert.
- **DELETE** schließlich bezeichnet das Entfernen von Datensätzen aus dem permanenten Speichermedium und entspricht im SQL dem `DELETE`.

Diese vier Aktionen umfassen im Grunde alles, was Sie in Ihren Programmen mit persistenten Daten machen können. Sicher hängt das konkrete SQL von der Struktur Ihrer Tabellen ab und auch die Darstellung der Daten wird sich von Anwendung zu Anwendung unterscheiden, aber im Prinzip können Sie alles auf diese vier Aktionen reduzieren.

4.1.2 Vorbereitungen

Legen Sie sich in Ihrem Webspace einen Ordner mit dem Namen `pdo_projekt` an, in diesem einen Ordner `inc`, und dort eine Datei namens `funktionen.inc.php`. Sie werden zwar für dieses Projekt nicht viele Funktionen benötigen, aber es schadet nie, von Anfang an sauber und strukturiert zu arbeiten.

Funktionen

Genau genommen werden wir im Moment nur eine Funktion benötigen, denn es gibt nur eine Sache, die Sie in jeder PHP-Seite ausführen müssen, nämlich die Datenbank-Verbindung aufzubauen. Dementsprechend enthält die Datei die Funktion `hole_datenbank()`.

Beispiel

```
1 <?php
2     function hole_datenbank()
3     {
```

```

4      $optionen = array(
5          PDO::ATTR_PERSISTENT                         => true,
6          PDO::MYSQL_ATTR_USE_BUFFERED_QUERY           => true,
7          PDO::ATTR_ERRMODE                           => PDO::ERRMODE_EXCEPTION,
8          PDO::ATTR_DEFAULT_FETCH_MODE                => PDO::FETCH_ASSOC
9      );
10     $db = new PDO('mysql:host=localhost;dbname pdo_projekt', 'root', '',
11                  $optionen);
12     $db->query('SET NAMES utf8');
13 }
14 ?>

```

Listing 4-1: inc/funktionen.inc.php

Diese Funktion erzeugt ein PDO-Objekt und gibt es anschließend zurück. Zusätzlich werden an dieser Stelle sämtliche Anpassungen vorgenommen, die wir für die Datenbank-Verbindung vornehmen wollen:

- Um Ressourcen zu sparen, wird in Zeile 5 auf persistente Datenbank-Verbindungen umgeschaltet.
- In Zeile 6 werden die gepufferten MySQL-Abfragen eingeschaltet. In unserem simplen Beispiel werden wir zwar in das Problem nicht hineinlaufen, aber auf diese Weise müssen wir uns später darüber keine Gedanken mehr machen.
- Zeile 7 bewirkt, dass wir die MySQL-Fehler auch sehen und das PHP-Programm mit einem Fehler abbricht, wenn von der Datenbank ein Problem gemeldet wird.
- Da wir ausschließlich mit assoziativen Arrays arbeiten werden, wird in Zeile 8 der Fetch-Modus auf eben diese eingestellt.
- In Zeile 11 schließlich wird die Datenbank-Verbindung mit SET NAMES auf Unicode umgeschaltet.

Datenbank-Struktur

Damit sind wir gerüstet, die Datenbank-Struktur anzulegen. Legen Sie im Projekt-Ordner eine Datei *installation.php* mit folgendem Inhalt an:

Beispiel

```

1  <?php
2  require_once 'inc/funktionen.inc.php';
3  $db = hole_datenbank();
4
5  $db->query('DROP TABLE IF EXISTS personen');
6  $sql = 'CREATE TABLE personen (' .
7      'id INT(5) AUTO_INCREMENT,' .
8      'vorname VARCHAR(255),' .
9      'nachname VARCHAR(255),' .
10     'email VARCHAR(255),' .
11     'PRIMARY KEY (id))' .
12     'DEFAULT CHARSET=utf8';
13 $db->query($sql);
14
15 $personen[] = array('Thorsten', 'Schneider', 'ts@example.com');
16 $personen[] = array('Marco', 'Emrich', 'me@example.com');
17 $personen[] = array('Marc', 'Remolt', 'mr@example.com');
18
19 $sql = 'INSERT INTO personen (vorname, nachname, email) VALUES (?, ?, ?)';
20 $abfrage = $db->prepare($sql);
21 foreach ($personen as $p) {
22     $abfrage->execute($p);
23 }
24 ?>
25 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
26 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
27 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="de" lang="de">
28 <head>

```

```

28 <title>Datenbank installieren</title>
29 <meta http-equiv="content-type" content="text/html; charset=utf-8" />
30 </head>
31 <body>
32 <h1>Datenbank installieren</h1>
33 <p>Die Tabellen wurden erfolgreich angelegt</p>
34 <p>
35 <a href="person_anlegen.php">Neue Person anlegen</a>
36 </p>
37 </body>
38 </html>

```

Listing 4-2: installation.php

- Nachdem die Datenbank-Verbindung aufgebaut ist, wird die Tabelle personen neu aufgebaut. Beachten Sie vor allem, wie in Zeile 12 die Tabelle auf UTF-8 umgestellt wird.
- Danach werden mit Hilfe eines *prepared statements* drei Datensätze eingefügt. Da Sie das execute() mehrfach ausführen können, wird an dieser Stelle elegant eine foreach-Schleife verwendet.
- Das anschließende HTML-Dokument teilt dem Benutzer nur mit, dass die Installation erfolgreich war und unter welchem Link er neue Personen anlegen kann.

Damit sind die Vorbereitungen im Prinzip abgeschlossen.

4.2 CREATE

4.2.1 Code

Kommen wir nun zum Anlegen von neuen Datensätzen in personen. Erzeugen Sie eine Datei *person_anlegen.php* mit folgendem Inhalt:

Beispiel

```

1 <?php
2   require_once 'inc/funktionen.inc.php';
3
4   if ($_POST) {
5     $db = hole_datenbank();
6     $sql = 'INSERT INTO personen (vorname, nachname, email) VALUES
7       (:vorname, :nachname, :email)';
8     $abfrage = $db->prepare($sql);
9     $abfrage->execute($_POST);
10
11    $nachricht = 'Person ' . $_POST['vorname'] . ' ' . $_POST['nachname'] .
12      ' wurde erfolgreich angelegt.';
13  }
14 ?>
15 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
16   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
17 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="de" lang="de">
18 <head>
19   <title>Person anlegen</title>
20   <meta http-equiv="content-type" content="text/html; charset=utf-8" />
21 </head>
22 <body>
23   <h1>Person anlegen</h1>
24   <?php if ($nachricht) { ?>
25     <p><?php echo $nachricht; ?></p>
26   <?php } ?>
27   <form action="<?php echo $_SERVER['PHP_SELF'] ?>" method="post">
28     <label for="vorname">Vorname:</label>
29     <input type="text" name="vorname" /><br />
30     <label for="nachname">Nachname:</label>

```

```
28      <input type="text" name="nachname" /><br />
29      <label for="email">E-Mail:</label>
30      <input type="text" name="email" /><br />
31      <input type="submit" value="anlegen" />
32  </form>
33  <p>
34  <a href="personen_ansehen.php">Personen ansehen</a>
35  </p>
36 </body>
37 </html>
```

Listing 4-3: person_anlegen.php

4.2.2 Erklärung

UTF-8-Encoding im HTML-Dokument

Um konsequent Unicode zu verwenden, ist es notwendig, dass auch der Browser weiß, dass er die Formulardaten UTF-8-codiert schicken muss. Dies teilen Sie ihm in Zeile 17 mit, indem Sie den Zeichensatz des Dokuments mit charset=utf-8 korrekt einstellen. Vergessen Sie auch nicht, in Ihrem Code-Editor den Zeichensatz der Datei passend einzustellen.

POST oder kein POST

Die selbe PHP-Datei wird verwendet, um das Eintragen-Formular anzuzeigen und auch, um es abzusenden. Würden wir die SQL-INSERT-Anweisung direkt in den PHP-Code schreiben, sprich, würde sie jedes Mal ausgeführt, hätten wir eine Menge leerer Datensätze in unserer Tabelle. Die INSERT-Anweisung soll nur dann ausgeführt werden, wenn das Formular abgesendet wurde.

Das können Sie leicht daran erkennen, dass die PHP-Seite mit POST geladen wurde, also das Array \$_POST nicht leer ist. Genau das prüfen wir in Zeile 4. Nur wenn die Bedingung erfüllt ist, wird die Datenbank-Verbindung aufgebaut und die Daten gespeichert.

Benannte Platzhalter und Namenskonventionen

In diesem Fall wurde bei der INSERT-Anweisung ein *prepared statement* mit benannten Platzhaltern verwendet. Warum ein *prepared statement* ist klar, denn die Benutzereingaben sollen nicht ungefiltert an unsere Datenbank übergeben werden¹⁹.

Die benannten Platzhalter bieten den Vorteil, dass wir direkt das assoziative Array \$_POST an unsere execute()-Methode übergeben können. Das funktioniert aber nur, wenn Sie die Platzhalter genau so nennen, wie die Input-Felder im HTML. Wenn das Input-Feld also den Namen nachname hat, dann muss der Platzhalter :nachname heißen.

Dem Benutzer Feedback geben

Wenn das Eintragen der Person erfolgreich war, wird noch eine Nachricht erzeugt, die unserem Benutzer zeigen soll, dass der Speichervorgang auch stattgefunden hat. Zu diesem Zweck wird eine Variable \$nachricht erzeugt und im HTML geprüft, ob sie existiert (Zeile 21). Ist das der Fall, wird die Nachricht ausgegeben.

4.3 READ

4.3.1 Code

Nun wollen wir sehen, ob die Datensätze auch wirklich in der Datenbank gelandet sind. Legen Sie eine Datei personen_ansehen.php mit folgendem Inhalt an:

19. Siehe Abschnitt 3.4.1 »SQL-Injektions«.

Beispiel

```

1 <?php
2   require_once 'inc/funktionen.inc.php';
3   $db      = hole_datenbank();
4   $abfrage = $db->query('SELECT * FROM personen');
5   $daten   = $abfrage->fetchAll();
6   $abfrage = null;
7 ?>
8 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
9   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
10 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="de" lang="de">
11 <head>
12   <title>Personen ansehen</title>
13   <meta http-equiv="content-type" content="text/html; charset=utf-8" />
14 </head>
15 <body>
16   <h1>Personen ansehen</h1>
17   <table>
18     <tr>
19       <th>Vorname</th>
20       <th>Nachname</th>
21       <th>E-Mail</th>
22       <th>&nbsp;</th>
23     </tr>
24     <?php foreach ($daten as $person) { ?>
25     <tr>
26       <td><?php echo $person['vorname'] ?></td>
27       <td><?php echo $person['nachname'] ?></td>
28       <td><?php echo $person['email'] ?></td>
29       <td>
30         <a href="person_bearbeiten.php?id=<?php echo $person['id']; ?>">
31           bearbeiten
32         </a>
33       </td>
34     </tr>
35   <?php } ?>
36 </table>
37 </body>
38 </html>
```

Listing 4-4: personen_ansehen.php

4.3.2 Erklärung

Datensätze auslesen

Im PHP-Teil des Dokuments passiert im Prinzip nichts Spektakuläres. Zu beachten ist, dass alle Datensätze mit `fetchAll()` auf einmal ausgelesen werden und die Abfrage danach geschlossen wird (Zeile 6), um die Ressourcen der Datenbank wieder frei zu machen.

Daten formatiert anzeigen

Da wir ein Array mit einem Datensatz pro Feld haben, bietet es sich an, dieses Array mit einer `foreach`-Schleife zu durchlaufen. Innerhalb des Datensatzes sprechen wir die einzelnen Spalten über den assoziativen Index des inneren Arrays an. Beachten Sie, dass wirklich nur die Daten selbst mit `echo`-Anweisungen ausgegeben werden. Alle HTML-Formatierungen wurden sauber außerhalb von PHP-Tags geschrieben, was die Lesbarkeit und Wartbarkeit des Codes erhöht.

Als Vorbereitung für den nächsten Schritt, das Bearbeiten von Personen, wird ein Link auf die URL `person_bearbeiten.php?id=xxx` angelegt, wobei `xxx` dem Inhalt der Spalte `id` des jeweiligen Datensatzes entspricht (Zeile 29).

4.4 UPDATE

4.4.1 Code

Nun wollen wir einen bereits existierenden Datensatz bearbeiten. Legen Sie zu diesem Zweck eine Datei `person_bearbeiten.php` mit folgendem Inhalt an:

Beispiel

```
1  <?php
2  require_once 'inc/funktionen.inc.php';
3  $db = hole_datenbank();
4
5  if ($_POST) {
6      $abfrage = $db->prepare('UPDATE personen SET vorname=:vorname,
nachname=:nachname, email=:email WHERE id=:id');
7      $abfrage->execute($_POST);
8      $nachricht = 'Person ' . $_POST['vorname'] . ' ' . $_POST['nachname'] .
' wurde erfolgreich bearbeitet.';
9  }
10 $abfrage = $db->prepare('SELECT * FROM personen WHERE id=?');
11 $abfrage->execute(array($_REQUEST['id']));
12 $person = $abfrage->fetch();
13 $abfrage = null;
14 ?>
15 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
16 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="de" lang="de">
17 <head>
18   <title>Person bearbeiten</title>
19   <meta http-equiv="content-type" content="text/html; charset=utf-8" />
20 </head>
21 <body>
22   <h1>Person bearbeiten</h1>
23   <?php if ($nachricht) { ?>
24     <p><?php echo $nachricht; ?></p>
25   <?php } ?>
26   <form action="<?php echo $_SERVER['PHP_SELF'] ?>" method="post">
27     <label for="vorname">Vorname:</label>
28     <input type="text" name="vorname" value="<?php echo $person['vorname'];
?>" /><br />
29     <label for="nachname">Nachname:</label>
30     <input type="text" name="nachname" value="<?php echo
$person['nachname']; ?>" /><br />
31     <label for="email">E-Mail:</label>
32     <input type="text" name="email" value="<?php echo $person['email'];
?>" /><br />
33     <input type="hidden" name="id" value="<?php echo $person['id'];
?>" />
34     <input type="submit" value="bearbeiten" />
35   </form>
36   <form action="person_loeschen.php" method="post">
37     <input type="hidden" name="id" value="<?php echo $person['id'];
?>" />
38     <input type="submit" value="löschen" />
39   </form>
40   <p>
41     <a href="personen_ansehen.php">zurück</a>
42   </p>
43 </body>
44 </html>
```

Listing 4-5: person_bearbeiten.php

4.4.2 Erklärung

Datensatz speichern

Für das Speichern der Formulardaten gelten hier die gleichen Regeln, wie beim CREATE. Wenn das Formular mit POST abgeschickt wurde, sollen die Daten gespeichert werden. Auch hier werden wieder benannte Platzhalter verwendet, um das \$_POST-Array direkt der execute()-Methode übergeben zu können.

Dieses Mal wird allerdings eine UPDATE-Anweisung verwendet, da ja ein bestehender Datensatz verändert werden soll. Vergessen Sie nicht die WHERE-Bedingung mit der id des Datensatzes, sonst verändern Sie alle Datensätze der Tabelle.

Woher die ID kommt

Wo wir schon beim Thema sind, wo kommt die ID des Datensatzes eigentlich her? Genau genommen erhalten Sie diese Information gleich aus zwei Quellen. Wenn Sie auf den Editieren-Link der Seite *personen_ansehen.php* geklickt haben, wurde die ID des Datensatzes mit übergeben (siehe Listing 4-4 Zeile 29).

Wenn Sie auf der aktuellen Seite das Formular abschicken, um den Datensatz zu verändern, muss die ID aus diesem Formular kommen, also wurde sie mit einem versteckten Feld übergeben (Zeile 33).

Also kommt die ID also einmal aus einem POST und einmal aus einem GET-Request.

Datensatz auslesen und anzeigen

Das stellt uns vor das Problem, dass wir beim Auslesen des passenden Datensatzes die ID entweder aus \$_GET['id'] oder aber aus \$_POST['id'] erhalten. Wir könnten nun eine if-else-Bedingung schreiben, die überprüft, welche Situation gerade zutrifft.

Erfreulicherweise nimmt uns PHP diese Mühe ab. Das Array \$_REQUEST verhält sich exakt so wie \$_GET und \$_POST, nur werden in diesem Array die Informationen aus beiden Request-Arten gespeichert. Mit anderen Worten finden Sie in \$_REQUEST['id'] immer den gewünschten Wert, egal ob er mit POST oder GET übertragen wurde. Also verwenden wir genau diese Variable in der execute()-Methode (Zeile 11).

Dieses mal verwenden wir nur fetch() und nicht fetchAll(), da wir ohnehin nur einen Datensatz erhalten werden und schließen daraufhin sofort wieder die Abfrage (Zeilen 12, 13).

Um das Formular mit den Werten aus der Datenbank zu befüllen, müssen Sie die value-Attribute der input-Tags mittels echo-Anweisungen befüllen (Zeilen 28, 30, 32).

Als Vorbereitung auf das Löschen von Datensätzen erzeugen wir ein weiteres Formular mit einem Submit-Button und einem versteckten Feld, durch das die ID übergeben wird (Zeilen 36-39). Die Mühe, ein eigenes Formular zu erstellen, anstatt einfach einen Link mit der ID zu erzeugen, machen wir uns aus Gründen der Sicherheit.

Wie Sie wissen, sollte eine Operation, die Daten in Ihrer Anwendung verändert, niemals mit GET abgesendet werden. Da Löschen definitiv eine Veränderung des Datenbestands darstellt, müssen wir einen POST-Request erzwingen. Also benötigen wir ein Formular.

4.5 DELETE

4.5.1 Code

Zuletzt implementierten wir die Funktion, einen Datensatz wieder aus der Datenbank zu entfernen. Erstellen Sie eine Datei *person_loeschen.php* mit folgendem Inhalt:

Beispiel

1 <?php

```
2  require_once 'inc/funktionen.inc.php';
3  if ($_POST) {
4      $db      = hole_datenbank();
5      $abfrage = $db->prepare('DELETE FROM personen WHERE id=?');
6      $abfrage->execute(array($_POST['id']));
7
8      $nachricht = 'Die Person wurde erfolgreich gelöscht.';
9  }
10 ?>
11 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
12   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
13 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="de" lang="de">
14 <head>
15   <title>Person löschen</title>
16   <meta http-equiv="content-type" content="text/html; charset=utf-8" />
17 </head>
18 <body>
19   <h1>Person löschen</h1>
20   <?php if ($nachricht) { ?>
21     <p><?php echo $nachricht; ?></p>
22   <?php } ?>
23   <p>
24     <a href="personen_ansehen.php">zurück</a>
25   </p>
26 </body>
27 </html>
```

Listing 4-6: person_loeschen.php

4.5.2 Erklärung

Hier passiert im Prinzip nichts Neues mehr. Auch hier wird die SQL-Anweisung, in diesem Fall `DELETE FROM`, nur dann ausgeführt, wenn die Daten per POST versendet wurden. Dann wird per *prepared statement* der Datensatz mit der passenden ID gelöscht und danach eine Erfolgsmeldung erzeugt. Das folgende HTML gibt eigentlich nur die Nachricht aus, wenn eine erzeugt wurde.

Aufgaben zur Selbstkontrolle

Übungen

1. Erstellen Sie das in dieser Lektion vorgestellte Projekt. Überlegen Sie, was man noch verbessern könnte.

Optionale Übungen

2. Erstellen Sie in dem Projekt eine zweite Tabelle `adressen` mit den Spalten `id`, `strasse`, `plz`, `ort` und `person_id`, die in einer 1-n-Beziehung zu der Tabelle `personen` steht. Die Spalte `person_id` ist der Fremdschlüssel zu der Tabelle `personen`.
3. Programmieren Sie eine Verwaltung für die Tabelle `adressen`, halten Sie sich dabei an das CRUD-Prinzip. Überlegen Sie, wie Sie den Fremdschlüssel `person_id` im HTML-Code optisch am elegantesten darstellen.
4. Falls Sie das Weblog-Projekt aus dem Anhang des Lernhefts *Grundlagen der PHP-Programmierung* umgesetzt haben, schreiben Sie das Projekt von Datei-basiert auf Datenbank-basiert um. Verwenden Sie, wo immer sinnvoll, *prepared statements* für Ihr SQL.

Lösungen

2 »Einführung in PDO«

➤ Fragen

1. Welchen Datentyp hat in PHP eine Datenbank-Verbindung mit PDO?
→ Es handelt sich um Objekte der Klasse PDO.
2. Wie können Sie mit PDO eine SQL-Anweisung ausführen?
→ Mit der Methode query() des PDO-Objekts.
3. Auf welche Arten können Sie Datensätze aus einem PDO-Ergebnis auslesen?
→ Mit den Methoden fetch() und fetchAll().
4. Wann müssen Sie bei SQL INSERT, wann UPDATE verwenden?
→ INSERT verwenden Sie, um einen Eintrag neu anzulegen, UPDATE um einen vorhandenen Datensatz zu verändern.
5. Wie können Sie alle Datensätze einer Tabelle löschen?
→ DELETE FROM tabellename ohne eine WHERE-Bedingung.
→ DELETE FROM tabellename mit einer WHERE-Bedingung, die auf alle Datensätze zutrifft.
→ TRUNCATE TABLE tabellename.
6. Wie können Sie eine Tabelle löschen?
→ Mit DROP TABLE tabellename.

3 »Fortgeschrittene PDO-Funktionen«

➤ Fragen

1. Was müssen Sie in PHP tun, um standardmäßig jeden Fehler den MySQL liefert, zu sehen?
→ Sie müssen das PDO-Attribut PDO::ATTR_ERRMODE entweder auf PDO::ERRMODE_WARNING oder auf PDO::ERRMODE_EXCEPTION setzen.
2. Welchen Vorteil haben buffered queries?
→ Sie können mehrere SQL-Abfragen gleichzeitig offen halten.
3. Welche Schritte sind notwendig, um von PHP aus Unicode-Daten in MySQL verwalten zu können?
→ Sie müssen bei der SQL-Anweisung CREATE TABLE am Ende mit der Option DEFAULT CHARSET=utf8 die Tabelle auf UTF-8 umstellen.
→ Sie müssen nach dem Verbindungsaufbau mit der SQL-Anweisung SET NAMES utf8 die Verbindung auf UTF-8 umstellen.
4. Wie unterscheiden sich prepared statements von herkömmlichen SQL-Anweisungen?
→ Sie können mehrfach verwendet werden.
→ Sie können Platzhalter (?) oder (:name) enthalten.
→ Strings in Platzhaltern werden automatisch in Anführungszeichen gesetzt.