

*Dynamische Webseiten erstellen*

**2. Auflage**  
**Behandelt PHP 5**



*Programmieren mit*

**PHP**

**O'REILLY®**

*Rasmus Lerdorf, Kevin Tatroe & Peter MacIntyre*

*Übersetzung von Peter Klicman & Lars Schulten*

2. AUFLAGE

---

# Programmieren mit PHP

*Rasmus Lerdorf, Kevin Tatroe  
und Peter MacIntyre*

*Deutsche Übersetzung von  
Lars Schulten & Peter Klicman*

**O'REILLY®**

Beijing · Cambridge · Farnham · Köln · Paris · Sebastopol · Taipei · Tokyo

Die Informationen in diesem Buch wurden mit größter Sorgfalt erarbeitet. Dennoch können Fehler nicht vollständig ausgeschlossen werden. Verlag, Autoren und Übersetzer übernehmen keine juristische Verantwortung oder irgendeine Haftung für eventuell verbliebene Fehler und deren Folgen.

Alle Warennamen werden ohne Gewährleistung der freien Verwendbarkeit benutzt und sind möglicherweise eingetragene Warenzeichen. Der Verlag richtet sich im Wesentlichen nach den Schreibweisen der Hersteller. Das Werk einschließlich aller seiner Teile ist urheberrechtlich geschützt. Alle Rechte vorbehalten einschließlich der Vervielfältigung, Übersetzung, Mikroverfilmung sowie Einspeicherung und Verarbeitung in elektronischen Systemen.

Kommentare und Fragen können Sie gerne an uns richten:

O'Reilly Verlag  
Balthasarstr. 81  
50670 Köln  
Tel.: 0221/9731600  
Fax: 0221/9731608  
E-Mail: kommentar@oreilly.de

Copyright der deutschen Ausgabe:  
© 2007 by O'Reilly Verlag GmbH & Co. KG  
1. Auflage 2003  
2. Auflage 2007

Die Originalausgabe erschien 2006 unter dem Titel  
*Programming PHP, 2nd Edition* bei O'Reilly Media, Inc.

Die Darstellung eines Kuckucks im Zusammenhang mit dem  
Thema PHP ist ein Warenzeichen von O'Reilly Media, Inc.

Bibliografische Information Der Deutschen Bibliothek  
Die Deutsche Bibliothek verzeichnet diese Publikation in der  
Deutschen Nationalbibliografie; detaillierte bibliografische Daten  
sind im Internet über <http://dnb.ddb.de> abrufbar.

Übersetzung und deutsche Bearbeitung: Lars Schulten, Köln & Peter Klicman, Köln  
Lektorat: Alexandra Follenius, Köln  
Korrektorat: Oliver Mosler, Köln  
Satz: G&U Language & Publishing Services GmbH, Flensburg  
Umschlaggestaltung: Ellie Volckhausen, Boston & Michael Oreal, Köln  
Produktion: Andrea Miß, Köln  
Belichtung, Druck und buchbinderische Verarbeitung:  
Druckerei Kösel, Krugzell; [www.koeselbuch.de](http://www.koeselbuch.de)

ISBN-10 3-89721-473-3  
ISBN-13 978-3-89721-473-6

Dieses Buch ist auf 100% chlorfrei gebleichtem Papier gedruckt.

# Inhalt

<b>Vorwort</b> .....	<b>IX</b>
<b>Einleitung</b> .....	<b>XI</b>
<b>1 Einführung in PHP</b> .....	<b>1</b>
Was macht PHP? .....	1
Eine kurze Geschichte von PHP .....	2
PHP installieren .....	8
Ein Rundgang durch PHP .....	10
<b>2 Grundlagen der Sprache</b> .....	<b>18</b>
Lexikalische Struktur .....	18
Datentypen .....	25
Variablen .....	33
Ausdrücke und Operatoren .....	38
Flusssteuerungsanweisungen .....	51
Code einbinden .....	60
Einbetten von PHP in Webseiten .....	62
<b>3 Funktionen</b> .....	<b>67</b>
Aufruf einer Funktion .....	67
Definition einer Funktion .....	68
Geltungsbereich von Variablen .....	71
Funktionsparameter .....	73
Rückgabewerte .....	76
Variablenfunktionen .....	77
Anonyme Funktionen .....	78

<b>4</b>	<b>Strings</b> .....	<b>79</b>
	Quoting von Stringkonstanten .....	79
	Ausgabe von Strings .....	83
	Zugriff auf einzelne Zeichen .....	87
	Strings säubern .....	88
	Codierung und Escaping .....	89
	Strings vergleichen .....	95
	Durchsuchen und Bearbeiten von Strings .....	98
	Reguläre Ausdrücke .....	105
	Reguläre Ausdrücke im POSIX-Stil .....	109
	Perl-kompatible reguläre Ausdrücke .....	114
<b>5</b>	<b>Arrays</b> .....	<b>128</b>
	Indizierte und assoziative Arrays .....	128
	Elemente eines Arrays identifizieren .....	129
	Daten in Arrays speichern .....	130
	Mehrdimensionale Arrays .....	132
	Mehrere Werte extrahieren .....	133
	Konvertierung zwischen Arrays und Variablen .....	137
	Arrays durchlaufen .....	138
	Sortierung .....	144
	Arbeiten mit vollständigen Arrays .....	149
	Arrays nutzen .....	151
<b>6</b>	<b>Objekte</b> .....	<b>154</b>
	Terminologie .....	155
	Ein Objekt erzeugen .....	156
	Zugriff auf Eigenschaften und Methoden .....	157
	Deklaration einer Klasse .....	158
	Introspektion .....	165
	Serialisierung .....	172
<b>7</b>	<b>Webtechniken</b> .....	<b>176</b>
	HTTP-Grundlagen .....	176
	Variablen .....	178
	Serverinformationen .....	179
	Verarbeitung von Formularen .....	180
	Response-Header setzen .....	194
	Zustände festhalten .....	198
	SSL .....	209

<b>8 Datenbanken</b> .....	<b>210</b>
PHP für den Zugriff auf Datenbanken .....	210
Relationale Datenbanken und SQL .....	212
PEAR DB-Grundlagen .....	214
Fortgeschrittene Datenbank-Techniken .....	220
Beispielanwendung .....	226
<b>9 Grafiken</b> .....	<b>241</b>
Ein Image in eine Seite einbetten .....	241
Die GD-Erweiterung .....	242
Grundlegende Grafikkonzepte .....	243
Images erzeugen und zeichnen .....	244
Images mit Text .....	249
Dynamisch generierte Buttons .....	252
Skalierung von Images .....	256
Farben .....	257
<b>10 PDF</b> .....	<b>263</b>
PDF-Erweiterungen .....	263
Dokumente und Seiten .....	263
Text .....	266
<b>11 XML</b> .....	<b>279</b>
Ein Blitzeinstieg in XML .....	279
XML generieren .....	281
XML parsen .....	283
XML mit DOM parsen .....	295
XML mit SimpleXML parsen .....	296
XML-Transformation mit XSLT .....	297
Web Services .....	299
<b>12 Sicherheit</b> .....	<b>304</b>
Eingaben filtern .....	304
Ausgaben maskieren .....	309
Cross-Site-Scripting .....	313
Session-Fixierung .....	314
Datei-Uploads .....	315
Dateizugriff .....	316
PHP-Code .....	320

Shell-Befehle .....	321
Mehr Informationen .....	322
Sicherheit kurz gefasst .....	322
<b>13 Anwendungstechniken .....</b>	<b>323</b>
Code-Bibliotheken .....	323
Template-Systeme .....	324
Verarbeitung von Ausgaben .....	328
Fehlerbehandlung .....	330
Performance-Tuning .....	336
<b>14 PHP erweitern .....</b>	<b>345</b>
Die Architektur im Überblick .....	345
Was Sie brauchen .....	346
Ihre ersten Erweiterungen erzeugen .....	348
Die Datei config.m4 .....	358
Speicherverwaltung .....	361
Der pval/zval-Datentyp .....	363
Umgang mit Parametern .....	368
Rückgabe von Werten .....	371
Referenzen .....	375
Globale Variablen .....	376
Variablen erzeugen .....	379
INI-Einträge für Erweiterungen .....	380
Ressourcen .....	382
Wie geht es weiter? .....	384
<b>15 PHP unter Windows .....</b>	<b>385</b>
Installation und Konfiguration von PHP unter Windows .....	385
Portablen Code für Windows und Unix entwickeln .....	389
Interfacing mit COM .....	392
Interaktion mit ODBC-Datenquellen .....	399
<b>A Funktionsreferenz .....</b>	<b>404</b>
<b>B Übersicht über die Erweiterungen .....</b>	<b>522</b>
<b>Index .....</b>	<b>535</b>

## KAPITEL 8

# Datenbanken

PHP unterstützt über 20 Datenbanken, einschließlich der populärsten kommerziellen und Open Source-Varianten. Relationale Datenbank-Systeme wie MySQL, PostgreSQL und Oracle bilden das Rückgrat der meisten modernen dynamischen Websites. In diesen werden Warenkorb-Informationen, Verkaufshistorien, Produktbesprechungen, Benutzerinformationen, Kreditkartennummern und manchmal sogar die Webseiten selbst abgelegt.

Dieses Kapitel beschreibt, wie man aus PHP heraus auf Datenbanken zugreift. Wir konzentrieren uns auf das PEAR DB-System, das es uns ermöglicht, die gleichen Funktionen für den Zugriff auf die verschiedensten Datenbanken zu verwenden. Auf diese Weise müssen wir uns nicht mit den vielen datenbankspezifischen Erweiterungen auseinandersetzen. In diesem Kapitel lernen Sie, wie man Daten aus einer Datenbank einliest, wie man Daten in einer Datenbank speichert und wie man Fehler behandelt. Wir schließen das Kapitel mit einer Beispielanwendung ab, die die verschiedenen Datenbank-Techniken in Aktion zeigt.

Dieses Buch kann nicht alle Details zur Entwicklung datenbankbasierter Webanwendungen mit PHP abdecken. Eine detaillierte Beschreibung der Zusammenarbeit von PHP und MySQL finden Sie in *Webdatenbank-Applikationen mit PHP und MySQL* von Hugh Williams und David Lane (O'Reilly Verlag).

## PHP für den Zugriff auf Datenbanken

Es gibt zwei Möglichkeiten, um aus PHP heraus auf Datenbanken zuzugreifen. Die eine ist die Verwendung einer datenbankspezifischen Erweiterung, die andere die Verwendung der datenbankunabhängigen PEAR DB-Bibliothek. Jeder Ansatz hat seine eigenen Vor- und Nachteile.

Wenn Sie eine datenbankspezifische Erweiterung verwenden, ist Ihr Code unmittelbar an die von Ihnen verwendete Datenbank gebunden. Funktionsnamen, Parameter und Fehlerbehandlung der MySQL-Erweiterung unterscheiden sich völlig von denen anderer Datenbank-Erweiterungen. Möchten Sie Ihre Datenbank von MySQL auf PostgreSQL verlagern, führt das zu signifikanten Änderungen an Ihrem Code. Die PEAR DB versteckt dagegen die datenbankspezifischen Funktionen vor Ihnen. Der Wechsel zwischen zwei Datenbanken kann so einfach sein wie das Austauschen einer Zeile in Ihrem Programm.

Die Portabilität einer Abstraktionsschicht wie der PEAR DB-Bibliothek hat ihren Preis. Spezifische Features einer bestimmten Datenbank (zum Beispiel das Auffinden des Wertes eines automatisch zugewiesenen eindeutigen Zeilen-Identifiers) stehen nicht zur Verfügung. Code, der die PEAR DB nutzt, ist üblicherweise auch ein wenig langsamer als Code, der eine datenbankspezifische Erweiterung nutzt.

Denken Sie daran, dass eine Abstraktionsschicht wie PEAR DB absolut nichts ausrichten kann, wenn es darum geht sicherzustellen, dass die eigentlichen SQL-Queries portabel sind. Wenn Ihre Anwendung irgendeine Art nicht generisches SQL verwendet, müssen Sie viel Arbeit investieren, um Ihre Queries, die Sie für eine Datenbank geschrieben haben, auf einer anderen nutzen zu können. Bei großen Anwendungen sollten Sie über die Entwicklung einer funktionalen Abstraktionsschicht nachdenken, d.h., für jede Datenbank, die Ihre Anwendung unterstützen muss, entwickeln Sie eine Reihe von Funktionen, die verschiedene Datenbank-Aktionen durchführen, etwa `get_user_record()`, `insert_user_record()` und was sonst noch so notwendig ist. Verwenden Sie dann eine Konfigurationsoption, die den Datenbank-Typ festlegt, mit dem Ihre Anwendung die Verbindung herstellt. Dieser Ansatz lässt Sie alle Spezialitäten jeder gewählten Datenbank verwenden, ohne die Geschwindigkeitseinbußen und Beschränkungen einer Abstraktionsschicht hinnehmen zu müssen. Es würde allerdings eine ganze Weile dauern, wollte man das von null aufbauen.

Bei einfachen Anwendungen ziehen wir PEAR DB den datenbankspezifischen Erweiterungen vor, und zwar nicht nur wegen der Portabilität, sondern auch wegen der einfachen Handhabung. Der Preis für die Geschwindigkeit und die Features ist selten hoch genug, um uns dazu zu zwingen, eine datenbankspezifische Erweiterung zu nutzen. Größtenteils enthält der Rest dieses Kapitels Beispielcode, der die PEAR DB-Abstraktionsobjekte nutzt.

Für die meisten Datenbanken müssen Sie PHP mit den entsprechenden Datenbank-Treibern neu kompilieren. Dabei spielt es keine Rolle, ob Sie die PEAR DB-Bibliothek nutzen oder nicht. Die Hilfsinformationen des `configure`-Befehls in der PHP-Quelldistribution liefern Angaben darüber, wie PHP kompiliert werden muss, um die verschiedenen Datenbanken zu unterstützen. Hier sehen Sie einige Beispiele:

```
--with-mysql[=DIR]    Include MySQL support. DIR is the MySQL base
                      directory. If unspecified, the bundled MySQL
                      library will be used.
--with-oci8[=DIR]     Include Oracle-oci8 support. Default DIR is
                      ORACLE_HOME.
```

```
--with-ibm-db2[=DIR]  Include IBM DB2 support.  DIR is the DB2 base
                       install directory, defaults to
                       /home/db2inst1/sqllib.
--with-pgsql[=DIR]    Include PostgreSQL support.  DIR is the PostgreSQL
                       base install directory, defaults to
                       /usr/local/pgsql.
```

PHP kann nicht für die Unterstützung einer Datenbank kompiliert werden, deren Client-Bibliotheken nicht auf Ihrem System vorliegen. Wenn Sie beispielsweise die Oracle-Client-Bibliotheken nicht vorliegen haben, können Sie kein PHP kompilieren, das Oracle-Datenbanken unterstützt.

Verwenden Sie die Funktion `phpinfo()`, um zu prüfen, welche Datenbanken von Ihrer PHP-Installation unterstützt werden. Wenn Sie im Konfigurationsbericht zum Beispiel einen Abschnitt über MySQL sehen, dann wissen Sie, dass MySQL unterstützt wird.

Neu in PHP Version 5 ist die kompakte und kleine Datenbankbindung namens SQLite. Wie der Name andeutet, handelt es sich um ein kleines und leichtgewichtiges Datenbankwerkzeug. Dieses Datenbankprodukt kommt mit PHP 5 und hat das bisherige Default-Datenbankwerkzeug MySQL ersetzt. Sie können MySQL immer noch mit PHP verwenden, aber Sie müssen etwas Arbeit investieren, um es einzurichten. SQLite ist sofort einsatzbereit, wenn Sie PHP installieren. Wenn Sie also nach einem leichtgewichtigen und kompakten Datenbankwerkzeug suchen, sollten Sie sich unbedingt über SQLite informieren.

## Relationale Datenbanken und SQL

Ein Relationales Datenbank Management System (RDBMS) ist ein Server, der Daten für Sie verwaltet. Die Daten sind in Tabellen strukturiert, wobei jede Tabelle eine Reihe von Spalten besitzt, von denen jede einen Namen und einen Typ aufweist. Um zum Beispiel Science-Fiction-Bücher festzuhalten, könnten wir eine »books«-Tabelle anlegen, die den Titel (einen String), das Jahr der Veröffentlichung (eine Zahl) und den Autor enthält.

Tabellen werden zu Datenbanken gruppiert, d.h., eine Science-Fiction-Buch-Datenbank könnte Tabellen für Zeitperioden, Autoren und die Schurken besitzen. Ein RDBMS hat normalerweise ein eigenes Benutzersystem, das die Zugriffsrechte für Datenbanken kontrolliert (z.B. »Benutzer Fred kann die Datenbank Autoren aktualisieren«).

PHP kommuniziert mit relationalen Datenbanken wie MySQL und Oracle über die so genannte *Structured Query Language* (kurz SQL). Sie können SQL zum Anlegen, Modifizieren und Abfragen relationaler Datenbanken verwenden.

Die Syntax für SQL teilt sich in zwei Bereiche auf. Der erste, die so genannte *Data Manipulation Language* oder kurz DML, wird verwendet, um Daten aus einer existierenden Datenbank einzulesen oder zu modifizieren. Die DML ist bemerkenswert kompakt und be-

steht nur aus vier (englischen) Verben: SELECT, INSERT, UPDATE und DELETE. Der Satz von SQL-Befehlen, der zur Erzeugung und Modifikation der Strukturen verwendet wird, die die Daten enthalten, wird *Data Definition Language* oder DDL genannt. Die Syntax der DDL ist nicht so standardisiert wie diejenige für die DML, aber PHP sendet einfach alle von Ihnen übergebenen SQL-Befehle an die Datenbank. Das heißt, Sie können alle SQL-Befehle übergeben, die Ihre Datenbank unterstützt.



Die SQL-Befehlsdatei, mit der diese Beispiel-Bibliotheksdatenbank erzeugt wird, finden Sie in der Datei *library.sql*.

Angenommen, Sie besitzen eine Tabelle namens books, dann würde die folgende SQL-Anweisung eine neue Zeile einfügen (den Quellcode finden Sie auf der Website zum Buch unter <http://www.oreilly.de/catalog/progphp2ger/>):

```
INSERT INTO books VALUES (4, 'I, Robot', '0-553-29438-5', 1950)
```

Die folgende SQL-Anweisung fügt eine neue Zeile ein, gibt aber auch die Spalten an, denen die Werte zugeordnet werden sollen:

```
INSERT INTO books (authorid, title, ISBN, pub_year) VALUES (4, 'I, Robot', '0-553-29438-5', 1950)
```

Um alle Bücher zu löschen, die im Jahr 1979 veröffentlicht wurden (falls es welche gab), könnten Sie diese SQL-Anweisung verwenden:

```
DELETE FROM books WHERE pub_date = 1979
```

Um das Jahr für *Roots* auf 1983 zu ändern, verwenden Sie diese SQL-Anweisung:

```
UPDATE books SET pub_year=1983 WHERE title='Roots'
```

Um nur die in den 80ern veröffentlichten Bücher einzulesen, verwenden Sie:

```
SELECT * FROM books WHERE pub_year > 1979 AND pub_year < 1990
```

Sie können auch die Felder festlegen, die Sie zurückgeliefert haben möchten:

```
SELECT title, pub_year FROM books WHERE pub_year > 1979 AND pub_year < 1990
```

Sie können Abfragen (Queries) formulieren, die Informationen aus mehreren Tabellen zusammenfassen. Zum Beispiel fasst die folgende Abfrage die book- und author-Tabellen zusammen und zeigt uns, wer welches Buch geschrieben hat:

```
SELECT authors.name, books.title FROM books, authors
WHERE authors.authorid = books.authorid.
```

Mehr zu SQL erfahren Sie in *SQL in a Nutshell* von Kevin Kline (O'Reilly Verlag).

## PEAR DB-Grundlagen

Beispiel 8-1 ist ein Programm, das eine HTML-Tabelle mit Informationen zu Science-Fiction-Büchern aufbaut. Es zeigt, wie die PEAR DB-Bibliothek (die zum Lieferumfang von PHP gehört) eingesetzt wird, um die Verbindung zu einer Datenbank herzustellen, Queries anzustoßen, auf Fehler zu prüfen und die Ergebnisse der Abfragen in HTML umzuwandeln. Dabei sollten Sie prüfen, wie PEAR in Ihrer PHP-Einrichtung eingeschaltet wird, da das bei Unix/Linux etwas anders ist als unter Windows. Einen Ausgangspunkt für Installationshinweise finden Sie unter <http://www.pear.php.net/manual/en/installation.getting.php>. Die Bibliothek ist objektorientiert und eine Mischung aus Klassenmethoden (`DB::connect()`, `DB::iserror()`) und Objektmethoden (`$db->query()`, `$q->fetchInto()`).

*Beispiel 8-1: Buchinformationen anzeigen*

```
<html><head><title>Bücher</title></head>
<body>

<table border=1>
<tr><th>Buch</th><th>Erscheinungsjahr</th><th>Autor</th></tr>
<?php
// verbinden
require_once('DB.php');
$db = DB::connect("mysql://librarian:passw0rd@localhost/library");
if (DB::iserror($db) {
    die($db->getMessage());
}

// die Abfrage ausführen
$sql = "SELECT books.title,books.pub_year,authors.name
        FROM books, authors
        WHERE books.authorid=authors.authorid
        ORDER BY books.pub_year ASC";

$q = $db->query($sql);
if (DB::iserror($q) {
    die($q->getMessage());
}

// die Tabelle generieren
while ($q->fetchInto($row) {
?>
<tr><td><?=$row[0] ?></td>
    <td><?=$row[1] ?></td>
    <td><?=$row[2] ?></td>
</tr>
<?php
}
?>
```

Die Ausgabe von Beispiel 8-1 ist in Abbildung 8-1 zu sehen.

Buch	Erscheinungsjahr	Autor
The Hobbit	1937	J.R.R. Tolkien
I, Robot	1950	Isaac Asimov
Foundation	1951	Isaac Asimov
Foundation and Empire	1952	Isaac Asimov
Second Foundation	1953	Isaac Asimov
The Two Towers	1954	J.R.R. Tolkien
The Return of The King	1955	J.R.R. Tolkien
The Best of Isaac Asimov	1973	Isaac Asimov
Roots	1974	Alex Haley
Foundation's Edge	1982	Isaac Asimov
Exploring the Earth and the Cosmos	1982	Isaac Asimov
The Sum of All Fears	1991	Tom Clancy
Forward the Foundation	1993	Isaac Asimov
Isaac Asimov: Gold	1995	Isaac Asimov
Executive Orders	1996	Tom Clancy
Rainbow Six	1998	Tom Clancy
Red Rabbit	2000	Tom Clancy
Teeth of the Tiger	2003	Tom Clancy

Abbildung 8-1: Die Bücher-Seite

## Data Source Names

Ein *Data Source Name* (DSN), zu Deutsch etwa Datenquellen-Name, ist ein String, der festlegt, wo die Datenbank liegt, um welche Art von Datenbank es sich handelt, welcher Benutzername und welches Passwort bei der Verbindung zur Datenbank notwendig sind und vieles mehr. Die Komponenten des PEAR-DSN sind in einem URL-ähnlichen String zusammengestellt:

```
typ(dbsyntax)://benutzername:passwort@protokoll+hostspect/datenbank
```

Das einzige Pflichtfeld ist der *typ*, der festlegt, welches PHP-Datenbank-Backend verwendet werden soll. Tabelle 8-1 führt die Datenbanktypen auf, die implementiert waren, als diese Zeilen geschrieben wurden.

Tabelle 8-1: PHP-Datenbanktypen

Name	Datenbank
Mysql	MySQL
mysqli	MySQL (für MySQL >= 4.1)
Pgsql	PostgreSQL
Ibase	InterBase
Msql	Mini SQL
Mssql	Microsoft SQL Server
oci8	Oracle 7/8/8i
Odbc	ODBC
Sybase	SyBase
Ifx	Informix
Fbsql	FrontBase
Dbase	DBase
Sqlite	SQLite

Das *protokoll* gibt das zu verwendende Kommunikationsprotokoll an. Die beiden gängigen Werte sind "tcp" und "unix", was den Internet- und Unix-Domain-Sockets entspricht. Nicht jedes Datenbank-Backend unterstützt jedes Kommunikationsprotokoll.

Hier beispielhaft einige gültige DSNs:

```
mysql:///library
mysql://localhost/library
mysql://librarian@localhost/library
mysql://librarian@tcp+localhost/library
mysql://librarian:password@localhost/library
```

In Beispiel 8-1 haben wir die Verbindung mit der MySQL-Datenbank *library* hergestellt und dabei den Benutzernamen *librarian* und das Passwort *password* verwendet.

Eine gängige Entwicklungstechnik besteht darin, den DSN in einer PHP-Datei abzulegen und diese Datei in jede Seite einzufügen, die die Datenbank-Verbindung benötigt. Auf diese Weise müssen Sie nicht jede Seite ändern, wenn sich diese Informationen ändern. In einer etwas ausgereifteren Einstellungsdatei könnten Sie die DSNs sogar abhängig davon wechseln, ob sich die Anwendung im Entwicklungs- oder Anwendungsmodus befindet.

## Verbindung herstellen

Sobald Sie einen DSN besitzen, stellen Sie die Verbindung zur Datenbank über die Methode `connect()` her. Das liefert ein Datenbank-Objekt zurück, das Sie für solche Aufgaben wie die Abfrage der Datenbank und das Quoting von Parametern verwenden:

```
$db = DB::connect(DSN [, optionen ]);
```

Der *optionen*-Wert kann entweder ein Boolescher Wert sein und festlegen, ob die Verbindung persistent sein soll oder nicht, oder er ist ein Array mit Optionseinstellungen. Die *optionen*-Werte stehen in Tabelle 8-2.

Tabelle 8-2: Verbindungsoptionen

Option	steuert
persistent	Verbindung bleibt zwischen den Zugriffen erhalten
optimize	Was optimiert werden soll
debug	Ausgabe der Debugging-Informationen

Standardmäßig ist die Verbindung nicht persistent, und es werden auch keine Debugging-Informationen ausgegeben. Gültige Werte für *optimize* sind 'performance' und 'portability'. Voreingestellt ist 'performance'. Das folgende Codefragment aktiviert das Debugging und optimiert für Portabilität:

```
$db = DB::connect($dsn, array('debug' => 1, 'optimize' => 'portability'));
```

## Fehlerprüfung

PEAR DB-Methoden geben `DB_ERROR` zurück, wenn ein Fehler aufgetreten ist. Sie können das mit `DB::isError()` überprüfen:

```
$db = DB::connect($datasource);
if (DB::isError($db)) {
    die($db->getMessage());
}
```

Die Methode `DB::isError()` gibt `true` zurück, wenn es während der Arbeit mit dem Datenbank-Objekt zu einem Fehler gekommen ist. Wenn ein Fehler aufgetreten ist, besteht das übliche Verhalten darin, das Programm anzuhalten und die von `getMessage()` gelieferte Fehlermeldung auszugeben. Sie können `getMessage()` auf jedes PEAR DB-Objekt anwenden.

## Eine Abfrage absetzen

Die `query()`-Methode, angewandt auf ein Datenbank-Objekt, sendet SQL an die Datenbank:

```
$ergebnis = $db->query($sql);
```

Eine SQL-Anweisung, die die Datenbank nicht abfragt (z.B. `INSERT`, `UPDATE`, `DELETE`), gibt bei Erfolg die Konstante `DB_OK` zurück. SQL, das eine Abfrage durchführt (z.B. `SELECT`), gibt ein Objekt zurück, das Sie nutzen können, um auf die Ergebnisse zuzugreifen.

Sie können den Erfolg mit `DB::isError()` überprüfen:

```

$q = $db->query($sql);
if (DB::isError($q)) {
    die($q->getMessage());
}

```

## Ergebnisse einer Abfrage abrufen

PEAR DB stellt zwei Methoden zur Verfügung, mit denen Sie Daten aus einem Query-Ergebnisobjekt abrufen können. Eine Methode liefert ein der nächsten Zeile entsprechendes Array zurück, die andere speichert das Zeilen-Array in einer als Parameter übergebenen Variablen ab.

### Die Zeile zurückgeben

Die `fetchRow()`-Methode führt bei einem Abfrage-Ergebnis zur Rückgabe eines Arrays der nächsten Zeile des Ergebnisses:

```
$zeile = $ergebnis->fetchRow([ modus ]);
```

Zurückgeliefert wird entweder ein Array mit Daten, NULL, wenn es keine weiteren Daten gibt (oder es von Anfang an keine gab – die Ergebnismenge leer war), oder `DB_ERROR` bei einem Fehler. Der Parameter *modus* bestimmt das Format des zurückgegebenen Arrays und wird später noch beschrieben.

Dieses gängige Idiom verwendet die `fetchRow()`-Methode, um ein Ergebnis Zeile für Zeile zu verarbeiten:

```

while ($zeile = $result->fetchRow()) {
    if (DB::isError($zeile)) {
        die($zeile->getMessage());
    }
    // Zeile verarbeiten
}

```

### Die Zeile speichern

Die Methode `fetchInto()` liest nicht nur die nächste Zeile ein, sondern speichert sie auch in einer als Parameter übergebenen Array-Variablen:

```
$erfolg = $ergebnis->fetchInto(array, [modus]);
```

Wie `fetchRow()` gibt auch `fetchInto()` NULL zurück, wenn es keine weiteren Daten gibt, bzw. `DB_ERROR`, wenn ein Fehler aufgetreten ist.

Das Idiom zur Verarbeitung aller Ergebnisse mit `fetchInto()` sieht wie folgt aus:

```

while ($erfolg = $ergebnis->fetchInto($zeile)) {
    if (DB::isError($erfolg)) {
        die($erfolg->getMessage());
    }
    // Zeile verarbeiten
}

```

### Innerhalb des Zeilen-Arrays

Was sind das nun für Zeilen, die da zurückgegeben werden? Standardmäßig handelt es sich um indexierte Arrays, bei denen die Positionen im Array der Reihenfolge der Spalten im zurückgegebenen Ergebnis entsprechen. Ein Beispiel:

```
$zeile = $ergebnis->fetchRow();
if (DB::isError($row)) {
    die($zeile->getMessage());
}
var_dump($zeile);
array(3) {
    [0]=>
    string(5) "Foundation"
    [1]=>
    string(4) "1951"
    [2]=>
    string(12) "Isaac Asimov"
}
```

Sie können einen *modus*-Parameter an `fetchRow()` oder `fetchInto()` übergeben, um das Format des Zeilen-Arrays zu kontrollieren. Das vorhin gezeigte Standardverhalten wird mit `DB_FETCHMODE_ORDERED` festgelegt.

Der Modus `DB_FETCHMODE_ASSOC` erzeugt ein Array, dessen Schlüssel die Spaltennamen und dessen Werte die entsprechenden Werte dieser Spalten sind:

```
$zeile = $ergebnis->fetchRow(DB_FETCHMODE_ASSOC);
if (DB::isError($row)) {
    die($zeile->getMessage());
}
var_dump($zeile);
array(3) {
    ["title"]=>
    string(5) "Foundation"
    ["pub_year"]=>
    string(4) "1951"
    ["name"]=>
    string(12) "Isaac Asimov"
}
```

Der Modus `DB_FETCHMODE_OBJECT` macht aus der Zeile ein Objekt mit einer Eigenschaft für jede Spalte der Ergebniszeile:

```
$zeile = $ergebnis->fetchRow(DB_FETCHMODE_OBJECT);
if (DB::isError($row)) {
    die($zeile->getMessage());
}
var_dump($zeile);
object(stdClass)(3) {
    ["title"]=>
    string(5) "Foundation"
    ["pub_year"]=>
```

```

    string(4) "1951"
    ["name"]=>
    string(12) "Isaac Asimov"
}

```

Um auf Daten im Objekt zuzugreifen, verwenden Sie die Notation `$objekt->eigenschaft`:

```

echo "{$zeile->title} wurde {$zeile->pub_year} veröffentlicht und von {$zeile->author}
geschrieben";
Foundation wurde 1951 veröffentlicht und von Isaac Asimov geschrieben

```

### Freigabe des Ergebnisses

Ein Query-Ergebnisobjekt enthält normalerweise alle von der Abfrage zurückgegebenen Zeilen. Das kann sehr viel Speicher verbrauchen. Um den verbrauchten Speicher wieder an das Betriebssystem zurückzugeben, verwenden Sie die Methode `free()`:

```
$ergebnis->free();
```

Das ist nicht wirklich notwendig, weil `free()` automatisch für alle Queries aufgerufen wird, wenn das PHP-Skript endet.

### Verbindung abbauen

Um mit PHP die Verbindung zu einer Datenbank abzubauen, wenden Sie die Methode `disconnect()` auf das Datenbank-Objekt an:

```
$db->disconnect();
```

Auch das ist nicht unbedingt notwendig, da alle Datenbank-Verbindungen abgebaut werden, wenn das PHP-Skript beendet wird.

## Fortgeschrittene Datenbank-Techniken

PEAR DB geht über die vorhin gezeigten Datenbank-Primitiva hinaus. Es besitzt mehrere Kurzformen zum Einlesen von Ergebniszeilen, ein eindeutiges Zeilen-ID-System und separate Prepare/Execute-(Vorbereiten/Ausführen)-Schritte, die die Performance von wiederholt ausgeführten Queries verbessern können.

### Platzhalter

Genau wie `printf()` einen String aufbaut, indem es Werte in ein Template einfügt, kann PEAR DB eine Abfrage aufbauen, indem es Werte in ein Template einfügt. Übergeben Sie der `query()`-Funktion SQL mit `?` anstelle bestimmter Werte, und hängen Sie einen zweiten Parameter an, der die Werte enthält, die in den SQL-Code einzufügen sind:

```
$ergebnis = $db->query(SQL, werte);
```

Zum Beispiel fügt dieser Code drei Einträge in die `books`-Tabelle ein:

```

$buecher = array(array('Foundation', 1951),
                 array('Second Foundation', 1953),
                 array('Foundation and Empire', 1952));
foreach ($buecher as $buch) {
    $db->query('INSERT INTO books (title,pub_year) VALUES (?,?)', $buch);
}

```

Es gibt drei Zeichen, die Sie als Platzhalter in einer SQL-Abfrage verwenden können:

- ? Ein String oder eine Zahl, der bzw. die bei Bedarf einem Quoting unterzogen wird (empfohlen)
- | Ein String oder eine Zahl, der bzw. die nie einem Quoting unterzogen wird
- & Ein Dateiname; der Dateiinhalt wird in die Anweisung eingefügt (z.B. um eine Image-Datei in einem BLOB-Feld zu speichern)

## Prepare/Execute

Wenn die gleiche Abfrage immer wieder abgesetzt wird, kann es effizienter sein, die Abfrage einmal zu kompilieren und dann wiederholt auszuführen. Das ist mit den Methoden `prepare()`, `execute()` und `executeMultiple()` möglich.

Der erste Schritt besteht im Aufruf von `prepare()` für die Abfrage:

```
$kompiliert = $db->prepare(SQL);
```

Das liefert ein kompiliertes Query-Objekt zurück. Die `execute()`-Methode füllt alle Platzhalter in der Abfrage aus und sendet sie an das RDBMS:

```
$antwort = $db->execute(kompiliert, werte);
```

Das `werte`-Array enthält die Werte für die Platzhalter. Der Rückgabewert ist entweder ein Query-Response-Objekt oder `DB_ERROR`, wenn ein Fehler aufgetreten ist.

Zum Beispiel könnten wir folgendermaßen mehrere Werte in die `books`-Tabelle einfügen:

```

$buecher = array(array('Foundation', 1951),
                 array('Second Foundation', 1953),
                 array('Foundation and Empire', 1952));
$kompiliert = $q->prepare('INSERT INTO books (title,pub_year) VALUES (?,?)');
foreach ($buecher as $buch) {
    $db->execute($kompiliert, $buch);
}

```

Die Methode `executeMultiple()` arbeitet mit einem zweidimensionalen Werte-Array:

```
$antworten = $db->executeMultiple(kompiliert, werte);
```

Das `werte`-Array muss bei 0 beginnend numerisch indiziert sein. Die Werte sind Arrays von Werten, die in die Datenbank eingefügt werden sollen. Die kompilierte Abfrage wird für jeden Eintrag in `werte` einmal ausgeführt und die Query-Responses werden in `$antworten` gesammelt.

Eine bessere Variante des Codes, mit dem Sie Einträge in der Buchdatenbank vornehmen, sieht daher so aus:

```
$buecher = array(array('Foundation', 1951),
                 array('Second Foundation', 1953),
                 array('Foundation and Empire', 1952));
$kompiliert = $q->prepare('INSERT INTO books (title, pub_year) VALUES (?,?)');
$db->insertMultiple($kompiliert, $buecher);
```

## Kurzformen

PEAR DB stellt eine Reihe von Methoden zur Verfügung, die in einem Schritt eine Abfrage durchführen und die Ergebnisse einlesen: `getOne()`, `getRow()`, `getCol()`, `getAssoc()` und `getAll()`. All diese Methoden erlauben Platzhalter.

Die `getOne()`-Methode liest die erste Spalte der ersten Zeile der von einer SQL-Abfrage zurückgegebenen Ergebnisse ein:

```
$wert = $db->getOne(SQL [, werte ]);
```

Ein Beispiel:

```
$wann = $db->getOne("SELECT avg(pub_year) FROM books");
if (DB::isError($when)) {
    die($wann->getMessage());
}
```

```
echo "Das durchschnittliche Erscheinungsdatum der Bücher in der Bibliothek ist $wann";
```

**Das durchschnittliche Erscheinungsdatum der Bücher in der Bibliothek ist 1974**

Die Methode `getRow()` gibt die erste von einer SQL-Abfrage zurückgelieferte Datenzeile zurück:

```
$zeile = $db->getRow(SQL [, werte ]);
```

Das ist nützlich, wenn Sie wissen, dass nur eine Zeile zurückgegeben wird. Ein Beispiel:

```
list($titel, $autor) = $db->getRow(
    "SELECT books.title, authors.name FROM books, authors
    WHERE books.pub_year=1950 AND books.authorid=authors.authorid");
echo "($titel, geschrieben von $autor)";
(I, Robot, geschrieben von Isaac Asimov)
```

Die `getCol()`-Methode gibt eine einzelne Spalte der durch eine SQL-Abfrage zurückgelieferten Daten zurück:

```
$ergebnis = $db->getCol(SQL [, spalte [, werte ]]);
```

Der `spalte`-Parameter kann eine Zahl (die voreingestellte 0 entspricht der ersten Spalte) oder ein Spaltenname sein.

Das folgende Beispiel gibt die Titel aller Bücher in der Datenbank sortiert nach Erscheinungsjahr aus:

```

$alle_titel = $db->getAll("SELECT title FROM books ORDER BY pub_year ASC");
foreach ($alle_titel as $titel) {
    echo "$titel\n";
}
the Hobbit
I, Robot
Foundation
...

```

Die `getAll()`-Methode gibt ein Array mit allen von einer Abfrage zurückgelieferten Zeilen zurück:

```
$alle = $db->getAll($SQL [, $werte [, $modus ]]);
```

Der folgende Code erzeugt beispielsweise eine Auswahlbox mit den Titeln der Bücher. Die ID des gewählten Buchs wird als Parameterwert übertragen.

```

$ergebnis = $db->getAll("SELECT bookid,title FROM books ORDER BY pub_year ASC");
echo "<select name='buecher'>\n";
foreach ($results as $result) {
    echo "<option value={$ergebnis[0]}>{$ergebnis[1]}</option>\n";
}
echo "</select>";

```

Alle `get*()`-Methoden geben bei einem Fehler `DB_ERROR` zurück.

## Details zu einer Query-Response

Vier PEAR DB-Methoden versorgen Sie mit Informationen zu einem Query-Ergebnisobjekt: `numRows()`, `numCols()`, `affectedRows()` und `tableInfo()`.

Die Methoden `numRows()` und `numCols()` liefern Ihnen die Anzahl der von einer SELECT-Abfrage zurückgegebenen Zeilen und Spalten:

```

$wie_viele = $response->numRows();
$wie_viele = $response->numCols();

```

Die `affectedRows()`-Methode gibt an, wie viele Zeilen von einer INSERT-, DELETE- oder UPDATE-Operation betroffen sind:

```
$wie_viele = $response->affectedRows();
```

Die `tableInfo()`-Methode liefert detaillierte Informationen zu den von einer SELECT-Operation zurückgegebenen Feldtypen und -Flags:

```
$info = $response->tableInfo();
```

Der folgende Code schreibt die Tabelleninformationen in eine HTML-Tabelle:

```

// verbinden
require_once('DB.php');
$db = DB::connect("mysql://librarian:passwOrd@localhost/library");
if (DB::isError($db)) {
    die($db->getMessage());
}

```

```

$sql = "SELECT * FROM BOOKS";

$q = $db->query($sql);
if (DB::isError($q)) {
    die($q->getMessage());
}

$info = $q->tableInfo();
a_to_table($info);

function a_to_table ($a) {
    echo "<html><head><title> Table Info </title></head>";
    echo "<table border=1>\n";
    foreach ($a as $key => $value) {
        echo "<tr valign=top align=left><td>$key</td><td>";
        if (is_array($value)) {
            a_to_table($value);
        } else {
            print_r($value);
        }
        echo "</td></tr>\n";
    }
    echo "</table>\n";
}

```

Abbildung 8-2 zeigt die Ausgabe unseres kleinen Tabellen-Informationsprogramms.

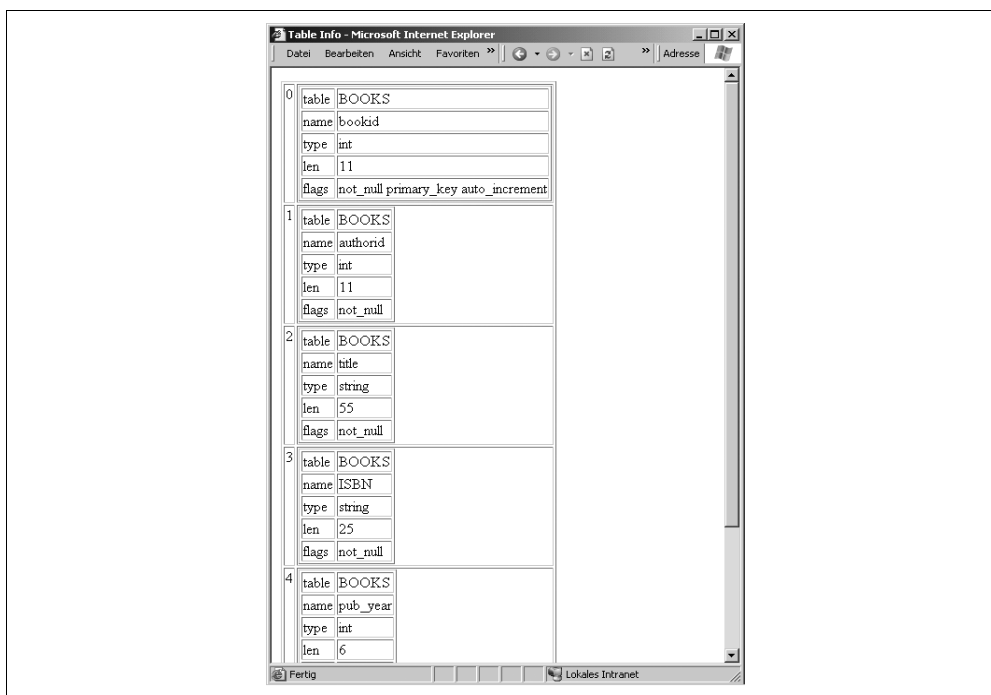


Abbildung 8-2: Die Informationen aus tableInfo()

## Sequenzen

Nicht jedes RDBMS besitzt die Fähigkeit, eindeutige Zeilen-IDs zuzuweisen, und diejenigen, die das können, liefern diese Informationen in sehr unterschiedlicher Form zurück. PEAR DB-Sequenzen sind eine Alternative zur datenbankspezifischen ID-Zuweisung (etwa zu `AUTO_INCREMENT` von MySQL).

Die `nextID()`-Methode liefert die nächste ID für eine gegebene Sequenz zurück:

```
$id = $db->nextID(sequenz);
```

Normalerweise besitzen Sie eine Sequenz für jede Tabelle, für die Sie eindeutige IDs benötigen. Das folgende Beispiel fügt Werte in die `books`-Tabelle ein und weist jeder Zeile einen eindeutigen Identifier zu:

```
$buecher = array(array('Foundation', 1951),
                 array('Second Foundation', 1953),
                 array('Foundation and Empire', 1952));

foreach ($buecher as $buch) {
    $id = $db->nextID('books');
    splice($buch, 0, 0, $id);
    $db->query('INSERT INTO books (bookid,title,pub_year) VALUES (?,?,?)', $book);
}</pre
```

Eine Sequenz ist in Wirklichkeit eine Tabelle in der Datenbank, in der die zuletzt zugewiesene ID festgehalten wird. Sie können Sequenzen explizit mit den Methoden `createSequence()` und `dropSequence()` erzeugen bzw. freigeben:

```
$res = $db->createSequence(sequenz);
$res = $db->dropSequence(sequenz);
```

Das Ergebnis entspricht dem Ergebnis-Objekt der `create-` oder `drop-Query` bzw. `DB_ERROR` bei einem Fehler.

## Metadaten

Mit der `getListOf()`-Methode können Sie die Datenbank nach Informationen über verfügbare Datenbanken, Benutzer, Views und Funktionen abfragen:

```
$daten = $db->getListOf(was);
```

Der `was`-Parameter ist ein String, der das aufzuführende Datenbank-Feature beschreibt. Die meisten Datenbanken unterstützen "databases"; einige unterstützen "users", "views" und "functions".

Das folgende Beispiel speichert eine Liste aller verfügbaren Datenbanken in `$dbs`:

```
$dbs = $db->getListOf("databases");
```

## Transaktionen

Einige RDBMSs unterstützen *Transaktionen*, bei denen eine Folge von Datenbank-Änderungen bestätigt (»commit«, d.h., alle Änderungen werden auf einmal vorgenommen) oder verworfen (»rollback«, d.h., keinerlei Änderungen werden vorgenommen) werden kann. Zum Beispiel muss in einer Bank die Entnahme von einem Konto und die Einzahlung auf ein anderes Konto in einem Schritt erfolgen – keine Aktion darf ohne die andere erfolgen, und zwischen diesen beiden Operationen darf keine Zeitverzögerung entstehen. PEAR DB bietet die Methoden `commit()` und `rollback()`, um Sie bei Transaktionen zu unterstützen:

```
$res = $db->commit();
$res = $db->rollback();
```

Wenn Sie `commit()` oder `rollback()` auf Datenbanken aufrufen, die keine Transaktionen unterstützen, geben die Methoden `DB_ERROR` zurück.



Prüfen Sie unbedingt, ob Ihr zugrunde liegendes Datenbanksystem Transaktionen unterstützt.

## Beispielanwendung

Weil datenbankbasierte Webanwendungen eine wesentliche Stütze der Webentwicklung sind, haben wir uns dazu entschieden, in diesem Kapitel eine vollständige Beispielanwendung zu zeigen. In diesem Abschnitt entwickeln wir ein sich selbst pflegendes Branchenverzeichnis. Unternehmen fügen eigene Einträge in die Datenbank ein und wählen die Kategorie oder Kategorien aus, in denen sie vertreten sein möchten.

Zwei HTML-Formulare sind notwendig, um die Datenbank-Tabellen mit Leben zu füllen. Ein Formular stellt dem Site-Administrator die Mittel zur Verfügung, um Kategorie-IDs, Titel und Beschreibungen zu verwalten. Das zweite Formular wird von den sich selbst eintragenden Unternehmen genutzt. Es sammelt die Unternehmensdaten und lässt den Benutzer die Daten mit einer oder mehreren Kategorien verknüpfen. Eine separate Seite gibt die Daten nach Kategorien sortiert auf der Webseite aus.

## Datenbank-Tabellen

Unsere Anwendung verwendet drei Tabellen: `businesses` für die Adressdaten aller Unternehmen, `categories` zur Benennung und Beschreibung jeder Kategorie und eine assoziative Tabelle namens `biz_categories`, die die Einträge der beiden anderen Tabellen miteinander verknüpft. Die Tabellen und ihre Beziehungen sind in Abbildung 8-3 zu sehen.

Beispiel 8-2 zeigt einen Dump der Tabellen-Schemata im MySQL-Format. Abhängig von den Features Ihrer Datenbank, können diese Schemata bei Ihnen etwas anders aussehen.

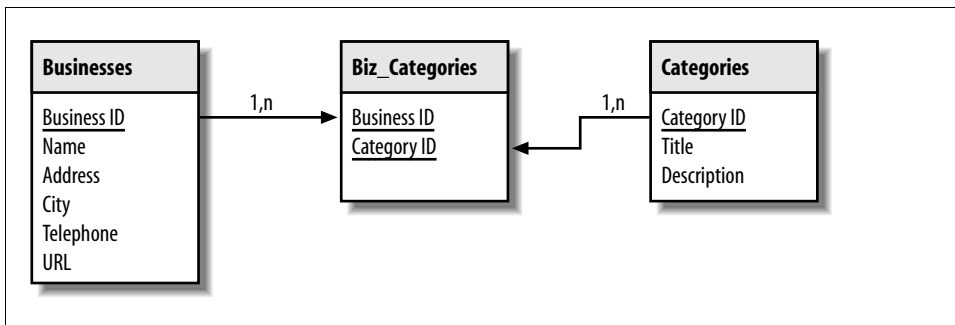


Abbildung 8-3: Datenbank-Design für unser Branchenverzeichnis

Beispiel 8-2: Datenbank-Schemata

```

# -----
#
# Tabellenstruktur für Tabelle 'biz_categories'
#
CREATE TABLE biz_categories (
    business_id int(11) NOT NULL,
    category_id char(10) NOT NULL,
    PRIMARY KEY (business_id, category_id),
    KEY business_id (business_id, category_id)
);

# -----
#
# Tabellenstruktur für Tabelle 'businesses'
#
CREATE TABLE businesses (
    business_id int(11) NOT NULL auto_increment,
    name varchar(255) NOT NULL,
    address varchar(255) NOT NULL,
    city varchar(128) NOT NULL,
    telephone varchar(64) NOT NULL,
    url varchar(255),
    PRIMARY KEY (business_id),
    UNIQUE business_id (business_id),
    KEY business_id_2 (business_id)
);

# -----
#
# Tabellenstruktur für Tabelle 'categories'
#
CREATE TABLE categories (
    category_id varchar(10) NOT NULL,

```

*Beispiel 8-2: Datenbank-Schemata (Fortsetzung)*

```

title varchar(128) NOT NULL,
description varchar(255) NOT NULL,
PRIMARY KEY (category_id),
UNIQUE category_id (category_id),
KEY category_id_2 (category_id)
);

```

## Datenbank-Verbindung

Wir haben die Seiten so entworfen, dass sie mit einem MySQL-, PostgreSQL- oder Oracle 8i-Backend zusammenarbeiten. Das einzige sichtbare Zeichen hierfür ist die Tatsache, dass der PHP-Code nach jedem Update ein `commit()` verwendet. Wir haben die datenbankspezifischen Dinge in einer `db_login.php`-Bibliothek untergebracht (siehe Beispiel 8-3), die den entsprechenden DSN für MySQL, PostgreSQL oder Oracle wählt.

*Beispiel 8-3: Abstraktionsskript für die Datenbank-Verbindung (db\_login.php)*

```

<?php
require_once('DB.php');

// Setup der Datenbank-Verbindung

$username = 'user';
$password = 'seekrit';
$hostspec = 'localhost';
$databse = 'phpbook';

// Wählen Sie einen dieser drei Typen für $phptype

// $phptype = 'pgsql';
// $phptype = 'oci8';
$phptype = 'mysql';

// Prüfung für Oracle 8, da sich die DSN-Syntax unterscheidet
if ($phptype != 'oci8'){
    $dsn = "$phptype://$username:$password@$hostspec/$databse";
} else {
    $net8name = 'www';
    $dsn = "$phptype://$username:$password@$net8name";
}

// Verbindung herstellen

$db = DB::connect($dsn);
if (DB::isError($db)) {
    die ($db->getMessage());
}
?>

```

## Die Administrator-Seite

Beispiel 8-4 zeigt die Backend-Seite, die dem Administrator die Aufnahme zusätzlicher Kategorien in das Branchenbuch ermöglicht. Die Eingabefelder für die Aufnahme eines neuen Datensatzes erscheinen nach der Ausgabe der aktuellen Daten. Der Administrator füllt das Formular aus und klickt auf den Button *Kategorie hinzufügen*. Die Seite wird dann mit dem neuen Datensatz erneut ausgegeben. Ist eines der drei notwendigen Felder nicht ausgefüllt worden, gibt die Seite eine Fehlermeldung aus.

*Beispiel 8-4: Administrator-Backend*

```
<html>
<head>
<?php
    require_once('db_login.php');
?>

<title>
<?php
    // Fenstertitel und Überschrift ausgeben
    $doc_title = 'Kategorie-Administration';
    echo "$doc_title\n";
?>
</title>
</head>
<body>
<h1>
<?php
    echo "$doc_title\n";
?>
</h1>

<?php
    // Kategorie hinzufügen - Eingabebereich

    // Werte aus $_REQUEST extrahieren
    $Cat_ID = $_REQUEST['Cat_ID'];
    $Cat_Title = $_REQUEST['Cat_Title'];
    $Cat_Desc = $_REQUEST['Cat_Desc'];
    $add_record = $_REQUEST['add_record'];

    // Länge der Eingabefelder ermitteln
    $len_cat_id = strlen($_REQUEST['Cat_ID']);
    $len_cat_tl = strlen($_REQUEST['Cat_Title']);
    $len_cat_de = strlen($_REQUEST['Cat_Desc']);

    // Validieren und einfügen, wenn Skript
    // mit dem »Kategorie hinzufügen«-Button aufgerufen wurde
    if ($add_record == 1) {
        if (($len_cat_id > 0) and ($len_cat_tl > 0) and ($len_cat_de > 0)){
            $sql = "insert into categories (category_id, title, description)";
```

*Beispiel 8-4: Administrator-Backend (Fortsetzung)*

```

        $sql .= " values ('$Cat_ID', '$Cat_Title', '$Cat_Desc')";
        $result = $db->query($sql);
        $db->commit();
    } else {
        echo "<p>F&uuml;llen Sie alle Felder aus, und versuchen Sie ";
        echo "es erneut.</p>\n";
    }
}

// Kategorien ausgeben

// Alle Datensätze aus der Tabelle einlesen, da
// im obigen Teil eventuell eine Änderung stattgefunden hat
$sql = "select * from categories";
$result = $db->query($sql);
?>

<form method="POST" action="cat_admin.php">

<table>
<tr><th bgcolor="#e0e0e0">Kategorie-ID</th>
    <th bgcolor="#e0e0e0">Titel</th>
    <th bgcolor="#e0e0e0">Beschreibung</th>
</tr>

<?php
// Ausgabe aller Datensätze der Datenbank
// und der Eingabezeile für eine neue Kategorie
while ($row = $result->fetchRow()){
    echo "<tr><td>$row[0]</td><td>$row[1]</td><td>$row[2]</td></tr>\n";
}
?>

<tr><td><input type="text" name="Cat_ID"    size="15" maxlength="10"></td>
    <td><input type="text" name="Cat_Title" size="40" maxlength="128"></td>
    <td><input type="text" name="Cat_Desc" size="45" maxlength="255"></td>
</tr>
</table>
<input type="hidden" name="add_record" value="1">
<input type="submit" name="submit" value="Kategorie hinzuf&uuml;gen">
</body>
</html>

```

Wenn der Administrator einen neuen Kategorie-Eintrag sendet, erzeugen wir eine Abfrage, um die Kategorie in die Datenbank aufzunehmen. Eine weitere Abfrage gibt eine Tabelle aller aktuellen Kategorien aus. Abbildung 8-4 zeigt die Seite mit fünf Einträgen.

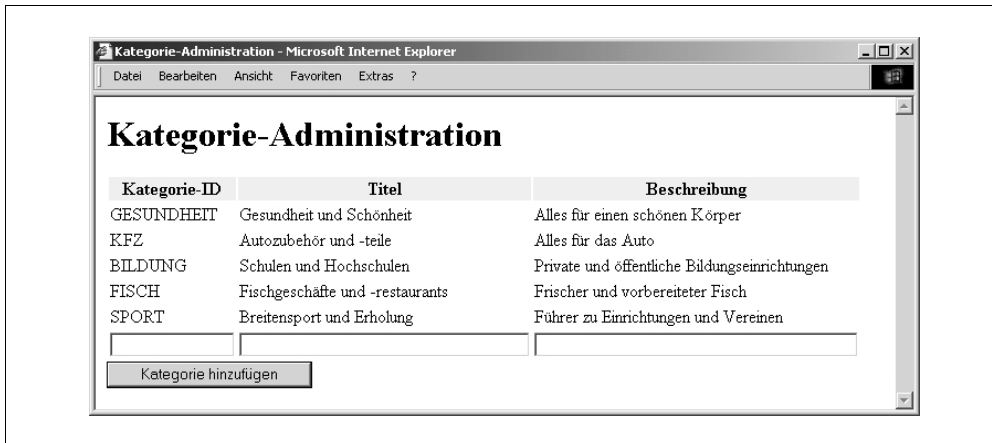


Abbildung 8-4: Die Administrator-Seite

## Ein Unternehmen aufnehmen

Beispiel 8-5 zeigt die Seite, mit deren Hilfe ein Unternehmen Daten in die Tabellen `businesses` und `biz_categories` einfügen kann. Abbildung 8-5 zeigt das Formular.



Abbildung 8-5: Registrierung des Unternehmens

Wenn ein Benutzer Daten eingibt und den Button *Unternehmen aufnehmen* anklickt, ruft sich das Skript selbst auf, um eine Bestätigungsseite auszugeben. Abbildung 8-6 zeigt eine Bestätigungsseite für ein Unternehmen, das in zwei Kategorien vertreten ist.



Abbildung 8-6: Eintrag, der zwei Kategorien zugeordnet ist

Auf der Bestätigungsseite ist der *Unternehmen aufnehmen*-Button durch einen Link ersetzt worden, der eine frische Instanz des Skripts aufruft. Eine Erfolgsmeldung wird zu Beginn der Seite ausgegeben. Anweisungen zur Verwendung der Scroll-Liste werden durch einen erläuternden Text ersetzt.

Wie in Beispiel 8-5 zu sehen ist, bauen wir die Scroll-Liste aus einer Abfrage auf, um alle Kategorien auszuwählen. Während wir den HTML-Code für jedes Ergebnis-Element der Abfrage erzeugen, prüfen wir auch, ob die aktuelle Kategorie eine von denjenigen war, die für das neue Unternehmen übergeben wurden. Ist das der Fall, tragen wir einen neuen Datensatz in der `biz_categories`-Tabelle ein.

*Beispiel 8-5: Aufnahme eines Unternehmens*

```
<html>
<head>
<title>
<?php
    $doc_title = 'Unternehmens-Registrierung';
    echo "$doc_title\n";
?>
</title>
</head>
<body>
<h1>
<?= $doc_title ?>
</h1>
```

*Beispiel 8-5: Aufnahme eines Unternehmens (Fortsetzung)*

```

<?php
require_once('db_login.php');

// Abfrage-Parameter einlesen
$add_record = $_REQUEST['add_record'];
$Biz_Name = $_REQUEST['Biz_Name'];
$Biz_Address = $_REQUEST['Biz_Address'];
$Biz_City = $_REQUEST['Biz_City'];
$Biz_Telephone = $_REQUEST['Biz_Telephone'];
$Biz_URL = $_REQUEST['Biz_URL'];
$Biz_Categories = $_REQUEST['Biz_Categories'];

$pick_message = 'Klicken Sie eine oder <BR>mehrere ';
$pick_message .= 'Kategorien an:';

// Neues Unternehmen aufnehmen
if ($add_record == 1) {
    $pick_message = 'Gew&auml;hlte Kategorien<BR>sind hervorgehoben: ';
    $sql = 'INSERT INTO businesses (name, address, city, telephone, '
    $sql .= ' url) VALUES (?, ?, ?, ?, ?)';
    $params = array($Biz_Name, $Biz_Address, $Biz_City, $Biz_Telephone, $Biz_URL);
    $query = $db->prepare($sql);
    if (DB::isError($query)) die($query->getMessage());
    $resp = $db->execute($query, $params);
    if (DB::isError($resp)) die($resp->getMessage());
    $resp = $db->commit();
    if (DB::isError($resp)) die($resp->getMessage());
    echo '<P CLASS="message">Sie wurden wie nachfolgend zu sehen aufgenommen.</P>';
    $biz_id = $db->getOne('SELECT max(business_id) FROM businesses');
}
?>

<form method="POST" action="<?=$PHP_SELF ?>">
<table>
<tr><td class="picklist"><?=$pick_message ?>
    <p>
    <select name="Biz_Categories[]" size="4" multiple>
    <?php
    // Scroll-Liste für die Kategorien aufbauen
    $sql = "SELECT * FROM categories";
    $result = $db->query($sql);
    if (DB::isError($result)) die($result->getMessage());
    while ($row = $result->fetchRow()){
        if (DB::isError($row)) die($row->getMessage());
        if ($add_record == 1){
            $selected = false;
            // Wenn diese Kategorie gewählt wurde, neue biz_categories-Zeile einfügen
            if (in_array($row[1], $Biz_Categories)) {
                $sql = 'INSERT INTO biz_categories';
                $sql .= ' (business_id, category_id)';
                $sql .= ' VALUES (?, ?)';
            }
        }
    }
    </?php
    </p>
    </td></tr>
</table>

```

*Beispiel 8-5: Aufnahme eines Unternehmens (Fortsetzung)*

```

        $params = array($biz_id, $row[0]);
        $query = $db->prepare($sql);
        if (DB::isError($query)) die($query->getMessage());
        $resp = $db->execute($query, $params);
        if (DB::isError($resp)) die($resp->getMessage());
        $resp = $db->commit();
        if (DB::isError($resp)) die($resp->getMessage());
        echo "<option selected>$row[1]</option>\n";
        $selected = true;
    }
    if ($selected == false) {
        echo "<option>$row[1]</option>\n";
    }
} else {
    echo "<option>$row[1]</option>\n";
}
}
?>

</select>
</td>
<td class="picklist">
    <table>
    <tr><td class="FormLabel">Unternehmensname:</td>
        <td><input type="text" name="Biz_Name" size="40" maxlength="255"
            value="<?=$Biz_Name ?>"</td>
    </tr>
    <tr><td class="FormLabel">Adresse:</td>
        <td><input type="text" name="Biz_Address" size="40" maxlength="255"
            value="<?=$Biz_Address ?>"</td>
    </tr>
    <tr><td class="FormLabel">Stadt:</td>
        <td><input type="text" name="Biz_City" size="40" maxlength="128"
            value="<?=$Biz_City ?>"</td>
    </tr>
    <tr><td class="FormLabel">Telefon:</td>
        <td><input type="text" name="Biz_Telephone" size="40" maxlength="64"
            value="<?=$Biz_Telephone ?>"</td>
    </tr>
    <tr><td class="FormLabel">URL:</td>
        <td><input type="text" name="Biz_URL" size="40" maxlength="255"
            value="<?=$Biz_URL ?>"</td>
    </tr>
    </table>
</td>
</tr>
</table>
<p>
<input type="hidden" name="add_record" value="1">

```

*Beispiel 8-5: Aufnahme eines Unternehmens (Fortsetzung)*

```

<?php
// Submit-Button bei neuen Formularen ausgeben; bei Bestätigungen auf
// frische Registrierungsseite verweisen
if ($add_record == 1){
    echo '<p><a href="" , $PHP_SELF, '>Weiteres Unternehmen aufnehmen</a></p>';
} else {
    echo '<input type="submit" name="submit" value="Unternehmen aufnehmen">';
}
?>

</p>
</body>
</html>

```

**Ausgabe der Datenbank**

Beispiel 8-6 zeigt eine Seite, die die Informationen in der Datenbank ausgibt. Die Links auf der linken Seite werden aus der categories-Tabelle erzeugt und verweisen zurück zu dem Skript, das eine Kategorie-ID hinzufügt. Die Kategorie-ID bildet die Grundlage für eine Abfrage der businesses- und der biz\_categories-Tabelle.

*Beispiel 8-6: Ausgabe der Unternehmen*

```

<html>
<head>
<title>
<?php
    $doc_title = 'Branchenbuch';
    echo "$doc_title\n";
?>
</title>
</head>
<body>
<h1>
<?= $doc_title ?>
</h1>

<?php
// Datenbank-Verbindung herstellen

require_once('db_login.php');

$pick_message = 'Klicken Sie eine Kategorie an, um sich die Unternehmen anzeigen zu
lassen: ';
?>

<table border=0>
<tr><td valign="top">
    <table border=5>
    <tr><td class="picklist"><strong><?= $pick_message ?></strong></td></tr>

```

*Beispiel 8-6: Ausgabe der Unternehmen (Fortsetzung)*

```

<p>
<?php
// Scroll-Liste für die Kategorien aufbauen
$sql = "SELECT * FROM categories";
$result = $db->query($sql);
if (DB::isError($result)) die($result->getMessage());
while ($row = $result->fetchRow()){
    if (DB::isError($row)) die($row->getMessage());
    echo '<tr><td class="formlabel">';
    echo "<a href=\"\$PHP_SELF?cat_id=\$row[0]\">";
    echo "$row[1]</a></td></tr>\n";
}
?>
</table>
</td>
<td valign="top">
<table border=1>
<?php
if ($cat_id) {
    $sql = "SELECT * FROM businesses b, biz_categories bc where";
    $sql .= " category_id = '$cat_id'";
    $sql .= " and b.business_id = bc.business_id";
    $result = $db->query($sql);
    if (DB::isError($result)) die($result->getMessage());
    while ($row = $result->fetchRow()){
        if (DB::isError($row)) die($row->getMessage());
        if ($color == 1) {
            $bg_shade = 'dark';
            $color = 0;
        } else {
            $bg_shade = 'light';
            $color = 1;
        }
        echo "<tr>\n";
        for($i = 0; $i < count($row); $i++) {
            echo "<td class=\"\$bg_shade\">$row[$i]</td>\n";
        }
        echo "</tr>\n";
    }
}
?>
</table>
</td></tr>
</table>
</body>
</html>

```

Die Ausgabe unseres Branchenbuchs finden Sie in Abbildung 8-7.

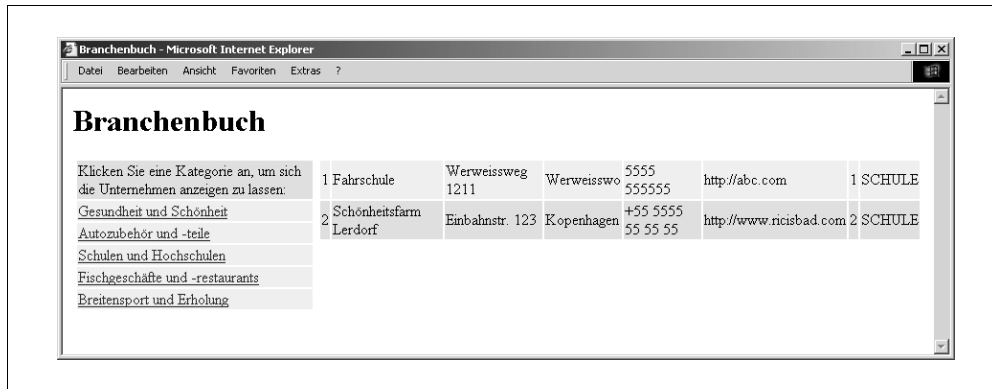


Abbildung 8-7: Ausgabe des Branchenbuchs

## PHP Data Objects

Es gibt ein weiteres Verfahren, um auf Datenbankinformationen zuzugreifen. Es handelt sich dabei um eine Datenbankeerweiterung namens PDO (PHP Data Objects), und die php.net-Website hat dazu Folgendes zu sagen:

Die PHP Data Objects-Erweiterung (PDO) definiert eine leichtgewichtige, konsistente Schnittstelle für den Datenbankzugriff in PHP. Jeder Datenbank-Treiber, der die PDO-Schnittstelle implementiert, kann datenbankspezifische Features als gewöhnliche Erweiterungsfunktionen veröffentlichen. Beachten Sie, dass Sie über die PDO-Erweiterung allein keinerlei Datenbankfunktionen ausführen können. Sie müssen einen datenbankspezifischen PDO-Treiber verwenden, um auf einen Datenbankserver zuzugreifen.

Die Hinzufügung und Erweiterung durch dieses neue Produkt war für Version 5.1 geplant und es sollte, wenn Sie das hier lesen, allgemein in Gebrauch sein. Im Grunde ist es ein weiteres Verfahren, um auf abstrakte Weise Verbindungen mit einer Datenbank aufzubauen. Obwohl es dem PEAR::DB-Ansatz ähnelt, den wir gerade behandelt haben, hat es (unter anderem) die folgenden einzigartigen Features:

- PDO ist eine native C-Erweiterung.
- PDO nutzt die Vorteile der neuesten PHP 5-Internas.
- PDO nutzt gepuffertes Lesen für die Daten der Ergebnismenge.
- PDO liefert allgemeine DB-Features als Grundlage.
- PDO ist immer noch dazu in der Lage, auf DB-spezifische Funktionen zuzugreifen.
- PDO kann transaktionsbasierte Techniken verwenden.
- PDO kann mit LOBS (Large Objects) in der Datenbank interagieren.

- PDO kann vorbereitete und ausführbare SQL-Anweisungen mit gebundenen Parametern verwenden.
- PDO kann scrollfähige Cursor implementieren.
- PDO hat Zugriff auf SQLSTATE-Fehlercodes und bietet eine sehr flexible Fehlerbehandlung.

Da das eine Vielzahl von Features ist, werden wir hier nur ein paar davon streifen, um Ihnen zu zeigen, wie vorteilhaft PDO sein kann.

Zuerst etwas über PDO. Es wird Treiber für fast alle existierenden Datenbank-Engines haben, und die Treiber, die PDO nicht unterstützt, sollten über PDOs generische ODBC-Verbindung zugreifbar sein. PDO ist darin modular, dass es verlangt, dass mindestens zwei Erweiterungen eingeschaltet sein müssen, damit es aktiv ist: die PDO-Erweiterung selbst und die spezifische PDO-Erweiterung für die Datenbank, auf die Sie zugreifen wollen. In der Online-Dokumentation unter <http://ca.php.net/pdo> erfahren Sie, wie Sie die Verbindungen für die Datenbank Ihrer Wahl einrichten. Um PDO auf einem Windows-Server für die MySQL-Interaktion einzurichten, geben Sie einfach die folgenden beiden Zeilen in Ihre *php.ini*-Datei ein und starten Ihren Server neu:

```
extension=php_pdo.dll
extension=php_pdo_mysql.dll
```

Die PDO-Bibliothek ist ebenfalls eine objektorientierte Erweiterung (das werden Sie in den nachfolgenden Codebeispielen sehen).

### Eine Verbindung herstellen

Für PDO müssen Sie als Erstes eine Verbindung zur fraglichen Datenbank herstellen und diese Verbindung in einer Verbindungs-Handle-Variablen wie im folgenden Code festhalten:

```
$ConnHandle = new PDO ($dsn, $username, $password);
```

*\$dsn* steht für den Data Source Name und die anderen beiden Parameter sind selbsterklärend. Für eine MySQL-Verbindung würden Sie beispielsweise den folgenden Code schreiben:

```
$ConnHandle = new PDO('mysql:host=localhost;dbname=library',
    'petermac', 'abc123');
```

Natürlich könnten (sollten) Sie die Parameter für Benutzername und Passwort aus Gründen der Wiederverwendbarkeit und Flexibilität variablenbasiert verwalten.

### Interaktion mit der Datenbank

Nachdem Sie eine Verbindung mit der Datenbank-Engine und der Datenbank haben, mit der Sie interagieren möchten, können Sie diese Verbindung verwenden, um SQL-Befehle an den Server zu senden. Eine einfache UPDATE-Anweisung würde folgendermaßen aussehen:

```
$ConnHandle->query("UPDATE books SET authorid=4 "
    . "WHERE pub_year = 1982");
```

Dieser Code aktualisiert einfach die Buchtabelle und gibt die Abfrage frei. Auf diese Weise würden Sie normalerweise einfache SQL-Befehle (UPDATE, DELETE, INSERT), die keine Ergebnismenge erzeugen, über PDO an die Datenbank senden, es sei denn, Sie verwenden vorbereitete Anweisungen, ein komplexeres Verfahren, das im nächsten Abschnitt besprochen wird.

### PDO und vorbereitete Anweisungen

PDO bietet außerdem eine Einrichtung, die man als vorbereitete Anweisungen bezeichnet. Dazu werden PDO-Aufrufe stufen- oder schrittweise ausgeführt. Betrachten Sie den folgenden Code:

```
$stmt = $ConnHandle->prepare( "SELECT * FROM books");
$stmt->execute();

while ($row = $stmt->fetch()) { // Zeilen zeilenweise abrufen
    print_r ($row);
    // oder etwas Sinnvolleres mit jeder zurückgelieferten Zeile tun
}
$stmt = null;
```

In diesem Code »bereiten« wir den SQL-Code vor und »führen ihn dann aus«. Anschließend gehen wir das Ergebnis mit dem while-Code durch und geben am Ende das Ergebnisobjekt frei, indem wir der Variablen null zuweisen. In diesem einfachen Beispiel sieht das vielleicht nicht so mächtig aus, aber es gibt andere Features, die mit vorbereiteten Anweisungen verwendet werden können. Betrachten Sie nun den folgenden Code:

```
$stmt = $db->prepare("INSERT INTO authors"
    . "(authorid, title, ISBN, pub_year)"
    . "VALUES (:authorid, :title, :ISBN, :pub_year)");

$stmt->execute(array('authorid' => 4,
    'title' => 'Foundation',
    'ISBN' => 0-553-80371-9,
    'pub_year' => 1951)
);
```

Hier bereiten wir die SQL-Anweisung mit vier benannten Platzhaltern vor: *authorid*, *title*, *ISBN* und *pub\_year*. Diese Namen stimmen zufälligerweise mit den Namen der Datenbankspalten überein. Das machen wir nur der Klarheit halber. Die Platzhalternamen können alles sein, was für Sie sinnvoll erscheint. Im execute()-Aufruf ersetzen wir die Platzhalter durch die tatsächlichen Daten, die wir in dieser bestimmten Abfrage verwenden wollen. Einer der Vorteile von vorbereiteten Anweisungen ist, dass Sie den gleichen SQL-Befehl mehrfach verwenden und dabei über das Array jedes Mal unterschiedliche Werte übergeben können. Sie können diese Art von Anweisungsvorbereitung auch mit positionellen Platzhaltern verwenden (ohne ihnen Namen zu geben), die durch ein ? ange-

zeigt werden, das das positionelle Element repräsentiert, das ersetzt werden soll. Betrachten Sie die folgende Abwandlung des vorigen Codes:

```
$stmt = $db->prepare("INSERT INTO authors"
    . "(authorid, title, ISBN, pub_year)"
    . "VALUES (?, ?, ?, ?)");

$stmt->execute(array(4, 'Foundation', 0-553-80371-9, 1951));
```

Dieser Code bewirkt das Gleiche, benötigt dazu aber weniger Code, da der Wertebereich der SQL-Anweisung die zu ersetzenden Elemente nicht benennt und das Array in der `execute()`-Anweisung nur noch die nackten Daten ohne Namen senden muss. Sie müssen nur sicher sein, welche Position die Daten haben, die Sie an die vorbereitete Anweisung senden.

Das war nur ein kurzer Überblick über das, was die neue PDO-Bibliothek für Sie im Datenbankbereich von PHP tun kann. Wenn Sie diese neue Bibliothek ausführlicher erforschen wollen, sollten Sie Ihre Forschungen und Ihre Tests unbedingt durchführen, bevor Sie sie in einer Produktionsumgebung einsetzen. Informationen zu PDO finden Sie unter <http://ca.php.net/pdo>.