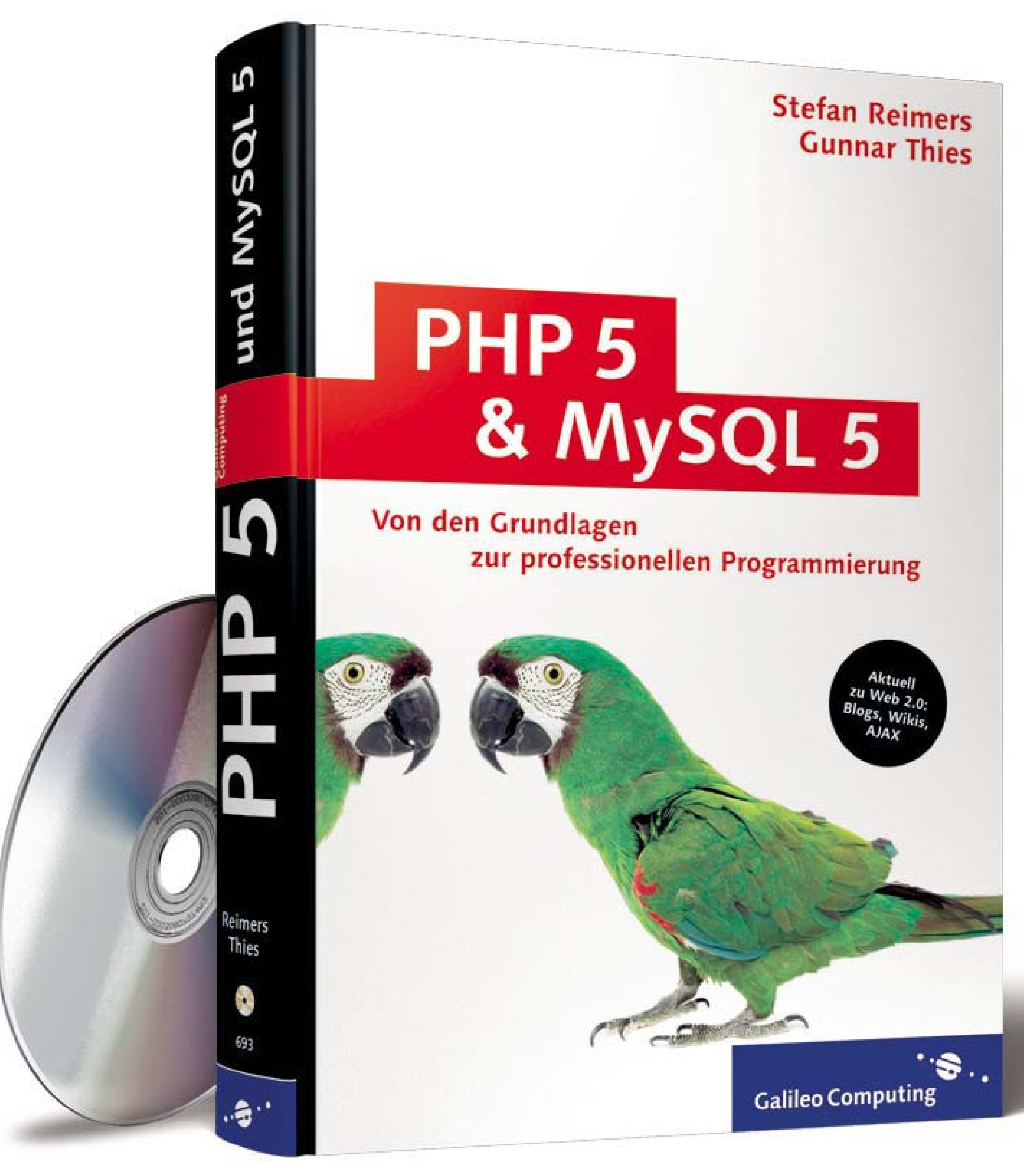

Stefan Reimers
Gunnar Thies
PHP 5
& MySQL 5
Von den Grundlagen
zur professionellen Programmierung
Aktuell
zu Web 2.0:
Blogs, Wikis,
AJAX
Galileo Computing
PHP 5 und MySQL 5
Reimers
Thies
693

Auf einen Blick

Vorwort 13
1 Einleitung 15
2 Grundpfeiler der Webentwicklung 23
3 Installation 37
4 Einführung in PHP 59
5 Objektorientierung in PHP 129
6 Einführung in MySQL 155
7 MySQLi 211
8 Wichtige PHP-Funktionalitäten 245
9 Fortgeschrittenes MySQL 303
10 MySQL Storage Engines 359
11 Sicherheit 377
12 Datenbankentwurf 403
13 Ein Basissystem mit PHP und MySQL 415
14 Sichere Webanwendungen 435
15 Mehrbenutzersysteme 497
16 Blogs und Bildergalerien 545
17 Dauerhafte Objektspeicherung 605
18 Automatische Formularerstellung 647
19 AJAX im Einsatz 669
A PHP-Referenz 691
B MySQL-Referenz 719
C Open Source in der Praxis: Lizenzen 741
D Glossar 747
E Inhalt der CD-ROM 751
Index 753

Inhalt

Vorwort 13

1 Einleitung 15

1.1 Konzeption 18
1.2 Feedback 20

2 Grundpfeiler der Webentwicklung 23

2.1 Das weltweite Netz 24
2.2 Das HTTP-Protokoll 28
2.3 Anbieter und Anwender 32
2.3.1 Anbieter/Ressourceninhaber 32
2.3.2 Nutzer Ihres Systems 34

3 Installation 37

3.1 Microsoft Windows 39
3.1.1 Installation des XAMPP-Basispaketes 39
3.1.2 Installation von XAMPP Lite 42
3.1.3 Starten und Beenden der Server 43
3.2 Linux 44
3.3 Konfiguration von XAMPP 45
3.3.1 Sicherheitslücken schließen 46
3.3.2 Konfigurationsdateien anpassen 49
3.4 Aktualisierung der Komponenten 55

4 Einführung in PHP 59

4.1 Strukturen einer PHP-Seite 60
4.2 Variablen 62
4.2.1 Grundlegende Syntax 62
4.2.2 Datentypen 64
4.2.3 Namenskonventionen 86
4.3 Konstanten 87
4.4 Kommentare 88
4.5 Funktionen 90
4.5.1 Syntax 92
4.5.2 Gültigkeitsbereiche 93
4.5.3 Namenskonventionen 95

4.6 Kontrollkonstrukte ... 95
4.6.1 Bedingte Entscheidungen ... 95
4.6.2 Wiederholungen ... 103
4.7 Vordefinierte Informationen ... 112
4.7.1 Superglobale Arrays ... 112
4.7.2 vordefinierte Konstanten ... 122
4.8 Einbinden externer Dateien ... 124

5 Objektorientierung in PHP ... 129

5.1 Die Modellierungssprache UML ... 129
5.2 Klassen und Objekte ... 131
5.2.1 Konstruktoren und Destruktoren ... 133
5.2.2 Zugriffsmodifizerer ... 134
5.2.3 Funktionen oder Methoden ... 138
5.2.4 Die Implementierung der Klasse Fahrzeug ... 138
5.3 Klassenbeziehungen ... 140
5.3.1 Vererbung ... 140
5.3.2 Klonen ... 146
5.4 Automatisches Laden von Klassen ... 148
5.5 Objektorientierte Fehlerbehandlung ... 149

6 Einführung in MySQL ... 155

6.1 Relationale Datenbanksysteme ... 159
6.2 MySQL und SQL ... 162
6.2.1 Eine Server-Verbindung aufbauen ... 163
6.2.2 Grundlegende SQL-Kommandos ... 171
6.2.3 Datentypen ... 189
6.3 Zugriffswerkzeuge ... 203
6.3.1 MySQL Administrator ... 203
6.3.2 MySQL Query Browser ... 204
6.3.3 phpMyAdmin ... 206

7 MySQLi ... 211

7.1 MySQLi in PHP einsetzen ... 211
7.2 MySQLi-Klassen ... 212
7.2.1 mysqli ... 213
7.2.2 mysqli_result ... 227
7.2.3 mysqli_stmt ... 237

8 Wichtige PHP-Funktionalitäten ... 245
8.1 Datum- und Zeitfunktionen ... 245
8.1.1 Erstellung eines Datums ... 246
8.1.2 Erstellung von Zeitstempeln ... 247
8.1.3 Mikrosekunden ... 248
8.1.4 Umgangssprachliche Zeitkalkulation ... 250
8.2 Datei- und Verzeichnisfunktionen ... 252
8.2.1 Auslesen und Schreiben von Dateien ... 252
8.2.2 Arbeiten mit Verzeichnissen ... 256
8.2.3 Prüfungen im Dateisystem ... 258
8.3 Reguläre Ausdrücke ... 259
8.3.1 Syntax ... 260
8.3.2 Reguläre Ausdrücke in PHP ... 267
8.3.3 Reguläre Ausdrücke in der Praxis ... 270
8.4 PEAR und PECL ... 273
8.4.1 PEAR ... 273
8.4.2 PECL ... 275
8.5 Datenabstraktion ... 276
8.5.1 Abstraktion im Kleinen: DBX ... 277
8.5.2 PDO ... 283

9 Fortgeschrittenes MySQL ... 303
9.1 Benutzerverwaltung ... 303
9.2 Kontrollfluss und Aggregationen ... 308
9.2.1 Bedingte Auswertung ... 308
9.2.2 Aggregationen ... 310
9.3 Performanter Datenbankzugriff ... 316
9.3.1 JOIN-Syntax ... 316
9.3.2 INDIZES ... 320
9.4 Metadaten ... 325
9.4.1 INFORMATION SCHEMA ... 326
9.4.2 Metadaten-Anweisungen ... 329
9.5 Views ... 330
9.5.1 Anlegen ... 331
9.5.2 Editierbare und erweiterbare Sichten ... 335
9.5.3 Ändern und löschen ... 337
9.5.4 Ein praktisches Beispiel ... 338
9.6 Stored Procedures ... 340
9.6.1 Anlegen ... 340
9.6.2 Aufrufen ... 344

9.6.3 Ändern und Löschen 345
9.6.4 Variablen 346
9.6.5 Kontrollstrukturen 347
9.7 Trigger 352
9.7.1 Anlegen 352
9.7.2 Wozu sind Trigger aber notwendig? 355
9.7.3 Löschen 357

10 MySQL Storage Engines 359

10.1 MyISAM 361
10.2 InnoDB 364
10.2.1 Transaktionen 366
10.2.2 Referenzielle Integrität 369
10.3 Memory 372
10.4 Archive 374

11 Sicherheit 377

11.1 Formulardaten und Validierung 379
11.2 Verschlüsselung 384
11.2.1 Ein-Weg-Verschlüsselung 385
11.2.2 Zwei-Wege-Verschlüsselung 388
11.2.3 SSL 392
11.3 Angriffsmethoden und Schutzmaßnahmen 394
11.3.1 Cross-Site-Scripting (XSS) 395
11.3.2 SQL Injection 398
11.3.3 Angriffe auf Sitzungen 399
11.3.4 HTTP Response Splitting 401
11.3.5 Fazit 402

12 Datenbankentwurf 403

12.1 ERM 404
12.2 Normalisierung 408
12.2.1 Normalformen 408
12.2.2 Denormalisierung 412

13 Ein Basissystem mit PHP und MySQL 415

13.1 Konfigurationsdateien 416
13.1.1 common.php 416

13.1.2 settings.php 418
13.1.3 includeAllClasses.php 418
13.2 Die Basisklassen 419
13.2.1 Die Klasse für HTML 419
13.2.2 Die Datenbankverbindungsklasse 422
13.2.3 Die Sicherheitsklasse 428

14 Sichere Webanwendungen 435

14.1 Benutzer authentifizieren 436
14.1.1 Klasse Login 436
14.1.2 Login-Klasse anwenden 440
14.2 Sitzungen mit der Datenbank verwalten 442
14.2.1 Die Klasse der Sitzungsverwaltung 442
14.2.2 Sitzungsverwaltung anwenden 449
14.2.3 Probleme mit der Sitzungsverwaltung 449
14.2.4 »Race Hazard« bei datenbankbasierter Sitzungsverwaltung 451
14.2.5 Benutzerstatus abfragen 452
14.2.6 Benutzer abmelden 454
14.3 Passwörter sicher gestalten 455
14.3.1 Passwortstrategie 455
14.3.2 Zufalls-Passwörter generieren 455
14.3.3 Passwort-Syntax überprüfen 458
14.4 Logging realisieren 460
14.4.1 Daten speichern 461
14.4.2 Klasse Logging 462
14.4.3 Daten mittels JPGraph darstellen 465
14.4.4 Klasse Chart 465
14.4.5 Daten als PDF-Dokument archivieren 469
14.4.6 Klasse PDFMaker 471
14.4.7 PDFMaker-Klasse anwenden 477
14.5 Einfache »Intrusion Detection« implementieren 479
14.5.1 Konfigurationsdatei für das »IntrusionDetectionLogin« 481
14.5.2 Klasse für »Intrusion Detection« 482
14.6 Eigene Fehlerbehandlung einbauen 488
14.6.1 Konfigurationsdatei für Fehlerbehandlung 489
14.6.2 Fehlerbehandlungsklasse 491
14.6.3 Fehlerbehandlung in das Basissystem integrieren 496

15 Mehrbenutzersysteme 497

15.1 Das Hauptproblem: 2 Benutzer – 1 Datensatz 498
15.1.1 Szenario 1: Wer zuerst kommt … ein Änderungsschlüssel 498
15.1.2 Szenario 2: Datensätze explizit sperren 498
15.2 Sperren von MySQL-Datensätzen 499
15.2.1 Die Klasse Locks 500
15.2.2 Beispielanwendung mit Sperren versehen 505
15.3 Transaktionen im praktischen Einsatz 509
15.3.1 Klasse Bank 511
15.3.2 Sichere und unsichere »Banktransaktionen« verwenden ... 514
15.4 Mehrsprachige Weboberflächen 516
15.4.1 Klasse LanguageSupport 517
15.4.2 Mehrsprachige Benutzeroberflächen realisieren 521
15.4.3 Erweiterungsmöglichkeiten 522
15.5 Ein konkretes Mehrbenutzersystem: WIKI 526
15.5.1 Die Klasse Wiki 528
15.5.2 Wiki in der Praxis 539

16 Blogs und Bildergalerien 545

16.1 Blog 546
16.1.1 Klasse Blog 547
16.1.2 Blog in der praktischen Anwendung 558
16.2 Bildergalerie 563
16.2.1 Klassenübersicht: Bildergalerie 564
16.2.2 Klasse AbstractPictureGallery 568
16.2.3 Klasse AdminPictureGallery 569
16.2.4 Klasse PictureGallery 590
16.2.5 Klasse Picture 593
16.3 Bildergalerie als Flash-Variante 597
16.3.1 Klasse FlashPictureGallery 598
16.3.2 Klasse Picture erweitern 600

17 Dauerhafte Objektspeicherung 605

17.1 Persistenz 605
17.2 Umsetzung persistenter Objekte 605
17.2.1 Klasse Attribute 606
17.2.2 Klasse DBO 608

17.3 Gültigkeitsprüfung von Parametern 625
17.3.1 Konfigurationsdatei der Gültigkeitsprüfung 625
17.3.2 Gültigkeitsprüfungsklasse 629
17.3.3 Gültigkeitsprüfung in die Klasse DBO einbauen 644

18 Automatische Formularerstellung 647

18.1 Klasse SimpleAutomaticFormular 650
18.2 Automatische Formulargenerierung anwenden 666
18.3 Verbesserungsvorschläge 667

19 AJAX im Einsatz 669

19.1 Beispiel: Blog-Suchmaschine 670
19.1.1 Klasse AJAX 671
19.1.2 AJAXJavaScript.js 680
19.1.3 PHP-Skripte für das AJAX-Beispiel 686

Anhang 691

A PHP-Referenz 691
B MySQL-Referenz 719
C Open Source in der Praxis: Lizenzen 741
C.1 GPL 741
C.2 LGPL 743
C.3 BSD 744
C.4 PHP License 745
C.5 MySQL-Lizenz 745
C.6 Lizenzen im Überblick 746
D Glossar 747
E Inhalt der CD-ROM 751

Index 753

Herzlich Willkommen!

1 Einleitung

Gut zehn Jahre nach ihrer Entstehung wagen PHP und MySQL mit erfolgreichen Vorgängern im Rücken jeweils einen fünften Anlauf. Das israelische Unternehmen Zend Technology, Inc. und nicht zuletzt die vielen freiwilligen Entwickler haben es geschafft, die jüngste PHP-Generation schon vor dem Jubiläum zu veröffentlichen. Und während die Geburtstagsfeiern im Sommer 2005 noch im Gange waren, wurde bereits an PHP 5.1 gearbeitet und pro forma der Begriff PHP 6 in den Raum geworfen. Die Datenbankprogrammierer aus Schweden hatten zu diesem Zeitpunkt eine Betaversion von MySQL 5 zum Testen auf ihren Servern anzubieten. Die letztendliche Veröffentlichung folgte Ende 2005.

Mit dem Schritt zur Version 5 werden sowohl das Datenbankmanagementsystem MySQL als auch die Skriptsprache PHP ein Stück erwachsener. Als Open-Source-Entwicklungen hatten beide Projekte in der Vergangenheit mit dem Vorurteil zu kämpfen, man könne sie nicht mit gutem Gewissen im professionellen produktiven Umfeld einsetzen. Die enge Verknüpfung von PHP- und HTML-Code erschwere die klare Trennung von Logik und Darstellung. Und bei MySQL würden Funktionalitäten fehlen, die man von anderen, teils auch kostenfrei nutzbaren Datenbankmanagementsystemen gewohnt sei. Sicherlich sind das nicht die einzigen Bedenken gewesen, aber zumindest diejenigen, die am lautesten gerufen wurden. Wie nun schicken sich die Nachfolger in der neuen Version an, dieses Image loszuwerden?

PHP 5 vollzieht den Wechsel zur *Zend Engine 2*, dem Herzstück der Skriptsprache. Sie gibt PHP einen neuen Unterbau und verleiht Ihren Skripten neuen Schub. Aber auch für Sie als Programmierer, der sich vielleicht schon mit PHP 4 angefreundet hat, bedeutet der Wechsel Änderungen an mancher Stelle bzw. neue Funktionen und Möglichkeiten.

Die sicher am meisten auffallende und beschworene Neuerung für den Nutzer ist das überarbeitete Objektmodell. Zwar gab es Objekte als native Sprachwerte

schon in PHP 3, jedoch fehlte dem versiert objektorientiert arbeitenden Programmierer im Vergleich zu anderen Sprachen wie Java oder C++ so manches gewohnte Feature. Darüber hinaus verhielt sich der Code für Umsteiger aus oben genannten Programmiersprachen nicht immer wie erwartet. Die Erkenntnis, dass Objekte anders zu behandeln sind als die übrigen Datentypen, fand jedoch erst in PHP 5 Einzug bzw. in die Zend Engine 2. Das überarbeitete Objektmodell beseitigt diese alten Missstände folglich. Darüber hinaus unterstützt es die Konzepte der Objektorientierung in weit größerem Umfang; darunter sind beispielsweise

- Interfaces und abstrakte Klassen,
- public, protected und private Eigenschaften sowie
- Konstruktoren und Destruktoren.

PHP 5 macht auch einen Schritt in Richtung Geschäftswelt, nicht zuletzt aufgrund der Community, die mit selbst programmierten Erweiterungen die Anforderungen moderner und professioneller Informationstechnologie aufgreift. Die Unterstützung von XML, serviceorientierten Architekturen und Web Services gehört ebenso dazu wie die Erstellung von Client-seitiger Software – also PHP-Applikationen mit eigener grafischer Benutzeroberfläche, die direkt beim Anwender ohne einen Webserver laufen – und die Adressierung von Sicherheitsfragen.

Nicht zuletzt bekommt auch MySQL in PHP eine neue Schnittstelle: *ext/mysqli* – ein Pseudonym für »Extension/MySQL improved«. Auch Datenbankschnittstellen kommen also in die Jahre, zumal die MySQL-Client-Bibliothek, auf der sowohl *ext/mysqli* als auch sein Vorgänger *ext/mysql* beruhen, weiter entwickelt worden ist. Die neue Erweiterung wird für MySQL 4.1 und alle späteren Versionen eingesetzt, also auch für MySQL 5.

Auf Datenbankseite kümmert man sich in MySQL 5 ebenfalls um neue Funktionen und rückt damit dem SQL-Standard ein Stück näher.

Hintergrundwissen

SQL steht für »Structured Query Language« und ist die standardisierte Sprache für relationale Datenbanksysteme, wie beispielsweise MySQL, Oracle oder IBM DB2. Im Standard wird bestimmt, welche Funktionen von Datenbanken unterstützt werden müssen, die SQL als Anfragesprache verwenden. Kaum ein System erreicht hundertprozentige Standardkonformität, zumal der Standard laufend weiter entwickelt wird.

Wir bedienen uns in diesem Buch der im Web üblichen Fachbegriffe. Falls Ihnen einmal ein Begriff oder eine Abkürzung unbekannt sein sollte und nicht im Text erklärt ist, können Sie sie im Glossar in Anhang D dieses Buches nachschlagen.

Zusätzlich möchten wir Sie zu einer Online-Recherche ermuntern, bei der Sie zu mehr und breiteren Ergebnissen gelangen werden. Gute Anlaufstellen sind die Suchmaschine Ihrer Wahl und Wikipedia.

Man war sich bei der MySQL AB sehr wohl bewusst, welche Funktionen fehlten und von den Anwendern gewünscht waren, nicht nur, weil man in Foren und Newsgroups danach ausgefragt wurde. Das MySQL-Handbuch enthält eigens Abschnitte dazu, wie die fehlenden Funktionen in älteren Datenbankversionen mit einigen Handgriffen ausgeglichen werden können. Diese so genannten Workarounds wurden meist nicht in der Datenbank selbst, sondern auf Applikationsebene durchgeführt, also beispielsweise von einem PHP-Skript. Der zusätzliche Aufwand, der durch die Kommunikation und den Datentransport zwischen Datenbank und Applikationsschicht entsteht, macht solche Kniffe vergleichsweise langsam. Schneller geht es, wenn die Daten in der Datenbank verbleiben und dort verarbeitet werden.

Mit den neuen Funktionen aus MySQL 5 ist man wieder auf dem neuesten Stand und dem SQL:2003-Standard näher als mancher Konkurrent; die neuen Funktionen sind im Wesentlichen

- aktualisierbare Views,
- Trigger und
- Stored Procedures.

Neben den Neuerungen, die aus dem SQL-Standard herrühren, bereichern die Entwickler ihr Datenbanksystem um neue Storage Engines. Dabei handelt es sich um Grundkonfigurationen für einzelne Tabellen, die je nach Anforderung der Anwender z.B. auf Datensicherheit oder Anfragegeschwindigkeit ausgelegt sein können. Mit der neuen Vielfalt an Storage Engines ermöglicht MySQL den Einsatz in spezialisierten Anwendungsfällen, der sich letztendlich in Leistungszuwächsen beim Nutzer niederschlägt.

Um noch einmal die anfängliche Frage aufzugreifen: PHP 5 und MySQL 5 sind mit ihren neuen Möglichkeiten durchaus für den professionellen Einsatz gerüstet. Auch als Open-Source-Entwicklung muss man sich nicht hinter der Konkurrenz verstecken und das auch dank der lebhaften Anhängerschaft auf beiden Seiten. Sicherlich ist man nicht aufeinander angewiesen. Eine der Stärken von PHP ist es, Schnittstellen zu einer Vielzahl von Datenbanken zu bieten. Darüber hinaus hat sich eine breite Palette von Datenbankabstraktionsklassen entwickelt, die den einheitlichen Umgang mit unterschiedlichen Datenbanken garantieren soll. MySQL seinerseits bietet unter anderem Programmierschnittstellen zu Java, Tcl, Python und nicht zuletzt die offene Schnittstelle ODBC. Dennoch hat sich das

»dynamische Duo« den jahrelangen Erfolg zu großen Teilen gegenseitig zu verdanken. Bereits seit Mitte der neunziger Jahre wird MySQL von PHP unterstützt. Vorbild dafür war die bereits damals vorhandene *msql*-Schnittstelle, die von Rasmus Lerdorf, dem Autor der ersten PHP-Versionen, per Suchen-Ersetzen umgebaut wurde. Mittlerweile ist es allerdings die Regel, dass Webmaster Zugriff auf PHP und MySQL von ihren Hostern bekommen.

Vom einfachen Besucherzähler – das war der Ursprung von PHP – bis hin zu datenbankgestützten Applikationen: Dies alles lässt sich dank PHP und MySQL nach kurzer, aber intensiver Einarbeitungszeit erstellen. Diese Hilfe soll Ihnen unser Buch leisten. Viel Spaß und Erfolg bei der Lektüre.

1.1 Konzeption

Wir haben das Buch für Sie in drei unterschiedliche Teile gegliedert. Allem voran haben wir jedoch noch ein Kapitel gestellt, das Sie in die Welt des World Wide Web einführen soll.

Der erste Teil richtet sich an unerfahrene Leser, denen in den vier Kapiteln des Teils eine Starthilfe gegeben werden soll, mit PHP und MySQL umzugehen. Der Teil beinhaltet die Kapitel:

Installation

Am Anfang der Programmierung steht die Einrichtung einer Testumgebung. Sie müssen also MySQL und PHP auf ihrem eigenen Computer installieren. Wir stellen Ihnen ein Paket vor, das Ihnen die Komponenten in einem Rutsch lauffähig und vorkonfiguriert auf die Festplatte bringt. Damit müssen Sie den Webserver, die Datenbank und die Skriptsprache nicht einzeln installieren und einrichten.

Einführung in PHP und MySQL

Variablen, Funktionen und Schleifen gehören unter anderem zu den Grundbausteinen von Programmiersprachen und natürlich auch zu PHP. Wie Sie diese Konstrukte im Einzelnen nutzen, erfahren Sie ausführlich in diesen Kapiteln. Darüber hinaus beschreiben wir an dieser Stelle die Objektorientierung in PHP. Damit Sie sich zwischen Datenbankmanagementsystem, Datenbank, Tabellen, Attributen und Datensätzen nicht verhaspeln, erklären wir in der Einführung in MySQL den Unterschied. Außerdem lernen Sie die gängigsten SQL-Kommandos kennen, mit denen Sie den Datenbank-Alltag bestreiten können.

Im zweiten Teil orientieren wir uns mehr in die theoretische Breite. Die Kapitel lassen sich dazu in drei Unterkategorien gruppieren.

Skriptkapitel

Die Grundbausteine der Programmierung wie Schleifen oder Funktionen bleiben davon unberührt. Objektorientierung ist mehr ein gedankliches Konstrukt der Programmierung. Daraufhin stellen wir Ihnen eine Reihe von Funktionalitäten vor, die zum »täglich Brot« gehören, insbesondere die neue Datenbankschnittstelle MySQLi. Abschließend möchten wir Sie noch mit der Datenbankabstraktion vertraut machen, mit der Sie ohne viel Aufwand auch andere Datenbankmanagementsysteme als MySQL ansprechen können.

Datenbankkapitel

Zum einen wollen wir Ihnen ein breiteres Wissen der Anfragesprache SQL vermitteln. Sie sollen lernen, wie die Benutzerverwaltung von MySQL funktioniert, wie Sie Ihre Abfragen optimieren können und wie Sie Informationen zu dem gespeicherten Datenbestand bekommen. Zum anderen wollen wir Ihnen die neuen Funktionalitäten näher bringen und die Storage Engines vorstellen.

Applikationskapitel

In diesem Teil gehen wir auf verschiedene Aspekte der Webentwicklung ein, wobei wir MySQL und PHP aus größerer Entfernung betrachten. Zu diesen Teilbereichen zählen wir den Entwurf eines »guten« Datenbankschemas und die Sicherheit eines webbasierten Systems.

Der ausgiebige dritte Teil des Buches widmet sich der praktischen Anwendung.

Grundfunktionen einer Webanwendung

Obwohl die einzelnen Programme des dritten Teils größtenteils überschneidungsfrei sind, liegt allen dennoch ein Basissystem zu Grunde. Die Funktionalität beschränkt sich zunächst auf die Datenbankschnittstelle sowie HTML-Grundfunktionalitäten, wächst aber mit fortschreitenden Kapiteln um die jeweils benötigten Module an.

Programmierung größerer Projekte

Zwei Kapitel befassen sich mit der Programmierung von Webapplikationen, die möglichst sicher sind und auf die Bedürfnisse von Benutzern eingehen können.

So werden in Kapitel 14, *Sichere Webanwendungen*, einige Sicherheitsaspekte der PHP-Programmierung praktisch gezeigt. Kapitel 15, *Mehrbenutzersysteme*, befasst sich daraufhin mit mehrsprachigen Weboberflächen, der Sicherung der Konsistenz der Daten bei gleichzeitiger Verwendung mehrerer Benutzer und realisiert eine praktische Anwendung: das »Wiki«.

Blogs und Bildergalerien

Kapitel 16 beschäftigt sich mit dem Einsatz von Blogs und Fotogalerien, die es erlauben, die eigene Site im Internet »persönlicher« zu gestalten.

Fortgeschrittene Themen der Webprogrammierung

Eines der Probleme mit Webapplikationen ist in der Zustandslosigkeit der PHP-Objekte bei der Programmierung zu sehen. Denn nach dem Beenden eines Skriptes sind die Objekte zerstört. In Kapitel 17, *Dauerhafte Objektspeicherung*, werden wir einen Ansatz programmieren, der es erlaubt, Objekte persistent in der Datenbank abzulegen. Anschließend beschäftigen wir uns mit der automatischen Generierung von HTML-Formularen aus der Datenbank, was einem sehr viel lästige Arbeit abnehmen kann. Als interaktiver Aspekt von XML wird AJAX vorgestellt, womit das Nachladen von Daten ohne »echten« Reload möglich wird.

Im Einführungs- und Theorieteil (den ersten beiden Teilen) verwenden wir in den Beispielen und Skripten deutsche Bezeichner für Variablen und Datenbankstrukturen. Ziel ist es, den Lerneffekt nicht noch durch englische Bezeichner zu erschweren. Im dritten Teil hingegen, dem Praxisteil, setzen wir durchweg auf englische Bezeichner. Sie werden bei der Arbeit mit quelloffenen Bibliotheken und sonstigen Skripten, die Sie aus dem Internet beziehen, größtenteils auf englisch programmierten und dokumentierten Code treffen. Unser Praxisteil soll Sie damit vertraut machen.

1.2 Feedback

Bei all den Vorzügen dieses Buch, das Sie in Händen halten, hat es leider einen Nachteil: Nach Drucklegung können wir es nicht immer auf den aktuellen Stand bringen, ohne eine neue Auflage herauszubringen. Die Entwicklung von PHP und MySQL geht derzeit laufend weiter.

Um dennoch aktuelle Fakten zu den hier behandelten Themen zu erhalten, haben wir jeweils Sektionen auf unseren Webseiten

http://www.gunnar-thies.de und

http://www.stefan-reimers.de

eingerichtet. Dort finden Sie Beispiele, Fehlerkorrekturen und nicht zuletzt Informationen über uns. Der Weg über die Webseiten ist auch die beste und schnellste Option, mit uns in Kontakt zu treten. Es interessiert uns, wenn Sie an einer Stelle den Faden verloren haben oder bestimmte Aspekte in diesem Buch vermissen.

Natürlich können Sie sich auch an den Verlag wenden. Kontaktmöglichkeiten finden Sie auf **http://www.galileocomputing.de.** Die Anfragen werden dann an uns weitergeleitet.

PHP gilt als sehr leicht zu erlernende Sprache und ist deshalb ein guter Einstieg in die Programmierung. Das folgende Kapitel legt den Grundstein für Ihre dynamischen Webseiten und Applikationen. Also, los geht's!

4 Einführung in PHP

Die Arbeit an Webapplikationen lässt sich mit den richtigen Werkzeugen deutlich erleichtern. Deshalb möchten wir Ihnen für die Programmierung mit PHP zweierlei Dinge ans Herz legen. Dazu gehört zum einen ein vernünftiges Programm, mit dem Sie Ihre Skripte schreiben und das Sie auch aktiv dabei unterstützt, also ein PHP-fähiger Editor. Außerdem sollten Sie die Befehlsreferenz, also quasi das Handbuch, in dem alle vorhandenen Befehle erklärt sind, jederzeit zur Hand haben. In manche Editoren ist das Handbuch bereits integriert.

Editoren gibt es zuhauf. Jeder hat so seine Vor- und Nachteile und ist mehr oder weniger komfortabel. Die Erfahrung zeigt, dass die Wahl des richtigen Programms stark von den eigenen Präferenzen abhängt. Was dem einen gefällt, empfindet der andere als störend, und so hat jeder Editor seine Fangemeinde. Eine Empfehlung zugunsten eines Programms wollen wir deswegen nicht aussprechen, aber Sie auch nicht mit der Wahl alleine lassen. Es gibt eine Reihe von Funktionen, die ein guter Editor beherrschen sollte:

- **Syntax-Highlighting**
  Dabei wird der Code in unterschiedlichen Farben hervorgehoben, wodurch die Übersichtlichkeit stark verbessert wird.
- **Fehlererkennung**
  Wenn einfache Syntaxfehler wie ein Vertippen oder fehlende Zeilenabschlusszeichen deutlich gekennzeichnet werden, können Sie dies an Ort und Stelle beheben, ohne dass Sie die Skripte im Browser testen und auf eine Fehlermeldung warten müssen.
- **Inhaltsübersichten zu verwendeten Dateien**
  Mitunter können Skripte sehr viele Funktionen enthalten. Ein Inhaltsverzeichnis bringt Sie schneller zu der Stelle im Skript, zu der Sie wollen.

- **Codevervollständigung**
  Wenn Sie einen Befehl zu tippen beginnen, werden Ihnen gültige Befehlsnamen vorgeschlagen. Das geht nicht nur schneller, sondern bewahrt Sie auch vor Schreibfehlern.

Die meisten Editoren können Sie sich über das Internet besorgen. Bekannte Vertreter bei den reinen PHP-Editoren sind beispielsweise *Zend Studio* oder *PHP-Edit*. Mit *Eclipse* oder *UltraEdit* greift man zu Programmen, die nicht auf PHP beschränkt sind, aber dennoch effektives Arbeiten erlauben. Die Spartaner, die nur beschränkt komfortabel sind, sind und bleiben *Notepad* unter Microsoft Windows bzw. *vi* unter Linux.

Die Befehlsreferenz können Sie sich von der offiziellen Webseite **http://www.php.net** herunterladen. Darin enthalten sind Definitionen aller verfügbaren Befehle und Funktionen mit Beschreibungen, wie sie zu benutzen sind. Die Referenz ist in HTML oder für Windows auch als navigierbare Hilfsdatei (*Compiled HTML Help – CHM*) verfügbar. Besser ist in jedem Fall die Online-Version, zumal sie zusätzlich mit Kommentaren und Beispielen von Programmierern auf der ganzen Welt angereichert ist. Wenn Sie also einen ständigen Zugang zum WWW haben, sei Ihnen diese Version empfohlen.

4.1 Strukturen einer PHP-Seite

PHP-Code wird in einfachen Textdateien gespeichert. Die Befehle müssen in gesonderten PHP-Bereichen von anderweitigem Inhalt wie beispielsweise HTML getrennt sein (Listing 4.1).

```
<html>
   <head/>
   <body>
   <?php
      echo "<h1>Willkommen bei PHP5 und MySQL 5</h1>";
   ?>
   </body>
</html>
```

Listing 4.1 PHP-Code kann in HTML eingebettet werden

Auf den ersten Blick sieht das Listing 4.1 aus wie eine ganz normale HTML-Datei, in der zuerst der Kopf und dann der Körper definiert werden. Der PHP-Bereich beginnt, sobald er durch ein `<?php` eingeleitet wird. Ab dieser Zeile werden alle Anweisungen vom PHP-Parser abgearbeitet, bis der Bereich mit einem `?>` endet. Code außerhalb des gesonderten Bereichs wird vom Parser überlesen.

Hinweis

Die Bereiche, in denen PHP vorkommen darf, können neben der vorgestellten Variante noch auf unterschiedliche Weise von anderweitigem Code getrennt werden, unter anderem durch ein Tag `<script language="php"/>` ähnlich zu JavaScript. Die von uns bevorzugte Alternative ist jedoch konform zu XHTML und XML.

Sie können in einer Datei beliebig viele Bereiche für PHP einfügen. Wichtig ist, dass Sie jeden Bereich wieder schließen, bevor Sie mit anderweitigem Inhalt fortfahren. Jede Anweisung, die der Parser nicht richtig deuten kann, wird mit einer Fehlermeldung quittiert.

Mit dem Befehl `echo` erzeugen Sie eine Ausgabe am Bildschirm. Die Inhalte aller Ausgaben erscheinen nach dem Parsen in der resultierenden Datei. Alternativ zu `echo` können Sie auch den Befehl `print` verwenden, der sich identisch verhält. Im vorigen Beispiel wird nur eine einzige Ausgabe gemacht. Nachdem der Parser das gesamte Skript verarbeitet hat, enthält es nur noch HTML:

```
<html>
   <head/>
   <body>
      <h1>Willkommen bei PHP5 und MySQL 5</h1>
   </body>
</html>
```

Listing 4.2 b: Ausgabe des obigen Skripts nach dem Parsen

Die Definition des HTML-, Kopf- und Körperbereichs wurde vom Parser links liegen gelassen, der Befehl `echo` resultiert in einer Überschrift erster Ordnung und alle darauf folgenden Teile sind ebenfalls unberührt geblieben.

Wie sich eine Datei, die Sie neu erstellen, von Grund auf am besten beginnen lässt, hängt von Ihrem Verständnis von PHP ab. Wenn es für Sie nur ein Mittel ist, um Ihre HTML-Webseiten mit ein wenig Dynamik auszustatten, dann betten Sie am besten die PHP-Bereiche in Ihr fertiges HTML ein. Wenn Sie hingegen das Ziel verfolgen, Webapplikationen zu erstellen und unter Umständen nicht nur HTML, sondern auch PDF-Dateien, Grafiken oder auch Excel-Tabellen auszugeben, dann sollte PHP für Sie die Basissprache sein, mit der Sie eine Datei beginnen. Ein Skript besteht dann in erster Linie aus einem einzigen PHP-Bereich von der ersten bis zur letzen Zeile. Dateiinhalte werden über `print` oder `echo` ausgegeben und mit den richtigen HTTP-Headern versehen, so dass sie von Programmen wie Ihrem Browser auch richtig verstanden werden.

4.2 Variablen

Variablen dienen dazu, Daten während der Abarbeitung eines Skriptes zu speichern. Wenn Sie eine Variable deklarieren und an einen Wert binden, ist dies nichts anderes, als dem Wert einen Namen zu geben. Wenn Sie genau diesen Wert zu einem späteren Zeitpunkt in Ihrem Skript wieder benötigen, können Sie ihn über den vergebenen Namen ansprechen.

Der Wert einer Variablen ist veränderlich. Wenn Sie Daten einer Variablen zuweisen, die bereits besteht, wird der bisherige Wert dadurch mit dem neuen überschrieben. Die Bindung eines Namens an einen Wert wird über den Zuweisungsoperator vorgenommen, symbolisiert durch ein einfaches Gleichheitszeichen (»=«). Eine gültige Zuweisung sieht dann wie folgt aus:

```
Name = Wert;
```

4.2.1 Grundlegende Syntax

Jeder Variablenname ist in einem PHP-Skript bzw. in einem Gültigkeitsbereich eindeutig. Gültige Namen werden immer eingeleitet durch ein Dollarzeichen (»$«), darauf folgt zwingend ein Unterstrich oder ein Buchstabe; danach kann der Name eine beliebige Abfolge von Buchstaben, Zahlen und Unterstrichen (»_«) enthalten.

```
<?php
$ersteVariable = 'Text';    //ein korrekter Name
$_zweiteVariable = '1234';    //auch ein korrekter Name
$3teVariable = 1.2;    //kein korrekter Name
?>
```

Nachdem eine Variable einmal belegt ist, lässt sich deren Inhalt so lange über den vergebenen Namen ansprechen, bis sie mit dem Befehl `unset()` gelöscht oder das Skript beendet wird. Ob eine Variable besteht, erfahren Sie über den Befehl `isset()`, der Ihnen wahr oder falsch zurück liefert, je nachdem, ob die Variable im aktuellen Gültigkeitsbereich definiert ist. Bei Variablennamen unterscheidet PHP zwischen Groß- und Kleinschreibung. Die folgenden beiden Variablen sind demnach nicht identisch:

```
<?php
$var = 15;
$Var = 51;   //das hat keinen Einfluss auf $var
$var = $Var;   //das verändert der Wert von $var
?>
```

Die Variablen `$var` und `$Var` sind also vollständig unabhängig voneinander. Nach der Belegung von `$Var` enthält die Variable `$var` immer noch den Wert 15. Erst durch die letzte Zuweisung wird der Wert von `$var` überschrieben und beträgt danach 51.

PHP unterstützt zwei Arten von Zuweisungen. Obiges Beispiel verwendet die Methode **pass by value**. Wird einer Variablen der Wert einer anderen zugewiesen, wird lediglich dessen Wert kopiert. Die Variablen sind danach immer noch unabhängig voneinander, d.h., eine Änderung von `$Var` im obigen Beispiel hätte keinen Einfluss auf `$var`. Zuweilen ist es aber durchaus erwünscht, dass zwei Variablen voneinander abhängig sind. Dazu dient die Methode **pass by reference**.

```
<?php
$quellVariable = 10;
$zielVariable = &$quellVariable;
echo $zielVariable;
$quellVariable = 17;
echo $zielVariable;
?>
```

Wie Sie sehen, ist der Unterschied in der Schreibweise nicht groß; in der Tat besteht er nur in dem kaufmännischen Und `&`, durch das die beiden Variablen logisch miteinander verknüpft werden. Eine Änderung von `$quellVariable`, beispielsweise die Zuweisung des Wertes 17, beeinflusst auch `$zielVariable`. Während die erste Ausgabe noch den Wert 10 erzeugt, ist das zweite Ergebnis 17.

Hintergrundinformationen

Um das Prinzip von Referenzen weiter zu verdeutlichen, muss man einen Blick in die Tiefen von PHP riskieren: Eine Variable ist ein Zeiger auf eine Stelle im Speicher, an der ihr Wert hinterlegt wird. Bei einer Zuweisung mit der Methode pass by value wird der Wert einer Quellvariablen an anderer Stelle im Speicher dupliziert, genau dorthin, wo der Wert der Zielvariablen hinterlegt ist. Eine Zuweisung über pass by reference hingegen leitet den Zeiger der Zielvariablen um, so dass er auf die Speicherstelle weist, auf die auch die Quellvariable zeigt.

Die Referenzierung mehrerer Variablen gilt natürlich in alle Richtungen:

```
<?php
//Zweiter Teil des Beispiels
$zielVariable = 20;
echo $quellVariable;
?>
```

Nach der neuen Belegung von `$zielVariable` ergibt die Ausgabe von `$quellVariable` ebenfalls 20.

PHP steht Ihnen noch eine ganze Reihe weiterer Freiheiten in der Syntax zu. Zum einen ist es erlaubt, mehrere Dollarzeichen nacheinander zu schreiben. Die Auswertung des Variablennamens erfolgt dann derart, dass von innen nach außen – also von rechts nach links – versucht wird, mehrere Namen aufzulösen und den entstehenden Wert mit dem nächsten Dollarzeichen als neue Variable anzusehen:

```
<?php
$pseudoName = 'variable';
$variable = 1000;
echo $$pseudoName;
?>
```

Die Ausgabe dieses Skriptes ist in der Tat die Zahl 1000. Es handelt sich also um einen variablen Variablennamen. Analog existieren variable Funktionsnamen. Die Funktion `strlen()` ist standardmäßig in PHP definiert und ermittelt die Länge einer Zeichenkette. Sie lässt sich nun wie folgt aufrufen:

```
<?php
$string = 'abcd';
$pseudoName = 'strlen';
echo $pseudoName($string);
?>
```

Auch hier wird das Skript fehlerfrei ausgeführt, die Funktion ergibt erfolgreich das Ergebnis 4, nämlich die Länge der Zeichenkette »abcd«.

4.2.2 Datentypen

Ihnen ist vielleicht aufgefallen, dass wir die Zuweisung von Variablen in den vorigen Beispielen unterschiedlich gehandhabt haben. Teils sind die Werte ohne Hochkommata, teils auch mit definiert worden. Das liegt an den verschiedenen Datentypen, die diese Werte haben.

Indem Sie Datentypen für Werte definieren, legen Sie gleichzeitig die Operationen fest, mit denen Sie die Daten bearbeiten können. Zum Beispiel lässt sich die Zeichenkette `"Das ist ein ganz normaler Satz"` schwerlich zu der Zahl 10 addieren, weil sich mit Sätzen im Allgemeinen nicht rechnen lässt. Ebenfalls ist durch den Datentyp festgelegt, welche Werte für die Operationen gültig sind.

All dies mag für Sie logisch und damit wie Haarspalterei erscheinen. Genauso intuitiv, wie Sie Daten in Ihrem Sprachgebrauch den Kategorien Zahlen, Buchstaben und so weiter zuordnen, können Sie sie auch in PHP einsetzen. Dafür stehen Ihnen acht Datentypen zur Verfügung (Tabelle 4.1).

Datentyp	Bezeichner	Beispiel / Beschreibung
Zeichenketten	String	‘a’, ‘Wort’, ‘ganze Sätze’
Ganzzahlige Werte	Integer	1, 2, 3, 100, 1000, –1, –15
Fließkommazahlen	Float/Double	1.5, 11.99999, 17.4e2
Boolesche Werte	Boolean	true, false
Arrays	Array	Mehrwertiger Datentyp
Objekte	Object	Mehrwertiger Datentyp
Ressourcen	Resource	Referenz auf externe Quellen
Null	Null	Typ für Variablen ohne Wert

Tabelle 4.1 PHP-Datentypen

PHP ist eine schwach getypte Sprache und nimmt Ihnen die Vergabe und Konvertierung von Datentypen weitgehend ab: Variablen können ihren Datentyp im Verlauf eines Skriptes mehrfach ändern, je nachdem in welchem Kontext sie gebraucht werden. Darin besteht ein großer Unterschied zu anderen Programmiersprachen wie Java, in denen eine Variable zum einen deklariert – also festgelegt – werden muss und seinen Typ danach nie verändern kann. Damit werden in PHP Operationen wie die folgende möglich:

```
echo 2 + '10 Säcke Kartoffeln';
```

Die Ausgabe der Anweisung ist 12. Der PHP-Parser erkennt, dass es sich bei der Operation um eine Addition handelt und die Operanden folglich beide Zahlen sein müssen. Der eine Operator ist bereits eine Zahl (2), der andere wird in eine Zahl umgewandelt. Für die Typkonvertierung gibt es in PHP mehrere Regeln. Zeichenketten werden beispielsweise in Zahlen überführt, indem von links nach rechts so viele Zeichen verwendet werden, wie sich als Zahl interpretieren lassen.

Natürlich können Sie Datentypen auch per Befehl abfragen und verändern. Für die meisten Datentypen sind eigene Funktionen vordefiniert (Tabelle 4.2).

Datentyp	Konvertierung	Abfrage
`String`	`(string)`	`is_string`
`Integer`	`(int), (integer)`	`is_int, is_integer`
`Float/Double`	`(float), (double)`	`is_float, is_double`
`Bool`	`(bool), (boolean)`	`is_bool`
`Array`	`(array)`	`is_array`
`Object`	`(object)`	`is_object`
`Null`	-	`is_null`

Tabelle 4.2 Funktionen zur Typprüfung und -konvertierung

Aus der Zeichenkette wird durch den folgenden Befehl eine Zahl:

```
$zahl = (int) '10 Säcke Kartoffeln';
```

Dies lässt sich mit `is_int($zahl)` bestätigen.

In der Tabelle sind nur die grundlegenden Funktionen aufgeführt. PHP stellt noch eine Reihe weiterer Funktionen zur Verfügung, beispielsweise `is_numeric()`, die Werte auf Zahlen oder Zeichenketten auf Zahlen prüft. Eine Konvertierung in Ressourcen oder Nullwerte ist nicht sinnvoll und deswegen nicht vorgesehen.

Zusätzlich existieren die Funktionen `gettype()` und `settype()`, mit denen sich die Datentypbezeichnungen abfragen und Typen setzen lassen. Im Folgenden wollen wir Ihnen die Datentypen einzeln vorstellen. Wir zeigen Ihnen die Besonderheiten auf und geben einen Überblick über die möglichen Operationen.

Strings

Zeichenketten stehen immer in einfachen oder doppelten Anführungszeichen:

```
<?php
echo 'ein gültiger String';
echo "noch ein gültiger String";
```

Strings können alle Zeichen enthalten, also neben Buchstaben auch Zahlen oder Sonderzeichen. Der Ausdruck `'1234'` aus einem der oberen Beispiele ist ein String, auch wenn er nur aus Ziffern besteht. Und das allein, weil er in Hochkommata eingeschlossen ist. Wichtig ist, dass Sie Anführungsstriche konsistent verwenden. Wenn Sie eine Zeichenkette mit einem doppelten Anführungsstrich beginnen, muss er auch mit einem doppelten enden.

Einfache Anführungsstriche

Eine Zeichenkette, die in einfache Anführungsstriche eingeschlossen ist, wird fast ohne Verarbeitung ausgegeben. Das bedeutet, darin enthaltene Variablen oder Sonderzeichen werden nicht ausgewertet. Die Anweisungen

```
$a = 'Test';
echo 'Ausgabe einer $a-Variablen';
```

führt nicht dazu, dass der Satz »Ausgabe einer Test-Variablen« auf Ihrem Bildschirm erscheint. Stattdessen ist »Ausgabe einer $a-Variablen« zu lesen. Ebenso wenig erscheint nach

```
echo 'Ausgabe einer \r\n Variablen';
```

ein Zeilenumbruch.

Hinweis

Zeilenumbrüche werden in unterschiedlichen Betriebssystemen verschieden angegeben. Unix und Linux verwenden einen einfachen Zeilenvorschub (Linefeed – \n), Mac OS benutzt einen so genannten Wagenrücklauf (Carriage Return – \r) und in Windows-Umgebungen müssen Sie beides nacheinander schreiben (\r\n).

Das Tag für einen Zeilenumbruch in HTML ist unabhängig von dem benutzten Betriebssystem das <br />, XHTML-konform mit einem abschließenden Slash nach einem Leerzeichen.

Doppelte Anführungsstriche

Strings in doppelten Anführungsstrichen werden vom PHP-Parser bei deren Verwendung verarbeitet. Es werden sowohl alle darin enthaltenen Zeichen als auch Sonderzeichen als auch die Variablen aufgelöst – und dabei in den Datentyp String konvertiert. Das vorige Beispiel

```
$a = "Test";
echo "Ausgabe einer $a-Variablen";
```

erzeugt demnach den Satz *»Ausgabe einer Test-Variablen«*. Ebenso werden Sonderzeichen wie der Zeilenumbruch in Strings mit doppelten Anführungszeichen richtig interpretiert. Andere Sonderzeichen, die Beachtung finden, sind beispielsweise der Backslash (»\«) oder das Dollarzeichen (»$«). Möchten Sie Sonderzeichen in einem String ausgeben, ohne dass sie interpretiert werden, müssen Sie sie mit einem Backslash »maskieren«. Beispielsweise erreichen Sie die Ausgabe des Satzes »PHP-Variablen beginnen mit einem $-Zeichen« wie folgt:

```
echo "PHP-Variablen beginnen mit einem \$-Zeichen";
```

Ohne den Backslash vor dem Dollarzeichen würde der PHP-Parser versuchen, die nachfolgenden Zeichen so weit wie möglich als Variablennamen auszuwerten.

Anführungsstriche des jeweils anderen Typs werden in Zeichenketten vom PHP-Parser nicht als störend angesehen. Ohne eine Form der Maskierung können Sie Strings wie

```
echo "Zeichen maskieren ist ein Synonym für 'escapen'";
echo 'Zeichen maskieren ist ein Synonym für "escapen"';
```

bilden und ausgeben lassen. Anders ist das jedoch, wenn Sie die gleichen Anführungszeichen ausgeben möchten, die Sie auch zum Begrenzen des Strings benutzen. In diesem Fall müssen Sie die auszugebenden Zeichen maskieren:

```
echo "Zeichen maskieren ist ein Synonym für \"escapen\"";
```

Doppelte Anführungszeichen sind somit weitaus flexibler. Allerdings sind sie auch weit verbreitet, um Attributwerte in XHTML-konformen Dokumenten zu kennzeichnen, z.B. in

```
<table border="1">...</table>
```

Um solchen HTML-Code mit PHP auszugeben, gibt es mehrere Möglichkeiten, die syntaktisch alle richtig, aber mehr oder weniger praktisch sind. Die folgenden Zeilen bewirken alle das Gleiche:

```
$bilddatei = 'bilddatei.png';
echo "<img src='$bilddatei' alt='Bild'/>";
echo "<img src=\"$bilddatei\" alt=\"Bild\"/>";
echo '<img src="bilddatei.png" alt="Bild"/>';
```

Alle drei Ausgaben sind konform zum W3C-Standard XHTML 1.0, in dem keine Aussage dazu getroffen wird, ob einfache oder doppelte Anführungsstriche verwendet werden sollen, wohl aber, dass Anführungszeichen benutzt werden müssen. Die erste Ausgabe mit doppelten Hochkommata hat den Vorteil, dass die enthaltene Variable ausgewertet wird. Die einfachen Anführungszeichen um die Attributwerte brauchen nicht maskiert zu werden, wie es in der zweiten Ausgabe der Fall ist. Sofern einfache Hochkommata benutzt werden, kann man keine variablen Bildnamen verwenden.

Bei allen drei Varianten mangelt es jedoch an Klarheit, was die Trennung von HTML- und PHP-Code betrifft. Nehmen Sie an, Sie haben Bilder, die Sie durchnummeriert haben: *Bild1.png*, *Bild2.png*, ... Eine flexible Ausgabe erreichen Sie mit

```
$nummer = 1;
echo "<img src='Bild$nummer.png' alt='Bild$nummer'/>";
```

Eine weitaus schönere Lösung ist es, die Bestandteile schon optisch voneinander unterscheiden zu können oder anders gesagt, Text und Variablen hintereinander zu hängen. PHP bietet Ihnen einen Operator, um mehrere Strings aneinander zu reihen: den Konkatenationsoperator (».«). Damit lassen sich zwei Zeichenketten nahtlos verbinden – Leerzeichen werden zwischen den Bestandteilen nicht eingefügt:

```
echo $satz1.$satz2;
```

Der obige Befehl wird dadurch um einiges besser lesbar.

```
echo "<img src='Bild".$nummer.".png' alt='Bild'/>";
```

Eine verbindliche Vorgabe, welche Schreibweise Sie zu benutzen haben, existiert nicht, zumal alle syntaktisch korrekt sind. Ihr Schreibstil ist damit den eigenen Vorlieben überlassen. Da die zuletzt vorgestellte Variante am eindeutigsten ist

und auch per Syntax-Highlighting unterstützt wird, verwenden wir sie in allen weiteren Skripten.

Im Folgenden werden wie Ihnen noch eine Auswahl an Befehlen vorstellen, die in Bezug auf Zeichenketten häufig benutzt werden. Sie alle entstammen der Gruppe der Stringfunktionen, wie sie in der Referenz zu PHP und im Anhang dieses Buches zu finden sind. Die von uns gewählte Schreibweise lehnt sich an die der offiziellen Referenz an.

Die Ausgabe von Strings erreichen Sie mit dem Befehl `echo`, den Sie schon aus den vorangegangenen Beispielen kennen:

```
echo $str;
```

Da `echo` keine Funktion, sondern ein Sprachkonstrukt ist, brauchen Sie für das Argument auch keine Klammerung. Synonym zu `echo` zu verwenden ist der Befehl `print`. Anders als `echo` gibt `print` jedoch wahr oder falsch zurück, je nachdem, ob die Ausgabe fehlerfrei ausgeführt wurde oder nicht.

Leerzeichen links und rechts eines Strings sind lästig, wenn sie zum Suchen in Texten benutzt oder im Browser ausgegeben werden sollen. Leerraum lässt sich bequem per `trim()` entfernen. Zu den entfernten Zeichen gehören nicht nur Leerzeichen, sondern auch Zeilenumbrüche oder Tabulatoren:

```
string trim(string $str);
string ltrim(string $str);
string rtrim(string $str);
```

Während die Funktion `trim()` Leerraum auf beiden Seiten des Strings `$str` entfernt, tun `ltrim()` und `rtrim()` das nur auf der jeweils linken und rechten Seite. Die Befehle geben jeweils den gekürzten String zurück:

```
$string = ' Webapplikation ';
echo trim($string);    //ergibt 'Webapplikation'
echo ltrim($string);   //ergibt 'Webapplikation '
echo rtrim($string);    //ergibt ' Webapplikation'
```

Eine andere Möglichkeit, Teile aus einem String zu extrahieren, bietet die Funktion

```
string substr(string $str, integer $start [, integer $laenge])
```

Voraussetzung ist dabei, dass Sie Kenntnis darüber haben, welche Teile Sie benötigen, zumal Sie nach dem String die Startposition angeben müssen. Ein positiver Wert des Parameters `$start` schneidet entsprechend viele Buchstaben von dem String am Anfang ab. Ist `$start` gleich 0, so beginnt der zurückgegebene String wie `$str`. Über den Parameter `$laenge` steuern Sie, wie viele Zeichen ab der

aktuellen Position zurückgeliefert werden. Bei einem negativen Wert des Parameters `$start` werden entsprechend viele Zeichen vom Ende der Zeichenkette zurückgegeben:

```
$string = 'Webapplikation';
echo substr($string, 0,3);   //ergibt 'Web'
echo substr($string, 2,4);   //ergibt 'bapp'
echo substr($string, -6,3);   //ergibt 'kat'
```

Obwohl `substr()` Teile aus Zeichenketten extrahiert, kann man dabei nicht von einer Suchfunktion sprechen.

Hinweis

Das Auffinden von Daten ist ein großes Thema in der Informatik. Wir werden uns im zweiten Teil dieses Buches eingehend damit beschäftigen, sowohl im Rahmen von PHP als auch mittels MySQL.

Um einen Teil-String zu suchen und ihn gegebenenfalls durch etwas anderes zu ersetzen, bringt PHP eine Vielzahl von Funktionen mit. Dabei wird unterschieden zwischen Funktionen, die nur die Teilaufgabe des Findens übernehmen, und solchen, die im Anschluss das Ersetzen mit erledigen. Die Funktionen `strpos()` und `strstr()` gehören in die erste Kategorie.

```
string strpos(string $str, string $suche [, $startposition])
integer strstr(string $str, string $suche)
```

Von `strpos()` wird die Position des ersten Vorkommens angegeben. Wird `$suche` in `$string` gar nicht gefunden, gibt die Funktion den Wahrheitswert falsch zurück. Der optionale Parameter `$startposition` erlaubt es Ihnen, die Suche erst ab einer bestimmten Position in `$str` zu beginnen.

```
$string = 'Webapplikation';
$position = strpos($string, 'p');   //$position ist dann 4
$position = strpos($string, 'a', 5);   //$position ist 9
```

Auch die Funktion `strstr()` sucht das erste Vorkommen von `$suche` in `$str`. Anders als bei `strpos()` wird hier jedoch der Rest-String (mitsamt dem gesuchten Zeichen) zurückgegeben.

```
$string = 'Webapplikation';
$ergebnis = strstr($string, 't');   //$position ist 'tion'
```

Das gleichzeitige Suchen und Ersetzen wird von der Funktion `str_replace()` vorgenommen, ist aber keineswegs beschränkt auf Zeichenketten, sondern kann

auch mit Arrays benutzt werden. Vorerst wird hier jedoch nur die Suche in Strings betrachtet:

```
string str_replace(string $alt, string $neu, string $str)
```

Die komplette Zeichenkette `$str` wird nach `$alt` abgesucht. Jedes Vorkommen wird durch `$neu` ersetzt. Als Rückgabewert erhalten Sie den String nach allen Ersetzungen:

```
$string = 'Webaplikation';
echo str_replace('p','pp',$string);
?>
```

Listing 4.3 Verarbeitung von Zeichenketten

Die Ausgabe ist folglich das richtig geschriebene Wort »Webapplikation«.

CD-ROM zum Buch

Die Listings für Zeichenketten finden Sie auf der CD-ROM zum Buch als Listing 4.2. Wie sich das Skript am Bildschirm darstellt, sehen Sie in Abbildung 4.1

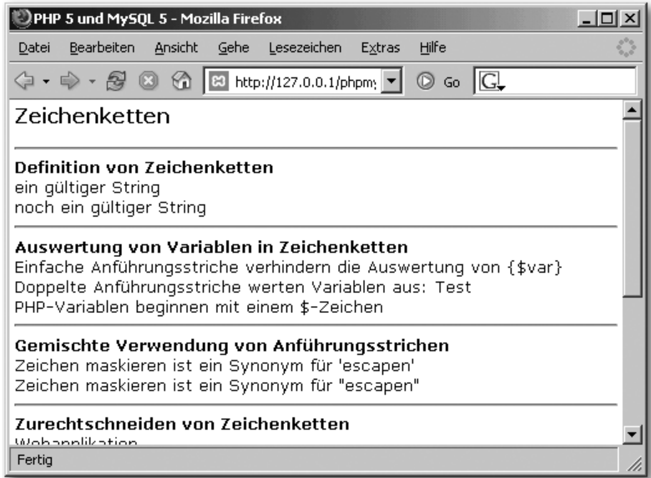

Abbildung 4.1 Ausgabe der Stringfunktionen

Ganzzahlige Werte

Ganzzahlige Werte sind Zahlen ohne Nachkommastellen. Dazu gehören die positiven und negative Zahlen sowie die Null. Als Synonym für Ganzzahlen ist der Begriff Integer anzusehen.

Für Zahlen sind die vier Grundrechenarten in PHP definiert, also die Addition (»+«), die Subtraktion (»-«), die Multiplikation (»*«) und die Division (»/«):

```
$addiert = 10 + 12;
$subtrahiert = 22 - 10;
$multipliziert = 12 * 10;
$dividiert = 120 / 12;
```

Zusätzlich zu den Operatoren der Grundrechenarten kennt PHP noch einen weiteren arithmetischen Operator: den Modulo (»%«). Der Modulo zweier Zahlen ist der Rest, der übrig bleibt, wenn die erste Zahl durch die zweite geteilt wird:

```
$rest = 48% 7;
```

In diesem Fall ist der Rest 6, denn das größte Vielfache von 7, das kleiner ist als 48, ist 42. Der Rest zweier Zahlen ist also minimal 0, wenn der linke Operand des Modulo ein Vielfaches von dem rechten Operanden ist, und maximal um 1 kleiner als der rechte Operand.

Häufig werden Zahlen als Zählvariablen benutzt, um die Häufigkeit eines bestimmten Ereignisses herauszufinden. Die Variable wird dann initialisiert, z.B. mit 0, und bei jedem Auftreten des Ereignisses um eins hochgezählt. Das nennt man inkrementieren. Eine Variable herunterzuzählen heißt dekrementieren. Für beide Varianten gibt es in PHP Operatoren. Um den Wert einer Variablen `$a` zu benutzen und dann um eins zu erhöhen, schreiben Sie:

```
$a = 10;
echo $a++;
```

Am Bildschirm wird Ihnen der Wert 10 ausgegeben. Nach dem `echo`-Befehl hat `$a` den Wert 11. Eine Dekrementierung erreichen Sie mit `$a--`. Diese Methode mit dem nachgestellten Operator heißt Post-Inkrementierung bzw. Post-Dekrementierung, weil `$a` erst nach Ausführung des `echo` erhöht bzw. verringert wird. Das Gegenteil dazu ist die Prä-Inkrementierung bzw. Prä-Dekrementierung:

```
$a = 10;
echo ++$a;
```

Die Variable wird zuerst verändert, bevor das `echo` ausgeführt wird. Die Ausgabe ist dementsprechend 11.

Fließkommazahlen

Bei Berechnungen mit Integer-Zahlen kann es sehr schnell vorkommen, dass Sie den Bereich ganzzahliger Werte verlassen. Das Ergebnis der Division

```
$number = 3;
$number = $number / 2;
```

lässt sich nicht als Integer abbilden. Zahlen mit Nachkommastellen heißen in PHP Fließkommazahlen. Die Programmierung kennt zwei verschiedene Arten von Fließkommazahlen, die in älteren Programmiersprachen wie Fortran, nicht aber von PHP unterschieden werden: float und double.

Hinweis

Das in Deutschland gebräuchliche Komma als Trennzeichen in Fließkommazahlen hat keinen Einzug in PHP gehalten. An dessen Stelle wird in der Programmierung der Punkt benutzt: 1.2 statt 1,2.

Analog zu ganzzahligen Werten können Sie mit Fließkommazahlen alle bereits vorgestellten arithmetischen Funktionen wie Addition oder Division ausführen. Neu hingegen ist die Kontrolle über die Anzahl der verwendeten Nachkommastellen. Bei »krummen« Berechnungen wird Ihnen das Ergebnis mit einer ganzen Reihe von Stellen hinter dem Komma angegeben. Während diese Genauigkeit für weiteres Rechnen durchaus wünschenswert ist, stört der Rattenschwanz aus Nachkommastellen spätestens bei der Ausgabe auf dem Bildschirm.

Wenn eine begrenzte Genauigkeit einer Zahl für Ihre weiteren Zwecke ausreichend ist, können Sie den Wert der Variablen auf die benötigte Anzahl von Nachkommastellen runden.

Achtung!

Das Zurechtschneiden von Zahlen mittels Funktionen wie `substr()` ist weder elegant noch mathematisch korrekt. Zum einen wird die Zahl dabei zu einer Zeichenkette konvertiert und zum anderen wird keine Rundung durchgeführt.

Nehmen wir als erstes Beispiel die Zahl PI:

```
$pi = pi();
```

Die Ausgabe von `echo $pi;` ergibt `3.14159265359`. Eine Rundung erreicht man mittels der Funktionen `floor()`, `ceil()` und `round()`. Die Angabe der Präzision ist allerdings nur bei `round` möglich:

```
echo round($pi, 4);
```

Damit wird `$pi` auf vier Nachkommastellen gerundet (3.1416). Die anderen beiden Befehle runden auf ganzzahlige Werte: `floor` (engl. Floor – Boden, Untergrenze) rundet auf die nächstkleinere ganze Zahl ab; `ceil` (engl. Ceiling – Decke, Obergrenze) liefert die nächsthöhere ganze Zahl.

Ein weiteres reales Beispiel ist die Berechnung der Mehrwertsteuer:

Geldbeträge werden in der Regel mit zwei Nachkommastellen angegeben, zumal kleinere Beträge nicht in Bargeld ohne Rundung ausgezahlt werden können. Rechnen Sie nun auf einen beliebigen Betrag die 16 oder auch 19 Prozent Mehrwertsteuer drauf, kann sich eine Zahl mit mehr als zwei Stellen hinter dem Komma ergeben, die wiederum gerundet werden muss: 17,73 * 1,16 = 20,5668:

```
$unbesteuert = 17.73;
$steuerfaktor = 1.16;
$besteuert = $unbesteuert * $steuerfaktor;
echo round($besteuert, 2);
?>
```

Listing 4.4 Umgang mit Zahlenwerten

Am Bildschirm wird damit 20,57 ausgegeben.

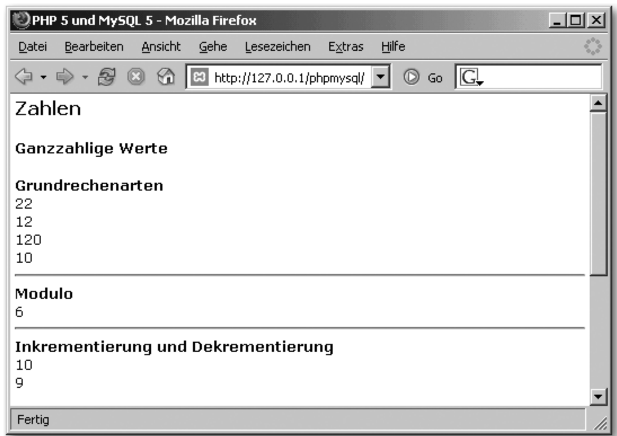

Abbildung 4.2 Ausgabe von Listing 4.3

Boolesche Wahrheitswerte

Bei booleschen Wahrheitswerten gibt es nur zwei unterschiedliche Ausprägungen: *wahr* und *falsch*; in PHP werden sie mit den Schlüsselwörtern `true` und `false` gekennzeichnet. Anwendung finden sie bei der Überprüfung von Gleichheit zweier oder mehrerer Werte. Wenn Sie beispielsweise zwei Variablen auf Gleichheit prüfen wollen, müssen Sie dafür ein doppeltes Gleichheitszeichen als Vergleichsoperator (»==«)benutzen. Das einfache Gleichheitszeichen ist ja bekanntlich für die Zuweisung reserviert und kann deshalb nicht für Gleichheitsprüfungen dienen:

```
Wert1 == Wert2
```

gibt Ihnen ein `true` oder `false` zurück, dahingegen überschreibt

```
Wert1 = Wert2
```

den ersten Wert durch den zweiten. Es ist also Vorsicht geboten. Die Prüfung auf Gleichheit zweier Werte mit dem doppelten Gleichheitszeichen ist nicht typsicher. Die Abfrage

```
10 == '10'
```

ist wahr, da eine automatische Typkonvertierung vorgenommen wird. Eine typsichere Variante ist das dreifache Gleichheitszeichen (»===«). Der Ausdruck

```
10 === '10'
```

ergibt den Wert `false`, da eine Zahl mit einem String verglichen wird. Die typunsichere Methode ist dennoch sinnvoll. Beispielsweise werden die übergebenen GET- und POST-Daten eines Formulars von PHP standardmäßig als String interpretiert. Rufen Sie Ihr Skript also mit `skript.php?zahl=10` auf und prüfen Sie darin, ob 10 eine Zahl ist, so werden Sie enttäuscht. Die Antwort ist `false`.

Zwei oder mehrere Bedingungen können paarweise über logische Operatoren miteinander verknüpft werden. Die verknüpften Teilbedingungen haben dann einen gemeinsamen Wahrheitswert. Nehmen Sie die folgenden mathematischen Bedingungen an:

```
Bedingung A:   10 > x
Bedingung B:   20 > x
```

Bei den beiden Bedingungen ergibt sich deren Wahrheitsgehalt durch einen Vergleich mit einer fest vorgegebenen Zahl mit einem variablen Schwellenwert. Die erste Bedingung ist wahr, solange das x einen Wert von 9 oder kleiner hat. Bei der zweiten Bedingung liegt die Schwelle um 10 Einheiten höher. Die Bedingungen A und B können dann anhand folgender Operatoren verknüpft werden.

Das logische Und

Zwei Bedingungen A und B sind immer dann gemeinsam wahr, wenn sowohl A als auch B wahr sind. Sobald auch nur eine Teilbedingung falsch ist, ist der gesamte Ausdruck falsch. Eine Übersicht über den gemeinsamen Wahrheitswert einer verknüpften Bedingung in Abhängigkeit von den Wahrheitswerten der Bestandteile zeigt die Tabelle links oben in Abbildung 4.3. Ein `t` in einer der Zellen bedeutet `true`/wahr, ein `f` heißt `false`/falsch.

In PHP gibt es sowohl das Schlüsselwort `and`, über das Sie Bedingungen miteinander verknüpfen können, als auch das doppelte kaufmännische Und (`&&`). In unserem Beispiel muss x einen Wert kleiner als 10 besitzen, damit der Ausdruck

`A && B` bzw. `A and B` wahr ist: Nehmen wir für x den Wert 8 an, so ist 10 > 8 wahr (Bedingung A) und 20 > 8 wahr (Bedingung B).

Das logische Oder

Zwei Bedingungen A und B sind gemeinsam immer dann wahr, solange mindestens einer der Bestandteile wahr ist. Das bedeutet, wenn eine Teilbedingung falsch ist, kann die Verknüpfung immer noch wahr sein. Dargestellt ist dies in der Tabelle oben rechts in Abbildung 4.3.

Das Logische Oder wird in PHP durch das Schlüsselwort `or` oder die doppelte Pipe (`||`) symbolisiert. Im Beispiel ist `A || B` oder `A or B` wahr für alle Zahlen kleiner oder gleich 19. Bei Zahlen größer 9 ist zwar die Bedingung A verletzt, jedoch stimmt Bedingung B noch. Erst ab x größer oder gleich 20 sind sowohl A als auch B und somit ihre Verknüpfung falsch.

Das logische exklusive Oder

Exklusiv in diesem Zusammenhang bedeutet ausschließend. Somit liegt das Exklusive Oder dichter an des Pudels Kern, wenn man das Wort Oder betrachtet. Gemeint ist, dass entweder die eine Alternative **oder** die andere wahr ist, nicht aber beide gleichzeitig. Verknüpft man zwei Bedingungen über das exklusive Oder miteinander, so muss genau eine der Bedingungen falsch und die andere wahr sein. Der Unterschied zum »normalen« Oder ergibt sich also nur in der Alternative, dass sowohl A als auch B wahr sind, zu sehen in der Tabelle unten links in Abbildung 4.3.

In PHP existiert das Schlüsselwort `xor` für das Exklusive Oder, ein weiteres Symbol wie für die vorigen Operatoren besteht nicht. In unserem Beispiel besteht nur ein sehr begrenzter Bereich an Werten von x, für die ein `A xor B` wahr ergibt. Das ist der Fall für alle Zahlen zwischen 10 und 19, bei denen A schon zu falsch ausgewertet wird.

Das logische Nicht

Dieser letzte Operator, den wir Ihnen bei den booleschen Werten vorstellen wollen, bezieht sich nicht wie die anderen auf die Verknüpfung zweier Bedingungen, sondern ist ein unärer (mit nur einem Argument) Operator, bezieht sich also nur auf einen Wahrheitswert. Der Wert der Bedingung wird dadurch verneint, `true` wird zu `false` und umgekehrt. In PHP wird das Nicht durch das Ausrufezeichen (»`!`«) symbolisiert. Das Schlüsselwort `not` existiert nicht, sondern ist nur in Abbildung 4.3 der Analogie halber aufgeführt.

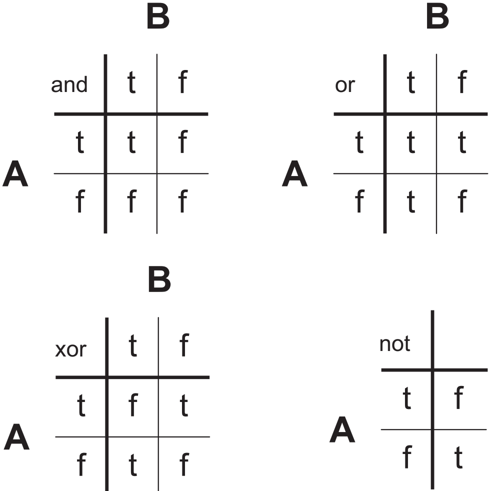

Abbildung 4.3 Verknüpfung von Bedingungen über logische Operatoren

Verknüpfungen von booleschen Termen können beliebig komplex sein. Um die Auswertungsreihenfolge eindeutig zu kennzeichnen, können bzw. müssen Sie die Terme klammern. Nehmen Sie die drei Bedingungen X (falsch), Y (falsch) und Z (wahr). Die Verknüpfung `X && Y || Z` ergibt wahr, da zunächst `X && Y` zu falsch ausgewertet wird und `false || Z` dann wahr ist. Um zuerst Y und Z miteinander zu verknüpfen, setzen wir sie in Klammern: `X && (Y || Z)`. Nun ergibt der gesamte Term falsch.

PHP besitzt die Eigenart, dass nicht nur die Schlüsselwörter `true` und `false` als boolesche Wahrheitswerte verwendet werden können. Auch die übrigen Datentypen eignen sich dafür. Im booleschen Sinne wahr sind beispielsweise:

- Strings, die nicht leer sind, also z.B. »b«, »wort«, »falsch« und »false«,
- alle Zahlen außer der 0, also z.B. 2809, 2912, 17, –1, –1000,
- Arrays und Objekte, solange sie Inhalt haben.

Arrays

In einem Array können Sie mehrere Werte zusammenschließen, die semantisch verwandt sind. Jedes Element ist entweder über einen eindeutigen Namen (assoziatives Array) oder eine Nummer (numerisches Array) ansprechbar, den so genannten Schlüssel. Ein Array ist also eine Ansammlung von Schlüssel-Wert-Paaren. Sofern Sie beim Hinzufügen eines Elementes zum Array keinen Schlüssel vergeben, wird automatisch eine fortlaufende Nummer verwendet, beginnend bei 0.

```
<?php
$array = array();
$array[] = 'erstes Element';
```

Zunächst wird das Array leer initialisiert. Dies geschieht mit dem Sprachkonstrukt

```
array();
```

Dann wird ein Element hinzugefügt, das fortan über den Bezeichner `$array[0]` angesprochen und verändert werden kann. Die leere Initialisierung müssen Sie nicht voranstellen. PHP ist auch in diesem Fall sehr kulant und quittiert das Befüllen eines nicht existenten Arrays nicht mit einer Fehlermeldung.

Es ist nicht zwingend notwendig, dass Sie eine fortlaufende Reihenfolge der Schlüssel einhalten. Im Falle eines assoziativen Arrays ist eine numerische Reihenfolge auch nicht immer möglich. PHP hält für jedes Array, das Sie erstellen, eine interne Ordnung aufrecht. Die Ordnung eines Arrays wird nur dadurch verändert, wenn Sie das Array nach den enthaltenen Werten sortieren. Durch die folgenden Anweisungen werden die Arrayfelder 3 und 5 belegt.

```
$array[3] = 'zweites Element';
$array[5] = 'drittes Element';
```

Die Schlüssel 1, 2 und 4 sind somit ausgelassen worden. Die Daten, die Sie in dem Array speichern, können zudem von unterschiedlichen Datentypen sein. Sie können beispielsweise Strings mit Zahlen mischen:

```
$array[7] = 'nächstes Element';
$array[9] = 5;
```

Sofern Sie nur skalare Werte in einem Array speichern, also Zeichenketten, boolesche Wahrheitswerte sowie Ganz- und Fließkommazahlen, spricht man von einem eindimensionalen Array. Allerdings kann ein Arrayfeld auch ein weiteres Array enthalten. Somit entsteht eine Hierarchie, ein so genanntes mehrdimensionales Array.

```
$array[10] = array('a', 'b', 'c', 'd');
```

Das Unter-Array in `$array[10]` wurde nicht leer initialisiert, sondern mit vier Werten befüllt. Es hat keinen eigenen Namen, sondern ist anonym. Die Werte des Unter-Arrays sind über die Schlüssel `$array[10][0]` bis `$array[10][3]` ansprechbar.

Die vorangegangenen Beispiele zeigen, dass Sie das Sprachkonstrukt `array()` in verschiedener Weise benutzen können. Zum einen ist die leere Initialisierung eines Arrays möglich. Zum zweiten lassen sich damit Arrays in einem Schritt

erstellen und befüllen. Allerdings gilt dies nicht nur für numerische Arrays, so wie es bei `array('a', 'b', 'c', 'd')` der Fall war. Ebenso gut können Sie den Werten nicht-numerische und unsortierte Bezeichner geben:

```
$assozArray = array(' weitererSchluessel '=>'a', 'key'=>'b');
?>
```

Listing 4.5 Definition von Arrays

Ein Schlüssel-Wert-Paar wird durch den Operator => verbunden. Auch für assoziative Arrays bleibt die definierte Ordnung der Elemente bestehen, auch wenn keine Sortierung besteht.

Analog zu den skalaren Datentypen lassen sich auch Arrays durch einen einfachen Befehl am Bildschirm ausgeben. Die Funktion `print_r()` stellt dabei nicht nur die Werte, sondern auch die Struktur der Daten dar. Das angehängte `_r` im Namen des Befehls bedeutet rekursiv; es werden also auch eventuelle Unter-Arrays beachtet. Wie ein Array über `print_r()` am Bildschirm ausgegeben wird, sehen Sie in Abbildung 4.4.

Abbildung 4.4 Ausgabe des Arrays $assozArray

Hinweis

Wir verwenden die Begriffe indiziertes Array und numerisches Array synonym und machen auf diese Weise die Unterscheidung zu assoziativen Arrays. Selbstverständlich können jedoch auch assoziative Arrays numerische Schlüssel besitzen. Das ändert aber nichts an der Tatsache, dass sie assoziativ sind.

Arrays können komplexe Datenstrukturen werden, auf denen eine Suche bzw. eine Sortierung sinnvoll eingesetzt werden kann. In PHP existieren mehrere alternative Sortieroptionen. Die gängigsten Formen sind:

```
sort(array $array [,$sortierTyp])
asort(array $array [,$sortierTyp])
ksort(array $array [,$sortierTyp])
rsort(array $array [,$sortierTyp])
```

Der Befehl sort() nimmt eine »normale« Sortierung in alphabetisch oder numerisch aufsteigender Folge vor. Die Reihenfolge der Schlüssel wird dabei nicht beibehalten. Das bedeutet, nach der Sortierung sind die Schlüssel weiterhin sortiert wie zuvor. Die Beziehung von Schlüssel und Wert wird bei asort() gewährleistet. Nach der Sortierung sind die Schlüssel demnach nicht mehr zwingend in der Reihenfolge wie davor. ksort() führt die Suche nicht nach den Werten, sondern den Schlüsseln des Arrays durch. Und rsort() führt die Sortierung letztlich in umgekehrter Reihenfolge durch. Keine der Funktionen hat einen Rückgabewert.

Der optionale Parameter $sortierTyp bestimmt dabei, wie die Werte sortiert werden:

Bezeichnung	Beschreibung
SORT_REGULAR	Führt Suche durch, ohne die Werte vorab zu konvertieren.
SORT_NUMERIC	Führt Suche durch, als wären die Werte Zahlen.
SORT_STRING	Führt Suche durch, indem alle Werte als Zeichenkette interpretiert werden.

Tabelle 4.3 Optionen für die Arraysortierung

Als Beispiel zur Verdeutlichung der Sucharten soll uns das folgende Array dienen:

```
<?php
$bundeslaender = array('a' => 'Brandenburg',
   'b' => 'Baden-Württemberg',
   'c' => 'Schleswig-Holstein',
   'd' => 'Nordrhein-Westfalen');
```

Für jedes der vier nächsten Beispiele nehmen wir $bundeslaender in obiger Form als Ausgangszustand an.

Nach sort($bundeslaender) ist die Sortierung wie folgt:

```
'0' => 'Baden-Württemberg',
'1' => 'Brandenburg',
'2' => 'Nordrhein-Westfalen',
'3' => 'Schleswig-Holstein'
```

Nordrhein-Westfalen und Schleswig-Holstein haben ihre Position getauscht und die Schlüssel wurden durch numerische, aufsteigend sortierte ersetzt. Die umgekehrte Reihenfolge der Werte erhalten wir durch die Anweisung rsort($bundeslaender):

```
'0' => 'Schleswig-Holstein',
'1' => 'Nordrhein-Westfalen',
```

```
'2' => 'Brandenburg',
'3' => 'Baden-Württemberg'
```

Auch hierbei gehen allerdings die Schlüssel-Wert-Beziehungen verloren. Mit `asort($bundeslaender)` ergibt sich das gleiche Bild wie bei der ersten Sortierung, allerdings wurden die Schlüssel nicht ersetzt:

```
'b' => 'Baden-Württemberg',
'a' => 'Brandenburg',
'd' => 'Nordrhein-Westfalen',
'c' => 'Schleswig-Holstein'
```

Eine Sortierung nach Schlüsseln mit `ksort($bundeslaender)` führt letztlich zu folgender Aufstellung:

```
'a' => 'Brandenburg',
'b' => 'Baden-Württemberg',
'c' => 'Schleswig-Holstein',
'd' => 'Nordrhein-Westfalen'
```

Arrays lassen sich genau wie Strings durchsuchen. Dazu dienen die beiden Befehle

```
bool in_array( mixed $suche, array $array [, bool $typ])
mixed array_search( mixed $suche, array $array [, bool $typ])
```

Gesucht wird nach den Werten und nicht nach den Schlüsseln im Array. Im ersten Fall wird nur wahr oder falsch zurückgegeben, je nachdem, ob `$suche` in `$array` vorhanden ist. Die Funktion `array_search()` hingegen liefert im Erfolgsfall den Schlüssel; bei Misserfolg wird auch falsch zurück geliefert. Als Beispiel soll wieder `$bundeslaender` dienen, und zwar in der Form, in der es ursprünglich definiert wurde:

```
in_array('Bayern', $bundeslaender)   //ergibt wahr
in_array('Sachsen', $bundeslaender)   //ergibt falsch
array_search('Bayern', $bundeslaender)   //ergibt 'b'
```

Der dritte optionale Parameter schaltet zwischen typsicherer und nicht typsicherer Suchmethode um. Im Standardfall ist `$typ` auf falsch gesetzt, dabei wird mit dem Operator == verglichen und die Suche ist nicht typsicher. Bei `$typ = true` wird stattdessen === für die Suche verwendet.

Der Parameter Suche kann unterschiedliche Datentypen besitzen – deswegen auch die Angabe des Pseudo-Datentyps *mixed*. Dies umfasst nicht nur skalare Datentypen. Ebenso gut lässt sich auch nach Arrays in Arrays suchen. Um dies zu verdeutlichen, führen wir die Suche in `$array` durch, das wir in diesem Abschnitt zuallererst definiert hatten:

```
$gesucht = array('a', 'b', 'c', 'd');
echo array_search($gesucht, $array);   //die Ausgabe ist 10
```

Um in einem Array nach einem Schlüssel zu suchen, können Sie den Befehl

```
bool array_key_exists(mixed $suche, array $array)
```

benutzen. Analog zu `in_array()` erhalten Sie dabei nur wahr oder falsch als Antwort. Alternativ dazu können Sie allerdings auch die Existenz über die allgemeine Funktion `isset()` erfragen. Beide kommen zu identischen Ergebnissen:

```
array_key_exists('a',$bundeslaender)   //ergibt wahr
isset($bundeslaender['a'])   //wahr und semantisch gleich
?>
```

Listing 4.6 Suchen und Sortieren in Arrays

Jedes Array hat einen internen Zeiger, der immer auf eines der bestehenden Elemente verweist. Dieser Zeiger wird beispielsweise beim Durchlaufen des Arrays sukzessive über alle Felder bewegt, um jedes Element genau einmal anzusprechen. Das geschieht transparent, d.h., das Bewegen des Zeigers wird Ihnen vom Parser abgenommen. Allerdings können Sie den Zeiger auch manuell verändern. Dazu gibt es eine Reihe von Befehlen, die in Abbildung 4.5 zu erkennen sind. Das Array ist demnach eine sortierte Liste von Feldern. Der Einfachheit halber wird in der Abbildung davon ausgegangen, dass die Felder nur skalare Werte beinhalten, es sich also um ein eindimensionales Array handelt. In mehrdimensionalen Arrays wird die Zeigerstruktur demnach komplexer. Manipulieren lässt sich die Zeigerstellung über die Funktionen, die in Tabelle 4.4 aufgelistet sind.

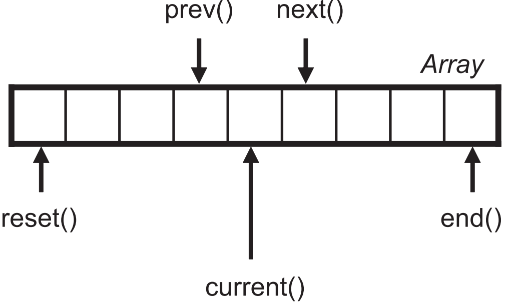

Abbildung 4.5 Zeigerfunktionen für Arrays

Befehl	Beschreibung
`current()`	Zeigt auf das aktuelle Element, also das Feld, auf den der Zeiger verweist.
`prev()`	Setzt den Zeiger um eine Position der internen Ordnung zurück.

Tabelle 4.4 Positionsfunktionen für Arrays

Befehl	Beschreibung
`next()`	Analog zu prev() wird der Zeiger auf das Feld hinter dem aktuellen Element gesetzt.
`reset()`	Setzt den Zeiger zurück auf das erste Element.
`end()`	Der Zeiger wird auf das letzte Feld des Arrays gesetzt.

Tabelle 4.4 Positionsfunktionen für Arrays (Forts.)

Sofern die Struktur des Arrays nicht durch Hinzufügen oder Löschen von Elementen verändert wird, zeigen `reset()` und `end()` also immer auf dieselben Elemente. Die anderen drei Befehle rücken bei jedem Aufruf von `prev()` oder `next()` immer eine Position in die entsprechende Richtung weiter. An den »Randbereichen« können zwei Befehle, etwa `next()` und `end()` also auf dasselbe Element zeigen. Ist kein voriges bzw. weiteres Element mehr vorhanden, so geben `next()` bzw. `prev()` statt einem Element `false` zurück.

Ein Array lässt sich bekanntermaßen über direkte Zuweisung oder Löschung von Elementen verändern. Durch die veränderliche Größe und flexible Struktur eignen sich Arrays für zahlreiche Datenstrukturen, wie etwa einen Stapelspeicher (engl. Stack). Das Element, das zuletzt ans Ende des Arrays angefügt wurde, wird zuerst wieder entfernt (LIFO-Prinzip – »Last In, First Out«). Stapelspeicher werden in PHP durch die beiden Befehle

```
int array_push(array $stack, mixed $var)
mixed array_pop(array $stack)
```

unterstützt. Mittels `array_push()` wird ein neuer Wert `$var` an die letzte Stelle des Arrays `$stack` hinzugefügt. Dies hat den gleichen Effekt wie

```
$stack[] = $var;
```

Es können auch gleichzeitig mehrere Werte angefügt werden, angegeben in einer durch Kommata getrennten Liste. Das Array vergrößert sich dadurch um die entsprechende Anzahl von Werten. Als Rückgabewert liefert `array_push()` die neue Größe des Arrays. Das Gegenstück `array_pop()` entfernt das letzte Element aus dem Array und gibt es zurück. Das Entfernen mehrerer Elemente ist nur durch wiederholtes Aufrufen der Funktion möglich.

Hintergrundwissen

Neben dem LIFO-Prinzip eines Stack existieren natürlich noch weitere Prinzipien, wie etwa das FIFO-Prinzip (»First In, First Out«) für Warteschlangen: Elemente werden ans Ende angefügt und vom Anfang entfernt. Weiter das HIFO (»Highest In, First Out«), das LOFO (»Lowest In, First Out«), usw. Alle Prinzipien lassen sich durch Arrays in PHP realisieren.

Objekte

Objekte sind, wie der Name schon verrät, die Grundbausteine objektorientierter Programmierung. Ein Objekt hat eine Reihe fest vorgeschriebener *Attribute*. Jedes Attribut hat einen eindeutigen Bezeichner, über den es sich ansprechen und manipulieren lässt. So weit ist ein Objekt einem Array sehr ähnlich. Was ein Objekt von einem Array abhebt, sind vordefinierte *Methoden*, über die die Attribute eines Objektes zur Laufzeit angepasst und Berechnungen durchgeführt werden können. Dass Methoden und Attribute festgelegt sind, bedeutet unter anderem, dass weitere Elemente zur Laufzeit nicht ohne weiteres zum Objekt hinzugefügt oder daraus entfernt werden können.

Mit Objekten lassen sich reale Dinge beschreiben. Gängige Beispiele in Tutorials sind Bücher im Rahmen eines Online-Shops oder Hunde, deren Verhaltensweisen zu Methoden gereichen. Als Einführung wollen wir ein Auto als Objekt betrachten. Wir abstrahieren bei der Darstellung eines Objekts als Datentyp gezielt von der genauen Syntax. Stattdessen wollen wir hier vorerst nur in die Gedankenwelt der Objektorientierung einführen

Für das Auto definieren wir eine Reihe von Attributen:

- die aktuelle Geschwindigkeit, der Einfachheit halber als Integer,
- die eingetragene Höchstgeschwindigkeit, auch als Integer,
- einen Status, ob das Auto gestartet ist, als booleschen Wert.

Im Startzustand des Objektes soll der Motor aus sein, das Statusflag ist also `false`. Sofern keine widrigen Umstände vorliegen, beträgt damit auch die aktuelle Geschwindigkeit `0`. Die Höchstgeschwindigkeit hängt von der Art des Autos ab. Für das Beispiel nehmen wir einmal moderate `180` Stundenkilometer an.

Sinnvolle Methoden für den Zugriff auf die Objektattribute sind das Ein- und Ausschalten des Motors sowie das Beschleunigen und Bremsen. Durch Aufrufen der Methoden werden die Objektattribute wie folgt manipuliert:

- `starteMotor` setzt das Statusflag von `false` auf `true`, sofern der Motor nicht bereits läuft.
- `stoppeMotor` arbeitet genau gegensätzlich dazu und setzt das Statusflag auf `false`.
- `beschleunige` setzt die aktuelle Geschwindigkeit auf einen angegebenen Wert, höchstens jedoch auf die eingetragene Höchstgeschwindigkeit.
- `bremse` wirkt wiederum andersherum und setzt die aktuelle Geschwindigkeit auf einen übergebenen Wert herab.

Alle definierten Attribute und Methoden lassen sich auf beliebige Autos anwenden, egal ob es sich dabei um Porsche oder Peugeot handelt. Es ist also möglich, die Gemeinsamkeiten in einer Art Schablone festzuhalten und zu definieren. Diese Vorlagen heißen in der Objektorientierung *Klassen*. Jedes Objekt, das aus dieser Klasse abgeleitet wird, ist eine *Instanz*. Dass eine Instanz, die einen Peugeot beschreibt, eine geringere Höchstgeschwindigkeit hat als eine Porsche-Instanz, ist logisch. Die entsprechenden Attribute müssen dem realen Vorbild bei der Erzeugung des Objektes bzw. der Instanz angepasst werden.

Ressourcen

Ressourcen bieten Ihnen eine Möglichkeit, auf externe Datenquellen zuzugreifen. Darunter fallen unter anderem Dateien, die über PHP geöffnet, gelesen und geschrieben werden, sowie Ergebnisse von Datenbankabfragen.

Wenn Sie beispielsweise eine Datei öffnen, wird eine Ressource erstellt. Bei jedem weiteren Befehl, mit dem die Datei verarbeitet werden soll, müssen Sie die Ressource wieder angeben, damit der PHP-Parser weiß, auf welche Datei sich der Befehl bezieht. Besonders deutlich wird die Notwendigkeit von Ressourcen, wenn Sie zwei Dateien geöffnet haben. Damit nicht fälschlicherweise Daten in die falsche Datei geschrieben werden, hilft Ihnen die Ressource.

Ressourcen belegen – mitunter sehr viel – Speicherplatz auf Ihrem Server und können deshalb gezielt wieder freigegeben werden. In der Regel müssen Sie sich als Programmierer darum aber nicht kümmern. PHP verwaltet die Ressourcen intern mit einer Zugriffsliste. Sobald der letzte Zugriff auf eine Ressource beendet ist, wird die Ressource automatisch freigegeben.

Null

Variablen haben den Wert `NULL`, wenn ihnen sonst kein Wert zugewiesen ist. Dies ist der Fall, wenn sie noch nicht initialisiert sind oder mit dem Befehl `unset()` wieder gelöscht wurden. Der Wert `NULL` ist somit auch die einzige Ausprägung des Datentyps.

Analog zu den übrigen Datentypen lässt sich auch mit dem Befehl `is_null($variable)` abfragen, ob eine Variable den Wert `NULL` hat. Dies ist jedoch nicht zu verwechseln mit dem Befehl `empty($variable)`, der zurückgibt, ob eine Variable leer ist. Eine leere Variable muss nicht immer den Wert bzw. Datentyp `NULL` haben; wenn `$variable=""` gesetzt wird, ist `$variable` zwar leer, der Befehl `empty()` ergibt also `true`, ist aber vom Typ String und somit nicht `NULL`.

Der Einsatz von Variablen des Typs NULL ist begrenzt. Anwendung finden sie vor allem dann, wenn eine Variable initialisiert werden soll, jedoch Datentyp oder Wert im Vorhinein unbekannt sind. Zugute kommt Ihnen dabei, dass PHP eine schwach getypte Sprache ist und sich die Datentypen bequem ändern lassen. Solange Sie nur vom Typ NULL zu einem der anderen Typen wechseln, sind Sie auch vor den Problemen beim Mixen von Datentypen gefeit, die wir eingangs des Kapitels beschrieben haben.

4.2.3 Namenskonventionen

Es existieren keine »Gesetze des Programmierens«, in denen festgehalten ist, wie Sie Ihren Code zu schreiben haben. Das Programmieren, egal in welcher Sprache, ist in weiten Teilen eine kreative Arbeit.

Dennoch ist es in vielen Fällen sinnvoll, sich an einige Richtlinien zu halten. Gerade wenn Sie im Team arbeiten, wenn Sie ihren Quellcode für andere freigeben oder wenn Sie nach Längerem wieder auf Ihre alten Skripte schauen, erleichtert eine einheitliche Programmierweise die Einarbeitung und beschleunigt somit das Verständnis und die Wartung des Codes. Selbst auferlegte Konventionen beginnen bereits bei der Namensgebung von Variablen.

In der Praxis nutzen wir die so genannte Höckerschreibweise. Wer sich allzu gewählt ausdrücken will, nennt das auch schon einmal Binnenmajuskel, was wörtlich nicht mehr heißt als »Großbuchstabe im Wortinnern«. Und obwohl man mit diesem Ausdruck eher auf fragende Gesichter stößt, beschreibt er die von uns bevorzugte Notation für Variablen sehr gut.

Variablennamen bestehen zuweilen aus mehreren Wörtern. Bei der Höckerschreibweise werden alle Namensbestandteile ohne Trennzeichen aneinandergehängt und der erste Buchstabe eines jeden Wortes wird großgeschrieben. Und das ungeachtet dessen, ob es sich bei dem Wort um ein Nomen, ein Verb oder sonstiges handelt. Der Ähnlichkeit zu einem Kamelrücken hat die Höckerschreibweise ihren Namen zu verdanken. Im Umkehrschluss der Regel für Großbuchstaben werden alle anderen Zeichen kleingeschrieben. Die einzige Ausnahme von dieser Regel ist der allererste Buchstabe des Namens. Obwohl auch er einen Wortanfang markiert, schreiben wir ihn trotzdem klein:

```
$betragOhneSteuern = 1000;
$lockDuration = 5;
$nummer = 1;
```

Die Höckernotation können Sie unabhängig von der Sprache benutzen. Die englisch benannte Variable $lockDuration beispielsweise, die Sie im dritten Teil des Buches im Abschnitt über Mehrbenutzersysteme wieder treffen werden, folgt

den gleichen Regeln wie der `$betragOhneSteuern`. Bei Variablennamen, die nur aus einem Wort bestehen, stellen sich die meisten Fragen der Höckernotation gar nicht, man muss sich nur an das erste kleingeschriebene Zeichen gewöhnen.

4.3 Konstanten

Im Gegensatz zu Variablen können Konstanten nach deren Definition nicht mehr verändert werden. Sowohl der Name als auch der Wert stehen unumstößlich fest. Ein weiterer Unterschied zu Variablen ist, dass Sie keine direkte Zuweisung verwenden können, sondern stattdessen den Befehl `define()` benutzen müssen, um eine Konstante anzulegen.

```
define("KONSTANTE",1234);
```

Die Funktion empfängt genau zwei Parameter. Als Erstes den Namen der Konstante, angegeben als String. Der zweite Parameter ist der Wert. Benutzen können Sie die Konstante wie eine Variable, d.h., Sie können damit rechnen, den Wert mit etwas vergleichen oder sie ausgeben:

```
echo KONSTANTE;
$abc = 4321 + KONSTANTE;
```

Daraus, wie Sie den Wert der Konstante bei der Definition angeben, ergibt sich deren Datentyp. Diesen können Sie per `gettype(KONSTANTE)` abfragen, jedoch nicht über `settype()` neu setzen. Das Thema Datentypen spielt bei Konstanten eine geringere Rolle als bei Variablen. Der Datentyp kann auch vom PHP-Parser selbst nicht zur Laufzeit verändert werden. Stattdessen wird intern mit Kopien der Konstante gearbeitet: so wird für die Ausgabe von `KONSTANTE` aus dem obigen Beispiel ein String und keine Zahl gebraucht.

Für die Namensgebung von Konstanten gelten andere Regeln als bei Variablen. Der Name einer Konstanten darf nicht mit einem Dollarzeichen (`$`) beginnen, weil anderenfalls Kollisionen mit Variablennamen nicht ausgeschlossen werden können. In allen weiteren Beispielen in diesem Buch erkennen Sie Konstanten daran, dass wir ihre Namen durchgängig in Großbuchstaben schreiben. Das ist keine syntaktische Vorgabe in PHP, grenzt aber Variablen und Konstanten schon optisch voneinander ab.

Konstanten werden vorerst für Einstellungen in einem System verwendet. Sie können also alle Werte, die von zentraler Bedeutung in Ihrem System sind, und die für jedes Skript die gleichen Werte haben sollen, als Konstante definieren. Das betrifft beispielsweise Verbindungsdaten für Ihre Datenbank oder Pfadangaben:

```
define("DBSERVER",'localhost');
define("IMAGEDIR",'images/');
```

4.4 Kommentare

Ein Kommentar ist Freitext, den Sie inmitten Ihres Quellcodes schreiben und der beschreibt, was an dieser Stelle im Code passiert. Angemessen kommentierter Code eignet sich wesentlich besser zur Wiederverwendung; und das nicht nur, wenn Sie den Code für andere Programmierer freigeben, sondern auch wenn Sie ihn selber nach einiger Zeit wieder zur Hand nehmen. Auch mit einem guten Gedächtnis werden Sie sich nicht immer daran erinnern können, wie Sie ein bestimmtes Problem zu einem früheren Zeitpunkt gelöst haben. Mit Kommentaren können Sie sich in solchen Fällen langes Grübeln ersparen.

Für kurze Bemerkungen eignet sich ein einzeiliger Kommentar. Dieser wird in PHP mit einem doppelten Slash (»//«) eingeleitet und gilt bis zum Zeilenende. Einzeilige Kommentare müssen Sie nicht von Hand beenden. Alle Anweisungen innerhalb der Kommentarzeile, die nach dem // stehen, werden vom Parser ignoriert.

```
// Zuweisung des aktuellen Datums in deutscher Notation
$date = '28.02.2006';
```

Erklärungen, die mehr Platz brauchen, können Sie in mehrzeiligen Kommentaren schreiben. Diese beginnen mit /* und enden mit einem */. Eine maximale Länge ist nicht vorgegeben. Umfangreiche Kommentare sind beispielsweise dann angebracht, wenn Sie die Funktion eines Skriptes generell beschreiben oder schwierig zu verstehende Passagen erklären wollen.

```
/* In den folgenden Zeilen werden Daten aufbereitet.
   Zuerst werden die Daten aus der MySQL-Datenbank geholt.
   Das DB-Ergebnis wird in einer Tabelle dargestellt.
*/
$link = mysql_connect('host', 'user', 'pass');
...
```

Wenn Sie beim Kommentieren Ihres Codes einige Konformitäten beachten, können Sie sich daraus automatisch die technische Dokumentation Ihrer Skripte erstellen lassen. Mit PHP-Bordmitteln erreichen Sie die automatische Dokumentation allerdings nicht. Stattdessen können Sie das kostenlose Tool *PHPdoc* nutzen, das Ihre Skripte auswertet und Ihre Kommentare in eine übersichtliche und strukturierte Form bringt. Inhalt einer technischen Dokumentation ist die Beschreibung Ihrer Programmierschnittstellen, also unter anderem Ein- und Aus-

gabeparameter einzelner Skripte oder Funktionen. Inwieweit Sie Ihren Code in der Dokumentation erklären, bleibt Ihrem eigenen Fleiß überlassen. Kommentare, die konform zu PHPdoc sind, haben eine festgelegte Struktur:

```
/**
 * Einzeilige Kurzbeschreibung des Abschnitts
 *
 * Umfangreiche Beschreibung nach einer Leerzeile
 * Der detaillierte Kommentar kann auch mehrzeilig sein.
 *
 * @version 1.0
 */
```

Herkömmliche mehrzeilige Kommentare werden von PHPdoc überlesen. Damit Kommentare geparst werden, müssen Sie sie mit `/**` beginnen, also mit zwei Sternen. Jede neue Zeile des Kommentars beginnt mit einem Stern. Manche Editoren wie beispielsweise *Eclipse* beherrschen PHPdoc-konforme Kommentare und unterstützen Sie bei der Dokumentation mit Syntax-Highlighting und Codevervollständigung. Ein Kommentar muss immer die drei Elemente *Kurzbeschreibung – Langbeschreibung – Tags* in genau dieser Reigenfolge enthalten. Während die Beschreibungstexte so von PHPdoc in die Dokumentation übernommen werden, wie Sie sie schreiben, werden die Tags verarbeitet. Es gibt eine ganze Reihe von Tags, die immer durch einen Klammeraffen (»@«) eingeleitet werden. Am wichtigsten neben `@version` sind noch `@param`, das einen Eingabeparameter einer Funktion mitsamt Datentyp spezifiziert, und `@return` für die Rückgabewerte von Funktionen.

Ausgabeformat von PHPdoc ist XML, das sich in unterschiedliche Repräsentationsformen konvertieren lässt, darunter navigierbares HTML in unterschiedlichem Layout, PDF oder Docbook. Letzteres ist zwar ebenfalls XML, jedoch als offener Standard festgeschrieben.

PHPdoc beziehen Sie über **http://www.phpdoc.de/**. Der Satz von Skripten ist selbst in PHP geschrieben und als quelloffene Software auch für kommerzielle Nutzung einsetzbar. Das Thema Dokumentation verdient eine genauere Betrachtung; deswegen haben wir ihm ein weiteres Kapitel im zweiten Teil dieses Buches gewidmet.

Hinweis

Die umfangreichen Listings der folgenden Abschnitte sind zu großen Teilen mit PHPdoc dokumentiert. Dadurch können Sie sich an den Umgang mit Kommentaren gewöhnen, sowohl an das Lesen als auch an das Schreiben von Kommentaren. In der Praxis empfiehlt es sich, Code nicht im Nachhinein, sondern gleich beim Programmieren zu kommentieren. Eine zugegeben lästige, aber sinnvolle Mehrarbeit.

der Angabe eines Spaltennamens `attrName` können Sie das Ergebnis weiter eingrenzen. Erlaubt sind für `attrName` bestehende oder durch Platzhalter (»_« für ein Zeichen, »%« für ein oder mehrere Zeichen) umschriebene Spaltennamen der Tabelle. Das Ergebnis hat immer das gleiche Format, das Sie aus Tabelle 9.4 ablesen können.

Spaltenname	Beschreibung
field	Spaltenname.
type	Datentyp, z.B. VARCHAR(100)
null	»YES«, wenn Nullwerte erlaubt sind, anderenfalls »NO«.
key	Enthält nur einen der folgenden Werte, wenn das Attribut einen Index hat: PRI (Primärschlüssel), UNI (eindeutiger Index), MUL (mehrwertiger Index).
default	Standardwert der Spalte.
extra	Zusätzliche Optionen, z.B. AUTO_INCREMENT.

Tabelle 9.4 Ergebnisstruktur einer DESCRIBE-Anweisung

Zu exakt demselben Ergebnis kommt auch der Befehl

```
EXPLAIN tblName.
```

9.5 Views

Sichten (engl. »View«) sind Abbilder bestehender Tabellen bzw. von deren Teilmengen. Sie haben einen eigenen Namen und ihre Struktur ist dauerhaft in der Datenbank hinterlegt. Sichten basieren auf den Ergebnissen von `SELECT`-Anfragen.

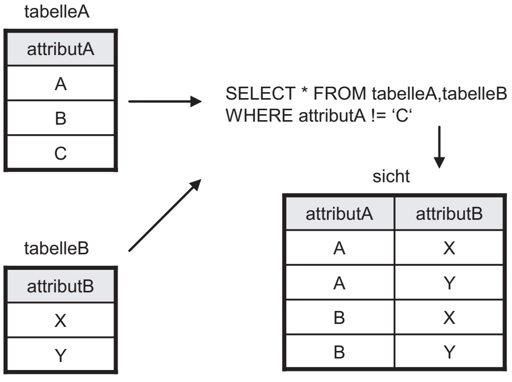

Abbildung 9.13 Sichten basieren auf bestehenden Tabellen

Nützlich ist dies für häufig ermittelte Tupelmengen, die Basis weiterer Verarbeitungsschritte sind. Anstatt die SELECT-Anfrage aus Abbildung 9.13 jedes Mal aufs Neue an die Datenbank zu senden – womöglich noch als Subquery einer weiteren komplizierten Anweisung – wird sie als Sicht gespeichert.

9.5.1 Anlegen

Die Syntax zur Definition einer Sicht lautet wie folgt:

```
CREATE [OR REPLACE]
[ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
VIEW vName
[(attrListe)]
AS selectKommando
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

Setzen Sie OR REPLACE, um eine vorhandene Sicht mit einer neuen Definition zu ersetzen. Dies ist kein Ersatz für einen ALTER-Befehl, der natürlich auch existiert. Jede Sicht muss einen eindeutigen Namen besitzen, der anstelle des vName eingesetzt wird. Es gelten die gleichen Vorgaben wie für Tabellennamen. Sichten und Tabellen teilen sich einen gemeinsamen Namensraum. Das bedeutet, deren Namen können kollidieren. Sie dürfen also für eine Sicht keinen Bezeichner benutzen, der schon für eine Tabelle vergeben ist – und umgekehrt. Als selectKommando geben Sie eine syntaktisch korrekte Anfrage an. Die Möglichkeiten entsprechen weitgehend dem, was Sie in der MySQL-Einführung im ersten Teil dieses Buches über der Befehl SELECT kennen gelernt haben. Beschränkungen existieren jedoch: Eine Sicht darf nicht auf Basis temporärer Tabellen erstellt werden, zumal die dauerhafte Funktionsfähigkeit der Sicht dadurch nicht gewährleistet werden kann. Ferner darf die FROM-Klausel keine Subquery enthalten; stattdessen darf sie nur Tabellen und Sichten referenzieren, die bereits existieren. Letztlich darf eine Sicht kein Attribut der Basistabelle mehrfach enthalten. Ein weiteres Manko, das sich allerdings auch für Tabellen nicht mit einem SELECT-Befehl bewerkstelligen lässt, ist das Fehlen von Indizes. Diese müssen auf den darunter liegenden Tabellen angelegt werden. Um die Sicht zu erzeugen, wie sie in Abbildung 9.13 vorgegeben ist, müssen Sie diese Anweisung ausführen:

```
CREATE VIEW sicht AS
SELECT * FROM tabelleA, tabelleB WHERE attributA != 'C';
```

Auf Sichten lassen sich Abfragen dann genauso stellen wie auf den Basistabellen (in Abbildung 9.13 sind dies tabelleA und tabelleB). Einfüge-, Lösch- und Aktualisierungsoperationen sind nur in manchen Sichten möglich. Dazu muss eine

Reihe von Bedingungen erfüllt sein, die wir im Laufe dieses Abschnitts genauer betrachten wollen. Die Anfrage

```
SELECT * FROM tabelleA, tabelleB WHERE attributA != 'C';
```

kann allerdings auf jeden Fall im Weiteren durch

```
SELECT * FROM sicht;
```

ersetzt werden. Sichten sind allerdings keine Momentaufnahmen bestehender Tabellen. Änderungen in den Basistabellen schlagen sich auch in der Sicht nieder. Fügen wir einen neuen Datensatz in die `tabelleA` ein:

```
INSERT INTO tabelleA SET attributA='D';
```

Wenn wir uns nun den gesamten Inhalt der Sicht anzeigen lassen, erhalten wir ein Ergebnis mit sechs Datensätzen, dargestellt in Abbildung 9.14.

sicht

attributA	attributB
A	X
A	Y
B	X
B	Y
D	X
D	Y

Abbildung 9.14 Sichten sind stets aktuell

Ähnlich verhält es sich mit den Funktionen von MySQL. Bei der Definition einer Sicht können Sie innerhalb des `SELECT`-Kommandos zum Beispiel auch Zeit- oder Datumsfunktionen wie `NOW()`, `CURDATE()` oder `CURTIME()` einsetzen. Deren Ergebnis wird nicht in einem Feld der Sicht gespeichert, sondern die Funktion wird bei einer Abfrage immer wieder aufs Neue ausgewertet. So erhalten Sie bei jeder Abfrage einen aktuellen Wert. Nicht alle Funktionen, die MySQL bereitstellt, können im `SELECT`-Teil einer Sicht benutzt werden. Grundsätzlich gilt: Jede parameterlose Funktion ist einsetzbar.

In unserem bisherigen Beispiel übernehmen wir die Attributsnamen von den Basistabellen. Wir können allerdings auch eigene Bezeichner verwenden. Dazu geben wir in der Sichtendefinition die (`attrListe`) an. Die einzelnen Bezeichner werden durch ein Komma voneinander getrennt. Wir erzeugen eine weitere Sicht auf den beiden Tabellen. Anstatt nun die Spalten der Sicht `attributA` und `attributB` zu nennen, vergeben wir zwei neue Namen:

```
CREATE VIEW sicht2 (variable,zuweisung) AS
SELECT * FROM tabelleA, tabelleB WHERE attributB = 'X';
```

Daraus resultiert eine Tabelle, deren Struktur und Inhalt Sie in Abbildung 9.15 sehen können. Die Bestandteile der Attributsliste korrespondieren mit den Spalten der darunter liegenden Tabellen. Die Attribute in der Liste müssen mit denen in der SELECT-Anweisung von der Menge her übereinstimmen. Die Zuweisung der Paarungen entsteht je nach Ordnung der beiden Listen. Das bedeutet, sicht2.variable korrespondiert mit tabelleA.attributA und zuweisung gehört zu tabelleB.attributB.

sicht2

variable	zuweisung
A	X
B	X
C	X
D	X

Abbildung 9.15 Sicht mit eigenen Bezeichnern

Der Inhalt einer Sicht kann dynamisch anhand zweier Algorithmen berechnet werden. Sofern Sie nicht MySQL die Wahl des richtigen Algorithmus überlassen wollen, können Sie mit ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE} in der Sichtdefinition explizit eine Herangehensweise vorschreiben. Der Wert UNDEFINED ist die Voreinstellung. Er besagt nichts anderes, als wenn Sie den ALGORITHM-Parameter weglassen: MySQL soll die Wahl automatisch treffen. Der MERGE-Algorithmus schreibt die Anfrage an die Sicht anhand der Sichtdefinition auf ein passendes Gegenstück auf die darunter liegende(n) Tabelle(n) um. So wird

```
SELECT * FROM sicht;
```

zu

```
SELECT * FROM tabelleA, tabelleB WHERE attributA != 'C';
```

Dies ist die von MySQL bevorzugte Alternative der Sichtauflösung, wenn sie auch einige Beschränkungen hat. Damit der MERGE-Algorithmus benutzt werden kann, muss eine Eins-zu-eins-Beziehung (*1:1*) zwischen Datensätzen in der Sicht und denen in den darunter liegenden Tabellen bestehen. Das impliziert, dass keine Aggregat-Funktionen (MAX(), MIN(), AVG() ...) mit MERGE verarbeitet werden können. Erzwingen Sie also diesen Algorithmus in der Sichtdefinition, die Aggregationen beinhaltet, führt das zwangsläufig zu einem Fehler oder einer Warnung.

Als letzte Alternative existiert der TEMPTABLE-Algorithmus. Bei jeder Anfrage an die Sicht wird eine temporäre Tabelle angelegt, die von der Struktur her mit der Sicht identisch ist. Als Datenbasis für die Anfrage wird dann diese temporäre Tabelle verwendet:

```
SELECT * FROM sicht;
```

wird zu

```
CREATE TEMPORARAY TABLE temp AS SELECT * FROM tabelleA, tabelleB ⤦
WHERE attributA != 'C';
SELECT * FROM temp;
```

Der Aufwand, kurzfristig eine weitere Tabelle zu erstellen, kann sich lohnen, wenn die Datenbank gut frequentiert ist. Dann werden relativ viele Sperren gesetzt und eine Anfrage unter Umständen von anderen oder von schreibenden Zugriffen blockiert. Für temporäre Tabellen besteht die Restriktion der Aggregatfunktionen nicht. Ebenso wenig problematisch sind `DISTINCT`, `UNION`, `GROUP BY` oder `HAVING`, die beim `MERGE`-Algorithmus Schwierigkeiten bereiten können.

Ist eine Sicht derart beschaffen, dass Sie Daten in sie einfügen können, lassen sich `INSERT`-Operationen überwachen. Einfügeoperationen auf Sichten greifen bis auf die darunter liegenden Tabellen durch, d.h., der Datensatz wird nicht wirklich in der Sicht angelegt, sondern in deren Basistabellen. Wollen Sie eine Überprüfung von Eingaben erwirken, können Sie das über den Parameter `WITH [CASCADED | LOCAL] CHECK OPTIONS` tun. Nehmen Sie die folgende Einfüge-Anweisung:

```
INSERT INTO sicht SET attributA='C';
```

Syntaktisch ist diese Anfrage vollkommen korrekt. Nur haben wir in der Definition von `sicht` explizit ausgeschlossen, dass `attributA` den Wert C haben darf. Diese Einfügeoperation wird jedoch ohne Beanstandung durchgeführt. Erzeugen wir eine neue Sicht. Sie soll strukturell identisch sein mit `sicht`, nur werden die Eingaben überprüft:

```
CREATE VIEW sicht3 AS
SELECT * FROM tabelleA, tabelleB WHERE attributA != 'C'
WITH CHECK OPTION;
```

Die Anweisung

```
INSERT INTO sicht3 SET attributA='C';
```

führt dann zu dem folgenden Fehler:

```
ERROR 1369 (HY000): CHECK OPTION failed 'phpmysql.sicht3'
```

Die optionalen Parameter `CASCADED` und `LOCAL` beziehen sich auf Sichten, deren Basis ebenfalls Sichten sind. `CASCADED` ist die mehrstufige Variante der `CHECK [LOCAL] OPTION`. Es werden also nicht nur die Bedingungen der aktuellen Sicht, sondern auch die der Basissicht überprüft. Anhand dessen wird dann entschieden, ob ein Datensatz eingefügt wird oder nicht.

9.5.2 Editierbare und erweiterbare Sichten

Wir haben schon mehrfach editierbare Sichten angesprochen. Unter welchen Umständen aber lassen sich die Inhalte einer Sicht bearbeiten? Generell lassen sich alle Sichten über ein `SELECT` abfragen. Editierbar bedeutet dann, dass `UPDATE`- und `DELETE`-Anweisungen ausgeführt werden. Dies betrifft nur eine Teilmenge aller Sichten. Und nur eine Untermenge aller editierbaren Sichten ist auch über `INSERT`-Kommandos erweiterbar.

Es gibt eine Handvoll Kriterien, die die Editierbarkeit einer Sicht verhindern. Dies sind zum einen bestimmte SQL-Fragmente wie `UNION`, `UNION ALL`, `DISTINCT`, `GROUP BY` und `HAVING`. Ist eines dieser Elemente in der Sichtdefinition vorhanden, lässt sich diese nicht editieren. Des Weiteren lässt sich eine Sicht nicht aktualisieren, wenn im `SELECT`-Teil der Definition eine Unterabfrage steht:

```
CREATE VIEW sicht4 AS
SELECT (SELECT attributA FROM tabelleA), attributB FROM tabelleB;
```

Ebenso verhält es sich, wenn eine Unterabfrage im `WHERE`-Teil zu finden ist, die dieselbe Basistabelle betrifft, die auch im `FROM`-Teil benutzt wird:

```
CREATE VIEW sicht5 AS
SELECT * FROM tabelleA
WHERE attributA != (SELECT MAX(attributA) FROM tabelleA);
```

Und letztlich ist eine Sicht dann nicht editierbar, wenn ihre Basis eine nicht editierbare Sicht ist oder wenn der `TEMPTABLE`-Algorithmus zur Berechnung erzwungen wird.

Um Daten in eine Sicht einfügen zu können, muss sie zum einen aktualisierbar sein und zum anderen die folgenden drei Bedingungen erfüllen.

1. Die Sicht muss alle Spalten der unterliegenden Tabelle(n) enthalten, die keinen Standardwert haben. Logischerweise kann eine Sicht nur denjenigen Spalten neue Werte zuweisen, die sie selbst umfasst. Beim Einfügen eines neuen Datensatzes in eine Basistabelle werden alle übrigen Attribute mit ihren Standardwerten belegt. Dabei darf kein Attribut leer bleiben.
2. Die referenzierten Tabellenattribute dürfen im `SELECT`-Teil der Sichtdefinition nicht durch beispielsweise Funktionen verarbeitet werden. Funktionen sind Abbildungen von Eingabeparametern auf Ausgabewerte. Für jede Kombination an Eingaben erzeugen sie eine immer gleiche Ausgabe. Im Umkehrschluss ist die Rückverfolgung von Ausgabewerten zu ihren Eingabeparametern nicht eindeutig. Nehmen Sie als Beispiel das PHP-Listing 9.1. Für die Zahl *10* können wir mit Bestimmtheit sagen, dass das Ergebnis der Funktion *z* lautet. Aber wenn wir das Ergebnis *z* kennen, wissen wir nicht, ob der Konverter mit dem

Wert *10* aufgerufen wurde; es könnte auch die *9* gewesen sein oder die *8* usw. Genauso verhält es sich auch mit Funktionen in SQL. Das Durchreichen von Werten aus einer Sicht in die Basistabellen kommt so einer Rückrechnung gleich.

```
<?php
function konverter ($zahl)
{
   return ($zahl > 10)?'a':'z';
}
?>
```

Listing 9.1 Zahl-Buchstaben-Konverter

3. Jedes Attribut im `SELECT`-Statement darf nur einmal vorhanden sein. Die `sicht6` im nächsten Absatz importiert `attributA` einmal unter dem Namen `faktor` und ein zweites Mal als `quantitaet`. Die folgende Einfügeoperation schlägt fehl, weil zwei verschiedene Werte in das `attributA` der Basistabelle eingetragen werden sollen.

```
CREATE VIEW sicht6 AS
SELECT attributA AS faktor, attributA AS quantität FROM tabelleA;
INSERT INTO sicht6 (faktor, quantität) VALUES (17,13);
```

Wir sind am Anfang dieses Abschnitts auf die Aktualität einer Sicht eingegangen. Die Inhalte der Sicht sind dynamisch erzeugte Anfrageergebnisse auf die Basistabellen. Diese Aktualität hat ihre Grenzen bei der Struktur der Sicht. Die von uns erstellte erste Sicht mit dem bezeichnenden Namen `sicht` umfasste alle Attribute aus dem kartesischen Produkt der Basisrelationen `tabelleA` und `tabelleB`, in denen `attributA` nicht den Wert C hat. Wie wir ferner erklärt haben, bilden die beiden verfügbaren Algorithmen die Struktur der Sicht bei Anfragen ab, der eine als Projektion auf die Ursprungstabelle, der andere erzeugt temporäre Tabellen. Was geschieht nun in einem solchen Fall, wenn wir die Basisrelation verändern? Zur Verdeutlichung noch einmal die Sichtdefinition und Projektion, wie der `MERGE`-Algorithmus sie benutzt:

```
CREATE VIEW sicht AS
SELECT * FROM tabelleA, tabelleB WHERE attributA != 'C';
```

Die Anfrage

```
SELECT * FROM sicht;
```

wird zu

```
SELECT * FROM tabelleA, tabelleB WHERE attributA != 'C';
```

Nun fügen wir ein neues Attribut zu einer der Basistabellen hinzu:

```
ALTER TABLE tabelleA ADD attributC CHAR(1) DEFAULT 'K';
```

Nun müsste die Anfrage `SELECT *` auch das neue Attribut umfassen. Dem ist aber nur so, wenn Sie die Relationen direkt abfragen. Die Sicht enthält weiterhin nur zwei Attribute. So konsequent ist MySQL allerdings nur beim Hinzufügen von Spalten zu Basistabellen. Entfernen Sie stattdessen Spalten oder modifizieren Sie sie in einer Form, die von der Sicht nicht unterstützt wird (Datentypänderungen etc.), sind teils nicht mehr brauchbare Sichten die Folge. Hier offenbart sich eine Schwäche des noch jungen Sichtenkonstrukts, die daraus resultiert, dass dazugehörige Kontrollmechanismen (`RESTRICT`, `CASCADE`) noch fehlen.

9.5.3 Ändern und löschen

Bislang haben wir nur Sichten angelegt und darauf verwiesen, dass Sie sie auch verändern und löschen können. Einen Weg, Modifikationen zu erreichen, kennen Sie bereits: Das `OR REPLACE` aus dem `CREATE VIEW`, mit dem eine Sicht komplett überschrieben wird. Beim ersten Hinsehen merken Sie, dass ein `ALTER VIEW` nichts anderes tut:

```
ALTER
[ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
VIEW vName
[(attrListe)]
AS selectKommando
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

Insbesondere ist nicht angedacht, dass nachträglich Spalten zur Sicht hinzugefügt oder gelöscht werden. Ähnlich eindeutig wie simpel ist auch das Kommando zum Löschen einer Sicht. Das

```
DROP VIEW [IF EXISTS]
vName [, vName...]
```

erinnert stark an ein `DROP TABLE`. Für die Verarbeitung von Sichten existieren Rechte, die teils durch Rechte auf Tabellen ersetzt werden und teils zusätzlich vergeben werden müssen. Um eine Sicht anzulegen, benötigen Sie das Recht *CREATE VIEW*. Sofern Sie die Option `OR REPLACE` einsetzen wollen, benötigen Sie zusätzlich das Recht *DROP*. Dieses ist auch zum Löschen einer Sicht notwendig. Ein explizites Recht zum Verändern einer Sicht gibt es nicht. Aus oben genannten Gründen reichen dafür die vorigen Rechte aus. Außerdem existiert das Recht *SHOW VIEW*, mit dem Sie sich unter anderem ansehen können, mit welchen SQL-Anweisungen Sichten erstellt wurden.

9.5.4 Ein praktisches Beispiel

Wir haben viel über die technischen Möglichkeiten von Sichten geschrieben, ohne auf deren praktischen Nutzen einzugehen. Dabei gibt es für Sichten diverse Einsatzmöglichkeiten. Zum einen stellen sie darunter liegende Relationen aufbereitet dar – mit zusätzlichen, vorberechneten oder aggregierten Attributen, die für weitere Verarbeitungsschritte als Basis dienen können. Zum anderen – und das wollen wir im Weiteren erläutern – können Sie damit eine gezielte Untermenge Ihres Datenbestandes für andere Datenbankbenutzer zugänglich machen. Wir werden nun

- eine Tabelle anlegen, die teils öffentliche und teils sensible Daten enthält,
- eine Sicht erzeugen, die nur die unbedenklichen Daten enthält,
- einen Datenbankbenutzer erstellen und seine Rechte so einrichten, dass er ausschließlich auf die Sicht zugreifen kann.

Als Beispiel dient uns eine Tabelle, in der Daten von Benutzern eines webbasierten Systems hinterlegt sind. Die umfassen unter anderem Benutzernamen, Anzahl der Einlog-Vorgänge, deren Geburtsdatum und Passwörter.

```
CREATE TABLE benutzer(
name VARCHAR(100) PRIMARY KEY,
anzahlLogins SMALLINT(5) UNSIGNED DEFAULT 0 NOT NULL,
geburtsdatum DATE,
passwort CHAR(32) NOT NULL);
```

Wir stören uns auch diesmal nicht an der schwedischen Collation. Unsere Passwörter werden natürlich nicht im Klartext in der Datenbank hinterlegt, sondern mit dem MD5-Algorithmus verschlüsselt.

Hinweis

Auch MD5 ist keine Garantie dafür, dass Ihre Passwörter sicher sind. Neben Versuchen, den Algorithmus durch *Trial&Error* umzukehren, existieren auch frei zugängliche Datenbanken im Internet, die zu vielen tausend Zeichenketten – potenziellen Passwörtern – die entsprechenden MD5-Schlüssel bereitstellen. Damit wird eine Rückwärtssuche möglich.

Als Merkmal, anhand dessen wir entscheiden können, ob ein Datensatz in die Sicht aufgenommen werden soll, fügen wir der Tabelle noch ein weiteres Attribut hinzu:

```
ALTER TABLE benutzer ADD frei TINYINT(1) NOT NULL DEFAULT 0;
```

Die Sicht `logStatistik` soll nur die Attribute `name` sowie `anzahlLogins` umfassen und nur diejenigen Datensätze enthalten, bei denen `frei` einen anderen Wert hat als die initiale 0. Diese Anforderungen führen zum folgenden `CREATE`-Befehl:

```
CREATE VIEW logStatistik (benutzer, login) AS
SELECT name, anzahlLogins FROM benutzer WHERE frei <> 0;
```

Nun erzeugen wir einen neuen Datenbankbenutzer namens `statistiker` und weisen ihm tabellenspezifische Rechte für die Sicht zu:

```
CREATE USER statistiker;
GRANT SELECT ON 'phpmysql'.'logStatistik' TO 'statistiker'@'%';
```

Vom Ergebnis können wir uns leicht in phpMyAdmin überzeugen. Abbildung 9.16 zeigt, wie sich die Oberfläche für den neuen Benutzer darstellt. Obwohl sich in unserer Datenbank *phpmysql* mehr als 20 Tabellen befinden, sieht der `statistiker` nur eine davon, nämlich die Sicht `logStatistik`. Deren Struktur stellt sich wie gewünscht dar: es sind lediglich zwei Attribute zu sehen. Auch auf eine andere Datenbank kann der Benutzer nicht zugreifen.

Hinweis

Um den Zugriff auf nur eine Datenbank zu beschränken, haben wir uns der Bordmittel von PHPMyAdmin bedient. Die dortige Konfiguration erlaubt uns, unter den Eintrag `$cfg['Servers'][$i]['only_db']` eine einzige Datenbank festzulegen, auf deren Zugriff ein Benutzer begrenzt wird.

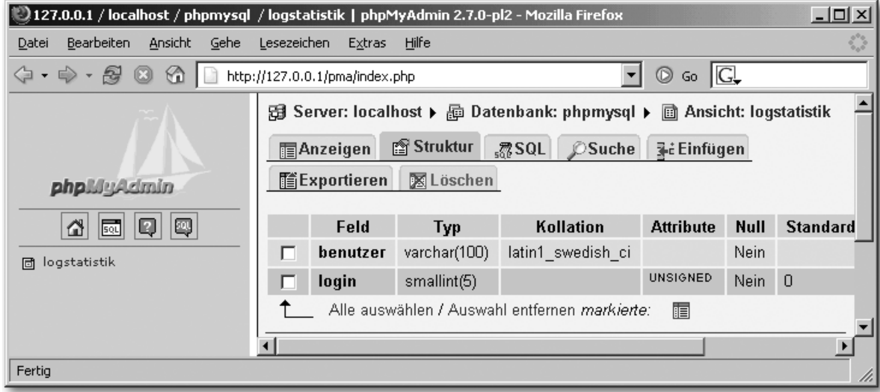

Abbildung 9.16 Dem neuen Benutzer zeigt sich ein sehr beschränktes Bild

9.6 Stored Procedures

Eine Stored Procedure (engl. für »gespeicherte Prozedur«) kapselt eine oder mehrere SQL-Anweisungen für den wiederholten Gebrauch. Daraus ergeben sich die gleichen Vorteile, die Kapselung auch in PHP – oder anderen Programmiersprachen – mit sich bringt:

- Die Abfolge der Anweisungen ist permanent gespeichert, auch über die Grenzen einer Datenbankverbindung oder die Laufzeit einer Serverinstanz hinweg.
- Anstatt die SQL-Kommandos zu späteren Zeitpunkten mehrfach manuell auszuführen, reicht ein Prozeduraufruf aus.
- Die Speicherung der Kommandos an mehreren Stellen wird vermieden.
- Darüber hinaus bietet MySQL noch den Vorteil der Zugriffsberechtigung, die es beispielsweise in PHP in dieser Art nicht gibt. Um eine Prozedur zu erstellen, zu verändern, zu löschen und auszuführen, muss der Datenbankbenutzer spezifische Rechte besitzen. Im Einzelnen sind das die Rechte *CREATE ROUTINE*, *ALTER ROUTINE* – zum Ändern und Löschen – und *EXECUTE*.

Eine Prozedur ist immer genau einer Datenbank zugeordnet. Das unterscheidet Routinen von Sichten, die wir im vorigen Abschnitt kennen gelernt haben, und Triggern, die auf dieses Unterkapitel folgen. Dabei wird die Zugehörigkeit auf Tabellen festgelegt.

9.6.1 Anlegen

Eine Prozedurdefinition ist nach dem folgenden Schema aufgebaut.

```
CREATE PROCEDURE [dbName.]spName
([param1 [, param2 ...]])
[LANGUAGE SQL]
Charakteristika der Prozedur
[SQL SECURITY {DEFINER | INVOKER}]
spKoerper
```

Eingeleitet wird die Definition durch `CREATE PROCEDURE`, gefolgt von dem eindeutigen Prozedurnamen `spName`. Sofern keine Angabe über die zugehörige Datenbank `dbName` gemacht wird, wird die Prozedur der aktuellen Datenbank zugerechnet. Es gelten wiederum dieselben Regeln für die Namensvergabe wie bei den bislang bekannten MySQL-Objekten.

Tipp

Wenn Sie eine Prozedur schreiben, deren Name identisch ist mit einer MySQL-eigenen Funktion, dann setzen Sie ein Leerzeichen zwischen Prozedurnamen und Parameter-Klammern. Am besten aber erwägen Sie erst gar nicht, eine gleichnamige Prozedur zu schreiben, da die Verwechslungsgefahr – für Sie und für andere Anwender – zu groß ist.

Gefolgt wird der Prozedurname von der Parameterliste. Die Liste ist nicht optional, muss also bei jeder Definition vorhanden sein. Eine Prozedur ohne Parameter muss eine leere Parameterliste enthalten. Jeder einzelne Parameter ist nach dem Muster

```
[IN | OUT | INOUT] name typ
```

aufgebaut. `IN` ist die Standardeinstellung und bezeichnet Eingabeparameter. `OUT` steht demnach für ein Ergebnis der Prozedur. Hat eine Prozedur Parameter vom Typ `INOUT`, so empfängt dieser einen Wert aus einer Systemvariablen, verarbeitet ihn innerhalb der Routine und schreibt den neuen Wert in die Variable zurück. Der Unterschied zwischen den Parametertypen ist in Abbildung 9.17 grafisch verdeutlicht. Der Parametername muss innerhalb der Liste eindeutig sein. `typ` bezeichnet einen der von MySQL unterstützten Datentypen.

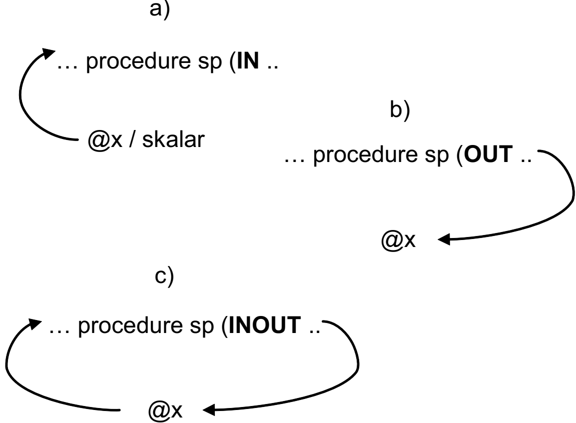

Abbildung 9.17 Parameterübergabe in Stored Procedures

`LANGUAGE SQL` gibt an, in welcher Sprache die Prozedur geschrieben ist. SQL ist Standard und in MySQL 5.0 die einzige Alternative, die benutzt werden kann. Es wird Unterstützung für weitere Programmiersprachen geben; eine der ersten wird PHP sein. Erst wenn es mehrere Sprachalternativen gibt, wird die Angabe von `LANGUAGE` relevant. In den kommenden Beispielen geben wir es aber dennoch mit an.

Die Charakteristika einer Prozedur enthalten Angaben darüber, ob und in welcher Form SQL innerhalb des Prozedurkörpers zum Einsatz kommt. Die alternativen Möglichkeiten sind:

- `CONTAINS SQL`
- `NO SQL`
- `READS SQL DATA`
- `MODIFIES SQL DATA`

Diese Angaben haben keinen Einfluss auf den Ablauf der Routine. Deshalb wollen wir nicht weiter darauf eingehen.

`SQL SECURITY` ist eine Neuerung der SQL-Version SQL:2003. Damit kann festgelegt werden, aus wessen Sicht die Prozedur ausgeführt werden soll. Setzen Sie `SQL SECURITY DEFINER`, dann hat die Prozedur die Rechte des Datenbankbenutzers, der die Prozedur geschrieben hat. Hingegen ist `SQL SECURITY INVOKER` auf die Berechtigungen beschränkt, die der aktuelle Benutzer besitzt, der die Routine aufruft.

Wichtig sind die Berechtigungen deshalb, weil innerhalb des Prozedurkörpers `spKoerper` ein großer Anteil der SQL-Anweisungen erlaubt sind, die Sie bislang kennen gelernt haben. Dies schließt unter anderen alle Punkte aus der nachstehenden Liste mit ein:

- Befehle der Daten-Definitions-Sprache (DDL), wie etwa `CREATE`, `DROP`, `ALTER`
- Anweisungen der Daten-Manipulations-Sprache (DML) wie `INSERT`, `UPDATE`, `DELETE`
- Kommandos zum Starten und Beenden von Transaktionen, wie `START TRANSACTION`, `COMMIT`, `ROLLBACK`
- Befehle, die eine Menge zurückgeben, wie `SELECT` ohne die Angabe von `INTO`, `EXPLAIN`
- Kontrollkonstrukte

Hingegen sind die folgenden Befehle nicht von einer Prozedur aufrufbar:

- `LOAD DATA INFILE`
- `USE datenbank`
- `{CREATE | ALTER | DROP} PROCEDURE`
- `{CREATE | DROP} TRIGGER`

Wenn Sie im Prozedurkörper mehr als eine Anweisung geben, müssen Sie den Anfang und das Ende des Befehlsblocks mit `BEGIN` und `END` kennzeichnen, wie im nachstehenden Code zu sehen ist:

```
CREATE PROCEDURE holeQuadrat (IN p INT)
LANGUAGE SQL
SQL SECURITY DEFINER
BEGIN
   ...
END;
```

Zwischen den einzelnen SQL-Anweisungen des Körpers wird das Semikolon (»;«) als Trennzeichen verwenden. In diesem Fall kommt es zu syntaktischen Komplikationen, denn das Semikolon gilt standardmäßig als Befehlsabschlusszeichen in SQL. Das bedeutet, der SQL-Interpreter hält die Prozedurdefinition für beendet, sobald das erste Semikolon im Kommando auftritt. Um diesem Fehler aus dem Weg zu gehen, können Sie alternative Abschlusszeichen für den SQL-Interpreter setzen.

Hinweis

Sie haben schon richtig gelesen: Wir ändern damit nicht etwa das Trennzeichen, das innerhalb des Prozedurkörpers die einzelnen Befehle voneinander trennt. Wir bringen stattdessen den SQL-Interpreter dazu, bei allen anderen Anweisungen ein anderes Zeichen als Abschluss zu akzeptieren.

Das neue Befehlsabschlusszeichen legen Sie mit dem Befehl `DELIMITER` fest. Es gilt, solange Sie die Verbindung zur Datenbank nicht abbrechen. Sofern Sie das Kommandozeilenwerkzeug *mysql* benutzen, kennen Sie bereits ein weiteres vordefiniertes Zeichen. Es wird Ihnen bei jedem Start des Programms angezeigt:

```
Welcome to the MySQL Monitor. Commands end with ; or \g.
...
mysql>
```

Sie können also `\g` als Befehlsabschlusszeichen verwenden. Der Begriff »Zeichen« ist wie Sie sehen irreführend. Es lassen sich auch Zeichenketten zum Abschluss eines Befehls festlegen. Diese dürfen dann aber logischerweise nirgendwo in der Anweisung vorkommen. Die offizielle MySQL-Referenz nutzt den doppelten Slash (»//«), und so wollen wir es auch halten:

```
DELIMITER //
```

Nach Ausführung dieses Befehls ist das Semikolon frei für den Gebrauch als Trennzeichen in einer Prozedur. Wir füllen nun die eingangs definierte Stored

Procedure mit Leben. Sie soll eine Datenbankabfrage ausführen und den Eingabeparameter einbeziehen:

```
CREATE PROCEDURE holeQuadrat (IN p INT)
LANGUAGE SQL
SQL SECURITY DEFINER
BEGIN
   SET @x = p*p;
   SELECT * FROM tabelle WHERE limit > @x;
END;//
```

Zuerst wird eine Variable `@x` mit dem Quadrat des Eingabeparameters belegt. Danach wird `@x` in einer `WHERE`-Klausel der Abfrage eingesetzt. Diesmal haben wir das Befehlsabschlusszeichen explizit mit eingeschlossen.

9.6.2 Aufrufen

Als Resultat wird uns bei einem Prozeduraufruf wie

```
CALL holeQuadrat(4)//
```

entweder eine leere Menge oder eine Tabelle mit allen Datensätzen aus `tabelle` angezeigt, die der Bedingung `WHERE limit > 16` genügen. Jeder Prozeduraufruf bedarf des Schlüsselworts `CALL`. Dies dient der Unterscheidung von Prozeduren und Funktionen – die beide unter dem Namen Routinen zusammengefasst werden.

Aus dem Beispiel wird noch ein Vorteil der Kapselung in Stored Procedures deutlich. Der Datenbankbenutzer, der die Prozedur ausführt, hat keinerlei Informationen darüber, wie und welche Tabellen verarbeitet werden. Durch das `SQL SECURITY DEFINER` muss er auch nicht zwingend alle Rechte haben, die die Ausführung verlangt. Mit Prozeduren ist es also möglich, kontrolliert Operationen auf Tabellen zu erlauben, die derjenige, der die Routine aufruft, auf anderem Wege nicht ausführen könnte.

Es ist nicht möglich, eine Prozedur direkt von einer Abfrage (`SELECT`) aus aufzurufen. Stattdessen muss ein Umweg über eine Variable gemacht werden. Um dies zu verdeutlichen, schreiben wir eine neue Prozedur, die einen Parameter vom Typ `OUT` besitzt. Bei der Ausführung wird die Variable, die wir übergeben, mit einem Wert gefüllt, den wir daraufhin auslesen können:

```
CREATE PROCEDURE setzeP (OUT p INT)
LANGUAGE SQL
SQL SECURITY INVOKER
SET p=10//
```

Wenn wir nun diese Routine direkt aufzurufen versuchen, tritt ein Fehler auf:

```
SELECT CALL setzeP()//
```

Zum einen hat die Parameterliste nicht die korrekte Form und zum anderen lassen sich CALL und SELECT auf diese Weise nicht kombinieren. Stattdessen setzen wir eine Variable ein:

```
CALL setzeP(@variable)//
SELECT @variable//
```

Die Abfrage liefert uns die gewünschte Zahl 10. Analog dazu arbeiten Prozeduren mit Parametern vom Typ INOUT:

```
CREATE PROCEDURE verzehnfache (INOUT p INT)
LANGUAGE SQL
SQL SECURITY INVOKER
SET p=10*p//
```

Ein Aufruf bedarf als Eingabeparameter einer Systemvariablen, die auch dazu dient, den Rückgabewert aufzunehmen. Wir initialisieren also die Variable @x und stoßen damit die Prozedur verzehnfache an:

```
SET @x = 5//
CALL verzehnfache(@x)//
SELECT @x//
```

Das Resultat ist 50.

9.6.3 Ändern und Löschen

Prozeduren, die Sie einmal erstellt haben, sind nur bedingt modifizierbar. Mit dem Befehl ALTER PROCEDURE lassen sich lediglich Einstellungen zu SQL SECURITY und den Charakteristika der Prozedur machen. Es ist also nicht möglich, den Körper der Routine oder die Parameterliste abzuändern. Um eine so grundlegende Änderung durchzuführen, müssen Sie die vorhandene Version der Prozedur löschen und sie neu anlegen. Die (hier vereinfacht dargestellte) Syntax des ALTER-Befehls lautet:

```
ALTER PROCEDURE spName
SQL SECURITY {DEFINER | INVOKER}
```

Zum Löschen der Prozedur können Sie in Analogie zu ähnlichen Anweisungen das

```
DROP PROCEDURE [IF EXISTS] spName
```

nutzen. Das [IF EXISTS] verhindert, dass ein Fehler auftritt, wenn die Prozedur nicht vorhanden ist.

9.6.4 Variablen

Bislang haben wir innerhalb von Prozeduren nur mit Eingabeparametern gearbeitet, die mitunter aus globalen Variablen stammten. Es ist jedoch auch möglich, innerhalb einer Prozedur lokale Variablen zu deklarieren. Eine Prozedur hat einen eigenen Namensraum, der durch das BEGIN … END aufgespannt wird. Variablen, die nach dem BEGIN initiiert werden, sind nur so lange gültig, bis der SQL-Interpreter das dazugehörige END erreicht. Da es erlaubt ist, BEGIN ... END-Blöcke zu verschachteln, können in einer Routine mehrere Namensräume bestehen. Demnach sind gleichnamige lokale Variablen möglich.

Hintergrundwissen

Wir erinnern uns zurück an die Einführung in PHP: Eine Variablenzuweisung ist eine Name-Wert-Namensraum-Bindung.

Lokale Variablen erstellen Sie mit dem Befehl DECLARE, der direkt am Anfang eines BEGIN ... END Blocks stehen muss. Seine Syntax ist wie folgt:

```
DECLARE var_name typ [DEFAULT wert]
```

Jede Variable hat einen der 27 Datentypen, die wir im Einführungskapitel zu MySQL vorgestellt haben. Standardmäßig ist eine Variable nullwertig, solange Sie keine Angabe per DEFAULT machen. wert kann sowohl konstant als auch durch einen Ausdruck festgelegt werden, wie die folgende Prozedur mit dem Namen jetzt beweist:

```
CREATE PROCEDURE jetzt (OUT p CHAR(16))
LANGUAGE SQL
BEGIN
   DECLARE x TIMESTAMP DEFAULT NOW();
   SET p=DATE_FORMAT(x,'%e.%m.%Y %H:%i');
END;//
CALL jetzt(@z)//
SELECT @z//
```

Die Prozedur ist ein wirklich umständlicher Weg, das aktuelle Datum samt Uhrzeit an eine globale Variable zu binden, allerdings war hierbei der Zweck ja ein anderer. Einfacher wäre es auf die altbekannte Art gegangen:

```
SET @z = DATE_FORMAT(NOW(),'%e.%m.%Y %H:%i')//
```

Neue Werte werden den Variablen, wie gesehen, mit dem bekannten SET zugewiesen.

9.6.5 Kontrollstrukturen

Innerhalb von Stored Procedures können Sie auf eine Reihe von Kontrollstrukturen zugreifen, ähnlich denen, die Sie für die Programmiersprache PHP kennen gelernt haben. Das macht die Prozedur zu einem mächtigen Konstrukt (im Sinne der Informatik bedeutet mächtig kurz gesagt so viel wie flexibel bzw. reich an Funktionalität). Mit den Kontrollstrukturen können Sie sowohl Fallunterscheidungen für zwei und mehr Alternativen als auch Schleifen realisieren. Sie sind jedoch nicht identisch mit den Funktionen zum Kontrollfluss, die wir in einem der vorigen Kapitel zu MySQL vorgestellt haben. Um die Beispiele im Folgenden kurz und klar zu halten, verzichten wir auf Angaben wie LANGUAGE SQL oder SQL SECURITY.

Bedingte Befehlsausführung mit IF

Kern eines IF-Konstrukts ist eine Bedingung, also ein Ausdruck, der im booleschen Sinne zu wahr oder falsch ausgewertet werden kann. Abhängig davon werden ein oder mehrere alternative Ausführungsblöcke verarbeitet. Aufgebaut ist ein IF nach dem folgenden Muster:

```
IF bedingung THEN anweisungen
   [ELSEIF bedingung THEN anweisungen] ...
   [ELSE anweisungen]
END IF
```

Über die Option ELSEIF lassen sich weitere IF-Konstrukte definieren, die verschachtelt ausgeführt werden. Das Konstrukt endet immer mit einem END IF.

Innerhalb von bedingung können Sie die Vergleichsoperatoren von MySQL einsetzen. Bei der Prüfung auf den Wahrheitsgehalt der Bedingung wird das Zeichen = als Test auf Gleichheit interpretiert. Demnach ist es an dieser Stelle nicht möglich, Variablen zu setzen. Dies ist nur in anweisungen erlaubt. Beachten Sie, dass ELSEIF und ELSE optional sind, das THEN ist hingegen zwingend erforderlich.

Im folgenden Beispiel wollen wir abhängig von einem Wert p eine Ausgabe erreichen. Trifft die Bedingung zu, so wird die gesamte Tabelle produkte ausgegeben, anderenfalls lediglich die Zahl 0.

```
CREATE PROCEDURE bedingt (IN p BOOL)
BEGIN
   IF p THEN
```

```
      SELECT * FROM produkte;
   ELSE
      SELECT 0;
   END IF;
END; //
```

Hinweis

Wir verwenden in diesem Fall den Pseudo-Datentyp BOOL, den wir Ihnen bislang verschwiegen haben. Es handelt sich dabei um keinen wirklichen Datentyp, sondern ein Synonym für TINYINT(1), bei dem 0 zu falsch ausgewertet wird, 1 bis 9 hingegen sind wahr.

Fallunterscheidung mit CASE

Das CASE-Kostrukt besteht aus einer Reihe von Wenn-Dann-Blöcken, die je einen booleschen Test und eine damit verbundene Konsequenz enthalten. Die Alternativen werden der Reihe nach abgearbeitet. Sobald ein Test das erste Mal zu wahr ausgewertet wird, wird das CASE-Konstrukt verlassen. Ein CASE ist durch eine Reihe verschachtelter IF / ELSEIF ersetzbar.

Variante 1: Tests für eine Variable

Die Tests, die sich alle auf eine zentrale Variable beziehen, prüfen alle auf Gleichheit des Variableninhalts mit einem Referenzwert.

```
CASE variable
   WHEN wert THEN anweisungen
   [WHEN wert THEN anweisungen] ...
   [ELSE anweisungen]
END CASE
```

Im Kopf des Konstrukts geben Sie den Namen einer Variablen an, die in den WHEN-Blöcken geprüft werden soll. Theoretisch können Sie auch einen konstanten Wert als variable definieren, damit wird Ihre Prozedur aber statisch und mitunter sinnfrei. Das folgende Beispiel gibt in Abhängigkeit einer Zahl p eine Meldung mittels SELECT aus:

```
CREATE PROCEDURE fallunterscheidung1 (IN p TINYINT)
BEGIN
   CASE p
      WHEN 1 THEN SELECT 'p ist 1' AS Meldung;
      WHEN 2 THEN SELECT 'p ist 2' AS Meldung;
      ELSE SELECT 'p ist weder 1 noch 2' AS Meldung;
   END CASE;
END; //
```

Variante 2: Unabhängige Tests

Bei dieser Abwandlung der vorigen Methode enthalten die `WHEN`-Anweisungen selber Tests, die sich nicht zwingend auf eine zentrale Variable bzw. einen zentralen Wert beziehen müssen. Stattdessen sind die Einzelprüfungen unabhängig voneinander.

```
CASE
   WHEN bedingung THEN anweisungen
   [WHEN bedingung THEN anweisungen] ...
   [ELSE anweisungen]
END CASE
```

Die Angabe einer Variablen nach dem Schlüsselwort `CASE` entfällt. Alle Vorkommen von `bedingung` sind boolescher Natur.

```
CREATE PROCEDURE fallunterscheidung2 (IN p TINYINT)
BEGIN
   CASE
      WHEN p<5 THEN SELECT 'p kleiner 5 ' AS Meldung;
      WHEN p<10 THEN SELECT 'p kleiner 10' AS Meldung;
      WHEN CURDATE() = '2006-02-28' THEN SELECT NOW();
   END CASE;
END; //
```

Die dritte Abfrage bezieht sich nicht auf den Parameter `p` und führt nur dann zu einer Ausgabe, wenn `p` mindestens 10 ist, weil dann die vorigen Abfragen fehlschlagen und der aktuelle Tag der 28. Februar 2006 ist.

Schleifen mit LOOP

Bei dem `LOOP`-Konstrukt handelt es sich um eine einfache Schleife ohne explizite Abbruchbedingung. Das bedeutet, die Schleife läuft endlos, es sei denn, Sie beenden sie mit dem Befehl `LEAVE`, mit dem alle Prozedurschleifen abgebrochen werden können. Um die Schleife zum richtigen Zeitpunkt abzubrechen, benötigen Sie ein Hilfskonstrukt in Form eines `IF` oder `CASE`, das im Schleifenrumpf ausgeführt wird.

```
[bezeichner:] LOOP
   anweisungen;
END LOOP [bezeichner]
```

Mit dem optionalen `bezeichner` können Sie für die Schleife einen Namen vergeben. Bezeichner werden von `LEAVE bezeichner` benutzt, um gezielt Schleifen zu beenden und bei geschachtelten Schleifen möglicherweise mehrere auf einmal abzubrechen. Es ist möglich, den Bezeichner einer `LOOP`-Schleife nur zu Beginn zu

setzen und den Endbezeichner wegzulassen. Sofern Sie beide Bezeichner einsetzen, müssen diese identisch sein. Sie dienen damit der Übersicht. Das folgende Beispiel nutzt eine Zählvariable, um eine Einfügeoperation mehrfach auszuführen.

```
CREATE PROCEDURE ZehnNeueDaten ()
BEGIN
   DECLARE x INT DEFAULT 0;
   schleife: LOOP
      INSERT INTO tabelle SET id=x, zeit=NOW();
      SET x = x + 1;
      IF x >= 10 THEN
         LEAVE schleife;
      END IF;
   END LOOP;
END;//
```

Bei einer Ausführung werden in `tabelle` zehn neue Datensätze angelegt, durchnummeriert von 0 bis 9.

Neben `LEAVE` existiert noch ein weiteres Kommando, das die Ausführung eines Schleifendurchlaufs beeinflussen kann: `ITERATE`. Dadurch wird der aktuelle Durchlauf beendet, jedoch anders als bei `LEAVE` ein neuer begonnen. Idealerweise wird `ITERATE` ebenfalls in einem eingebetteten Kontrollkonstrukt eingesetzt.

```
CREATE PROCEDURE GeradeNeueDaten ()
BEGIN
   DECLARE x INT DEFAULT 0;
   schleife: LOOP
      SET x = x + 1;
      IF x % 2 THEN
         ITERATE schleife;
      END IF;
      IF x > 10 THEN
         LEAVE schleife;
      ELSE
         INSERT INTO tabelle SET id=x, zeit=NOW();
      END IF;
   END LOOP;
END;//
```

Durch das ... `IF x%2 THEN ITERATE` ... werden alle Schleifendurchläufe abgebrochen, bei denen `x` einen ungeraden Wert hat.

Schleifen mit REPEAT

Mit einem REPEAT implementieren Sie Schleifen mit Abbruchbedingung, die mindestens einmal durchlaufen werden. Das bedeutet, der Schleifenrumpf wird vor dem Test auf Gültigkeit der Bedingung ausgeführt. Schematisch sieht das folgendermaßen aus:

```
[bezeichner:] REPEAT
   anweisungen
   UNTIL bedingung
END REPEAT [bezeichner]
```

Analog zur LOOP-Schleife wollen wir auch hiermit eine Einfügeoperation implementieren:

```
CREATE PROCEDURE wiederhole ()
BEGIN
   DECLARE x INT DEFAULT 0;
   REPEAT
      INSERT INTO tabelle SET id=x, zeit=NOW();
      SET x = x + 1;
      UNTIL x > 10
   END REPEAT;
END;//
```

Es werden elf Datensätze angelegt, durchnummeriert von 0 bis 10. Dass der Datensatz mit der Nummer 10 noch vorhanden ist, liegt an der oben beschriebenen Auswertungsreihenfolge.

Schleifen mit WHILE

Eine WHILE-Schleife ähnelt einem REPEAT, nur dass die Überprüfung der Abbruchbedingung vor dem Schleifenrumpf ausgewertet wird. Damit ist nicht zwangsläufig garantiert, dass der Rumpf überhaupt einmal verarbeitet wird. Die Syntax des Konstrukts stellt sich wie folgt dar:

```
[bezeichner:] WHILE bedingung DO
   anweisungen
END WHILE [bezeichner]
```

Die nachstehende Prozedur erledigt genau die gleichen Schritte wie die aus den vorigen Beispielen. Nur ist hier die Auswertungsreihenfolge vertauscht.

```
CREATE PROCEDURE ()
BEGIN
   DECLARE x INT DEFAULT 0;
   WHILE x < 10 DO
```

```
      INSERT INTO tabelle SET id=x, zeit=NOW();
      SET x = x + 1;
   END WHILE;
END;//
```

Wir erhalten nur noch zehn Datensätze. Das resultiert allerdings lediglich daraus, dass wir den Operator »echt kleiner« (»<«) einsetzen, der keine gleichwertige Alternative zu »echt größer« aus der vorigen Prozedur ist. Wir können das identische Ergebnis erhalten, indem wir stattdessen den Operator für »kleiner gleich« einsetzen.

9.7 Trigger

Mit Triggern (engl. für »Auslöser, Abzug«) lassen sich SQL-Befehle ereignisgesteuert ausführen. Jeder Trigger gehört zu einer Datenbanktabelle. Ereignisse, die einen Trigger starten lassen – in Analogie zu einem Auslöser bei einem Gewehr spricht man davon, dass ein Trigger feuert – können dann Einfüge-, Aktualisierungs- und Löschoperationen auf dieser Tabelle sein. Ein Trigger kann allerdings nur eines dieser Ereignisse zur Zeit überwachen – im Zweifel bedarf es deshalb mehrerer Trigger pro Tabelle.

9.7.1 Anlegen

Einen Trigger richten Sie wie folgt ein:

```
CREATE TRIGGER trigName
{BEFORE | AFTER}
{INSERT | UPDATE | DELETE}
ON tblName
FOR EACH ROW triggerBefehl
```

Jeder Trigger muss einen eindeutigen Namen `trigName` bekommen. Es gelten die gleichen Begrenzungen wie für Tabellen- und Sichtnamen. Ein Trigger darf also insbesondere nicht länger als 64 Zeichen sein.

Der SQL-Befehl, der von einem Trigger angestoßen wird, kann entweder vor oder nach der Tabellenoperation verarbeitet werden. Dafür existieren die exklusiven Schlüsselwörter `BEFORE` und `AFTER`. Zusammen mit den drei Tabellenoperationen `INSERT`, `UPDATE` und `DELETE` ergeben sich somit sechs Möglichkeiten zur Triggerausführung.

Da es nicht erlaubt ist, zwei oder mehr Trigger für dieselbe Kombination aus Tabelle und Ereignis zu definieren, kann es pro Tabelle somit auch nur bis zu sechs Triggern geben.

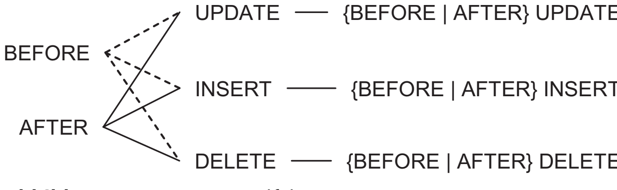

Abbildung 9.18 Triggervielfalt

Als `tblName` müssen Sie einen gültigen Tabellennamen angeben. Das bedeutet, Sie müssen beim Aufbau Ihrer Datenbank zuerst die nötigen Tabellen spezifizieren, um dem Fehler

```
ERROR 1146 (42S02): Table 'dbName.tblName' doesn't exist
```

zu entgehen. Es ist ebenfalls nicht möglich, einen Trigger für eine temporäre Tabelle oder für eine Sicht zu definieren. Das `FOR EACH ROW` verdeutlicht, dass ein Trigger auf jeden Datensatz (einzeln) einwirkt, der von der Datenoperation betroffen ist, und nicht nur ein einziges Mal auf die Tabelle. Innerhalb von `triggerBefehl` können Sie eine oder mehrere SQL-Anweisungen ausführen. Bei mehr als einem Befehl muss `triggerBefehl` durch `BEGIN` und `END` eingeschlossen werden. Erlaubt ist, was Sie im vorigen Abschnitt über Stored Procedures kennen gelernt haben. Nicht erlaubt ist allerdings der Aufruf einer Prozedur mittels `CALL` oder das Starten bzw. Beenden einer Transaktion mit `BEGIN` und `COMMIT` bzw. `ROLLBACK`.

Zusätzlich unterstützt MySQL in Zusammenhang mit Triggern die Schlüsselwörter `OLD` und `NEW`. Damit lassen sich Spalten ansprechen und manipulieren. `OLD` repräsentiert dabei den Datensatz in dem Zustand, wie er vor der Abarbeitung der Tabellenoperation vorzufinden ist. `OLD.attrName` dient also dazu, ein Attribut auszulesen. `NEW` hingegen steht für den Datensatz nach Bearbeitung. Mit `NEW.attrName` können Sie dementsprechend neue Werte auslesen oder für einzelne Spalten vergeben. Veranschaulichen wir das mit einem Beispiel. Wir betrachten die Benutzertabelle, die wir im Abschnitt über Sichten erzeugt haben, und wollen nun Trigger darauf definieren. Hier sehen Sie noch einmal die (komplette) Definition:

```
CREATE TABLE benutzer(
name VARCHAR(100) PRIMARY KEY,
anzahlLogins SMALLINT(5) UNSIGNED DEFAULT 0 NOT NULL,
geburtsdatum DATE,
passwort CHAR(32) NOT NULL,
frei TINYINT(1) NOT NULL DEFAULT 0);
```

Wir wollen nun mit unserem Trigger Folgendes bewirken:

- Der Benutzername soll durchweg aus Kleinbuchstaben bestehen und
- das Passwort soll vor dem Speichern MD5-codiert werden.

Damit übernimmt die Datenbank Aufgaben, die wir auch auf Applikationsebene hätten erledigen können. Was im Einzelfall performanter ist, kann von Ihrer Systemumgebung abhängen. Besonders wenn Applikationsserver und Datenbankserver nicht auf demselben Rechner laufen, können sich die Auslastung und die Hardware stark unterscheiden. Im Zweifel helfen nur Leistungstests. Wichtig für unser Beispiel ist nur, dass wir die Aufgaben nicht doppelt – also von der Applikation und der Datenbank – erledigen lassen. Im Fall der Großbuchstaben-Elimination wäre eine mehrfache Verarbeitung vollkommen unproblematisch. Ein doppeltes Codieren hingegen macht dem Nutzer das Einloggen unmöglich. Unseren Trigger legen wir wie folgt an:

```
CREATE TRIGGER checkeBenutzer
BEFORE INSERT
ON benutzer
FOR EACH ROW
BEGIN
   SET NEW.name = LOWER(NEW.name);
   SET NEW.passwort = MD5(NEW.passwort);
END;//
```

Hinweis

Beachten Sie, dass bei der Benutzung von `BEGIN … END` der Einsatz des Befehls `DELIMITER` vor der Triggerdefinition notwendig ist. Wir benutzen ab hier wieder den doppelten Slash als Befehlsabschlusszeichen.

Mit den beiden Anweisungen innerhalb von `triggerBefehl` überschreiben wir die eingefügten Werte mit denen, die wir aus den Funktionen `LOWER()` und `MD5()` bekommen.

`OLD` und `NEW` beziehen sich also nicht darauf, was vor und nach Ausführung des Triggers für Werte bestehen. Der Bezugspunkt ist die Einfügeoperation (oder in anderen Fällen die Aktualisierung bzw. Löschung). Das impliziert, dass Sie *vor* einem `INSERT` keine Daten über ein `OLD` referenzieren können, denn bevor ein Datensatz eingefügt wird, existiert er logischerweise nicht. Genauso wenig existiert *nach* einem `DELETE` ein `NEW.attrName`, denn nachdem das Tupel gelöscht ist, hat es keine Werte mehr. Alleine bei der Verwendung von `UPDATE` existieren `NEW` und `OLD` sowohl vorher als auch hinterher.

9.7.2 Wozu sind Trigger aber notwendig?

Wie gesagt, viele Aufgaben, die Sie mit Triggern angehen können, lassen sich auf Applikationsebene bearbeiten. Eine Faustregel lässt sich etwa so formulieren: **Wenn die Ablauflogik es nicht erfordert, dass Sie Daten von der Applikation zur Datenbank schicken, tun Sie es auch nicht. Wenn die Datenbank stattdessen zwingend eingesetzt wird, bearbeiten Sie die Eingaben dort.** Diese Faustregel gilt dann, wenn nicht eine Komponente signifikant schneller ist als die andere. Sie beruht darauf, dass der Transfer der Daten zwischen den Systemebenen das entscheidende Kriterium ist, über das Ihre Webapplikation ausgebremst wird.

Die nachstehende Liste enthält einige Beispiele zur Verdeutlichung. In allen Fällen kann die Aufgabe von der Datenbank (Triggern) oder der Applikation übernommen werden.

1. Eingabevalidierung: Bevor Sie Benutzerdaten in Ihrer Datenbank ablegen, sollten diese syntaktisch und logisch überprüft werden – Sicherheitschecks lassen wir in diesem Fall außer Acht (dafür ist die Applikationsebene verantwortlich). Es geht also beispielhaft darum, ob eine deutsche Postleitzahl ein fünfstelliger Integer ist. Wenn ja, dann wird sie in der Datenbank abgelegt, wenn nicht, dann tritt ein Fehler auf und der Anwender muss von der Applikation darüber informiert werden. In diesem Fall die Datenbank zu bemühen, bedeutet zusätzlichen und unnötigen Datentransfer, siehe Abbildung 9.19 a) und b). Unsere Empfehlung: Verarbeitung auf der Applikationsebene.
2. Eingaben verändern: Dies ist genau das, was wir mit dem Trigger im vorigen Beispiel gemacht haben. Bevor die Daten in den Tabellen abgelegt werden, werden sie über Funktionen oder mathematisch verarbeitet – z.B. inkrementiert, gerundet usw. Unabhängig von der Verarbeitung landen die Daten letztlich in der Datenbank, müssen also mindestens einmal zwischen Applikation und Datenbank verschickt werden, siehe Abbildung 9.19 c) und d). Für diese Aufgabe ist die Datenbank eine gute Wahl.
3. Überwachung von Datenbewegungen: Logging ist per se ereignisgesteuert (genau wie PHP und Trigger in MySQL). Wenn Sie SQL-Anweisungen im laufenden System protokollieren möchten (z.B. Einfügeoperationen), können Sie Ihre Log-Tabellen von der Applikationsebene mit SQL-Befehlen füllen. Pro Abfrage wird dann ein weiteres Kommando an die Datenbank geschickt. Auf Datenbankebene reagiert ein Trigger auf eine SQL-Operation und führt weitere aus. Die Applikationsvariante erzeugt demnach mehr Datentransfer (bis zu 200 %, abhängig vom Ausmaß der Protokollierung), dargestellt in Abbildung 9.19 e) und f). Auch hier sind Trigger die bessere Wahl – wenn man davon absieht, dass es keinen `SELECT`-Trigger gibt.

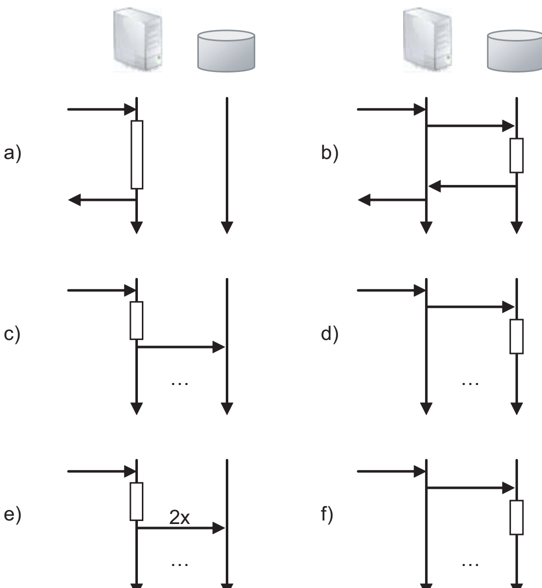

Abbildung 9.19 Datentransfer zwischen Applikation und Datenbank

Wir möchten allerdings betonen, dass es sich bei den Beispielen nur um Empfehlungen handelt. Wenn Ihre Applikation immer noch performant ist, obwohl Sie das Logging von der Applikationsebene aus verrichten, spricht nichts dagegen, es so zu belassen.

Auf das Szenario f) aus Abbildung 9.19 wollen wir in einem weiteren Beispiel näher eingehen. Wir legen dazu eine neue Tabelle an, die Log-Tabelle. Wann immer ein Benutzer seinen Zugang löscht oder vom Administrator entfernt wird, halten wir das in der Protokolltabelle fest. Wir hinterlegen dort den Namen und das Geburtsdatum des alten Benutzers. Sinn und Zweck dieser Speicherung zusätzlicher Daten ist es, bei Neuanmeldungen nachvollziehen zu können, ob ein gesperrter bzw. gelöschter Benutzer versucht, erneut Zugriff zu dem System zu bekommen. Hier ist unsere Logtabelle:

```
CREATE TABLE benutzerLog (
zeitstempel TIMESTAMP NOT NULL DEFAULT NOW(),
name VARCHAR(100) NOT NULL,
geburtsdatum DATE)//
```

Die Definition der Attribute `name` und `geburtsdatum` müssen mit denen aus der `benutzer`-Tabelle übereinstimmen, zumal wir von dort nur die Werte kopieren. Das bedeutet unter anderem, dass das Geburtsdatum auch nullwertig sein darf. Zusätzlich führen wir einen Zeitstempel ein, der den Zeitpunkt des Einfügevorgangs mitschreibt.

Bei der gewählten Struktur der Tabelle `benutzer` ist es nicht sinnvoll, die Daten gelöschter Benutzer darin zu belassen. Zum einen bläht das die Tabelle auf, immerhin wollen wir darin einen Überblick über alle aktiven Anwender haben, zum anderen ist das Attribut `name` als Primärschlüssel gekennzeichnet. Ist Benutzer A mit dem Namen `sepp` gelöscht, soll es Benutzer B aber immer noch möglich sein, ein gleichnamiges Profil zu erstellen. Dies ist technisch jedoch nicht möglich, solange alte Werte in der Tabelle zu finden sind.

Der Trigger soll nun auf alle Löschoperationen in der Tabelle `benutzer` achten. Als Aktion führt er ein `INSERT` in der Tabelle `benutzerLog` aus.

```
CREATE TRIGGER protokolliereBenutzer
BEFORE DELETE
ON benutzer
FOR EACH ROW
INSERT INTO benutzerLog SET
   name=OLD.name, geburtsdatum=OLD.geburtsdatum//
```

Wir wählen mit Bedacht die Option `BEFORE DELETE`, weil dann bei einem Fehler in der Protokollierung der eigentliche Datensatz nicht gelöscht wird. Wir stufen das Logging somit als wichtig ein. Die Alternative `AFTER DELETE` geht das Risiko ein, dass Benutzerdaten gelöscht werden können, ohne dass sie protokolliert sind.

9.7.3 Löschen

Um einen Trigger wieder zu löschen, benutzen Sie

```
DROP TRIGGER [tblName].trigName
```

Die MySQL-Referenz gibt vor, dass der Tabellenname beim Löschen eines Triggers angegeben wird. Aus unseren praktischen Erfahrungen geht jedoch hervor, dass dies nicht immer zwingend notwendig ist. So lässt sich der obige Trigger mit

```
DROP TRIGGER protokolliereBenutzer//
```

entfernen. Dies geht einher mit der Tatsache, dass sich keine gleichnamigen Trigger für unterschiedliche Tabellen einer Datenbank erzeugen lassen, es also auch nicht zu Namenskonflikten kommen kann. Aus diesem Grund haben wir `tblName` – entgegen der Referenz! – als optional gekennzeichnet.

Der Einsatz von Triggern in der Praxis ist darüber hinaus nicht überall möglich. Natürlich ist die Unterstützung dafür in jeder Version ab MySQL 5.0 vorhanden. Allerdings kann es gerade dann zu Problemen kommen, wenn Sie die Datenbank bei einem Webhoster gemietet haben. Um einen Trigger anzulegen, benötigen Sie das *SUPER*-Recht, das Ihnen sicherlich nicht immer gewährt wird. Und so bleibt Ihnen in vielen Fällen nur übrig, die Applikation zu bemühen.

Index

$_COOKIE 118
$_ENV 120
$_FILES 115, 579
$_GET 114
$_POST 113
$_REQUEST 116
$_SERVER 120
$_SESSION 118
$GLOBALS 121, 494
<embed> 599
<object> 599
__clone() 148
__construct() 133
__desctruct() 134
Normalform
- *erste* 409
- *zweite* 409
- *dritte* 410
- *vierte* 411
- *fünfte* 412

A

Abbruchbedingung 106
absoluter Pfad 747
abstract 143
ACID 366
ACSII → American Standard Code for Information Interchange
ActiveX 397
ADDDATE() 199, 719
addslashes() 383, 691
ADDTIME() 719
AES_DECRYPT() 392, 719
AES_ENCRYPT() 392, 719
AFTER 352
Aggregation 310
ALTER 720
ALTER DATABASE 186
ALTER PROCEDURE 345
ALTER TABLE 186
ALTER VIEW 337
American Standard Code for Information Interchange 747
AMP-Systeme 38
Anfrageexpansion 325
Anführungsstriche 66
Anführungsstriche, doppelte 67
Apache 37
Apache Friends 38
Apache License 745
API → Application Programming Interface
Application Programming Interface 747
ARCHIVE 374
Archivtabellen 374
Array
- *assoziatives* 77, 170, 231, 256, 293
- *numerisches* 231, 293
- *superglobales* 167
- *zweidimensionales* 158

array() 78, 691
array_diff() 642, 691
array_key_exists() 82, 615, 655, 691
array_map 429
array_map() 692
array_pop() 83, 692
array_push() 83, 502, 624, 692
array_search() 81, 692
Arrays 65, 77
ASCII 747
asort() 80, 692
Attribute, statische 146
Authentifizierung 436, 747
AUTO_INCREMENT 174, 220
AUTOCOMMIT 367
Autorisierung 747
AVG() 312, 721

B

Backslash 262
basename() 466, 651, 692
B-Baum 320
bedingte Auswertung 308
BEFORE 352
BEGIN 366, 721
Benutzerkreis 747
Berkeley Software Distribution 744
BETWEEN 183
Binärdaten 747

binäre Texttypen 194
Binärtypen 195
Binnenmajuskel 86
Bitmuster 202
Blackbox 27, 91
Blackbox-Prinzip 747
Blogs 545
boolesche Suchanfragen 324
boolesche Werte 65
break 101
Brute-Force 387
BSD 744–746
Buffer-Overflow → Pufferüberlauf

C

Cache 747
CALL 344, 721
Carriage Return 67
Cascading Stylesheets → CSS
CASE 309, 348, 722
CAST 190
CAST() 723
CD-ROM zum Buch 39, 71
ceil() 73, 193, 693, 723
CHAR 194
checkdate() 248, 628, 639, 693
CHM 60
chr() 693
class 131
class_exists() 132, 693
classid 599
clone 148
Codevervollständigung 60, 89
Collation 163, 172, 186, 312, 321, 329, 361
COMMIT 367, 723
Compiled HTML Help → CHM
CONCAT() 195, 723
CONCAT_WS() 196, 723
const 133
constant() 635, 693
Cookie 117
Copyleft 741–742
COUNT() 311, 724
count() 508, 693
CREATE 724
CREATE DATABASE 172
CREATE INDEX 322
CREATE PROCEDURE 340
CREATE TABLE 173
CREATE TRIGGER 352
CREATE USER 303
CREATE VIEW 331
Creole 276, 605
Cronjob 33, 747
Cross-Site-Scripting 304, 395, 430
  clientseitiges 397
  serverseitiges 395
CSS 747
CSV-Datei 157
current() 82, 693

D

Data Source Name 285
date() 246, 507, 548, 694
DATE_FORMAT() 191, 197, 246, 726
Datenabstraktion 276
Datenbank-Administrations-Sprache 163
Datenbankentwurf 403
Datenbankmanagementsystem 276
Datenbankserver 24
Datenbanksystem, relationale 159
Daten-Definitions-Sprache 163, 172, 185, 305, 342
Daten-Manipulations-Sprache 163, 171, 175, 239, 305, 342
Datenmodelle 403
Datensicherheit 366
Datenstruktur 320
Datentypen 64, 173
DATETIME 197
DAYNAME() 199, 727
DAYOFMONTH() 199, 727
DAYOFWEEK() 199, 727
DAYOFYEAR() 199, 727
DBA 276
DBX 276–277
dbx_close() 279, 695
dbx_connect() 278, 695
dbx_error() 281, 695
dbx_escape_string() 281, 695
dbx_fetch_row() 281, 696
dbx_query() 279, 696
DECLARE 346, 728
DEFAULT 174
default 103

define() 87, 696
Defragmentierung 363
Dekrementierung 72
DELETE 185, 728
DELIMITER 343, 728
Denormalisierung 412
deprecated 417
DESCRIBE 215, 329, 728
Destruktoren 133
Deutsches Institut für Normung 747
die() 122, 562, 696
DIN → Deutsches Institut für Normung
DIN-Norm 747
DISTINCT 179, 312
do...while 105
DOUBLE 192
DROP 188, 728
DROP DATABASE 188
DROP PROCEDURE 345
DROP TABLE 188
DROP TRIGGER 357
DROP USER 305
DROP VIEW 337
DTD 159

E

echo() 61, 69, 696
Eclipse 60, 89
ELSE 309
elseif 99
Elternklasse 140
empty() 379, 696
enctype 578
END 309
End of File 254
end() 83, 697
Endlosschleife 748
ENGINE 359
Entität 160, 404
Entitätstyp 404
Entity-Relationship-Modell 403–404, 410
ENUM 201
Ergebnismenge 748
error_reporting() 123, 152, 697
EXPLAIN 215, 330, 729
explode() 269, 639, 643, 697
extends 141, 569
Extensible Markup Language 750
externe Dateien 124

F

false (falsch) 74
fclose() 253, 495, 697
Fehlerbehandlung 488
Fehlererkennung 59
feof() 254, 698
fetching 170, 227
fgetc() 253, 698
fgets() 253, 698
Fibonacci 111
file() 255, 698
file_exists() 258, 698
file_get_contents() 255, 698
file_put_contents() 255, 698
filesize() 584, 699
final 145
FIND_IN_SET() 202, 729
Flag 748
Fließkommazahlen 65
FLOAT 192
FLOOR() 193, 729
floor() 73, 699
fopen() 252, 495, 584, 699
for 106
foreach 107
FOREIGN KEY() 729
Forum 748
Fotoskalierung 582
FPDF 469
fread() 584, 699
Fremdschlüssel 161, 328, 366, 370, 409
FROM 178
FROM_UNIXTIME() 200, 730
FTP 748
FTP-Server 24
funktionale Abhängigkeit 409
Funktionen 90, 138
  Gültigkeitsbereiche 93
  Namenskonventionen 95
  Syntax 92
Funktionsrumpf 90
fwrite() 254, 495, 699

G

ganzzahlige Werte 65
Garbage Collector 444, 448
General Public License 741, 746
GET 750
get_class() 132, 700
get_class_methods() 136, 700
get_class_vars() 136, 700
get_defined_constants() 112, 700
get_defined_functions() 112, 700
get_defined_vars() 112, 700
get_magic_quotes_gpc() 700
getimagesize() 583, 701
Getter-Methoden 150
Getter-Setter-Methoden 136, 606
gettype() 66, 701
global 94
globale Gültigkeit 93
globale Variable 748
GLOBALS 121
GPL → General Public License
GRANT 171, 305, 730
gregorianischer Kalender 197, 748
GROUP BY 179, 315
Grundrechenarten 72
Gruppierung 313, 315

H

Hacker 35
Haftung 743
Hash 748
Hashing 384
header() 114, 401, 539, 559, 561, 701
HEAP-Engine 372
Höckerschreibweise 86
host 257
HTACCESS 158
htmlentities() 548, 701
HTML-Header 748
htmlspecialchars() 166, 431, 702
HTML-Tags
  input 663
  noscript 671
  option 672
  script 671
  select 672
  span 679
HTTP 748
  GET 29
  Header 30, 61, 384, 392
  POST 29
  Protokoll 29
  Statuscodes 30, 50
HTTP → HTTP-Protokoll
HTTP Response Splitting 377, 384, 401, 561
httpd.conf 49
HTTP-Protokoll 28, 377
Hypertext Transfer Protocol 748

I

IF 308, 347
if 96
IF EXISTS 188
IF() 732
IFNULL 308
IFNULL() 732
IFrame 748
if-then-else 97
IGNORE 176
imagecopyresampled() 584, 702
imagecreatefromjpeg() 583, 703
imagecreatetruecolor() 584, 703
imagedestroy() 585, 703
imagejpeg() 703
implements 145
implode() 255, 676, 703
IN 183
in_array() 81, 493, 641, 703
include() 124, 148
include_once() 704
Index 320, 361
INDIZES 320
Infizierung 742
INFORMATION_SCHEMA 305, 326
ini_set() 113
Initialisierung 133
Inkrementierung 72
InnoDB 364, 510, 650
INSERT 175, 732
INSERT...SELECT 183
INSERT...SET 176
INSERT...VALUES 176
instanceof 132
Instanz 85

Instanziierung 216
Integrität 497
interface 144
INTO OUTFILE 179
Intrusion Detection 479
intval() 530, 704
Inversion 262
IP-Adresse 748
IS 183
is_array() 65, 704
is_bool() 65, 704
is_dir() 258, 704
is_double() 65
is_file() 258, 704
is_finite() 704
is_float() 65, 705
is_int() 65, 379, 530, 705
is_integer() 65
is_null() 65, 705
is_numeric() 635, 705
is_object() 65, 705
is_readable() 259, 705
is_string() 65, 379, 705
is_writeable() 259, 705
isset() 62, 379, 706
ITERATE 350, 733
Iteration 103

J

JavaScript 23, 397, 585, 749
  alert() 681
  getElementById() 682
  onClick 672
  onKeyup 673
  split() 684
  unescape() 685
JOIN 178, 316
  LEFT 318
  NATURAL 318
  RIGHT 319
JPGraph 462

K

Kalender 748
Kapselung 91, 344
Kardinalität 405
kartesisches Produkt 178, 317
Kindklasse 140
Klassen 85, 131
  abstrakte 143
  automatisches Laden 148
  finale 145
Klassenbeziehungen 140
Klassendiagramm 130
Klassenhierarchie 142, 145
Klassenrumpf 131
Klonen 146
Kommentare 88
Konkatenation 68
Konsistenz 497
Konstanten 87
Konstruktoren 133
Kontrollkonstrukte 95
Kontrollstrukturen 347
ksort() 80, 706

L

Laufindex 106
LEAVE 349, 733
Lesser General Public License 743, 746
LGPL → Lesser General Public License
Library General Public License 743
LIKE 181, 183
LIMIT 179, 182, 184
Linefeed 67
Linux 743
Linux-Distribution 749
list() 583, 706
Logging 460
Logging-Tabelle 374
lokale Gültigkeit 93
LONGBLOB 195
LOOP 349, 733
LTRIM() 196, 733
ltrim() 69, 706

M

Magic Quotes 428
mail() 494, 706
MAKE_SET() 202, 733
Manipulation 749
Maskierung 67, 215, 247, 281, 286, 383, 398
MATCH 323

MATCH...AGAINST()   734
mathematische Funktionen   193–194
MAX()   312, 734
mcrypt_create_iv()   390, 707
mcrypt_decrypt()   391, 707
mcrypt_encrypt()   389, 707
mcrypt_get_iv_size()   390, 707
MD5   385
MD5()   386, 734
md5()   386, 707
md5_file()   386, 707
MD5-Algorithmus   338
MEDIUMTEXT   195
Mehrbenutzerbetrieb   368
Mehrbenutzersystem   497
mehrwertige Abhängigkeit   411
MEMORY   372
Metadaten   149, 243, 305, 325, 329
Methode   84
Methoden   138
  abstrakte   143
  finale   145
  statische   146
microtime()   248, 708
MIME → Multipurpose Internet Mail Extensions
mktime()   247, 474, 708
MIN()   312, 734
Min-Max-Notation   405
Modulo   72
Multipurpose Internet Mail Extensions   30, 749
Mustersuche   259
my.cnf   52
MyISAM   361
myisampack   364
MySQL
  Einführung   155
  Geschichte   15
  Lizenz   745
  my.cnf   52
MySQL AB   17
MySQL Administrator   203
MySQL Query Browser   204
MySQL-Datentypen   189
  Datums- und Zeittypen   196
  Mengentypen   201
  numerische Typgruppen   191
  Zeichenketten   194
MySQLI
  mysqli_result   227
MySQLi   28, 211
  __construct()   214
  affected_rows   220, 502
  auto_commit()   368
  change_user()   216
  character_set_name()   217
  client_encoding()   217
  client_info   219
  close()   214
  commit()   368
  connect()   214
  errno   223
  error   223
  Fehlerbehandlung   222
  host_info   219
  info   220
  init()   224
  insert_id   220
  kill()   221
  more_results()   226
  multi_query()   226
  mysqli_real_escape_string()   212
  mysqli_stmt   237
  next_result()   226
  options()   225
  query()   214
  real_connect()   224
  real_escape_string()   212, 215
  rollback()   369
  select_db()   216
  server_info   219
  sqlstate   223
  stdClass   234
  stmt_init()   238
  store_result()   226
  thread_id   220
  use_result()   226
  warning_count   223
mysqli   207
mysqli_affected_rows()   168, 708
mysqli_close()   169, 708
mysqli_connect()   167, 708
mysqli_error()   169, 708
mysqli_fetch_array()   709
mysqli_fetch_assoc()   170, 709
mysqli_init()   224
mysqli_num_rows()   168, 709

mysqli_query() 167, 709
mysqli_real_escape_string() 709
MySQLi_Result
data_seek() 235
fetch_array() 231
fetch_assoc() 233
fetch_field() 229
fetch_field_direct() 229
fetch_object() 234
fetch_row() 233
field_count 230
field_seek() 229
free_result() 235
num_rows 235
result_metadata() 242
MySQLi_Stmt
affected_rows 242
bind_param() 239
bind_result() 240
close() 238
errno 242
error 242
execute() 240
fetch() 240
num_rows 242
prepare() 238
MySQLi-Klassen 212
MySQLi-Result
fetch_fields() 229
num_fields 230
MySQL-Rechte 171, 303, 327
ALTER ROUTINE 340
CREATE 171
CREATE ROUTINE 340
CREATE USER 303
CREATE VIEW 337
DELETE 304
DROP 337
EXECUTE 340
GRANT OPTION 303
SHOW DATABASES 327
SHOW VIEW 337
SUPER 357
Übersicht 305
USAGE 305, 326
MySQL-Systemtabellen, KEY_COLUMN_USAGE 649
MySQL-Variablen 310

N

Namenskonventionen 86
new 132
Newsgroups 749
next() 83, 633, 710
nl2br() 536, 677, 710
Normalform 408
Normalisierung 408
NOT 183
Notepad 60
NOW() 190, 197, 734
NULL 174
Null 65
NULLIF 309

O

object cloning 146
Object Management Group → OMG
Objekte 65, 131
objektorientierte Fehlerbehandlung 149
Objektorientierung 129, 140
ODBC 277
OMG 129
Open Source 15, 37, 203, 741
Open Source Initiative 741
OpenOffice.org 744
OPTIMIZE 734
OPTIMIZE TABLE 363
Oracle 276
ORDER BY 179, 184, 315
OSI 741
Overhead 749

P

Parameterliste 92
parent 141
parse_url() 257, 710
Parser 26, 749
PASSWORD() 304
Passwort 455
Patent 742
path 257
pathinfo() 256, 710
PDF-Format 469
PDFlib 469

PDFMaker 471
PDO 283
__construct() 285
beginTransaction() 290
commit() 290
errorCode() 287
errorInfo() 287
exec() 286
getAttribute() 288
Prepared Statements 298
query() 285
quote() 286
rollback() 290
setAttribute() 288
PDOStatement
bindParam() 299
bindValue() 299
closeCursor() 297
columnCount() 297
execute() 300
fetch() 292
fetchAll() 295
fetchColumn() 296
fetchObject 297
rowCount() 297
setFetchMode() 294
PEAR → PHP Extension and Application Repository
PECL → PHP Extension Community Library
Performance 356, 359, 413, 749
Perl 261
PHP
Arrays 77
Datentypen 64
Destruktoren 133
Einführung 59
Funktionen 138
ganzzahlige Werte 71
Geschichte 15
grundlegende Syntax 62
Klassen 131
Konstruktoren 133
Lizenz (Licence 3.0) 745
Methoden 138
Objekte 84, 131
Objektorientierung 129
php.ini 53
Ressourcen 85
Strings 66
Variablen 62
PHP 5 15–17, 26, 131, 133, 138, 149, 211, 276, 283, 383, 454, 465, 469
PHP 5.1 15, 247, 276–277, 386
PHP 5.1.2 402
PHP Extension and Application Repository 43, 245, 273, 391
DB 276
MDB 276
MDB2 276
PHP Extension Community Library 245, 273, 275, 283
php.ini 53, 252, 284
allow_url_fopen 252, 386, 396
include_path 252
magic_quotes_gpc 383, 700
register_globals 112, 377, 454
PHP-Bereich 60
PHPdoc 88
PHPEdit 60
PHP-Editor 59
phpinfo() 120, 710
PHP-Module 27, 54
phpMyAdmin 47, 206, 307, 322
pi() 73
Pipe 262, 264
Platzhalter 239, 311
Port 749
port 257
POST 750
PostgreSQL 276
pow() 628, 710
POWER() 735
preg_grep() 268, 710
preg_match() 267, 459, 639, 711
preg_match_all() 267, 459, 637, 711
preg_replace() 269, 711
preg_split() 269, 712
Prepared Statements 211, 237, 291
prev() 82
Primärschlüssel 160, 437
PRIMARY KEY 173
print 61
print_r() 79, 712
Programmierung
objektorientierte 212
prozedurale 212
Propel 605

Prüfsumme 748
Puffer 749
Pufferüberlauf 747

R

Race Hazard 451
RAM → Random Access Memory
RAND() 237, 735
rand() 512, 712
Random Access Memory 750
rawurlencode() 712
realpath() 258, 712
Redhat 749
Referenz 93
referenzielle Integrität 161, 365, 369
REGEXP 735
REGEXP() 266
register_shutdown_function() 443
reguläre Ausdrücke 259, 267, 270, 380
Rekursion 109
rekursive Funktion 109
Relation 159, 404
Relationstypen 404
relativer Pfad 750
Release Candidate 750
Reload-Problematik 750
REPEAT 351, 735
require() 124, 148
require_once() 127, 713
reservierte Sonderzeichen 262
reset() 83, 632, 713
Ressourcen 65
REVOKE 307, 735
ROLLBACK 367, 736
ROUND() 193
round() 73, 713
ROW_FORMAT 364
rsort() 80, 713
RTRIM() 196, 736
rtrim() 69, 713
Rückgabewert 90
Rundung 73

S

Safe Mode 48, 55
scheme 257
Schleifen 103
Schnittstellen → Interfaces
SELECT 163, 177, 215, 332, 736
  SELECT-FROM-WHERE 178
  Vergleichsmöglichkeiten 182
serverseitige Skriptsprache 25
SESSION 118
Session Fixation 400
Session Hijacking 378, 399
session_destroy() 119, 447, 454, 713
session_id() 119, 713
session_name() 119, 713
session_regenerate_id() 400, 452, 714
session_save_path() 120, 714
session_set_save_handler() 443, 714
session_start() 118, 443, 714
Session-ID 117
SET 202, 736
SET PASSWORD 304
set_error_handler() 491, 714
setcookie() 117, 714
settype() 66, 715
SHA1 387
SHA1() 388, 737
sha1() 387, 715
sha1_file() 388, 715
SHOW 215, 326, 737
Sicherheit 377
Sicherheitskopie 203
Simple Mail Transfer Protocol 750
Sitzung 116
Sitzungsverwaltung 442
SMTP → Simple Mail Transfer Protocol
Sonderzeichen 66
sort() 80, 715
Sourceforge 39
Speicherung
  dynamische 364
  komprimierte 364
  statische 364
Sperren 368
Spooling 748
Sprach-Debug-Modus 523
SQL 16, 162
SQL Injection 398
SQL-Anfrage 748
SQLite 745
SQL-Kommandos 171
SQLSTATE 223, 287
SQRT() 737

SSL 392
Stack 83
StarOffice 744
START TRANSACTION 367, 737
static 146
Steuerzeichen 750
Stoppwortliste 323, 750
Storage Engines 162, 290, 320, 359
Stored Procedures 340
str_replace() 70
STRICT_ALL_TABLES 201
stripos() 633, 678, 715
stripslashes() 429, 715
strlen() 64, 640, 679, 716
strpos() 70
strripos() 643
strstr() 70, 716
strtolower() 633, 716
strtotime() 250, 448, 716
strtoupper() 716
SUBDATE() 199, 737
Subselect 185
substr() 70, 637, 679, 716
SUBSTRING() 196, 738
Suchfunktion 261
Suchmaschine 323
Suchmuster 260
SUM() 313, 737
SUSE 749
switch-Konstrukt 100
Syntax-Highlighting 59, 69, 89, 205

T

tblName 176
TDDSG → Teledienstdatenschutzgesetz
Teledienstdatenschutzgesetz 121
TEMPORARY 173
TEXT 195
time() 245, 484, 717
TIMESTAMP 197
TINYINT 191
Transaktion 342, 353, 364
transitive Abhängigkeit 410
Traversion 111
Trigger 352
trigger_error() 423, 717
TRIM() 196, 738
trim() 69, 717
true (wahr) 74
try-catch 150
Tupel 160, 750
Type Hint 138
Typkonvertierung 66, 190, 380

U

UltraEdit 60
UML → Unified Modeling Language
Unified Modeling Language 129
UNION 314, 738
UNIQUE 174
UNIX_TIMESTAMP() 200, 738
unscharfe Suchkriterien 259
unset() 62, 717
UNSIGNED 191
Unteranfrage 183
UPDATE 183, 738
Urheberrecht 742
USE 738
usleep() 512, 717

V

Validierung 750
VALUES 176
VARCHAR 194
Variablen 62, 346
Verarbeitungsprinzipien
FIFO 83
HIFO 83
LIFO 83
LOFO 83
Vererbung 140
Verschlüsselung 384
Ein-Weg- 385
Zwei-Wege- 388, 390
Verzeichnisfunktionen 256
Verzeichnisschutz 47
vi 60
Views 330
Virtueller Privater Server 34
Volltextindex 322

W

Wagenrücklauf 67
Wartung 91

Webbrowser 24
Webhosting 32
Webserver 24
WHERE 178
WHILE 351, 739
while 103
Whitelist 402
Wiederholungen 103
Wiederholungsgruppe 409
Wiederverwendung 91
Windows-Dienst 41
Workaround 750

X

XAMPP 38
Control Panel 43
Installation 39
Sicherheitslücken 46
XAMPP Lite 42
XML → Extensible Markup Language
XML Schema 159
XQuery 159

Z

Zeichenketten 65, 259, 262
Zeichensatz 361
Zeiger 82, 226, 301
Zeitstempel 197, 245, 750
Zend Engine 15
Zend Studio 60
ZEROFILL 191
Zugriffsmodifizierer
private 134
protected 134
public 134
Zuweisung 62
pass by reference 63, 94
pass by value 63, 93