Datenbank MySQL PHP

Christoph Reeg
Co-Autor:
Jens Hatlak

19. Juni 2005

Diese Anleitung, die sich gleichermaßen an Anfänger wie allgemein Interessierte in Sachen Datenbanken, SQL und PHP richtet, versucht schrittweise und spielerisch in die Geheimnisse der dynamischen Informationsverarbeitung im WWW einzuführen. Von der Datenbank-Theorie und -Praxis (am Beispiel MySQL) über die webfreundliche Scriptsprache PHP (inklusive Themen wie Datenbank-Anbindung und Objektorientierung) bis hin zu XML wird auf leicht verständliche Art beschrieben, welche Möglichkeiten moderne Technologien bieten, Webseiten dynamisch zu gestalten. Nicht fehlen dürfen dabei natürlich die zahlreichen Beispiele und und Hinweise; so gibt es u. a. auch ein Kapitel, das sich mit der Behandlung und Vermeidung von Fehlern beschäftigt.

Die aktuelle Version sowie verschiedene Formate zum Herunterladen befinden sich unter *http://reeg.net/*.

Inhaltsverzeichnis

1 Vorwort

> The difference between theory and practice is that in theory there is no difference between theory and practice but in practice there is.

Dieses Tutorial erklärt die Grundlagen einer Datenbank in Verbindung mit der Abfragesprache SQL und der Scriptsprache PHP. Dafür wird entlang eines Beispiels ein Datenmodell entwickelt und seine Umsetzung beschrieben. Anhand von einigen Beispielen wird dabei versucht, die Theorie etwas aufzulockern.

Die Kenntnis der Sprache HTML wird vorausgesetzt und in dem vorliegenden Tutorial werden im Prinzip keine HTML-Befehle beschrieben. Wer meint, noch Nachholbedarf zu haben, kann sich unter [7] noch ein Nachschlagewerk besorgen.

Dieses Tutorial wurde ursprünglich für Skripte geschrieben, die auf dem Server der Sites *http://ffm.junetz.de/* und *http://of.junetz.de/* Verwendung finden. Sie kann aber auch für alle anderen Server benutzt werden, die dieselben Grundkomponenten nutzen, wie in Abschnitt 2.2 beschrieben. Etwaige Server-spezifische Besonderheiten müssen dabei natürlich außer acht gelassen werden.

1.1 Aktuelle Version/Download

Diese Anleitung wird ständig aktualisiert und verbessert (wenn ich es schaffe, den inneren Schweinehund zu überwinden). Weiterhin befindet sich unter *http://reeg.net/* immer die aktuelle Version. Es gibt neben der Online-HTML Variante auch noch PostScript und PDF in verschiedenen Versionen zum herunterladen und ausdrucken.

Neue Versionen werden per E-Mail angekündigt, wenn du auch eine E-Mail bekommen willst, kannst du dich auf der Mailingliste[1] eintragen.

Wenn du Anregungen hast, was noch verbessert werden kann, würden wir uns über eine E-Mail an *dsp-autor@ml.junetz.de* sehr freuen.

1.2 Was ist das hier oder was ist es nicht?

Dieses Tutorial will die Grundlagen in Datenbankdesign, SQL und PHP vermitteln. Das Wissen sollte für normale und auch etwas aufwendigere Webseiten ausreichen. Wer allerdings Datenbanken mit Millionen von Einträgen oder Webpräsenzen mit mehreren hundert Seiten und tausenden von Abrufen pro Tag plant, sollte dies nur als Einführung sehen und sich Hilfe in Form eines Profis besorgen.

[1] *http://ml.junetz.de/list/listinfo/dsp-announce/*

Im SQL- und PHP-Teil werden die grundlegenden Befehle vorgestellt. Für eine ausführliche Befehlsübersicht müssen die Original-Anleitungen zu Rate gezogen werden.

1.3 Weitere Informationsquellen oder Hilfe

Wie oben schon gesagt, sollten zur Befehlsübersicht immer die Original-Anleitungen zu Rate gezogen werden. Die Anleitung von MySQL [4] gibt es inzwischen nicht nur in englisch, sondern auch in deutsch; diese ist sehr ausführlich und eignet sich auch als Einführung für SQL.

Bei PHP [6] sieht die Sache etwas anders aus. Die Anleitung gibt es in verschiedenen Sprachen, auch wenn nicht alle komplett aus dem Englischen übersetzt wurden. Sie eignet sich nur dann zum Erlernen von PHP, wenn man schon vorher programmieren konnte und nur die neue Syntax lernen muß. Sie eignet sich aber auf jeden Fall als gutes Nachschlagewerk. Bei der (englischen) Online-Variante gibt es die Möglichkeit, Kommentare zu den einzelnen Seiten zu hinterlassen, die später auch in die Dokumentation eingebaut werden. Die Kommentare sind häufig sehr wertvoll, besonders wenn die Dokumentation nicht ganz klar formuliert ist.

Eine weitere Informationsquelle sind die FAQs[2]. Die FAQ von MySQL [9] ist noch im Aufbau und deshalb noch nicht so ausführlich; sie enthält jedoch schon einige nützliche Informationen. Die PHP-FAQ [8] müßte eigentlich AFAPQ[3] heißen: Es gibt für fast jede erdenkliche Frage eine Antwort.

Neben der PHP-FAQ gibt es noch unzählige weitere Informationsquellen zu PHP (und häufig auch MySQL) im Internet. Gute Startpunkte dafür sind die PHP-Portale, wie z. B. [10] oder [11]. Nimm dir einfach mal einen Abend frei und surf los. Wenn du Glück hast, findest du das, was du selbst programmieren wolltest, schon irgendwo fertig oder zumindest Teile davon.

Trotz ausführlicher Anleitungen kann es einem passieren, daß man an der ein oder anderen Stelle nicht weiterkommt. Für solche Fälle gibt es Newsgroups und Mailinglisten im Internet, wo andere Leute in ihrer Freizeit versuchen, zu helfen. Die Mailingliste zu MySQL findet sich auf der MySQL-Homepage [4] unter „Documentation“; die deutschsprachige Newsgroup heißt ‚de.comp.datenbanken.mysql‘. Zu PHP gibt es gleich mehrere Mailinglisten in Englisch, die sich auf der PHP-Homepage [6] unter „support“ finden. Die deutsche Mailingliste wird vom PHP-Center [10] betrieben. Die Newsgroups zu PHP finden sich unter ‚de.comp.lang.php‘. Auch wenn man keine aktuellen Probleme hat, kann es sich lohnen, dort mitzulesen.

1.4 Unterteilung

Dieses Dokument ist in verschiedene Teile unterteilt, wobei die ersten drei auch einzeln für sich gelesen werden können.

[2] Frequently Asked Questions
[3] Answer For Any Possible Question

Erster Teil

Im ersten Abschnitt dieses Teils der Anleitung soll kurz geklärt werden, welche Software verwendet wird und welche Rolle sie spielt. Die Einzelheiten werden später im Tutorial behandelt.

Anschließend soll geklärt werden, was eine Datenbank ist, bzw. was mit Begriffen wie DB, DBS oder DBMS gemeint ist und wo die Unterschiede liegen.

Dieser Teil ist nicht existentiell, um nachher Datenbanken zu entwickeln; allerdings sollte man es sich schon einmal durchlesen. Und sei es nur, um einmal die Begriffe gehört zu haben ;-).

Wenn wir dann wissen, was eine DB ist, kommen wir zu der Frage, wie man aus einer Aufgabenstellung die passenden Datenstrukturen entwickelt. Dieser Teil ist für alle, die auch für den Entwurf der Datenstrukturen verantwortlich sind, besonders wichtig, weil man sich sonst viel Ärger einhandeln kann. Wer lediglich mit schon vorhanden Datenstrukturen arbeiten muß, kann diesen Teil überspringen, obwohl er eigentlich sehr interessant ist ;-).

Zweiter Teil

Nachdem wir uns dann im ersten Teil schon einige Zeit mit der Datenstruktur beschäftigt haben, wird es langsam Zeit, unsere Kenntnisse am Computer zu testen.

Als DBMS benutzen wir 'MySQL', das als Abfragesprache SQL[4] benutzt. SQL ist eine nach bestimmten Normen festgelegte Sprache, die von der Mehrheit der Relationalen DBS[5] benutzt wird; dazu zählen unter anderem: PostgreSQL, IBM DB2, Oracle, Adabas-D, MySQL, mSQL, Informix und Gupta.

Im Prinzip ist SQL standardisiert, allerdings unterstützen nicht alle Hersteller den kompletten Standard und jeder hat seine eigenen Erweiterungen. Soweit nicht explizit angegeben, sollten die in dieser Anleitung benutzten Befehle dem Standard entsprechen und auch mit anderen Datenbanken verwendet werden können.

All diejenigen, die mit Datenbanken arbeiten wollen, müssen diesen Teil vollständig verstanden haben - andernfalls kommen sie später mit Sicherheit stark ins Schleudern.

Du mußt übrigens nicht jeden Befehl auswendig kennen; viel wichtiger ist, daß du weißt, wo du nachschlagen kannst.

Dritter Teil

In diesem Teil wird die Scriptsprache PHP beschrieben. Mit ihr kann man sehr viel mehr machen als nur Datenbanken abzufragen, wobei der Schwerpunkt hier natürlich auf der Datenbankabfrage liegt. Wer mehr über diese Sprache lernen will, sollte sich einfach die Original-Dokumentation zu Gemüte führen.

Seit einiger Zeit gibt es PHP in der Version 4. Bis alle umgestellt haben, wird jedoch sicherlich noch einige Zeit lang PHP3 verbreitet sein. Im Prinzip sollten alle PHP3-Scripte auch unter PHP4 laufen. Soweit nicht anders angegeben, sind die Befehle, die in dieser Anleitung verwendet werden, sowohl unter PHP3 als auch unter PHP4 verwendbar. Wenn ein Befehl erst in PHP4 hinzu gekommen ist, wird dies explizit erwähnt (sofern ich es nicht vergessen habe ;-)).

[4] Structured Query Language
[5] Datenbanksysteme, mehr dazu im ersten Teil

Dieser Teil ist unbedingt notwendig, um in PHP programmieren zu können und sollte deshalb verstanden worden sein, was wie oben nicht heißen soll, daß du jeden Befehl auswendig kennen mußt, sondern nur, daß du im Zweifelsfall weißt, wo du nachschlagen mußt.

Vierter Teil

Hier werden ein paar Beispiele vorgestellt, in denen meines Erachtens Konstrukte benutzt werden, auf die man nicht immer auf Anhieb kommen würde. Hoffentlich helfen sie Dir. Sollte noch etwas fehlen: Meine E-Mail-Adresse kennst du ja. ;-)

Ein paar der Beispiele sind auch unter *http://reeg.net/* in Aktion zu sehen (wenn ich es endlich mal schaffe, sie zu programmieren :-( ).

1.5 Typographische Konventionen

Um die Übersichtlichkeit zu erhöhen, werden einige Sachen hervorgehoben. Die Tabelle 1.1 zeigt, auf welche Weise was hervorgehoben wird.

In den Beispielen wird der PHP-Code, HTML-Text, sowie teilweise der SQL-Code syntaktisch hervorgehoben. Laß dich davon nicht irritieren, sondern verstehe es einfach als Lesehilfe. Wie die jeweiligen Sprachelemente hervorgehoben werden, sollte eigentlich relativ einfach zu erkennen sein.

Wie	**Was**	**Beispiel**
kursiv	Internet-Adressen	Wenn du Anregungen hast, dann schreib mir doch eine E-Mail an *dsp-autor@ml.junetz.de*
in einfachen Anführungszeichen	Bezeichnungen, Namen	Wir benutzen hier ‚MySQL' und ‚PHP'.
in doppelten Anführungszeichen	Werte	Wenn die Variable Tag den Wert „13" und die Variable Monat den Wert „Mai" annimmt, hat ein wichtiger Mensch Geburtstag
nicht-proportionale Schriftart	Befehle bzw. Teile davon	Zum Mail-Lesen wird auf Unix-Maschinen der Befehl `rm` (read mail) benutzt. Es gibt noch die Optionen `-rf` (really fast), da geht das dann noch schneller. Um alle Mails zu lesen, muß man `rm *` eingeben (**auf eigene Gefahr!**).
fett	besonders wichtige Dinge, Hervorhebungen, Spalten-Bezeichnungen	Der Befehl `rm` bedeutet nicht „read mail", sondern **remove**. Und `-rf` steht für **recursive** und **force**.

Tabelle 1.1: Typogr. Konventionen

1.6 Autoren

Inzwischen schreibe ich zum Glück nicht mehr alleine an diesem Tutorial. Auf den jeweiligen Seiten steht in der Fußzeile, wer das Kapitel im Prinzip geschrieben hat[6].

Wenn du Anregungen zu dem Tutorial hast, kannst du gerne eine Mail an *dsp-autor@ml.junetz.de* schreiben; die wird dann automatisch an alle weitergeleitet und irgendjemand im Autorenteam[7] wird dann schon antworten.

Nun folgen ein paar Worte von den einzelnen Autoren (vielleicht werden es irgendwann ja auch noch mehr ;-)).

Christoph Reeg

Ich war derjenige, der mit dem Ganzen angefangen hatte. Eigentlich war es nur im Rahmen des Jugendnetz FFM/OF gedacht gewesen und nach der ersten Version[8] hatte ich auch keine große Lust mehr gehabt, weiter zu schreiben und hatte sie einfach auf meiner Homepage[9] liegen. Als ich dann aber in die Abrufstatistik von dem Server gesehen und festgestellt hatte, daß das Tutorial etliche Zugriffe verbuchen konnte, habe ich dann weiter geschrieben. In der Zwischenzeit habe ich dann auch noch etwas dazu gelernt, so daß auch ein paar Anfangsfehler verschwunden sind. So geht es nun weiter: in gelegentlichen Abständen ergreift mich der Ehrgeiz und ich setze mich wieder für ein paar Wochen dran und es entstehen ein paar neue Kapitel, und dann ruht es wieder.

Da ich bis jetzt leider noch nichts Vergleichbares im Internet gefunden habe, werde ich wohl auch noch die nächste Zeit damit beschäftigt sein und weiterschreiben. Auch die Mails, die ich gelegentlich bekomme, zeigen mir immer wieder, daß es einen gewissen Sinn hat, was ich da mache.

Jens Hatlak

Angefangen habe ich, wie Stefan Nagelschmitt, als Lektor. Anstatt jedoch einfach nur Christophs (zahlreiche ;-)) Rechtschreibfehler zu korrigieren, habe ich dann mit der Zeit immer mehr auch Neues hinzugefügt. Das ging dann irgendwann so weit, daß erst ein paar Abschnitte und schließlich ganze Kapitel (wie z. B. OOP) von mir geschrieben wurden. So langsam fällt mir aber gar nichts mehr ein, was noch dringend beschrieben werden müßte – aber es steht ja noch die Umstellung auf neue Rechtschreibung an, also werde ich zumindest als Lektor wohl immer was zu tun haben. ;-)

Tilman Brock

Ich bin einer von denen, die zusammen mit Christoph beim Jugendnetz Frankfurt arbeiten. Über das Jugendnetz und meine Programmierarbeit mit PHP habe ich das DSP schätzen gelernt; jetzt fühle ich mich in der Lage, auch meinen Teil zu dieser PHP/MySQL-Anleitung zu schreiben. Ich bin auch der Erste, der das DSP extensiv in die Schulen ge-

[6] Ich lasse es mir natürlich nicht nehmen, überall noch etwas zu korrigieren. Im Gegenzug ist Jens immer noch so freundlich und korrigiert meine Sachen ;-)
[7] Team = Toll, ein anderer macht's!
[8] die war irgendwann 1998
[9] die eigentlich gar nicht so toll war

bracht hat, meine komplette Schule[10] – soweit sie mit PHP/MySQL arbeitet – benutzt als erste Dokumentation *http://www.reeg.net*.

David Peter

Ich bin im Herbst 2001 per Zufall auf DSP gestoßen und hab kurz danach Christoph gefragt, ob ich helfen könnte, indem ich ein paar Artikel für DSP schreibe. Er hatte nichts dagegen und so konnte ich kurz darauf das erste Kapitel (über XML) für DSP bereitstellen. Danach war eine Weile lang nichts los, da weder ich noch die anderen Autoren viel an DSP gearbeitet haben und so wurde ich erst im Frühling 2002 wieder – durch eine Mail von Christoph – an DSP erinnert.

[10] die Ernst-Reuter Schule I in Frankfurt

Teil I

Theoretische Grundlagen

2 Benötigte Software

2.1 Apache: Der Webserver

Fangen wir als erstes mit der einfachsten Form von Webseiten an: den statischen Webseiten. Sie werden mit Hilfe eines ASCII-/Homepage-Editors erstellt und liegen dann als .html auf der Festplatte. Von dort kannst du sie dir dann mit jedem beliebigen Browser abrufen und anzeigen lassen. Damit jetzt jeder beliebige Surfer im Internet sich die Seiten ansehen kann, muß auch er irgendwie darauf zugreifen können. Dafür ist der Webserver[1] (z. B. Apache) da. Er erlaubt den Zugriff aus dem Netz auf die lokalen Dateien. Normalerweise lädt man seine Homepage aber auf den Server seines Providers und überläßt den dortigen Webservern die Bereitstellung der Webseiten.

Wenn es so einfach ist, eine Webseite zu erstellen, wozu braucht man dann noch Datenbanken, PHP und SQL? Weil es einige Bereiche gibt, wo statische Webseiten nicht ausreichen. Zum Beispiel bei Suchmaschinen: Die Ergebnisseite hängt von den Suchwörtern ab, die du eingegeben hast und es ist unmöglich für alle möglichen Suchwörter und deren Kombinationen statische Ergebnisseiten zu erstellen. Also muß es auf dem Webserver ein Programm geben, was die Benutzereingaben auswertet und dementsprechende Webseiten erstellt. Der Besucher der Webseite bekommt nichts davon mit, ob die Webseite statisch oder dynamisch ist: er bekommt immer HTML von dem Webserver zurück.

Der Webserver muß dafür sorgen, daß bei dynamischen Webseiten das entsprechende Programm ausgeführt wird und dessen Ausgabe an den Besucher zurück gegeben wird. Eine mögliche Programmiersprache für solche Programme ist PHP.

2.2 PHP - ist das gefährlich?

In den USA stand tatsächlich einmal ein Schüler unter dem schweren Verdacht, die Droge PHP konsumiert zu haben. Den lokalen Behörden war offensichtlich nicht klar, in welches Fettnäpfchen sie sich mit dieser Anklage gesetzt hatten, denn PHP ist kein Rauschmittel[2], sondern eine Programmiersprache, die für die Erfordernisse des modernen WWW – dynamische Webseiten – konzipiert wurde und sich großer Beliebtheit erfreut. Entstanden ist sie unter dem Einfluß so bekannter Programmiersprachen wie C, Java und Perl, aber auch neue und in dieser Form besondere Eigenheiten wurden hinzugefügt. Wie viele andere Programmiersprachen (z. B. ASP, ColdFusion; JSP) wird PHP in normale, statische HTML-Seiten eingebettet und bietet somit für den Programmierer einige Vorteile; auch gegenüber den Hochsprachen wie C oder Java finden sich dazu Beispiel wie der bewußte Verzicht auf strenge Typisierung – aber das würde hier zu weit führen, weshalb ich hier nur auf Kapitel 8 verweise.

[1] An dieser Stelle meine ich die Software, nicht den Rechner
[2] Oder? ;-)

2.3 MySQL

Häufig wird mit Hilfe von PHP auch Daten verarbeitet. Zum Beispiel bei einem Gästebuch werden die Einträge gespeichert. Die einfachste Variante wäre, das über Dateien zu lösen, allerdings muß sich dann der Programmierer um einige Sachen, wie zum Beispiel File-Locking[3], effizient Daten löschen, effiziente Suche, usw. selbst kümmern. Viel einfacher ist es, diese Aufgaben einem Spezialisten zu überlassen: Dem Datenbank Management System (kurz. DBMS). Im Web-Bereich ist MySQL sehr verbreitet.

2.4 LAMP / WAMP

Das Trio Apache, MySQL und PHP ist im Web-Bereich stark vertreten. Aus diesem Grund hat sich dafür auch eine Abkürzung etabliert: „LAMP“ bzw. „WAMP“. Das „L“ bzw. „W“ steht für das verwendete Betriebsystem. Im Normalfall ist das Betriebsystem für die Nutzung von PHP und MySQL unbedeutend.

Die Installation von MySQL, Apache und PHP wird in diesem Tutorial nicht beschrieben. Zu diesen Themen gibt es neben den Beschreibungen in der Dokumentation auch etliche Webseiten, die das Thema ausführlich behandeln. Zumindest für die Windows-Plattform gibt es aber ein fertiges Paket, das einen kompletten Webserver mit Unterstützung für PHP/MySQL und sogar SSL bereitstellt.[12]

[3] z. B. Was passiert wenn zwei Besucher zur genau selben Zeit die Seite aufrufen?

3 Datenbanksystem

3.1 Komponenten eines Datenbanksystems

Eine Datenbank (DB, engl. Data Base) ist eine systematische Sammlung von Daten. Zur Nutzung und Verwaltung der in der DB gespeicherten Daten benötigt der Anwender ein Datenbank-Verwaltungssystem (DBMS, engl. Data Base Management System). Die Kombination aus DB und DBMS ist das Datenbanksystem (DBS, engl.: Data Base System), das jedoch häufig fälschlicherweise als Datenbank bezeichnet wird.

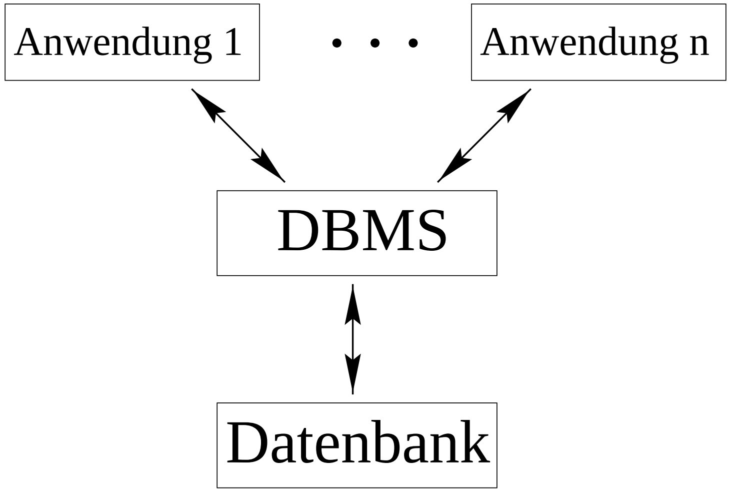

Abbildung 3.1: Struktur eines Datenbanksystems

Das DBMS besteht aus einer Vielzahl von Werkzeugen und Generatoren („Erzeugern“). Auf der einen Seite stellt es dem Entwickler die Instrumente zu Verfügung, mit denen er das Datenmodell beschreiben und einrichten kann. Auf der anderen Seite bietet es die Funktionen an, mit denen die einzelnen Anwender Daten eingeben, verändern, abfragen und ausgeben können.

Alle Funktionen des DBMS werden durch „was“ und nicht mehr „wie“ spezifiziert; soll heißen: Der Entwickler teilt dem Programm die Datenlogik mit und der Anwender formuliert seine Abfrage. Wie die Daten zu speichern und zu verwalten sind, ist Sache des

DBMS. Dieses ist also zuständig für die technische Umsetzung der Anforderungen des Entwicklers und der Anwender.

3.2 Ebenen eines Datenbanksystems

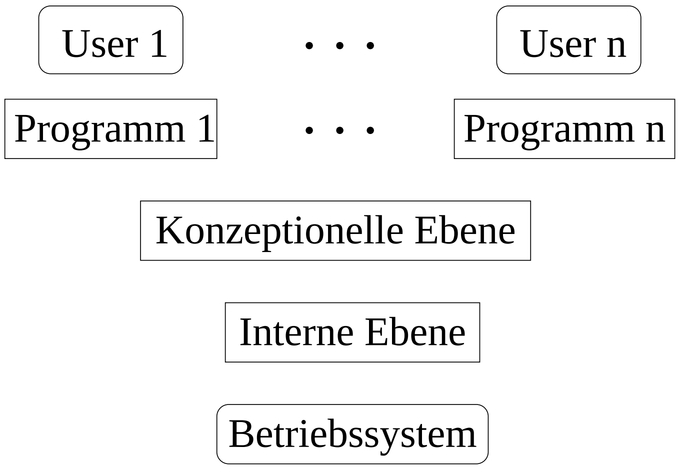

Abbildung 3.2: Die vier Ebenen eines DBS

Ein Datenbanksystem (DBS, engl.: Data Base System, = DB+DBMS) besteht aus den vier Ebenen:

3.2.1 Betriebssystem/Hardware

Dies ist die unterste Ebene, auf der jede Computeranwendung basiert. Neben dem DBS bauen auch alle anderen Programme auf dieser Ebene auf. Man kann diese Ebene aber noch weiter unterteilen: Zum einen ist da die Hardware als absolut unterste Ebene, deren Möglichkeiten vom Betriebssystem (BS) verwaltet werden. Das Betriebssystem zum anderen bietet Programmen die Hardwaremöglichkeiten an, ohne daß die Programme die Hardware direkt ansprechen müßten.

3.2.2 Interne Ebene

Auf der internen Ebene erfolgt die physische Speicherung der Daten. Die Speicherlogik, die dabei verwendet wird, hängt vom DBMS ab und kann dem Entwickler ziemlich egal sein, da er lediglich über die konzeptionelle Ebene auf die DB zugreift. Den Anwender

braucht weder die interne noch die konzeptionelle Ebene zu kümmern, da er erst über die oberste, nämlich die externe Ebene, auf die DB zugreift.

3.2.3 Konzeptionelle Ebene

Auf der dritten, der konzeptionellen Ebene, wird das Datenmodell beschrieben. Unter einem Datenmodell versteht man die datenmäßige Abbildung eines bestimmten Ausschnitts der realen Umwelt. Im Datenmodell sind die Strukturen der Daten und ihre Beziehung zueinander festgelegt. Nach der Art, wie die Beziehungen in dem Datenmodell geregelt werden, unterscheidet man zwischen hierarchischen, vernetzten, objektorientierten, objektrelationalen und relationalen Datenmodellen. Wir verwenden im Folgenden lediglich das relationale Datenmodell, da es (noch) die größte Verbreitung besitzt.

3.2.3.1 Tabellenstruktur

Beim relationalen Datenmodell werden die Daten in zweidimensionalen Tabellen angeordnet.

Jede Tabelle hat einen eindeutigen Relationsnamen. Alle Zeilen der Tabelle (ohne die Spaltenüberschriftszeile) werden als Relation, jede einzelne Zeile davon als Tupel bzw. Datensatz, die Spaltenüberschriften als Attributnamen oder Attribute und alle Attributnamen zusammen werden als Relationsschema bezeichnet.

Allgemein wird in jeder Zeile eine Entität abgebildet.

In Abbildung 3.3 wurde versucht, die Zusammenhänge grafisch darzustellen.

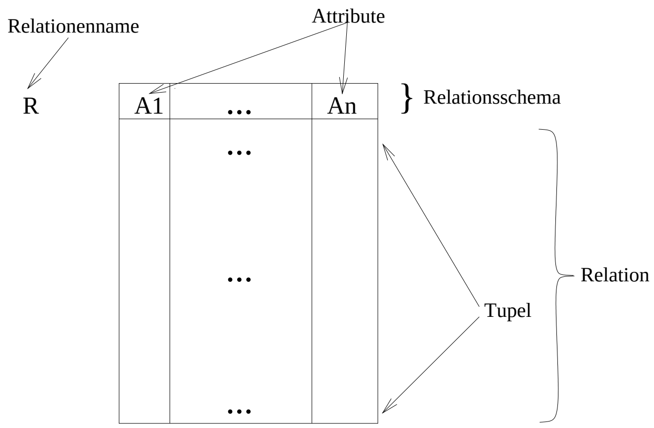

Abbildung 3.3: Tabellenstruktur

Um das Ganze etwas konkreter zu machen, habe ich in Tabelle 3.1 ein kleines Beispiel dargestellt.

Mitarbeiter				
MNr	**AbtNr**	**Name**	**GebDat**	**Telefon**
1	3	Christoph Reeg	13.5.1979	NULL
2	1	junetz.de	5.3.1998	069/764758
3	1	Uli	NULL	NULL
4	1	JCP	NULL	069/764758
5	2	Maier	NULL	06196/671797
6	2	Meier	NULL	069/97640232

Tabelle 3.1: Beispiel für Tabellenstruktur

Das Beispiel zeigt die Relation mit dem Namen 'Mitarbeiter'. Jeder Mitarbeiter hat die Attribute 'MNr', 'Name', 'GebDat' und 'Telefon'. In der Relation stehen 6 Datensätze bzw. Tupel.

3.2.3.2 Schlüssel

Damit man jede Zeile gezielt ansprechen kann, wird ein Schlüsselattribut eingeführt. Der Schlüssel muß immer eindeutig sein und wird auch als Primärschlüssel bezeichnet. Der Primärschlüssel muß nicht immer aus nur einem Attribut bestehen. Es ist auch möglich, mehrere Attribute zusammen als (zusammengesetzten) Primärschlüssel zu verwenden. Teilweise hat man in einer Relation mehrere Attribute, die eindeutig sind, d. h. Schlüssel sein könnten; in diesem Fall werden die anderen Attribute als Schlüsselkandidaten bezeichnet. Oder anders herum: Jeder Schlüsselkandidat kann jederzeit als Primärschlüssel benutzt werden. Es kann aber für eine Tabelle immer nur einen Primärschlüssel gleichzeitig geben.

Zum Einrichten der DB mit ihren Tabellen bedient man sich der Data Definition Language (DDL).

3.2.4 Externe Ebene

Auf der obersten Ebene befindet sich der Anwender, der auf das DBS mit einer Daten-Abfragesprache (DQL, engl.: Data Query Language), einer Daten-Manipulationssprache (DML, engl.: Data Manipulation Language) oder einer eigenen Anwendung, welche in unserem Beispiel die WWW-Seite ist, zugreift.

4 Datenbanken entwickeln

Umwege erhöhen die Ortskenntnisse

unbekannt

4.1 Vorgehensweise

Die Entwicklung einer DB vollzieht sich in mehreren Schritten. Zunächst ist festzustellen, welche Informationen die Anwender vom DBS erwarten, bzw. welche Informationen gespeichert werden sollen. Aufgrund dieser Erhebung kann man sich dann überlegen, welche Tabellen benötigt werden. Ferner muß festgelegt werden, welche Datentypen für die einzelnen Tabellenspalten benötigt werden. Diesen Prozeß bezeichnet man als Datenmodellierung. Erst wenn die Datenmodellierung abgeschlossen ist, können die Tabellen angelegt werden.

Man sollte sich für diesen Schritt ruhig ein wenig Zeit nehmen, weil es nachher häufig unmöglich ist, ohne großen Aufwand Fehler zu beheben.

4.2 Grundsätze

Um sich einigen Ärger zu ersparen, empfiehlt es sich, ein paar Grundsätze bei der Datenmodellierung zu beachten:

4.2.1 Keine Redundanz

Unter Redundanz versteht man das doppelte Vorhandensein einzelner Daten. Am folgenden Beispiel wird dies besonders deutlich:
In folgender Tabelle werden Adressen gespeichert:

Vor-/Nachname	**Vorname**	**Straße**
Hans Maier	Hans	Musterstr. 5
Jürgen Müller	Jürgen	In dem Muster 4
Christof Meier	Christof	Gibt es nicht 1

Wie man leicht erkennen kann, kommt der jeweilige Vorname in zwei Spalten vor. Dies bringt zwei Nachteile mit sich: Zum einen kostet es mehr Speicherplatz, was bei einigen 1000 Datensätzen schon etwas ausmacht; zum anderen werden Änderungen schwieriger, anfälliger für Fehler und auch aufwendiger, da ja zwei Attribute geändert werden müssen. Wenn dies nicht erfolgt, treten Inkonsistenzen auf.

Wenn zum Beispiel *Christof Meier* feststellt, daß ein *Christoph*, mit 'f' geschrieben, einfach nicht so gut aussieht und er es gerne in *Christoph* geändert haben würde, dabei aber nur das Attribut **Vorname** geändert wird, könnten zum Beispiel die Briefe weiter an *Christof Meier* geschickt werden, weil hier das Attribut **Vor-/Nachname** verwendet wird. An einer anderen Stelle im Programm würde aber wieder der korrigierte *Christoph* auftauchen.

4.2.2 Eindeutigkeit

Eine DB enthält Angaben zu den Eigenschaften einer Person oder Sache. Mittels dieser Angaben muß eindeutig ein bestimmtes Tupel identifizierbar sein.

Das DBMS verfügt nicht über einen definierten Zugriffsweg auf einen bestimmten Datensatz. Deshalb muß in jeder Zeile einer Tabelle ein Wert enthalten sein, der diesen Eintrag eindeutig kennzeichnet bzw. identifiziert. Um die Eindeutigkeit der Tabellenzeilen zu gewährleisten, erweitert man den Datensatz um ein Identifikationsmerkmal, z. B. wird einem Artikeldatensatz eine Artikelnummer zugeordnet. Dieses Merkmal nennt man Schlüssel.

Beim Festlegen des Schlüssels kann man einen Schlüssel selbst definieren oder einen fremddefinierten übernehmen. Bei einem Buch würde sich da die ISBN-Nummer anbieten. Um nicht Gefahr zu laufen, daß durch eine Änderung solcher fremddefinierten Schlüssel im DBS Inkonsistenzen auftreten, zum Beispiel, weil der Schlüssel nicht mehr eindeutig ist, empfiehlt es sich häufig, einen eigenen zu nehmen.

4.2.3 Keine Prozeßdaten

Prozeßdaten sind Daten, die durch einen Rechenprozeß aus gespeicherten Attributen gewonnen werden. Folgendes einfaches Beispiel: Neben dem Geburtsdatum wird auch noch das Alter gespeichert. Spätestens nach einem Jahr ist dieser Eintrag falsch. Deshalb sollten diese Prozeßdaten bei jeder Abfrage neu errechnet werden.

4.3 Datenmodelle entwickeln

Es gibt mehrere Vorgehensweisen. Eine Möglichkeit ist, erst einmal darüber nachzudenken, was man eigentlich machen will, dann die entsprechenden Prozeduren zu entwickeln und dabei zu sehen, welche Art von Daten man braucht. Diese Vorgehensweise kennen diejenigen, die schon einmal programmiert haben.

Andererseits kann man sich auch zuerst überlegen, welche Daten überhaupt anfallen und wie diese am besten organisiert werden. Anschließend kann man sich dazu die entsprechenden Funktionen ausdenken. Da Datenbanken in der Regel zum Speichern von Daten gedacht sind, empfiehlt sich letztere Vorgehensweise; man sollte aber trotzdem die benötigten Funktionen nicht aus dem Auge verlieren. Also zusammenfassend:
Als erstes muß man feststellen, welche Daten gebraucht werden bzw. anfallen und wie diese organisiert werden sollen. Im nächsten Schritt ist zu überlegen, ob alle Anforderungen realisierbar sind.

4.3.1 Tabellen erstellen

Wie im letzten Kapitel dargestellt, werden bei Relationalen Datenbanksystemen die Daten in Tabellen gespeichert. Demzufolge ist der letzte Schritt bei der Datenmodellierung das

Festlegen, wie die Tabellen aussehen sollen.

Um die benötigten Tabellen zu entwickeln, gibt es für einfache DBs im Prinzip zwei Möglichkeiten: Entweder stur nach Schema-F über die fünf Normalformen (Kapitel 4.4) oder etwas intuitiver über das ER-Modell (Kapitel 4.6), evtl. anschließend mit Kontrolle durch die fünf Normalformen (Kapitel 4.4).

Erst wenn man größere DBs entwickelt, muß man mit beiden Möglichkeiten gleichzeitig arbeiten. Das heißt, erst mit dem ER-Modell eine Grundstruktur festlegen und diese dann mit den fünf Normalformen überprüfen.

4.4 Die fünf Normalformen

Die Normalformen sind ein Regelwerk, mit deren Hilfe man überprüfen kann, ob sich ein Datenmodell sauber mit Hilfe eines Relationalen DBMS implementieren läßt. Man spricht auch von Normalisierung.

4.4.1 Die 1. Normalform

Definition:

> Ein Relationstyp ist in der **1. Normalform**, wenn **alle Attribute maximal einen Wert** haben. Am Kreuzungspunkt einer Spalte mit einer Reihe darf also maximal ein Datenwert stehen. Das Nichtvorhandensein von Daten ist zulässig.
>
> Mit anderen Worten: Wiederholungsgruppen sind nicht erlaubt. [3]

Ein kleines Beispiel:
Es sollen alle Bestellformulare eines Versandhandels in einer Datenbank gespeichert werden. Eine einzelne Bestellung enthält die Kundennummer, das Datum, die Auftragsnummer und natürlich die bestellten Artikel sowie deren Anzahl (weitere evtl. notwendige Werte werden der Einfachheit einfach mal unterschlagen). Siehe dazu auch folgende Tabelle 'Auftrag'.

Auftrag				
AuftragNr	**Datum**	**KundenNr**	**ArtikelNr**	**Anzahl**
4711	03.10.1999	12345	4692	5
			0567	2
			5671	3
			0579	1
0815	01.03.1998	54321	8971	2
			5324	5
			0579	9

Um die Wiederholungsgruppen zu vermeiden, wird die Relation 'Auftrag' in zwei gesonderte Relationen aufgespalten. Dadurch würden sich die folgenden beiden Tabellen ergeben:

best. Artikel	
ArtikelNr	**Anzahl**
4692	5
0567	2
5671	3
0579	1
8971	2
5324	5
0579	9

Auftrag		
AuftragNr	**Datum**	**KundenNr**
4711	3.10.1999	12345
0815	1.3.1998	54321

Jetzt ist aber die Zuordnung verloren gegangen. Wer hat welche(n) Artikel bestellt?

Dieses Problem ist einfach zu lösen: Wir müssen nur festhalten, welche Artikel zu welcher Bestellung gehören. Da die AuftragNr eindeutig ist, nehmen wir diese als Primärschlüssel[1] für 'Auftrag'. Nun fügen wir noch dieser Spalte entsprechend ihrer Werte der Relation 'best. Artikel' hinzu, und schon haben wir wieder unsere Zuordnung.

In dieser Konstellation wird die Spalte 'AuftragNr' in 'best. Artikel' als Fremdschlüssel bezeichnet.

Weiterhin wurde schon auf Seite 15 gefordert, daß jede Zeile eindeutig ansprechbar sein muß. Wie aber ist das in unserem Fall der bestellten Artikel zu erreichen?

Nun, die AuftragNr und die ArtikelNr kommen zwar mehrfach vor, trotzdem ist die Lösung aber ganz einfach: Die Kombination aus AuftragNr und ArtikelNr muß eindeutig sein. Wenn wir also diese Kombination wählen, ist die o. g. Forderung erfüllt. Diese Kombination wird übrigens als ‚zusammengesetzter Primärschlüssel[2]' bezeichnet.

Damit ergeben sich für unser Beispiel die folgenden beiden Relationen:

best. Artikel		
# AufragNr	**# ArtikelNr**	**Anzahl**
4711	4692	5
4711	0567	2
4711	5671	3
4711	0579	1
0815	8971	2
0815	5324	5
0815	0579	9

Auftrag		
# AuftragNr	**Datum**	**KundenNr**
4711	3.10.1999	12345
0815	1.3.1998	54321

[1] Primärschlüssel werden im folgenden durch eine führende Raute (#) gekennzeichnet
[2] bei zusammengesetzten Primärschlüsseln wird im Folgenden jeder Teil mit einer führenden Raute (#) gekennzeichnet

4.4.2 Die 2. Normalform

Definition:

> Ein Relationstyp ist in der **2. Normalform**, wenn er in der 1. Normalform ist und **jedes Attribut vom gesamten Primärschlüssel abhängt**.
>
> Relationstypen, die in der 1. Normalform sind, sind automatisch in der 2. Normalform, wenn ihr Primärschlüssel nicht zusammengesetzt ist. [3]

Ein kleines Beispiel:
Neben der AuftragNr, der ArtikelNr und der Menge soll auch der Hersteller des Artikels gespeichert werden. Damit würde sich die folgende Artikel-Tabelle ergeben.
AuftragNr und ArtikelNr sind der zusammengesetzte Primärschlüssel.

best. Artikel			
# AuftragNr	**# ArtikelNr**	**Menge**	**Hersteller**
4711	4692	5	Blech-AG
4711	0567	2	Keramik GmbH
4711	5671	3	Baustoff KG
4711	0579	1	Keramik GmbH
0815	8971	2	Keramik GmbH
0815	5324	5	Baustoff KG
0815	0579	9	Keramik GmbH

In diesem Beispiel ist das Attribut 'Hersteller' nur vom Teilschlüssel 'ArtikelNr' und nicht auch von 'AuftragNr' abhängig. Damit die Relation der 2. NF genügt, muß das Attribut 'Hersteller' aus der Relation herausgenommen und der (neuen) Relation Artikel zugeordnet werden.
Daraus würden dann die folgenden zwei Relationen entstehen:

best. Artikel		
# AufragNr	**# ArtikelNr**	**Anzahl**
4711	4692	5
4711	0567	2
4711	5671	3
4711	0579	1
0815	8971	2
0815	5324	5
0815	0579	9

Artikel	
# ArtikelNr	**Hersteller**
4692	Blech-AG
0537	Keramik GmbH
5671	Baustoff KG
0579	Keramik GmbH
8971	Keramik GmbH
5324	Keramik GmbH

4.4.3 Die 3. Normalform

Definition:

> Die **3. Normalform** ist erfüllt, wenn die 2. Normalform erfüllt ist und die **Nicht-Schlüssel-Attribute funktional unabhängig voneinander** sind.

Sind A und B Attribute eines Relationstyps, so ist B funktional abhängig von A, wenn für jedes Vorkommen ein und desselben Wertes von A immer derselbe Wert von B auftreten muß.

Eine funktionale Abhängigkeit kann auch von einer Gruppe von Attributen bestehen. [3]

Ein kleines Beispiel:
Zu den einzelnen Artikeln sollen die ArtikelNr, die Bezeichnung, der Hersteller und die HerstellerNr gespeichert werden. Als Primärschlüssel wird die ArtikelNr verwendet. Würde man die zusätzliche Spalte einfach in die vorhandene Tabelle Artikel einfügen, ergäbe sich damit folgende Tabelle:

Artikel			
# ArtikelNr	**Bezeichnung**	**HerstellerNr**	**Hersteller**
4692	Putzeimer	5410	Blech-AG
0567	Waschbecken	5647	Keramik GmbH
5671	Gummi	6740	Baustoff KG
0579	Teller	5647	Keramik GmbH
8971	Tasse	5647	Keramik GmbH
5324	Badewanne	5647	Keramik GmbH

Wie man unschwer erkennen kann, ist der Herstellername von der ArtikelNr über die HerstellerNr transitiv abhängig. Oder anders ausgedrückt: Der Herstellername ist funktional abhängig von der HerstellerNr, die wiederum abhängig von der ArtikelNr ist. Und diese funktionale Abhängigkeit der beiden Nicht-Schlüssel-Attribute HerstellerNr und Hersteller ist nicht erlaubt.

Was jetzt kommt, ist nicht schwer zu erraten: Die Tabelle 'Artikel' wird in die beiden Tabellen 'Artikel' und 'Hersteller' aufgespalten. Das heißt, es ergeben sich folgende Tabellen:

Artikel		
# ArtikelNr	**Bezeichnung**	**HerstellerNr**
4692	Putzeimer	5410
0567	Waschbecken	5647
5671	Gummi	6740
0579	Teller	5647
8971	Tasse	5647
5324	Badewanne	5647

Hersteller	
# HerstellerNr	**Hersteller**
5410	Blech-AG
5647	Keramik GmbH
6740	Baustoff KG

4.4.4 Die 4. Normalform

Definition:

Die **4. Normalform** ist erfüllt, wenn die 3. Normalform erfüllt ist und wenn **keine paarweise auftretenden mehrwertigen Abhängigkeiten** vorhanden sind.

Sind A, B und C Attribute eines Relationstypes, so ist C mehrwertig abhängig von A, falls für jeden Wert von A für alle Werte von B, die zusammen mit diesem Wert von A auftreten, jeweils die gleiche Wertemenge von C auftreten muß. Für verschiedene Werte von A können unterschiedliche Wertemengen von C auftreten.

Bei Verstoß gegen die 4. Normalform können „Gruppeninkonsistenzen" auftreten. [3]

Kurzes Beispiel:

Disposition		
ArtikelNr	**Lager**	**AuftragNr**
04532	SFO-4	2063
04532	NYC-4	2063
04532	SFO-4	2267
04532	NYC-4	2267
53944	ORD-1	2088
53944	SFO-1	2088
53944	LAX-1	2088
53944	ORD-1	2070
53944	SFO-1	2070
53944	LAX-1	2070

In der Relation *Disposition* sind folgende Informationen festgehalten:

- der Lagerort für jeden Artikel
- Aufträge, in denen ein Artikel vorkommt

Es soll *nicht* ausgesagt werden, aus welchem Lager der Artikel für einen Auftrag kommt. Folgende mehrwertige Abhängigkeiten liegen vor:

- 'Lager' ist mehrwertig abhängig von 'ArtikelNr :

  Für jeden Artikel muß für alle Aufträge, für die der Artikel bestellt ist, jeweils die gleiche Gruppe von Lagern auftreten.

- 'AuftragNr' ist mehrwertig von 'ArtikelNr' :

  Für jeden Artikel muß für alle Lager, in denen der Artikel aufbewahrt wird, jeweils die gleiche Gruppe von Aufträgen auftreten.

Damit die Relation der 4. NF genügt, muß sie in zwei neue Relationen (Artikel-Lager und Artikel-Auftrag) aufgespalten werden. Die erste Relation beschreibt, in welchem Zusammenhang Artikel und Lager stehen; die zweite den Zusammenhang zwischen Artikel und Auftrag.

4.4.5 Die 5. Normalform

Definition:

> Ein Relationstyp ist in der **5. Normalform**, wenn er in der 4. Normalform ist und er sich unter keinen Umständen durch Kombination einfacherer Relationstypen mit unterschiedlichen Schlüsseln bilden läßt. [3]

Das ist doch eigentlich selbsterklärend, oder? ;-)

4.4.6 Denormalisierung der Tabellen

Im Prinzip kann man die Tabellen, die man nach den fünf Normalisierungen erhalten hat, 1:1 in der DB verwenden. Es ist jedoch zu prüfen, ob man in der Normalisierungswut die Tabellen nicht zu sehr auseinandergerissen hat. Tabellen, die denselben Primärschlüssel haben, können ohne weiteres zusammengelegt werden, ohne gegen eine Normalisierungsform zu verstoßen.

Bei umfangreichen Datenbeständen mit hohen Zugriffszahlen kann es sich jedoch teilweise empfehlen, aus Performancegründen wieder eine gewisse Denormalisierung herzustellen. Da wir aber keine so hohen Zugriffszahlen und Datenbestände haben, daß der Server überlastet werden könnte, können wir diesen Schritt getrost übergehen.
Hierbei kann man sagen, daß es weniger problematisch ist, mit sich nicht ändernden Daten gegen die Normalformen zu verstoßen. Bei diesen entfällt nämlich das Problem, daß beim Ändern nicht alle Daten verändert werden und dadurch Widersprüche entstehen. Trotzdem sollte man sich immer im klaren darüber sein, wann man gegen die Normalformen verstoßen hat!

Als nächstes ist anhand des Streifendiagramms (Kapitel 4.5) zu überprüfen, ob die Tabellen den Anforderungen der Vorgänge entsprechen.

4.5 Streifendiagramm

Um die Tabellen grafisch dazustellen, gibt es verschiedene Methoden. Eine Methode, mit der man relativ schnell einen Überblick über die vorhandenen Relationen einschließlich deren Attribute und Beziehungen bekommt, ist das Streifendiagramm. Damit ist es dann möglich, anhand des Vorgangskatalogs zu überprüfen, ob alle Vorgänge möglich sind und die Relationen stimmen.

Als Beispiel habe ich die Relationen 'Auftrag', 'best. Artikel', 'Artikel' und 'Hersteller' aus dem obigen Beispiel in der Abbildung 4.1 dargestellt.

4.6 Das ER-Modell

Bei größeren Projekten unumgänglich, bei kleineren für manche schöner: das Entity Relationship-Modell. Wer ein wenig Erfahrung mit Datenbanken hat und gut nachdenkt, kann auch mit Hilfe des ER-Modells Datenmodelle entwickeln, die allen Normalformen entsprechen.

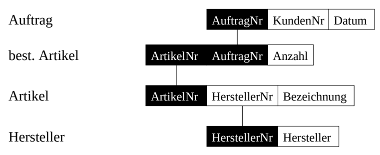

Abbildung 4.1: Streifendiagramm

Es gibt verschiedene Formen, das ER-Modell zu zeichnen. Ich benutze hier das sogenannte Krähenfuß-Diagramm.

Was bedeutet eigentlich Entity-Relationship ??
Für Entität gibt es verschiedene gebräuchliche Definitionen:

> **Entität** [mlat.]*die, -/-en, Philosophie:* die bestimmte Seinsverfassung (Wesen) des einzelnen Seienden, auch diese selbst. [1]
>
> **Entität** [lat.-mlat.] *die, -, -en:* 1. Dasein im Unterschied zum Wesen eines Dinges (Philos.). 2. [gegebene] Größe [2]

Wer jetzt weiß, was Entity bedeutet, kann diesen Absatz überspringen; für alle anderen versuche ich, es anders zu erklären: „Entity" kann man mit „Objekt" oder „Ding" ins Deutsche übersetzen, letztlich sind es konkrete Objekte der Realität. Beim ER-Modell sind Entities Objekte, die über Attribute weiter beschrieben werden können.

Nachdem wir jetzt hoffentlich wissen, was Entity bedeutet, sind wir beim zweiten Begriff angelangt: Relationship bedeutet so viel wie „Beziehung" oder „Relation".

Ein kleines Beispiel:

Für ein Unternehmen soll eine Datenbank entwickelt werden. Es sollen alle Mitarbeiter der Firma gespeichert werden. Jeder Mitarbeiter hat einen Vorgesetzten und gehört zu einer Abteilung. Außerdem verfügt jede Abteilung über einige oder keine PKWs aus dem Fuhrpark für die Mitarbeiter. Zusätzlich soll die Antwort auf die Frage möglich sein, wer wann mit welchem Wagen wie viele Kilometer gefahren ist.

Diese Realtitätsbeschreibung legt drei Entitäten nahe: Mitarbeiter, Abteilung, PKW. Die Zeichnung 4.2 stellt die Beziehung der drei Entitäten und deren Attribute dar.

Wie man unschwer erkennen kann, stellen die Rechtecke die Entitäten da. Sowohl die Entitäten, als auch nachher die Tabellen werden grundsätzlich in der Einzahl bezeichnet.

Die Striche zwischen den Entitäten stellen deren Beziehung da. Der durchgezogene Strich bedeutet **genau einer**, der gestrichelte **einer oder keiner**. Der Krähenfuß auf der anderen Seite bedeutet **einer oder mehrere**. Die Wörter an den Strichen zeigen, was die Beziehung aussagt.

Also z. B.: Ein Mitarbeiter *gehört* zu *genau einer* Abteilung. Eine Abteilung *kann* aus *keinem, einem oder mehreren* Mitarbeiter *bestehen*. Daß eine Abteilung keinen Mitarbeiter haben kann, mag auf den ersten Blick merkwürdig erscheinen. Was ist aber, wenn die Abteilung gerade erst eingerichtet wurde?

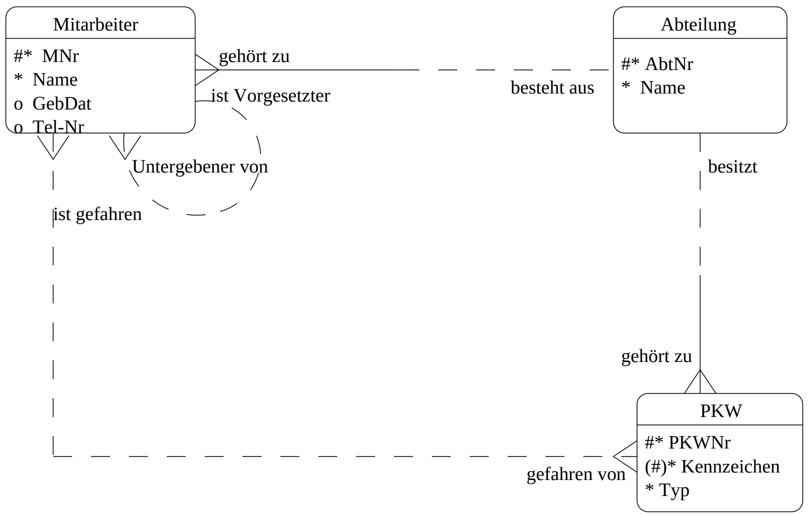

Abbildung 4.2: ER-Beispiel 1

Wie wir bei der Entität Mitarbeiter sehen, kann es durchaus auch eine Beziehung zwischen einer Entität mit sich selber geben. Über diese Beziehung wird ausgesagt wer Vorgesetzter von wem ist. Daß diese Beziehung von beider Seite eine ‚kann'-Beziehung sein soll, mag auf den ersten Blick komisch erscheinen. Aber nicht jeder Mitarbeiter hat einen Vorgesetzten, der Chef hat keinen.

In den Rechtecken stehen die wichtigsten Attribute. Primärschlüssel werden durch ein **#** gekennzeichnet, Primärschlüsselkandidaten dagegen durch **(#)** markiert. Attribute, bei denen ein Wert eingetragen werden muß, werden mit einem ***** versehen; bei den optionalen wird ein **o** verwendet.

Wenn man sich dieses ER-Modell einmal genauer ansieht, stellt man fest, daß es gegen die 1. NF verstößt. Die Beziehung zwischen 'Mitarbeiter' und 'PKW' ist eine n:m Beziehung. Um dieses Problem zu lösen, wird eine sog. Link-Relation erstellt. Wie das dann genau aussieht, ist in Abbildung 4.3 dargestellt.

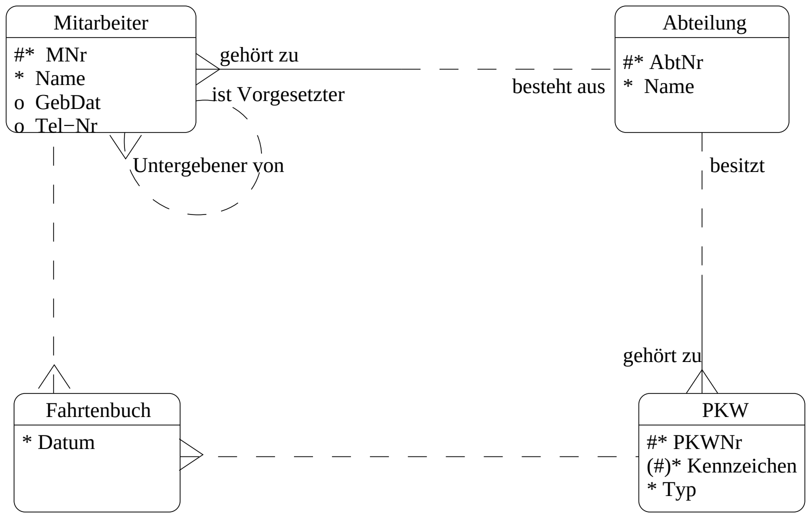

Abbildung 4.3: ER-Beispiel 2

4.7 Relationen erstellen

Last but not least müssen aus dem ER-Modell noch die Tabellen erstellt werden. Das ist allerdings kein großes Problem mehr - jede Entität wird einfach zu einer Tabelle. Als nächstes sucht man sich alle Primärschlüssel. In unserem Beispiel ergäbe sich folgendes:

Tabelle	Primärschlüssel
Mitarbeiter	MNr
Abteilung	AbtNr
Fahrtenbuch	MNr, PKWNr, Datum
PKW	PKWNr

Die Relation 'Fahrtenbuch' hat einen zusammengesetzten Primärschlüssel. Die 'MNr' oder auch das Datum für sich alleine wären nicht eindeutig, da ein Mitarbeiter ja nicht nur einmal Auto fährt und zu einem Zeitpunkt ja auch ggf. mehrere Leute gleichzeitig Autos fahren. In der Kombination jedoch sind die drei Attribute eindeutig.

Bei 'PKW' wurde nicht das Kennzeichen als Primärschlüssel genommen, obwohl es sich dafür eignen würde (mit Sicherheit eindeutig). Allerdings kann es passieren, daß ein PKW ein neues Kennzeichen bekommt. Um auch dann noch die Datenintegrität sicherstellen zu können, habe ich ein neues Attribut eingeführt.

Nachdem man die Primärschlüssel herausgesucht hat, müssen auch die Fremdschlüssel gesucht werden, damit man die Beziehung zwischen den Relationen herstellen kann. Das ist mit dem Krähenfußdiagramm relativ einfach. Alle Relationen, die einen Krähenfuß haben, bekommen den Primärschlüssel der anderen Relation. D. h. es muß z. B. die Abteilungsnummer in die Relation 'Mitarbeiter' eingefügt werden. Das ist auch logisch, denn eine Abteilung kann natürlich auch mehrere Mitarbeiter haben. Würde man dagegen die Mitarbeiter-Nummer in 'Abteilung' einfügen, hätte man einen Verstoß gegen die 1. NF.

Aus unserem Beispiel ergäben sich also folgende Relationen (mit Angabe der zu den Fremdschlüsseln gehörigen Primärschlüssel):

Tabelle	Primärschlüssel	Fremdschlüssel
Mitarbeiter	MNr	AbtNr [Abteilung(AbtNr)] VNr [Mitarbeiter(MNr)]
Abteilung	AbtNr	
Fahrtenbuch	MNr, PKWNr, Datum	MNr [Mitarbeiter(MNr)] PKWNr [PKW(PKWNr)]
PKW	PKWNr	AbtNr [Abteilung(AbtNr)]

Als letztes müssen noch die benötigten Attribute den Relationen hinzugefügt und für alle Attribute die entsprechenden Datentypen festgelegt werden.

4.8 Datentypen

Nachdem jetzt die Datenstruktur feststeht, muß noch festgelegt werden, welche Datentypen für die einzelnen Attribute verwendet werden sollen. Im Prinzip gibt es drei verschiedene Datentypen: Zahlen, Text und große Objekte. In den verschiedenen Datenbanken gibt es dazu dann entsprechend verschiedene Größen. Eine Auflistung möglicher Datentypen für MySQL befindet sich im Kapitel 6.2.

Bei der Überlegung, welcher Datentyp zu verwenden ist, sollte man nicht vom Normalfall ausgehen, sondern grundsätzlich alle möglichen Ausnahmen in Betracht ziehen. So ist es z. B. notwendig, für 'Hausnummer' einen Text-Typ zu wählen, da z. B. „5a" auch eine gültige Hausnummer ist.

Teil II

Praktischer Teil SQL

5 SQL benutzen

Ein DBS kann auf drei Arten gesteuert werden: Entweder dialogorientiert, im Batch-Betrieb oder durch andere Programme. Dialogorientiert bedeutet, daß man am Bildschirm seine Befehle eingibt und innerhalb von Sekunden das Ergebnis oder die Fehlermeldung erhält. Das ganze ist vergleichbar mit der Konsole[1] beim Betriebssystem. In beiden Fällen gibt es einen Prompt[2], an dem man seine Befehle eingibt und im selben Fenster erscheint dann die Ausgabe.

Über den MySQL-Prompt hat man natürlich die meisten Möglichkeiten, weil man jeden Befehl verwenden kann. Häufig ist aber auch ein wenig Unterstützung durch ein Programm praktisch, das z. B. die Tabellenstruktur anzeigt oder beim Ändern die alten Werte als Vorbelegung nimmt etc. Ein sehr schönes Programm dazu ist PHPMyAdmin, mehr dazu im Kapitel 7.3.

In diesem Kapitel werden wir dialogorientiert arbeiten. Alle Befehle, die hier direkt eingegeben werden, können aber auch in eine Text-Datei geschrieben werden, die dann „dialogorientiert“ abgearbeitet wird. Das nennt man dann Batch-Betrieb. Sehr empfehlenswert ist dies z. B. für die Erstellung von Tabellen. Dann kann man nämlich ohne großen Aufwand dieselbe Tabellenstruktur in verschiedenen DBs verwenden.

Bei der Benutzung durch andere Programme merkt der Benutzer nicht, daß eine oder welche Datenbank im Hintergrund arbeitet. So benutzen z. B. alle Suchmaschinen im Internet (Google, AltaVista, Yahoo) in irgendeiner Form eine Datenbank, um die Informationen zu speichern. Als normaler Nutzer sieht man aber nur die Suchmaske und bekommt dann die Ergebnisse schön formatiert angezeigt. Es muß dann ein Programm geben, das die eingegebene Anfrage an die Datenbank weiterreicht und dann die Ergebnisse formatiert ausgibt.

Dieses Programm kann im Prinzip in jeder beliebigen Sprache geschrieben werden. Häufig wird dafür PHP verwendet, was weiter unten noch beschrieben wird.

5.1 MySQL

Um mit dem DBMS zu „reden“, muß das Programm `mysql` von einem Rechner, von dem aus man Zugriff auf den Rechner mit dem DBMS hat, gestartet werden. Viele Provider erlauben dies leider aus Sicherheitsgründen nicht, so daß es empfehlenswert ist, sich zu Hause ein DBMS zu installieren[3]. Die mysql-Aufrufsyntax lautet:

```
mysql -p<Paßwort> -u <Benutzername> <DB-Name>
```

Für Paßwort, Benutzername und DB-Name sind natürlich die entsprechenden Werte einzutragen (ohne die spitzen Klammern!). Diese bekommst du vom Provider, der dir die

[1] unter Windows: MS-DOS Eingabeaufforderung
[2] unter DOS z. B. `c:`
`>`
[3] Das schließt natürlich das Starten des Datenbank-Servers mit ein!

Datenbank zur Verfügung stellt, bzw., im Falle einer eigenen DB, gibst du dir selbst die Daten. ;-)

Alternativ kannst du auch dein Paßwort abfragen lassen:

```
mysql -u <Benutzername> -p <DB-Name>
Enter password: ********
```

Wie du siehst, wird in diesem Fall das Paßwort explizit abgefragt. Diese Methode ist der Möglichkeit, das Paßwort hinter -p anzugeben, aus zwei Gründen vorzuziehen: Zum einen erscheint dann das Paßwort weder (unter Unix/Linux) in der .bash_history noch im Klartext auf dem Bildschirm und zum anderen hat man dann auch nicht das Problem, darauf achten zu müssen, daß zwischen -p und dem Paßwort kein Leerzeichen stehen darf, weil nach einem solchen je nach Reihenfolge der Parameter der Datenbankname erwartet wird ...

Wenn du keinen DB-Namen angibst, bekommst du zwar auch den mysql-Prompt, kannst dann aber einige Befehle nicht nutzen, weil du mit keiner Datenbank verbunden wirst. Dann mußt du noch mit Hilfe von USE <DB-Name> die Datenbank angeben.

Wir beim Jugendnetz richten immer zwei Accounts für dieselbe DB ein. Einen, für den sowohl Lese- als auch Schreibzugriffe erlaubt sind, und einen anderen, der nur lesend zugreifen darf. Der nur lesende Account bekommt ein '_ro' an das Ende des Benutzernamens angehängt. Solltest du einen eigenen Server mit MySQL betreiben, solltest du dir auch den Abschnitt 7.2 über MySQL-Nutzer durchlesen.

Wenn das alles geklappt hat, kommt folgende Ausgabe (oder ähnlich):

```
Welcome to the MySQL monitor.  Commands end with ; or \\g.
Your MySQL connection id is 1561 to server version: 3.22.32

Type 'help' for help.

mysql>
```

Immer dann, wenn in der letzten Zeile ein mysql> steht, kannst du deine Befehle eingeben. Die Groß-/Kleinschreibung ist bei den Befehlen egal, bei den Tabellen- und Spaltennamen (Attribute) sowie den eigentlichen Werten dagegen natürlich nicht!

5.1.1 Dateien abarbeiten

Wie oben schon erwähnt, kann man die Befehle auch mit einem ASCII[4]-Editor in eine Datei schreiben und diese dann abarbeiten lassen. Der Aufruf, der das Abarbeiten der Datei startet, ähnelt dem normalen mysql-Aufruf:

```
mysql -p{Paßwort} -u{Benutzername} {DB-Name} < dateiname
```

Alle (Fehler-)Meldungen, die normalerweise angezeigt werden, werden auf den Bildschirm geschrieben.

[4] American Standard Code for Information Interchange

5.1.2 Kommentare

Kommentare können wie in der Programmiersprache C mit /* und */ umrahmt in die Datei eingefügt werden. Die beiden folgenden Anweisungen sind identisch:

```
SELECT * FROM Mitarbeiter;

SELECT * /* Was soll alles ausgewählt werden? */
  FROM /* ... und aus welcher Tabelle? */ Mitarbeiter;
```

Es gibt auch noch # und - - als Kommentarzeichen. Bei diesen wird alles, was hinter dem Zeichen bis zum Zeilenende steht, als Kommentar interpretiert.

```
SELECT * FROM Mitarbeiter; # Wir wollen alles
SELECT * FROM Mitarbeiter; -- Diese Zeile macht dasselbe wie die darüber
```

ist kein Kommentarzeichen nach ANSI-Norm, d.h. andere Datenbanken können diese Kommentare nicht erkennen.

6 SQL-Befehle

> Das gute Gedächtnis ist wie ein Sack,
> es behält alles.
> Das bessere Gedächtnis ist wie ein
> Sieb; es behält nur, worauf es
> ankommt.
>
> Helmut Walters

Sobald man mit dem DBMS verbunden ist, kann man die Befehle eingeben, die auch von PHP aus aufgerufen werden.

Bei der dialogorientierten Benutzung und im Batch-Betrieb müssen die Befehle immer mit einem `;` abgeschlossen werden. Man kann ohne Probleme einen Befehl in mehrere Zeilen schreiben, indem man erst am Ende das Semikolon setzt.

Als theoretische Grundlage wird die Beispiel-Datenbank, die in Kapitel 4.6 beschrieben wurde, benutzt.

6.1 CREATE DATABASE

Was eine Datenbank theoretisch ist, haben wir im letzten Kapitel 4 gelernt. Wie aber sieht es auf dem Server, also technisch, aus? Im Falle von MySQL entspricht eine Datenbank hier ganz profan einem bestimmten Unterverzeichnis der MySQL-Installation. Relationen dieser Datenbank, die Tabellen, sind als Dateien realisiert, die sich in einem solchen Unterverzeichnis befinden. Diese Dateien bestehen aus Binärcode, d. h. sie können und sollten ausschließlich von MySQL selbst gelesen werden (anders als etwa .sql Dateien).

Im Allgemeinen wird wohl eine Datenbank vom Administrator (beim Jugendnetz: wir) eingerichtet werden. Von diesem bekommst du dann auch die nötigen Zugangsdaten wie Datenbankname, Benutzername und Paßwort. Bist du selbst jedoch der Administrator des Systems (wie z. B. bei der Installation eines Offline-Testwebservers), mußt du das Einrichten der Datenbank auch selbst erledigen; der MySQL-Befehl hierzu lautet `CREATE DATABASE`, gefolgt von dem gewünschten Namen der Datenbank.

Beim Anlegen wie auch beim späteren Benutzen der Datenbank ist folgendes zu bedenken: Unix/Linux unterscheidet Groß- und Kleinschreibung. Aufgrund o. g. Dateisystem-Abhängigkeit der Datenbank (Verzeichnischarakter) wäre die Datenbank „Termine“ nämlich eine andere als „termine“.
Als Grundregel solltest du dir einfach angewöhnen, bei jeglichen Eingaben von Datenbank-, Tabellen- und Feldnamen grundsätzlich auf die Groß- und Kleinschreibung zu achten und diese konsistent zu halten, also auch in allen SQL-Befehlen diese Namen genau so zu verwenden, wie sie angelegt wurden. Selbst wenn Windows (ja, auch hier läuft MySQL!) weniger restriktiv ist, vermeidest du durch dieses Vorgehen späteren Ärger z. B. bei Portierungen von Datenbanken nach Unix/Linux.

Nach dem Einrichten der Datenbank ist diese bereits benutzbar, d. h. du kannst jetzt Tabellen mit entsprechenden Feldern anlegen (mehr dazu im nächsten Abschnitt: 6.2). Arbeitest du hingegen auf dem `mysql`-Prompt, mußt du zuerst die Datenbank wechseln: das geht mittels `USE <DB-Name>`.

6.2 CREATE TABLE

Als erstes müssen wir natürlich die Tabellen erstellen. Dazu dient der Befehl `CREATE TABLE`. Die Syntax lautet:

```
CREATE TABLE table_name (create_definition,...)
```

Für table_name ist selbstverständlich der Name der zu erstellenden Tabelle einzusetzen. Die drei Punkte hinter ‘create_definition‘ bedeuten, daß mehrere ‘create_definitions‘ durch Kommas getrennt hintereinander stehen können. Die ‘create_definitions‘ müssen innerhalb der runden Klammern stehen!

Für ‘create_definition‘ kann man folgendes einsetzen:

```
    feld_name Typ [NOT NULL] [AUTO_INCREMENT] [UNIQUE] [PRIMARY KEY]
oder PRIMARY KEY (feld_name,...)
oder UNIQUE (feld_name,...)
oder FOREIGN KEY (feld_name,...) [reference_definition]
```

Für ‘reference_definition‘ muß man folgendes einsetzen:

```
REFERENCES table_name[(index_spalte,...)]
```

Mit `NOT NULL` wird festgelegt, daß ein Wert (das kann auch ein leerer sein) eingetragen werden muß. NULL ist nicht mit der Zahl 0 zu verwechseln; NULL bedeutet einfach „kein Wert“. Wenn bei `INSERT` kein Wert für dieses Feld angegeben wurde, wird der Standardwert genommen. Es gibt keine Fehlermeldung (nur MySQL)!

Wenn ein Zahlenfeld mit dem Schlüsselwort `AUTO_INCREMENT` angelegt wurde, wird, solange kein Wert für dieses Feld angeben wurde, der höchste Wert +1 genommen. `AUTO_INCREMENT` kann nur einmal pro Tabelle in einem Primärschlüsselfeld verwendet werden. `AUTO_INCREMENT` gibt es nur in MySQL.

Wenn `UNIQUE` bei einem Feld angegeben wurde, darf ein Wert in maximal einem Tupel vorkommen, d. h. jeder Wert muß eindeutig sein. Bei Primärschlüsseln wird automatisch `UNIQUE` angenommen.

Mit `PRIMARY KEY` wird der Primärschlüssel festgelegt. Bei zusammengesetzten Primärschlüsseln sind alle Felder, die Teil des Schlüssels sind, anzugeben. Primärschlüssel müssen eindeutig sein und es muß `NOT NULL` angegeben werden.

`UNIQUE` und `PRIMARY KEY` können entweder direkt hinter einer Spaltendefinition angegeben werden oder getrennt davon. Sobald erst eine Kombination mehrerer Spalten den Primärschlüssel bildet, d. h. eindeutig ist, muß der Primärschlüssel getrennt angegeben werden.

Mit `FOREIGN KEY` wird die Fremdschlüsselbeziehung angegeben. MySQL unterstützt bis jetzt noch keine Fremdschlüssel und ignoriert folglich die Angaben.

Für `Typ` ist einer der Datentypen aus Tabelle 6.1 zu verwenden.

Tabelle 6.1: Verfügbare Datentypen in SQL

Typ	**Beschreibung**
TINYINT	-128 .. 127
TINYINT UNSIGNED	0 .. 255
INT	-2.147.483.648 .. 2.147.483.647
INT UNSIGNED	0 .. 4.294.967.295
BIGINT	-3402823e+31 .. 3402823e+31
DECIMAL(length,dec)	Kommazahl der Länge 'length' und mit 'dec' Dezimalstellen; die Länge beträgt: Stellen vor dem Komma + 1 Stelle für Komma + Stellen nach dem Komma
VARCHAR(NUM) [BINARY]	Zeichenkette mit max 'NUM' Stellen ($1 \leq NUM \leq 255$). Alle Leerstellen am Ende werden gelöscht. Solange nicht 'BINARY' angegeben wurde, wird bei Vergleichen nicht auf Groß-/Kleinschreibung geachtet.
TEXT	Text mit einer max. Länge von 65535 Zeichen
MEDIUMTEXT	Text mit einer max. Länge von 16.777.216 Zeichen
TIME	Zeit; Format: HH:MM:SS, HHMMSS, HHMM oder HH
DATE	Datum; Format: YYYY-MM-DD, wobei '-' jedes nicht numerische Zeichen sein kann
TIMESTAMP	setzt einen Datumswert beim Einfügen/Updaten einzelner Felder automatisch auf das Systemdatum. Format: YYYYMMDDHHMMSS. Wenn mehrere Felder den Typ 'TIMESTAMP' haben, wird immer nur das erste automatisch geändert!

Tabelle 6.2: Bedeutung der YMHSDs

D	Tag (engl. day)
H	Stunde (engl. hour)
M	Monat (engl. month) oder Minute (engl. minute)
S	Sekunde (engl. second)
Y	Jahr (engl. year)

Die Bedeutung der YMHSDs ist in der Tabelle 6.2 erläutert.

Wenn ein Buchstabe mehrmals vorkommt, so bedeutet das, daß es mehrere Stellen gibt.

Ein kleines Beispiel:

Es soll eine Tabelle 'Mitarbeiter' erstellt werden. Zu jedem Mitarbeiter sind Name, Telefonnummer und die Mitarbeiter-Nummer (kurz MNr) zu speichern. Die MNr ist Primärschlüssel und soll automatisch um 1 erhöht werden. Der Name ist nicht optional. Daraus ergibt sich folgender Befehl:

```
CREATE TABLE Mitarbeiter (
   MNr        INT NOT NULL AUTO_INCREMENT,
   VNr        INT,
   AbtNr      INT NOT NULL,
   Name       VARCHAR(30) NOT NULL,
   GebDat     DATE,
   Telefon    VARCHAR(30),
   PRIMARY KEY(MNr),
   FOREIGN KEY(VNr) REFERENCES Mitarbeiter(MNr),
   FOREIGN KEY(AbtNr) REFERENCES Abteilung(AbtNr)
);
```

Die mehrfachen Leerstellen sind optional. Sie sind hier nur der Übersichtlichkeit halber eingefügt.

Wenn man versucht, die Mitarbeiter-Tabelle bei einem DBMS, das Fremdschlüssel unterstützt, als erstes anzulegen, wird man eine Fehlermeldung erhalten, weil die referenzierte Tabelle noch nicht existiert. In diesem Fall kann man die Tabellenstruktur nachträglich mit `ALTER TABLE` (6.11) anpassen.

Im Anhang A.1 sind noch einmal sämtliche 'CREATE TABLE'-Definitionen aufgeführt, die für unser Beispiel benötigt werden. Es sind der Vollständigkeit halber auch sämtliche Fremdschlüssel mit angegeben, obwohl sie unter MySQL keine Bedeutung haben.

6.3 SHOW

Man kann sich die nun erstellten Tabellen und deren Felder auch ansehen. Dazu dient der Befehl `SHOW`. Die Syntax lautet wie folgt (nur MySQL):

```
SHOW TABLES
```

oder

```
SHOW COLUMNS FROM table
```

`SHOW TABLES` zeigt alle angelegten Tabellen an. Mit `SHOW COLUMNS FROM table` lassen sich die Felder in der Tabelle 'table' anzeigen.

Nach unserem obigen `CREATE TABLE` würde sich folgende Ausgabe ergeben:

```
mysql> SHOW TABLES;
+--------------+
| Tables in cr |
```

```
+--------------+
| Mitarbeiter  |
+--------------+
1 row in set (0.01 sec)

mysql> SHOW COLUMNS FROM Mitarbeiter;
+---------+-------------+------+-----+---------+----------------+
| Field   | Type        | Null | Key | Default | Extra          |
+---------+-------------+------+-----+---------+----------------+
| MNr     | int(11)     |      | PRI | 0       | auto_increment |
| VNr     | int(11)     | YES  |     | NULL    |                |
| AbtNr   | int(11)     |      |     | 0       |                |
| Name    | varchar(30) |      |     |         |                |
| GebDat  | date        | YES  |     | NULL    |                |
| Telefon | varchar(30) | YES  |     | NULL    |                |
+---------+-------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)
```

In der unteren Tabelle ist zu erkennen, welche Felder mit welchen Typen und Attributen in der jeweiligen Tabelle vorhanden sind. Nur für 'VNr', 'GebDat' und 'Telefon' sind NULL-Werte zugelassen. Der Primärschlüssel ist 'MNr'. Wie man sieht, wurden die Fremdschlüssel ignoriert.

6.4 DROP TABLE

Da wir jetzt wissen, wie wir Tabellen erstellen, ist es auch wichtig zu wissen, wie wir sie wieder loswerden. Dies geschieht mit dem Befehl `DROP TABLE`.

Die Syntax lautet:

```
DROP TABLE table_name
```

Achtung: Es erfolgt **keine** Abfrage, ob du dies wirklich tun willst! Mit der Tabelle werden natürlich auch alle Daten der Tabelle unwiderruflich gelöscht!

6.5 INSERT INTO

Tabellen ohne Daten haben eigentlich keinen Sinn. Deshalb wollen wir mit dem Befehl `INSERT INTO` ein paar Daten einfügen. Die Syntax lautet:

```
INSERT INTO table_name [ (feld_name,...) ] VALUES (werte,...)
```

Die Feldnamen können weggelassen werden, wenn in alle Felder etwas eingefügt werden soll. In diesem Fall muß man aber die Werte in genau der Reihenfolge eingeben, in der die Felder in der CREATE TABLE Anweisung definiert wurden.

In der Regel, vor allem aber in Programmen, empfiehlt es sich, die Feldnamen anzugeben, weil man sich nie darauf verlassen sollte, daß sich die Reihenfolge der Spalten nicht ändert.

Bei den Werten müssen Zeichenketten und Datum in Hochkommata (Anführungszeichen) stehen, nur für Zahlen gilt das nicht.

In unsere oben erstellte Tabelle sollen folgende Werte eingefügt werden:

Name	**GebDat**	**Telefon**
Christoph Reeg	13.5.1979	
junetz.de	5.3.1998	069/764758

Damit ergeben sich folgende Befehle und Ausgaben (wie man hier bereits sieht, gibt der MySQL-Prompt automatisch die Zeichenfolge '->' aus, wenn ein mehrzeiliger Befehl eingegeben wird):

```
mysql> INSERT INTO Mitarbeiter (Name,GebDat)
    ->  VALUES ('Christoph Reeg','1979-5-13');
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO Mitarbeiter (Name,GebDat,Telefon)
    ->  VALUES ('junetz.de','1998-3-5','069/764758');
Query OK, 1 row affected (0.00 sec)
```

Wegen fehlender Fremdschlüssel konnten die Mitarbeiter in die DB eingefügt werden, obwohl wir keine Abteilung für sie angegeben haben. Deshalb muß man immer selbst für die Einhaltung der Beziehung(en) sorgen!

Um die Datenbasis für unser Beispiel in die DB einzutragen, wird der Befehl noch ein paarmal benutzt. Diese konkrete Anwendung kann im Anhang A.2 nachgeschlagen werden.

Bei den folgenden Kurzbeispielen gehen wir von der Datenbasis unseres Beispiels aus.

6.6 SELECT

Der Befehl `SELECT` ist der mächtigste Befehl in SQL. Die Grund-Syntax lautet:

```
SELECT [DISTINCT | ALL] select_expression,... FROM tables ...
[WHERE where_definition]
[GROUP BY feld_name,...]
[ORDER BY feld_name [ASC | DESC] ,...]
[LIMIT [offset,] rows]
```

Die kürzestmögliche `SELECT`-Anweisung lautet:

```
SELECT * FROM table
```

Es sollen z. B. alle Mitarbeiter ausgegeben werden:

```
mysql> select * from Mitarbeiter;
+-----+------+-------+----------------+------------+--------------+
| MNr | VNr  | AbtNr | Name           | GebDat     | Telefon      |
+-----+------+-------+----------------+------------+--------------+
|   1 | NULL |     3 | Christoph Reeg | 1979-05-13 | NULL         |
```

```
|   2 |    1 |     1 | junetz.de      | 1998-03-05 | 069/764758   |
|   3 |    1 |     1 | Uli            | NULL       | NULL         |
|   4 |    3 |     1 | JCP            | NULL       | 069/764758   |
|   5 |    1 |     2 | Maier          | NULL       | 06196/671797 |
|   6 |    5 |     2 | Meier          | NULL       | 069/97640232 |
+-----+------+-------+----------------+------------+--------------+
6 rows in set (0.00 sec)

mysql>
```

`DISTINCT` und `ALL` sind exklusive, optionale Parameter; soll heißen, es **kann** immer nur einer, **muß** aber keiner benutzt werden. `DISTINCT` sorgt dafür, daß jede identische Zeile nur einmal ausgegeben wird. Mit `ALL` werden die sich wiederholenden Werte auch mehrmals ausgegeben. Ohne Parameter verhält sich das DBMS normalerweise, als ob man `ALL` angeben würde.

Es sollen alle Telefonnummern aus der Mitarbeiter-Tabelle ausgeben werden:

```
mysql> SELECT Telefon from Mitarbeiter;
+--------------+
| Telefon      |
+--------------+
| NULL         |
| 069/764758   |
| NULL         |
| 069/764758   |
| 06196/671797 |
| 069/97640232 |
+--------------+
6 rows in set (0.01 sec)

mysql> SELECT ALL Telefon from Mitarbeiter;
+--------------+
| Telefon      |
+--------------+
| NULL         |
| 069/764758   |
| NULL         |
| 069/764758   |
| 06196/671797 |
| 069/97640232 |
+--------------+
6 rows in set (0.00 sec)

mysql> SELECT DISTINCT Telefon from Mitarbeiter;
+--------------+
| Telefon      |
+--------------+
```



```
| NULL         |
| 06196/671797 |
| 069/764758   |
| 069/97640232 |
+--------------+
4 rows in set (0.05 sec)

mysql>
```

6.6.1 ORDER BY

Mit `ORDER BY` wird festgelegt, nach welcher Spalte bzw. welchen Spalten sortiert werden soll. Mit `ASC` [1] werden die Zeilen aufsteigend, mit `DESC` [2] absteigend sortiert. Ist nichts angegeben, wird aufsteigend sortiert.

Als Beispiel alle Mitarbeiter, nach Namen sortiert:

```
mysql> SELECT * from Mitarbeiter ORDER BY Name;
+-----+------+-------+----------------+------------+--------------+
| MNr | VNr  | AbtNr | Name           | GebDat     | Telefon      |
+-----+------+-------+----------------+------------+--------------+
|   1 | NULL |     3 | Christoph Reeg | 1979-05-13 | NULL         |
|   4 |    3 |     1 | JCP            | NULL       | 069/764758   |
|   2 |    1 |     1 | junetz.de      | 1998-03-05 | 069/764758   |
|   5 |    1 |     2 | Maier          | NULL       | 06196/671797 |
|   6 |    5 |     2 | Meier          | NULL       | 069/97640232 |
|   3 |    1 |     1 | Uli            | NULL       | NULL         |
+-----+------+-------+----------------+------------+--------------+
6 rows in set (0.00 sec)

mysql> SELECT * from Mitarbeiter ORDER BY Name ASC;
+-----+------+-------+----------------+------------+--------------+
| MNr | VNr  | AbtNr | Name           | GebDat     | Telefon      |
+-----+------+-------+----------------+------------+--------------+
|   1 | NULL |     3 | Christoph Reeg | 1979-05-13 | NULL         |
|   4 |    3 |     1 | JCP            | NULL       | 069/764758   |
|   2 |    1 |     1 | junetz.de      | 1998-03-05 | 069/764758   |
|   5 |    1 |     2 | Maier          | NULL       | 06196/671797 |
|   6 |    5 |     2 | Meier          | NULL       | 069/97640232 |
|   3 |    1 |     1 | Uli            | NULL       | NULL         |
+-----+------+-------+----------------+------------+--------------+
6 rows in set (0.01 sec)

mysql> SELECT * from Mitarbeiter ORDER BY Name DESC;
+-----+------+-------+----------------+------------+--------------+
```

[1] engl. ascending
[2] engl. descending

```
| MNr | VNr  | AbtNr | Name           | GebDat     | Telefon      |
+-----+------+-------+----------------+------------+--------------+
|   3 |    1 |     1 | Uli            | NULL       | NULL         |
|   6 |    5 |     2 | Meier          | NULL       | 069/97640232 |
|   5 |    1 |     2 | Maier          | NULL       | 06196/671797 |
|   2 |    1 |     1 | junetz.de      | 1998-03-05 | 069/764758   |
|   4 |    3 |     1 | JCP            | NULL       | 069/764758   |
|   1 | NULL |     3 | Christoph Reeg | 1979-05-13 | NULL         |
+-----+------+-------+----------------+------------+--------------+
6 rows in set (0.00 sec)

mysql>
```

Als letztes soll nach dem Geburtsdatum sortiert werden.

```
mysql>  SELECT * from Mitarbeiter ORDER BY GebDat;
+-----+------+-------+----------------+------------+--------------+
| MNr | VNr  | AbtNr | Name           | GebDat     | Telefon      |
+-----+------+-------+----------------+------------+--------------+
|   3 |    1 |     1 | Uli            | NULL       | NULL         |
|   4 |    3 |     1 | JCP            | NULL       | 069/764758   |
|   5 |    1 |     2 | Maier          | NULL       | 06196/671797 |
|   6 |    5 |     2 | Meier          | NULL       | 069/97640232 |
|   1 | NULL |     3 | Christoph Reeg | 1979-05-13 | NULL         |
|   2 |    1 |     1 | junetz.de      | 1998-03-05 | 069/764758   |
+-----+------+-------+----------------+------------+--------------+
6 rows in set (0.01 sec)

mysql>
```

Die ersten vier Mitarbeiter haben kein Geburtsdatum eingetragen. Um sie dennoch irgendwie zu sortieren, ist bei ihnen ein zweites Sortierkriterium notwendig. Das kann einfach mit einem Komma getrennt hinter `ORDER BY` geschrieben werden.

Um zum Beispiel nach Geburtsdatum und, wenn das nicht eindeutig ist, dann nach Namen zu sortieren, ist folgende Anweisung notwendig:

```
mysql>  SELECT * from Mitarbeiter ORDER BY GebDat,Name;
+-----+------+-------+----------------+------------+--------------+
| MNr | VNr  | AbtNr | Name           | GebDat     | Telefon      |
+-----+------+-------+----------------+------------+--------------+
|   4 |    3 |     1 | JCP            | NULL       | 069/764758   |
|   5 |    1 |     2 | Maier          | NULL       | 06196/671797 |
|   6 |    5 |     2 | Meier          | NULL       | 069/97640232 |
|   3 |    1 |     1 | Uli            | NULL       | NULL         |
|   1 | NULL |     3 | Christoph Reeg | 1979-05-13 | NULL         |
|   2 |    1 |     1 | junetz.de      | 1998-03-05 | 069/764758   |
+-----+------+-------+----------------+------------+--------------+
```



```
6 rows in set (0.00 sec)

mysql>
```

6.6.2 GROUP BY

Die `GROUP BY`-Anweisung wird nur in Verbindung mit den Gruppenfunktionen aus Kapitel 6.7.5 benutzt, um mehrere Tupel mit identischen Attributen zu Gruppen zusammenzufassen.

Ein kleines Beispiel:
Es soll ausgegeben werden, wie viele Mitarbeiter in den jeweiligen Abteilungen arbeiten.

```
mysql> SELECT count(*), AbtNr from Mitarbeiter GROUP BY AbtNr;
+----------+-------+
| count(*) | AbtNr |
+----------+-------+
|        3 |     1 |
|        2 |     2 |
|        1 |     3 |
+----------+-------+
3 rows in set (0.00 sec)

mysql>
```

Die sechs Tupel (Datensätze), die wir in der Mitarbeiter-Tabelle haben, werden zu drei Gruppen zusammengefaßt; anschließend wird die Anzahl der Tupel pro Gruppe ausgeben.

Eigentlich dürfen in der `select_expression` nur Spalten angegeben werden, die in `GROUP BY` auftauchen, MySQL ignoriert dies allerdings. Folgende Anweisung wäre eigentlich unzulässig, zudem ist sie völlig sinnlos - was besagt schon die Spalte 'Name'?

```
mysql> SELECT Name,count(*),AbtNr from Mitarbeiter GROUP BY AbtNr;
+----------------+----------+-------+
| Name           | count(*) | AbtNr |
+----------------+----------+-------+
| junetz.de      |        3 |     1 |
| Maier          |        2 |     2 |
| Christoph Reeg |        1 |     3 |
+----------------+----------+-------+
3 rows in set (0.00 sec)

mysql>
```

6.6.3 LIMIT

Mit `LIMIT [offset,] rows` kann angegeben werden, wie viele Zeilen man maximal von der SELECT-Anweisung zurückgeliefert haben will. Mit 'offset' kann man festlegen, ab welcher Zeile angefangen werden soll. Ohne Angaben wird 0 angenommen. Mit 'rows' legt man fest, wie viele Zeilen man maximal ausgegeben haben möchte.

```
mysql> select * from table LIMIT 5,10;  # gibt die Zeilen 6-15 zurück
mysql> select * from table LIMIT 5;     # gibt die ersten 5 Zeilen zurück
mysql> select * from table LIMIT 0,5;   # dasselbe nochmal
```

6.6.4 select_expression

Für `select_expression` muß eingesetzt werden, was angezeigt werden soll. Im einfachsten Fall ist das ein '*'. Damit wird einfach alles angezeigt.

Im Normalfall werden die benötigten Spalten durch Kommas getrennt angegeben. Sobald die Spaltennamen nicht mehr eindeutig sind, weil mehrere Tabellen angegeben wurden, muß der Tabellenname, gefolgt von einem Punkt, vorangestellt werden; es können dabei auch sog. Aliase verwendet werden. Mehr dazu weiter unten.

Neben den reinen Spaltenwerten können auch Werte, die aus den Spaltenwerten berechnet oder durch sonstige Funktionen erstellt wurden, ausgegeben werden. In Kapitel 6.7 werden die Funktionen ausführlich beschrieben.

Als Beispiel sollen der Mitarbeitername und die Telefonnummer ausgegeben werden. Die beiden Ausgaben sind äquivalent, wie man unschwer erkennen kann. Solange nur aus einer Tabelle ausgelesen wird, ist die erstere allerdings eher empfehlenswert, da nicht zu umständlich.

```
mysql> select Name, Telefon from Mitarbeiter;
+----------------+--------------+
| Name           | Telefon      |
+----------------+--------------+
| Christoph Reeg | NULL         |
| junetz.de      | 069/764758   |
| Uli            | NULL         |
| JCP            | 069/764758   |
| Maier          | 06196/671797 |
| Meier          | 069/97640232 |
+----------------+--------------+
6 rows in set (0.02 sec)

mysql> select Mitarbeiter.Name, Mitarbeiter.Telefon from Mitarbeiter;
+----------------+--------------+
| Name           | Telefon      |
+----------------+--------------+
| Christoph Reeg | NULL         |
| junetz.de      | 069/764758   |
| Uli            | NULL         |
| JCP            | 069/764758   |
| Maier          | 06196/671797 |
| Meier          | 069/97640232 |
+----------------+--------------+
6 rows in set (0.00 sec)

mysql>
```


6.6.5 Alias

Alias bedeutet so viel wie ein „anderer Name“. Man kann sowohl für Spalten als auch für Tabellen Aliase definieren.

6.6.5.1 Tabellen-Alias

Tabellen-Aliase können sowohl bei der **`select_expression`** als auch bei der 'where_definition' zur eindeutigen Spaltenbeschreibung anstelle des Tabellennamens verwendet werden. Aliase werden verwendet, weil sie in der Regel kürzer sind als der Spaltenname. Aliase werden bei `tables` mit einer Leerstelle getrennt hinter dem Tabellennamen eingegeben.

Die folgenden zwei Anweisungen sind völlig identisch, abgesehen davon, daß erstere kürzer ist. Der einzige Unterschied liegt in den ersten beiden Zeilen. Im ersten Beispiel wird bei der `FROM`-Anweisung ein Alias definiert, welches in der ersten Zeile bei der `select_expression` benutzt wird.

```
mysql> select M.Name, M.Telefon, M.AbtNr
    -> FROM Mitarbeiter M
    -> WHERE M.AbtNr = 1;
+-----------+------------+-------+
| Name      | Telefon    | AbtNr |
+-----------+------------+-------+
| junetz.de | 069/764758 |     1 |
| Uli       | NULL       |     1 |
| JCP       | 069/764758 |     1 |
+-----------+------------+-------+
3 rows in set (0.00 sec)

mysql> select Mitarbeiter.Name, Mitarbeiter.Telefon,
    ->        Mitarbeiter.AbtNr
    -> FROM Mitarbeiter
    -> WHERE Mitarbeiter.AbtNr = 1;
+-----------+------------+-------+
| Name      | Telefon    | AbtNr |
+-----------+------------+-------+
| junetz.de | 069/764758 |     1 |
| Uli       | NULL       |     1 |
| JCP       | 069/764758 |     1 |
+-----------+------------+-------+
3 rows in set (0.01 sec)

mysql>
```

6.6.5.2 Spalten-Alias

Wenn man die Überschrift der Spalten ändern will, braucht man Spalten-Aliase. Auf den ersten Blick mag es vielleicht egal sein, wie die Spalten heißen. Später in Verbindung mit

PHP ist das allerdings wichtig und sobald Funktionen in der Abfrage verwendet werden, sind die Spaltenüberschriften auch nicht mehr so schön.

Zum Umbenennen der Spalten wird einfach hinter den Spaltennamen bzw. Ausdruck der Aliasname geschrieben. Alternativ kann auch `AS aliasname` benutzt werden.

Ein kleines Beispiel:
Es soll die Anzahl der Mitarbeiter ausgegeben werden. Dazu wird eine Funktion benötigt (siehe Kapitel 6.7, das soll uns aber nicht weiter stören).

```
mysql> SELECT count(*)
    -> FROM Mitarbeiter;
+----------+
| count(*) |
+----------+
|        6 |
+----------+
1 row in set (0.00 sec)

mysql>
```

Das einzige störende ist, daß die Spalte „count(*)“ heißt. Viel schöner wäre es doch, wenn sie z. B. „Anzahl“ heißen würde:

```
mysql> SELECT count(*) Anzahl
    -> FROM Mitarbeiter;
+--------+
| Anzahl |
+--------+
|      6 |
+--------+
1 row in set (0.00 sec)

mysql> SELECT count(*) AS Anzahl
    -> FROM Mitarbeiter;
+--------+
| Anzahl |
+--------+
|      6 |
+--------+
1 row in set (0.00 sec)

mysql>
```

Wie man unschwer erkennen kann, ist das `AS` optional.

6.6.6 where_definition

Da es nicht immer sinnvoll ist, alle Zeilen auszugeben, kann man über die ‘where_definition‘ angeben, welche Bedingungen erfüllt sein müssen, damit die Zeile ausgegeben wird.

Die ‘where_definition‘ wird auch beim Löschen (`DELETE`) und Ändern (`UPDATE`) von ausgewählten Datensätzen gebraucht.

Eine Bedingung kann aus mehreren Teilbedingungen, die mit AND und OR verknüpft werden müssen, bestehen. Eine Teilbedingung besteht aus einem Spaltennamen, einem Operator sowie entweder einer Konstanten, einer weiteren Spalte oder einer Funktion. Die Teilbedingungen können auch mit NOT verneint werden. Schreibfaule können an Stelle von `NOT` auch einfach `!` verwenden; das Ergebnis ist dasselbe. Die Reihenfolge der Teilbedingungen kann durch Klammern beeinflußt werden. Als Operatoren stehen die Vergleichsoperatoren sowie die Operatoren ‘LIKE‘, ‘BETWEEN‘ und ‘IN‘ zur Auswahl. Alle Vergleichsoperatoren aus Tabelle 6.3 stehen zur Verfügung. Bei Vergleichen mit Strings (=VARCHAR) wird im Normalfall nicht auf die Groß-/Kleinschreibung geachtet. Wenn man jedoch unterscheiden will, so muß beim Anlegen der Tabelle bei `VARCHAR` die Option `BINARY` angegeben werden.

Tabelle 6.3: Verfügbare Vergleichsoperatoren in SQL

Operator	Bedeutung	verneinender Operator
=	gleich	<> bzw. !=
<> oder !=	ungleich	=
>	größer	<=
<	kleiner	>=
>=	größer gleich	<
<=	kleiner gleich	>

Es sollen alle Mitarbeiter ausgegeben werden, bei denen die Abteilungsnummer größer als 2 ist:

```
mysql> SELECT Name, AbtNr
    -> FROM Mitarbeiter
    -> WHERE AbtNr > 2;
+----------------+-------+
| Name           | AbtNr |
+----------------+-------+
| Christoph Reeg |     3 |
+----------------+-------+
1 row in set (0.00 sec)

mysql> SELECT Name, AbtNr
    -> FROM Mitarbeiter
    -> WHERE NOT (AbtNr < 3);
+----------------+-------+
| Name           | AbtNr |
+----------------+-------+
| Christoph Reeg |     3 |
+----------------+-------+
1 row in set (0.00 sec)
```

```
mysql>
```

Es sollen alle Mitarbeiter ausgegeben werden, die der Abteilung 1 angehören und als Vorgesetztennummer ebenfalls die 1 haben.

```
mysql> SELECT Name, AbtNr, VNr
    -> FROM Mitarbeiter
    -> WHERE AbtNr = 1
    ->   AND VNr = 1;
+-----------+-------+------+
| Name      | AbtNr | VNr  |
+-----------+-------+------+
| junetz.de |     1 |    1 |
| Uli       |     1 |    1 |
+-----------+-------+------+
2 rows in set (0.02 sec)

mysql>
```

Es sollen alle Mitarbeiter ausgegeben werden, die keinen Vorgesetzten haben oder der Abteilung 1 angehören:

```
mysql> SELECT Name, AbtNr, VNr
    -> FROM Mitarbeiter
    -> WHERE AbtNr = 1
    ->    OR VNr IS NULL;
+----------------+-------+------+
| Name           | AbtNr | VNr  |
+----------------+-------+------+
| Christoph Reeg |     3 | NULL |
| junetz.de      |     1 |    1 |
| Uli            |     1 |    1 |
| JCP            |     1 |    3 |
+----------------+-------+------+
4 rows in set (0.01 sec)

mysql> SELECT Name, AbtNr, VNr
    -> FROM Mitarbeiter
    -> WHERE NOT (AbtNr <> 1)
    ->   OR VNr IS NULL;
+----------------+-------+------+
| Name           | AbtNr | VNr  |
+----------------+-------+------+
| Christoph Reeg |     3 | NULL |
| junetz.de      |     1 |    1 |
| Uli            |     1 |    1 |
```



```
| JCP            |     1 |    3 |
+----------------+-------+------+
4 rows in set (0.00 sec)

mysql> SELECT Name, AbtNr, VNr
    -> FROM Mitarbeiter
    -> WHERE NOT (AbtNr <> 1 AND VNr IS NOT NULL);
+----------------+-------+------+
| Name           | AbtNr | VNr  |
+----------------+-------+------+
| Christoph Reeg |     3 | NULL |
| junetz.de      |     1 |    1 |
| Uli            |     1 |    1 |
| JCP            |     1 |    3 |
+----------------+-------+------+
4 rows in set (0.00 sec)

mysql>
```

Bei der Überprüfung auf keinen Vorgesetzten ist `IS NULL` verwendet worden, da bei Vergleichen mit NULL-Werten nicht mit den normalen Operatoren gearbeitet werden kann. Statt dessen ist nur `IS NULL` oder, verneint, `IS NOT NULL` möglich.

Es gibt mit Sicherheit noch 1001 andere Möglichkeiten, auf diese Lösung zu kommen. Da der erste Korrekturleser mir das nicht glauben wollte, hier noch ein paar Möglichkeiten mehr:

```
mysql> SELECT Name, AbtNr, VNr
    -> FROM Mitarbeiter
    -> WHERE NOT (AbtNr < 1 OR AbtNr > 1)
    ->    OR VNr IS NULL;
+----------------+-------+------+
| Name           | AbtNr | VNr  |
+----------------+-------+------+
| Christoph Reeg |     3 | NULL |
| junetz.de      |     1 |    1 |
| Uli            |     1 |    1 |
| JCP            |     1 |    3 |
+----------------+-------+------+
4 rows in set (0.00 sec)

mysql> SELECT Name, AbtNr, VNr
    -> FROM Mitarbeiter
    -> WHERE (AbtNr <= 1 AND AbtNr >= 1)
    ->    OR VNr IS NULL;
+----------------+-------+------+
| Name           | AbtNr | VNr  |
+----------------+-------+------+
```

```
| Christoph Reeg |     3 | NULL |
| junetz.de      |     1 |    1 |
| Uli            |     1 |    1 |
| JCP            |     1 |    3 |
+----------------+-------+------+
4 rows in set (0.01 sec)

mysql> SELECT Name, AbtNr, VNr
    -> FROM Mitarbeiter
    -> WHERE AbtNr & 1 = 1
    ->    OR VNr IS NULL;
+----------------+-------+------+
| Name           | AbtNr | VNr  |
+----------------+-------+------+
| Christoph Reeg |     3 | NULL |
| junetz.de      |     1 |    1 |
| Uli            |     1 |    1 |
| JCP            |     1 |    3 |
+----------------+-------+------+
4 rows in set (0.00 sec)

mysql> SELECT Name, AbtNr, VNr
    -> FROM Mitarbeiter
    -> WHERE POW(13,AbtNr) = 13
    ->    OR POW(VNr,42) IS NULL;
+----------------+-------+------+
| Name           | AbtNr | VNr  |
+----------------+-------+------+
| Christoph Reeg |     3 | NULL |
| junetz.de      |     1 |    1 |
| Uli            |     1 |    1 |
| JCP            |     1 |    3 |
+----------------+-------+------+
4 rows in set (0.02 sec)

mysql>
```

Es sind noch nicht 1001 Möglichkeiten, aber wenn ich vieeel Zeit habe, werde ich daran weiterarbeiten ;-). Wer sich die beiden letzten Lösungen genau angesehen hat, wird feststellen, daß dort unbekannte Befehle verwendet werden; in Kapitel 6.7 werden diese noch genauer beschrieben.

Bei der letzten Möglichkeit sieht man, was passiert, wenn mit NULL-Werten gerechnet wird: Das Ergebnis ist NULL.

6.6.6.1 LIKE

Immer dann, wenn man in Textfeldern im Suchmuster Platzhalter oder Jokerzeichen (auch reguläre Ausdrücke[3] genannt) verwenden will, können die Vergleichsoperatoren nicht verwendet werden. Ein Beispiel zur Verdeutlichung:

Es sollen alle Mitarbeiter ausgegeben werden, bei denen die Telefon-Vorwahl auf „96" endet. Falls du eine Möglichkeit findest, das mit Vergleichsoperatoren (und ohne Funktionen) zu lösen, schicke mir bitte eine E-Mail an die Adresse dsp@reeg.net; ich bevorzuge übrigens den Operator `LIKE`.

```
mysql> SELECT Name, Telefon
    -> FROM Mitarbeiter
    -> WHERE Telefon LIKE '%96/%';
+-------+--------------+
| Name  | Telefon      |
+-------+--------------+
| Maier | 06196/671797 |
+-------+--------------+
1 row in set (0.00 sec)

mysql>
```

Es soll ein Mitarbeiter mit Namen „Meier" ausgegeben werden. Allerdings ist unbekannt, ob er sich mit „ei" oder „ai" schreibt. Es sollen also alle Möglichkeiten ausgegeben werden.

```
mysql> SELECT Name
    -> FROM Mitarbeiter
    -> WHERE Name LIKE 'M_ier';
+-------+
| Name  |
+-------+
| Maier |
| Meier |
+-------+
2 rows in set (0.00 sec)

mysql>
```

Wie wir sehen, gibt es zwei Jokerzeichen:

%	DOS-Pendant: *	steht für eine beliebige (auch 0) Anzahl beliebiger Zeichen
_	DOS-Pendant: ?	steht für genau ein beliebiges Zeichen

6.6.6.2 BETWEEN

Neben dem LIKE-Operator gibt es auch noch andere, zum Beispiel den BETWEEN-Operator. Er tut das, was man von ihm erwartet: er wählt alle Spalten aus, die zwischen dem oberen und unteren Wert liegen.

[3] engl.: regular expressions

Beispiel: Es sollen alle Mitarbeiter ausgewählt werden, deren Abteilungs-Nummer zwischen 2 und 5 liegt. Mit dem bisherigen Wissen würde man folgende Anweisung nehmen:

```
mysql> SELECT * FROM Mitarbeiter
    -> WHERE AbtNr >= 2 AND AbtNr <=5;
+-----+------+-------+----------------+------------+--------------+
| MNr | VNr  | AbtNr | Name           | GebDat     | Telefon      |
+-----+------+-------+----------------+------------+--------------+
|   1 | NULL |     3 | Christoph Reeg | 1979-05-13 | NULL         |
|   5 |    1 |     2 | Maier          | NULL       | 06196/671797 |
|   6 |    5 |     2 | Meier          | NULL       | 069/97640232 |
+-----+------+-------+----------------+------------+--------------+
3 rows in set (0.01 sec)

mysql>
```

Man kann es aber noch etwas vereinfachen:

```
mysql> SELECT * FROM Mitarbeiter
    -> WHERE AbtNr BETWEEN 2 AND 5;
+-----+------+-------+----------------+------------+--------------+
| MNr | VNr  | AbtNr | Name           | GebDat     | Telefon      |
+-----+------+-------+----------------+------------+--------------+
|   1 | NULL |     3 | Christoph Reeg | 1979-05-13 | NULL         |
|   5 |    1 |     2 | Maier          | NULL       | 06196/671797 |
|   6 |    5 |     2 | Meier          | NULL       | 069/97640232 |
+-----+------+-------+----------------+------------+--------------+
3 rows in set (0.00 sec)

mysql>
```

BETWEEN kann bei Textspalten, Datumsspalten und numerischen Spalten verwendet werden.

6.6.6.3 IN

Der Operator IN schließlich wird benutzt, wenn man nicht mit einem einzelnen Wert, sondern mit einer Wertemenge vergleichen will.

Zur Verdeutlichung: Es sollen alle Mitarbeiter ausgegeben werden, deren Telefonnummer „06196/671797“ oder „069/764758“ ist. Mit den bisherigen Operatoren würde sich folgende Abfrage ergeben:

```
mysql> SELECT * FROM Mitarbeiter
    -> WHERE Telefon = '06196/671797' OR Telefon = '069/764758';
+-----+------+-------+-----------+------------+--------------+
| MNr | VNr  | AbtNr | Name      | GebDat     | Telefon      |
+-----+------+-------+-----------+------------+--------------+
|   2 |    1 |     1 | junetz.de | 1998-03-05 | 069/764758   |
```



```
|   4 |    3 |     1 | JCP       | NULL       | 069/764758   |
|   5 |    1 |     2 | Maier     | NULL       | 06196/671797 |
+-----+------+-------+-----------+------------+--------------+
3 rows in set (0.00 sec)

mysql>
```

Das funktioniert zwar, aber da diese Möglichkeit bei großen Mengen von Werten sehr umständlich und unübersichtlich wird, hier das ganze nochmal mit dem IN-Operator:

```
mysql> SELECT * FROM Mitarbeiter
    -> WHERE Telefon IN ('06196/671797','069/764758');
+-----+------+-------+-----------+------------+--------------+
| MNr | VNr  | AbtNr | Name      | GebDat     | Telefon      |
+-----+------+-------+-----------+------------+--------------+
|   2 |    1 |     1 | junetz.de | 1998-03-05 | 069/764758   |
|   4 |    3 |     1 | JCP       | NULL       | 069/764758   |
|   5 |    1 |     2 | Maier     | NULL       | 06196/671797 |
+-----+------+-------+-----------+------------+--------------+
3 rows in set (0.00 sec)

mysql>
```

Der IN-Operator kann bei Textspalten, Datumsspalten und numerischen Spalten verwendet werden.

Die Verneinung des IN-Operators ist NOT IN. Als Beispiel sollen alle Mitarbeiter ausgegeben werden, deren Telefonnummer nicht „06196/671797“ oder „069/97640232“ ist. Erst umständlich mit `OR` und dann elegant…

```
mysql> SELECT * FROM Mitarbeiter
    -> WHERE NOT (Telefon = '06196/671797' OR Telefon = '069/764758');
+-----+------+-------+----------------+------------+--------------+
| MNr | VNr  | AbtNr | Name           | GebDat     | Telefon      |
+-----+------+-------+----------------+------------+--------------+
|   1 | NULL |     3 | Christoph Reeg | 1979-05-13 | NULL         |
|   3 |    1 |     1 | Uli            | NULL       | NULL         |
|   6 |    5 |     2 | Meier          | NULL       | 069/97640232 |
+-----+------+-------+----------------+------------+--------------+
3 rows in set (0.00 sec)

mysql> SELECT * FROM Mitarbeiter
    -> WHERE Telefon NOT IN ('06196/671797','069/764758');
+-----+------+-------+----------------+------------+--------------+
| MNr | VNr  | AbtNr | Name           | GebDat     | Telefon      |
+-----+------+-------+----------------+------------+--------------+
|   1 | NULL |     3 | Christoph Reeg | 1979-05-13 | NULL         |
|   3 |    1 |     1 | Uli            | NULL       | NULL         |
```

```
|   6 |    5 |     2 | Meier          | NULL       | 069/97640232 |
+-----+------+-------+----------------+------------+--------------+
3 rows in set (0.00 sec)

mysql>
```

6.7 Funktionen

Bei `select_expression` und `where_expression` können neben Konstanten und Spaltenwerten auch Funktionen verwendet werden. Es gibt zwei Arten von Funktionen, zum einen die sog. „singlerow"-Funktionen und zum anderen die Gruppenfunktionen. Singlerow-Funktionen werden auf jede Zeile angewendet, während die Gruppenfunktionen immer auf eine Gruppe von Zeilen angewendet werden.

Es wäre zum Beispiel Schwachsinn, zu versuchen, den Betrag von allen Stellenanzahlen auf einmal zu berechnen; bei der Summe paßt das schon eher.

6.7.1 Mathematische Funktionen

Es können nicht nur Spaltennamen angegeben werden, sondern auch mathematische Rechenoperationen mit Spalten und/oder Konstanten.

Eine Auswahl der mathematischen Funktionen ist in Tabelle 6.4 aufgeführt.

+ - * /	addieren/subtrahieren/multiplizieren/dividieren
%	modulo (ganzzahliger Rest)
ABS()	Betrag von
COS()	Cosinus in rad
DEGREES()	Umrechnung von rad in deg (Grad)
MOD()	Modulo (ganzzahliger Rest)
PI()	die Zahl Pi
POW(X,Y)	rechnet X hoch Y aus
RAND()	liefert eine Zufallszahl zwischen 0 und 1
ROUND()	rundet Wert
ROUND(x,n)	rundet Wert von x auf n Stellen
SQRT()	Wurzel (2. Grades)
TRUNCATE(x,n)	schneidet nach n Kommastellen von x ab

Tabelle 6.4: Mathematische Funktionen in SQL

6.7.2 Logische Operatoren

Die logischen bzw. booleschen Operatoren haben wir bereits im Kapitel 6.6.6 kennengelernt. In Tabelle 6.5 sind sie der Vollständigkeit halber noch einmal aufgeführt.

NOT !	logisches NOT. Wird genau dann wahr (gibt 1 zurück), wenn das Argument 0 ist (sonst Rückgabe 0). NOT NULL gibt NULL zurück.
AND &&	logisches UND. Wird genau dann wahr (gibt 1 zurück), wenn alle Argumente weder 0 noch NULL sind (sonst Rückgabe 0).
OR ||	logisches ODER. Wird genau dann wahr (gibt 1 zurück), wenn eines der Argumente weder 0 noch NULL ist (sonst Rückgabe 0).

Tabelle 6.5: Logische Funktionen in SQL

6.7.3 Sonstige Funktionen

In Tabelle 6.6 und 6.7 sind sonstige, teilweise praktische Funktionen aufgeführt. Die Tabellen sind nicht vollständig!

|	bitweises ODER
&	bitweises UND

Tabelle 6.6: Bit-Funktionen in SQL

CONCAT(str1, str2, ...)	Gibt den String zurück, der durch Zusammenführen der Argumente entstanden ist. Sobald ein Argument NULL ist, wird NULL zurückgegeben.
LEFT(str,n)	schneidet n Buchstaben von 'str' ab und gibt diese zurück
LTRIM(str)	löscht alle Leerzeichen am Anfang von 'str'
PASSWORD(str)	verschlüsselt den Klartext 'str'
REVERSE(str)	dreht 'str' um, d.h. letzter Buchstabe ist dann am Anfang
LCASE(str) LOWER(str)	Wandelt 'str' in Kleinbuchstaben und gibt das Ergebnis zurück
UCASE(str) UPPER(str)	Wandelt 'str' in Großbuchstaben und gibt das Ergebnis zurück

Tabelle 6.7: String-Funktionen in SQL

6.7.4 Datums-Funktionen

Mit Hilfe von `DATE_FORMAT` kann man Datumswerte aus Tabellen so formatieren, wie man sie gerne hätte. Die Funktion erwarten zwei Parameter. Zum einen das Datumsfeld, zum anderen den Formatierungs-String. Die Formatierungszeichen (siehe Tabelle 6.9) werden durch die entsprechenden Werte ersetzt. Alle anderen Zeichen werden so wie sie sind ausgegeben.

In PHP gibt es auch eine Datum-Formatierungsfunktion. Ob man nun mit der MySQL-Funktion das Datum formatiert und dann mit PHP ausgibt oder mit Hilfe der PHP-Funktion das Datum formatiert, ist häufig egal. Teilweise ist es praktischer, mit Hilfe der MySQL-Funktion das Datum zu formatieren, weil man dann die Formatierungsanweisung in der SQL-Abfrage hat und mit PHP nur ausgeben muß. Andererseits kann es aber auch praktischer sein, mit PHP zu formatieren, wenn man zum Beispiel dasselbe Datum an

DAYOFWEEK(date)	Gibt den Wochentag-Index des Datums zurück (1 = Sonntag, 2 = Montag, ..., 7 = Samstag)
DAYOFMONTH(date)	Gibt den Tag des Monats zurück
DAYOFYEAR(date)	Gibt den Tag im Jahr zurück
WEEK(date) WEEK(date,first)	Gibt die Woche des Datums zurück. Wenn 'first' nicht angegeben wird bzw. 0 ist, fängt die Woche mit Sonntag an. Ist 'first' z. B. 1, fängt die Woche mit Montag an.
MONTH(date)	Gibt den Monat zurück
YEAR(date)	Gibt das Jahr zurück
DATE_FORMAT(date,format)	Formatiert das Datum `date` entsprechend dem übergebenen `format` String.
UNIX_TIMESTAMP(date)	Gibt den Unix-Timestamp (Sekunden seit dem 1.1.1970) des Datums `date` zurück.

Tabelle 6.8: Datum-Funktionen in SQL

mehreren Stellen auf der Seite verschieden formatiert haben will. Wie gesagt, es kommt auf den Einzelfall und die Vorlieben des Programmierers an.

6.7.5 Gruppenfunktionen

Es können aber auch die sogenannten Gruppenfunktionen verwendet werden. Gruppenfunktionen heißen so, weil sie immer auf eine Gruppe von Tupeln (Datensätzen) angewendet werden.

In Verbindung mit Gruppenfunktionen darf streng genommen kein Spaltenname mehr mit in der `select_expression` stehen. Die einzige Ausnahme ist dann gegeben, wenn der Spaltenname in der `GROUP BY`-Anweisung steht. MySQL sieht das etwas lockerer und gibt keine Fehlermeldung bei Verwendung von Spaltennamen in der `select_expression` aus, allerdings hat dies im Normalfall wenig Sinn.

In Tabelle 6.10 sind ein Teil der verfügbaren Gruppenfunktionen aufgeführt.

Für 'expr' ist immer der Name der Spalte einzusetzen, mit der diese Operation erfolgen soll.

Beispiel: Es soll ausgegeben werden, wie viele Mitarbeiter ihren Geburtstag angegeben haben.

```
mysql> SELECT count(GebDat)
    -> FROM Mitarbeiter;
+---------------+
| count(GebDat) |
+---------------+
|             2 |
+---------------+
1 row in set (0.27 sec)

mysql>
```


`%W`	Wochentag
`%w`	Tag in der Woche (0 = Sonntag, ..., 6=Samstag)
`%d`	Tag des Monats (00 - 31)
`%e`	Tag des Monats (0 - 31)
`%j`	Tag im Jahr (001 - 366)
`%U`	Woche, mit Sonntag als 1. Tag der Woche (00 - 52)
`%u`	Woche, mit Montag als 1. Tag der Woche (00 - 52)
`%M`	Monatsname
`%m`	Monat, numerisch (01 - 12)
`%c`	Monat, numerisch (1 - 12)
`%Y`	Jahr (4stellig)
`%y`	Jahr (2stellig)
`%T`	Uhrzeit (24 Std.) (hh:mm:ss)
`%S`	Sekunden (00 - 59)
`%s`	Sekunden (00 - 59)
`%i`	Minuten (00 - 59)
`%H`	Stunde (00 - 23)
`%k`	Stunde (0 - 23)
`%h`	Stunde (00 - 12)
`%I`	Stunde (00 - 12)
`%l`	Stunde (0 - 12)
`%%`	%

Tabelle 6.9: mögl. Formatierungen für DATE_FORMAT

COUNT(expr)	zählt die Zeilen, deren Werte ungleich NULL sind
AVG(expr)	durchschnittlicher Wert
MIN(expr)	kleinster Wert
MAX(expr)	größter Wert
SUM(expr)	Summe

Tabelle 6.10: Gruppenfunktionen in SQL

6.8 Joins

Nachdem wir jetzt die Tabellen perfekt abfragen können, wollen wir mal ein paar Tabellen miteinander verbinden. Nach unserer Normalisierung (siehe Kapitel 4.4) befinden sich einige Informationen nämlich in verschiedenen Tabellen, obwohl sie eigentlich zusammengehören. Zum Beispiel würde ich gerne alle Mitarbeiter mit deren Abteilungen sehen. Hierbei aber nicht die Abteilungsnummer, sondern den Namen der Abteilung. Nach dem, was wir bis jetzt wissen, würden wir folgende Abfrage starten:

```
mysql> SELECT m.Name, m.AbtNr, a.Name, a.AbtNr
    -> FROM Mitarbeiter m, Abteilung a;
+----------------+-------+------------+-------+
| Name           | AbtNr | Name       | AbtNr |
+----------------+-------+------------+-------+
| Christoph Reeg |     3 | EDV        |     1 |
| junetz.de      |     1 | EDV        |     1 |
| Uli            |     1 | EDV        |     1 |
| JCP            |     1 | EDV        |     1 |
| Maier          |     2 | EDV        |     1 |
| Meier          |     2 | EDV        |     1 |
| Christoph Reeg |     3 | Verwaltung |     2 |
| junetz.de      |     1 | Verwaltung |     2 |
| Uli            |     1 | Verwaltung |     2 |
| JCP            |     1 | Verwaltung |     2 |
| Maier          |     2 | Verwaltung |     2 |
| Meier          |     2 | Verwaltung |     2 |
| Christoph Reeg |     3 | Chefetage  |     3 |
| junetz.de      |     1 | Chefetage  |     3 |
| Uli            |     1 | Chefetage  |     3 |
| JCP            |     1 | Chefetage  |     3 |
| Maier          |     2 | Chefetage  |     3 |
| Meier          |     2 | Chefetage  |     3 |
+----------------+-------+------------+-------+
18 rows in set (0.07 sec)

mysql>
```

6.8.1 Equi-Join

Die obige Abfrage ergibt allerdings nicht ganz das erwünschte Resultat. Bei dieser Art der Abfrage entsteht nämlich das kartesische Produkt, was so viel bedeutet wie „jeder mit jedem". Wie wir oben jedoch unschwer erkennen können, gehören nur die Mitarbeiter und Abteilungen zusammen, deren AbtNr übereinstimmen, deshalb hatten wir diese auch eingefügt. Eine entsprechend modifizierte Abfrage würde demnach folgendermaßen lauten:

```
mysql> SELECT m.Name, m.AbtNr, a.Name, a.AbtNr
    -> FROM Mitarbeiter m, Abteilung a
```



```
    -> WHERE m.AbtNr = a.AbtNr;
+----------------+-------+------------+-------+
| Name           | AbtNr | Name       | AbtNr |
+----------------+-------+------------+-------+
| junetz.de      |     1 | EDV        |     1 |
| Uli            |     1 | EDV        |     1 |
| JCP            |     1 | EDV        |     1 |
| Maier          |     2 | Verwaltung |     2 |
| Meier          |     2 | Verwaltung |     2 |
| Christoph Reeg |     3 | Chefetage  |     3 |
+----------------+-------+------------+-------+
6 rows in set (0.00 sec)

mysql>
```

Vor den Spaltennamen muß jeweils die entsprechende Tabelle genannt werden, da die Namen nicht eindeutig sind. Um nicht jedes Mal den kompletten Tabellennamen benutzen zu müssen, wurden Aliase verwendet (siehe Kapitel 6.6.5).

Über die AbtNr entsteht die Verbindung zwischen den beiden Tabellen. Falls bei der Verknüpfung von mehreren Tabellen nicht das gewünschte Ergebnis erscheint, fehlt häufig eine WHERE-Bedingung, so daß bei einigen Tabellen das kartesische Produkt entsteht. In solchen Fällen kann man einfach nachzählen: Wenn n Tabellen miteinander verknüpft werden sollen, werden (n-1) WHERE-Bedingungen benötigt.

Diese Art der Verbindung wird Equi[4]-Join[5] genannt.

6.8.2 Self-Join

So, wie man mehrere Tabellen miteinander verbinden kann, ist es auch möglich, eine Tabelle mit sich selbst zu verbinden. Notwendig ist dies in unserem Beispiel beim Vorgesetzten. Es ist zwar schön, daß man zu jedem Mitarbeiter die Mitarbeiter-Nummer seines Vorgesetzten abrufen kann, aber der Name wäre doch noch viel schöner:

```
mysql> SELECT m.Name, m.VNr, v.Name, v.MNr
    -> FROM Mitarbeiter m, Mitarbeiter v
    -> WHERE m.VNr = v.MNr;
+-----------+------+----------------+-----+
| Name      | VNr  | Name           | MNr |
+-----------+------+----------------+-----+
| junetz.de |    1 | Christoph Reeg |   1 |
| Uli       |    1 | Christoph Reeg |   1 |
| Maier     |    1 | Christoph Reeg |   1 |
| JCP       |    3 | Uli            |   3 |
| Meier     |    5 | Maier          |   5 |
+-----------+------+----------------+-----+
```

[4] Kurzform für engl. equal, in deutsch: gleich, Gleichheit
[5] deutsch: Verknüpfung

```
5 rows in set (0.13 sec)

mysql>
```

Die Tabelle 'Mitarbeiter' muß zwei Mal innerhalb der `FROM`-Anweisung auftauchen. Um nachher die Spalten eindeutig bezeichnen zu können, müssen Tabellen-Aliase[6] vergeben werden.

Der einzige Schönheitsfehler bei der Abfrage ist, daß der Mitarbeiter „Christoph Reeg“ nicht aufgelistet wird. Im Prinzip ist das logisch, da er keinen Vorgesetzten hat. Dumm ist es trotzdem, und deshalb kommen wir zur nächsten Form des Joins: dem Outer-Join.

6.8.3 Outer-Join

Um beim Join alle Tupel der Haupttabelle mit den dazu passenden Tupeln der nachgeordneten Tabelle zu bekommen, wenn nicht zu jedem Tupel der Haupttabelle ein passender Tupel existiert, wird der Outer-Join benötigt.

Die Syntax unter MySQL lautet (wo sonst die Tabellennamen stehen):

```
     haupttabelle LEFT JOIN tabelle2 ON bedingung
oder haupttabelle LEFT JOIN tabelle2 USING (spalte)
```

Bei der unteren Möglichkeit müssen die Spaltennamen in den beiden Tabellen, über die die Verbindung entsteht, gleich sein. Bei der oberen Möglichkeit muß an die Stelle, wo 'bedingung' steht, das eingesetzt werden, was man beim Equi-Join als 'where_condition' schreiben würde. In beiden Fällen kann auch `LEFT OUTER JOIN` anstelle von `LEFT JOIN` geschrieben werden – das `OUTER` ist optional und nur aus Gründen der SQL-Kompatibilität erlaubt.

Um beim oben gezeigten Beispiel zu bleiben: Es sollen alle Mitarbeiter mit deren Vorgesetzten (sofern vorhanden) angezeigt werden. Da die Spaltennamen ungleich sind (in der Haupttabelle 'VNr' und in der nachgeordneten Tabelle 'MNr'), muß die obere Syntax benutzt werden.

```
mysql> SELECT m.Name, m.VNr, v.Name, v.MNr
    -> FROM Mitarbeiter m LEFT JOIN Mitarbeiter v ON m.VNr = v.MNr;
+----------------+------+----------------+------+
| Name           | VNr  | Name           | MNr  |
+----------------+------+----------------+------+
| Christoph Reeg | NULL | NULL           | NULL |
| junetz.de      |    1 | Christoph Reeg |    1 |
| Uli            |    1 | Christoph Reeg |    1 |
| JCP            |    3 | Uli            |    3 |
| Maier          |    1 | Christoph Reeg |    1 |
| Meier          |    5 | Maier          |    5 |
+----------------+------+----------------+------+
6 rows in set (0.03 sec)

mysql>
```

[6] Wer vergeßlich ist, kann auf Seite 42 nachschlagen :-)

Die Mitarbeitertabelle mit dem Alias „m“ ist in unserem Fall die Haupttabelle.

Wie man unschwer erkennen kann, wird bei den Attributen, bei denen keine Werte existieren (in diesem Beispiel die beiden rechten Spalten in der ersten Zeile), NULL als Wert genommen.

Übrigens: Man muß natürlich nicht sowohl VNr als auch MNr abfragen (also zwischen `SELECT` und `FROM` angeben), da deren Werte wegen der `ON`-Bedingung (in der sie wiederum stehen müssen) immer gleich sind. Wenn man nur wissen möchte, in welcher Beziehung die einzelnen Mitarbeiter zu den Vorgesetzten stehen, kann man sogar ganz auf das Abfragen der Nummern verzichten. Um nicht völlig den Überblick zu verlieren, vergeben wir dabei noch Aliase:

```
mysql> SELECT m.Name AS Mitarb, v.Name as Vorges
    -> FROM Mitarbeiter m LEFT JOIN Mitarbeiter v ON m.VNr = v.MNr;
+----------------+----------------+
| Mitarb         | Vorges         |
+----------------+----------------+
| Christoph Reeg | NULL           |
| junetz.de      | Christoph Reeg |
| Uli            | Christoph Reeg |
| JCP            | Uli            |
| Maier          | Christoph Reeg |
| Meier          | Maier          |
+----------------+----------------+
6 rows in set (0.03 sec)

mysql>
```

Ein neues Beispiel:
Ich will alle Abteilungen aufgelistet haben, aber mit deren Autos.

```
mysql> SELECT a.Name, p.Kennzeichen, p.Typ
    -> FROM Abteilung a, PKW p
    -> WHERE a.AbtNr = p.AbtNr;
+-----------+-------------+---------+
| Name      | Kennzeichen | Typ     |
+-----------+-------------+---------+
| Chefetage | MTK-CR 1    | RR      |
| EDV       | F-JN 1      | VW-Golf |
+-----------+-------------+---------+
2 rows in set (0.04 sec)

mysql>
```

Wie man sieht, führt der Equi-Join nicht zum gewünschten Ergebnis[7]. Also das ganze nochmal als Outer-Join. Da die Schlüsselspalten denselben Namen haben, können in diesem Fall beide Syntaxvarianten verwendet werden:

[7] Sonst würde dieses Beispiel auch nicht im Kapitel Outer-Join stehen . . . ;-)

```
mysql> SELECT a.Name, p.Kennzeichen, p.Typ
    -> FROM Abteilung a LEFT JOIN PKW p ON a.AbtNr = p.AbtNr;
+------------+-------------+---------+
| Name       | Kennzeichen | Typ     |
+------------+-------------+---------+
| EDV        | F-JN 1      | VW-Golf |
| Verwaltung | NULL        | NULL    |
| Chefetage  | MTK-CR 1    | RR      |
+------------+-------------+---------+
3 rows in set (0.00 sec)

mysql> SELECT a.Name, p.Kennzeichen, p.Typ
    -> FROM Abteilung a LEFT JOIN PKW p USING (AbtNr);
+------------+-------------+---------+
| Name       | Kennzeichen | Typ     |
+------------+-------------+---------+
| EDV        | F-JN 1      | VW-Golf |
| Verwaltung | NULL        | NULL    |
| Chefetage  | MTK-CR 1    | RR      |
+------------+-------------+---------+
3 rows in set (0.00 sec)

mysql>
```

Als kleine Herausforderung hätte ich jetzt gerne die Abteilungen, die keinen PKW haben. Die Lösung ist eigentlich einfach. Wenn man sich das obige Ergebnis ansieht, stellt man fest (wie auch etwas weiter oben schon beschrieben), daß bei der Abteilung ohne PKW (nämlich „Verwaltung") 'Kennzeichen' und 'Typ' NULL sind.

```
mysql> SELECT a.Name, p.Kennzeichen, p.Typ
    -> FROM Abteilung a LEFT JOIN PKW p USING (AbtNr)
    -> WHERE Typ IS NULL;
+------------+-------------+------+
| Name       | Kennzeichen | Typ  |
+------------+-------------+------+
| Verwaltung | NULL        | NULL |
+------------+-------------+------+
1 row in set (0.03 sec)

mysql>
```

6.9 DELETE FROM

Immer nur Daten einfügen und anzeigen zu lassen, ist zwar ganz praktisch, aber nicht ausreichend. Um Daten wieder löschen zu können, wird der Befehl `DELETE FROM` benutzt. Die Syntax lautet:


```
DELETE FROM table_name [WHERE where_definition]
```

Für 'table_name' ist selbstverständlich der Name der Tabelle einzusetzen, in der gelöscht wird. Wenn **keine** 'where_definition' angegeben wird, werden **alle Daten ohne Nachfrage gelöscht!** Die 'where_definition' muß denselben Anforderungen genügen wie bei der `SELECT`-Anweisung.

Um vor dem Löschen zu testen, ob auch wirklich nur die richtigen Daten gelöscht werden, kann man erst ein `SELECT *` statt `DELETE` machen. Wenn nur Daten angezeigt werden, die gelöscht werden sollen, ersetzt man das `SELECT *` durch `DELETE`.

6.10 UPDATE

Als letztes fehlt noch das Ändern von Daten. Es ist doch etwas unpraktisch, erst die Daten zu löschen, um sie danach wieder geändert einzufügen. Die Syntax des `UPDATE`-Befehls lautet:

```
UPDATE table_name SET feld_name=expression,... [WHERE where_definition]
```

Für 'table_name' ist selbstverständlich der Name der Tabelle einzusetzen, in der einzelne Einträge geändert werden sollen. 'feld_name' ist durch den Namen des Feldes zu ersetzen, dessen Werte geändert werden sollen und 'expression' durch den neuen Wert. Falls Werte mehrerer Felder geändert werden sollen, können diese Zuweisungen einfach durch je ein Komma getrennt hintereinander geschrieben werden. Wenn **keine** 'where_definition' angegeben wird, werden **alle** Tupel geändert! Die 'where_definition' muß denselben Anforderungen genügen wie bei der `SELECT`-Anweisung.

6.11 ALTER TABLE

Schließlich kommen wir noch zum Ändern von Tabellen. Es ist möglich, bei Tabellen, die Daten enthalten, die Datentypen für einzelne Spalten zu ändern und Spalten hinzuzufügen bzw. zu löschen; beim Löschen gehen natürlich die Daten der gelöschten Spalte verloren! Der Befehl dafür ist `ALTER TABLE`. Die Syntax lautet:

```
ALTER TABLE table_name alter_spec

alter_spec:
     ADD [COLUMN] create_definition [AFTER column_name | FIRST]
oder CHANGE alter_feld_name create_definition
oder ADD PRIMARY KEY (index_spalte,...)
oder ADD INDEX (index_spalte,...)
oder ADD UNIQUE (index_spalte,...)
oder DROP feld_name
oder DROP PRIMARY KEY
oder RENAME new_table_name
```

Für 'create_definition' gilt dasselbe wie schon für 'CREATE TABLE'.

Um das Ganze etwas zu verdeutlichen, hier ein kleines praktisches Beispiel (der Befehl `SHOW COLUMNS` wird hierbei jeweils nur zur Verdeutlichung der Ergebnisse benutzt):
Es wird folgende Tabelle angelegt:

```
mysql> CREATE TABLE temp (
    -> eins     VARCHAR(10) NOT NULL,
    -> zwei     VARCHAR(20),
    -> PRIMARY KEY(eins)
    -> );
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW columns FROM temp;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| eins  | varchar(10) |      | PRI |         |       |
| zwei  | varchar(20) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)

mysql>
```

Als nächstes soll die Spalte 'fuenf' vom Typ `VARCHAR(30)` eingefügt werden.

```
mysql> ALTER TABLE temp
    -> ADD fuenf VARCHAR(30);
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SHOW columns FROM temp;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| eins  | varchar(10) |      | PRI |         |       |
| zwei  | varchar(20) | YES  |     | NULL    |       |
| fuenf | varchar(30) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql>
```

Nun wird die Spalte 'drei' vom Typ `VARCHAR(30)` nach der Spalte 'zwei' eingefügt.

```
mysql> ALTER TABLE temp
    -> ADD drei VARCHAR(30) AFTER zwei;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```



```
mysql> SHOW columns FROM temp;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| eins  | varchar(10) |      | PRI |         |       |
| zwei  | varchar(20) | YES  |     | NULL    |       |
| drei  | varchar(30) | YES  |     | NULL    |       |
| fuenf | varchar(30) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)

mysql>
```

So gefällt uns das aber noch nicht ganz. Also benennen wir einfach die Spalte 'fuenf' in 'vier' um:

```
mysql> ALTER TABLE temp
    -> CHANGE fuenf vier VARCHAR(30);
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SHOW columns FROM temp;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| eins  | varchar(10) |      | PRI |         |       |
| zwei  | varchar(20) | YES  |     | NULL    |       |
| drei  | varchar(30) | YES  |     | NULL    |       |
| vier  | varchar(30) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
4 rows in set (0.01 sec)

mysql>
```

Jetzt soll die Spalte 'eins' nicht mehr Primärschlüssel sein (nicht die erste Spalte der `SHOW`-Anweisung!):

```
mysql> ALTER TABLE temp
    -> DROP PRIMARY KEY;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SHOW columns FROM temp;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
```

```
+-------+-------------+------+-----+---------+-------+
| eins  | varchar(10) |      |     |         |       |
| zwei  | varchar(20) | YES  |     | NULL    |       |
| drei  | varchar(30) | YES  |     | NULL    |       |
| vier  | varchar(30) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
4 rows in set (0.01 sec)

mysql>
```

Die Spalte ‘drei‘ soll nun Primärschlüssel werden. Primärschlüssel müssen aber als ‘NOT NULL‘ definiert werden. Deshalb zuerst einmal eine kleine Änderung des Datentyps:

```
mysql> ALTER TABLE temp
    -> CHANGE drei drei  VARCHAR(30) NOT NULL;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SHOW columns FROM temp;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| eins  | varchar(10) |      |     |         |       |
| zwei  | varchar(20) | YES  |     | NULL    |       |
| drei  | varchar(30) |      |     |         |       |
| vier  | varchar(30) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)

mysql>
```

Und nun definieren wir den Primärschlüssel neu:

```
mysql> ALTER TABLE temp
    -> ADD PRIMARY KEY(drei);
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SHOW columns FROM temp;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| eins  | varchar(10) |      |     |         |       |
| zwei  | varchar(20) | YES  |     | NULL    |       |
| drei  | varchar(30) |      | PRI |         |       |
| vier  | varchar(30) | YES  |     | NULL    |       |
```



```
+-------+-------------+------+-----+---------+-------+
4 rows in set (0.01 sec)

mysql>
```

Jetzt, wo wir alles mühsam erstellt haben, fangen wir wieder an, alle Spalten der Reihe nach zu löschen. Als erstes soll die Spalte ‘drei‘ gelöscht werden.

```
mysql> ALTER TABLE temp
    -> DROP drei;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SHOW columns FROM temp;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| eins  | varchar(10) |      |     |         |       |
| zwei  | varchar(20) | YES  |     | NULL    |       |
| vier  | varchar(30) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql>
```

Der Name ‘temp‘ klingt aber irgendwie langweilig. Deshalb benennen wir die Tabelle in ‘test‘ um:

```
mysql> ALTER TABLE temp
    -> RENAME test;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW columns FROM temp;
ERROR 1146: Table 'cr.temp' doesn't exist
mysql> SHOW columns FROM test;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| eins  | varchar(10) |      |     |         |       |
| zwei  | varchar(20) | YES  |     | NULL    |       |
| vier  | varchar(30) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql>
```

Zum Schluß löschen wir wieder die ganze Tabelle:

```
mysql> DROP TABLE test;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW columns FROM test;
ERROR 1146: Table 'cr.test' doesn't exist
mysql>
```

Die letzte Fehlermeldung besagt, daß die Tabelle nicht existiert. Diese Fehlermeldung bekommt man auch, wenn man sich beim Tabellennamen verschrieben hat (auch Groß-/Kleinschreibung).

7 Tips & Tricks

7.1 zufällige Daten auswählen

Häufig benötigt man ein paar zufällige Datensätze aus einer Tabelle, oder will die vorhandenen Datensätze zufällig sortiert anzeigen.

Es gibt verschiedene Lösungen. Ab MySQL-Version 3.23 ist das etwas einfacher geworden. Ab dieser Version ist es nämlich möglich, `ORDER BY RAND()` zu nutzen. `RAND()` ist eine MySQL-Funktion, die zufällige Werte produziert. Wenn nach der Ausgabe dieser Funktion sortiert wird, ergeben sich folglich zufällige Reihenfolgen.

Bei älteren Versionen funktioniert das nicht: wir müssen etwas in die Trickkiste greifen. Der Trick besteht darin, der ausgegebenen Tabelle eine Spalte mit den zufälligen Werten anzufügen. Das nächste Beispiel zeigt die Idee.

```
mysql> SELECT MNr,Name,RAND() AS sort
    -> FROM Mitarbeiter
    -> ORDER BY sort;
+-----+----------------+--------+
| MNr | Name           | sort   |
+-----+----------------+--------+
|   1 | Christoph Reeg | 0.8742 |
|   2 | junetz.de      | 0.5510 |
|   3 | Uli            | 0.1324 |
|   4 | JCP            | 0.0092 |
|   5 | Maier          | 0.6487 |
|   6 | Meier          | 0.2160 |
+-----+----------------+--------+
6 rows in set (0.00 sec)
```

Die Spalte sort enthält schon die Zufallswerte, aber noch wird nicht richtig danach sortiert. Sobald jedoch eine Spalte mit in die Berechnung von sort einbezogen wird, funktioniert das Ganze, wie im folgenden Beispiel zu sehen ist.

```
mysql> SELECT MNr,Name,0*MNr+RAND() AS sort
    -> FROM Mitarbeiter
    -> ORDER BY sort;
+-----+----------------+--------+
| MNr | Name           | sort   |
+-----+----------------+--------+
|   3 | Uli            | 0.8871 |
|   6 | Meier          | 0.0881 |
|   1 | Christoph Reeg | 0.7790 |
|   5 | Maier          | 0.6311 |
```

```
|   4 | JCP            | 0.8183 |
|   2 | junetz.de      | 0.1982 |
+-----+----------------+--------+
6 rows in set (0.00 sec)
```

Der Summand $0 * MNr$ beeinflußt das Endergebnis nicht, trotzdem funktioniert das Sortieren.

Die Reihenfolge ist zufällig, auch wenn (komischerweise[1]) nicht nach 'sort' sortiert wurde.

Nachdem wir nun die zufällige Reihenfolge erhalten haben, können wir uns auch mit Hilfe von `LIMIT` (siehe auch 6.6.3) einen Datensatz ausgeben lassen.

Eine andere Möglichkeit, genau einen zufälligen Datensatz auszuwählen, wäre z. B. mit Hilfe von PHP: erst mit Hilfe von `COUNT` (siehe auch 6.7.5) bestimmen, wie viele Datensätze es gibt, dann mit PHP eine Zufallszahl bestimmen und genau den korrespondierenden Eintrag ausgeben. Funktioniert im Endeffekt genauso, ist nur etwas mehr Programmieraufwand. Was im Endeffekt schneller läuft, hängt von dem jeweiligen Datenbankdesign ab.

7.2 Nutzer in MySQL

Bei der Installation von MySQL wird standardmäßig ein Nutzer eingerichtet, der kein Paßwort[2], aber alle Rechte hat: Sein Name ist `root`. Mit diesem Nutzer solltest Dd also, wenn du einen eigenen (Offline-) Server betreibst, auf eine neu erstellte Datenbank problemlos zugreifen können.
Um die Datenbank effizient nutzen zu können, sollten allerdings später geeignete Nutzer angelegt werden, z. B. ein Abfragenutzer, der nur Leserechte hat und ein Adminnutzer, der vollen Zugriff auf die Datenbank hat. Generell müssen solche Nutzer dem MySQL-Ansatz der Rechtevergabe folgend in mysql.user (Datenbank mysql, Tabelle user) eingetragen werden. Aufgrund der Komplexität dieses Themas verweise ich aber an dieser Stelle nur auf das (englische) MySQL-Manual: Dort findest du durch Suche nach dem Begriff „privileges" eine ausführliche Beschreibung, wie man entsprechende Anpassungen vornimmt.

7.3 PHPMyAdmin

PHPMyAdmin ist ein PHP-basiertes MySQL-Administrierungssystem, d. h. es stellt eine Möglichkeit dar, über einen gewöhnlichen Web-Browser eine MySQL-Datenbank zu verwalten (Benutzer, Datenbanken, Tabellen und Datensätze anlegen, ändern und löschen). Dabei besteht die Möglichkeit, das System so einzurichten, daß alle MySQL-Nutzer eines Systems sich einloggen und dabei natürlich nur ihre eigenen Datenbanken (genauer: die, für die sie Rechte besitzen) sehen und bearbeiten können. Auch für den Systemverwalter gibt es einige Einstellungsmöglichkeiten wie das Anlegen neuer Nutzer inklusive dem obligatorischen MySQL-Neustart oder Rechtevergabe.

[1] IMHO sollte schon richtig nach der Spalte 'sort' sortiert werden
[2] Das solltest du aber, sobald es geht, ändern, d. h. eines setzen!

Richtig eingesetzt, kann PHPMyAdmin fast vollständig das ‚mysql' Kommandozeilenprogramm ersetzen, denn sogar Im- und Export von fertigen SQL-Dateien ist möglich. Von Vorteil ist natürlich die Umsetzung des Systems in HTML: Während beim Kommandozeilenprogramm schnell die Übersichtlichkeit verloren geht, wenn große Tabellen ausgegeben werden sollen, macht PHPMyAdmin von gewöhnlichen HTML-Tabellen Gebrauch und bietet dabei direkt sinnvolle Editiermöglichkeiten und Vereinfachungen.

Für Nutzer von Billig-Webhosting Angeboten ist ein Zugriff auf die mysql-Kommandozeile gar nicht möglich und dadurch PHPMyAdmin eine sehr einfache Alternative, trotzdem entsprechende Möglichkeit zu bekommen.

Auf der PHPMyAdmin-Homepage *http://www.phpmyadmin.net/* finden sich immer die aktuellen Versionen des Pakets. Zu beachten ist, daß es Versionen für zwei verschiedene Endungen (.php und .php3) gibt. Dies ist dann von besonderem Interesse, wenn, wie beim Jugendnetz Frankfurt/Offenbach, verschiedene Endungen verschiedenen PHP-Interpreter-Versionen zugeordnet sind. Dem Paket liegt zwar ein Script zum Ändern der Endungen in frei wählbare bei, dieses funktioniert aber leider nur mit der Bash, also i. A. nicht unter Windows. Ist also z. B. die Verwendung der Endung .php4 gewünscht, so ist Zugang zu einem Unix-basierten System von Vorteil. ;-)

Wenn man nachfragt, ist bei Anbietern von serverseitiger PHP- und MySQL-Unterstützung oftmals PHPMyAdmin schon deshalb installiert, damit die Administratoren selbst effizienter arbeiten können - so auch bei uns. Im Zweifelsfall lohnt es sich, einfach mal zu fragen, ob dieses System angeboten wird und für Mitglieder auch frei verfügbar ist. Für letzteres ist nämlich nicht viel zu machen, lediglich ein Nutzer mit Leserechten auf die „mysql"-Datenbank muß angelegt werden, was auch kein Sicherheitsrisiko darstellt angesichts der Tatsache, daß die dort abgelegten Paßworte durch den verwendeten Verschlüsselungs-Algorithmus praktisch unknackbar sind[3].

Sollten noch Fragen offen bleiben, die auch die PHPMyAdmin-eigene Dokumentation nicht beantworten kann und die von allgemeinem Interesse sind, kannst du diese gerne an die vorne angegebene E-Mail-Adresse schicken. Auf diese Weise halten häufig gestellte Fragen (FAQ) sicher Einzug in spätere DSP-Versionen.

7.4 Bäume in SQL

An der ein oder anderen Stelle hat man das Problem, daß man eine Baumstruktur in einer SQL-Datenbank speichern will. Jeder kennt mindestens eine Baumstruktur von seinem Computer, nämlich den Verzeichnisbaum. Die eine oder andere Datenbank bietet dazu proprietäre Funktionen (z. B. Oracle), andere nicht (z. B. MySQL). Im Laufe der Zeit wurden hier entsprechende Varianten entworfen, um trotzdem Bäume abspeichern zu können. Ich werde hier einige vorstellen und die jeweiligen Vor- und Nachteile aufzuzeigen versuchen.

7.4.1 Was ist ein Baum?

Jeder kennt sicher die natürliche Variante (die mit den komischen grünen Blättern). Der Informatiker dagegen versteht i. A. etwas anderes unter einem Baum. Es gibt natürlich

[3] Bei der Authentifizierung wird ja nur mit demselben Algorithmus verschlüsselt und dann verglichen - eine Entschlüsselung ist gar nicht vorgesehen (und sollte nicht möglich sein)

auch verschiedene Definitionen, was ein Baum ist.

> Ein Baum ist ein ungerichteter, zyklenfreier, zusammenhängender Graph.

> Ein Baum ist ein azyklischer Graph, in dem genau ein Knoten keinen Vorgänger hat und alle anderen Knoten mindestens einen Vorgänger haben.

Die letzte Definition kann man sogar fast verstehen, trotzdem versuche ich es anders zu erklären.

Bei dem guten alten DOS gab es ein Laufwerk `C:`, auf dem verschiedene Verzeichnisse (z. B. `DOS` und `WIN`) existierten, die wieder Unterverzeichnisse hatten (z. B. `SYSTEM` im `WIN`-Verzeichnis). Wenn man nun `C:` als Wurzel bezeichnet und `DOS` und `WIN` als Nachfolger von `C:`, sowie `SYSTEM` als Nachfolger von `WIN` hat man einen Baum.

An Stelle von Nachfolger und Vorgänger wird auch häufig Sohn und Vater gesagt. Die Elemente in der Mitte des Baumes werden Knoten genannt und die Elemente ohne Nachfolger heißen Blätter.

7.4.2 Beispieldaten

In diesem Kapitel werde ich von einem Beispielbaum ausgehen, wo ich die Elemente so benannt habe, daß eigentlich klar sein sollte, wo sie sich im Baum befinden sollten. In Abbildung 7.1 habe ich den Baum einmal grafisch dargestellt.

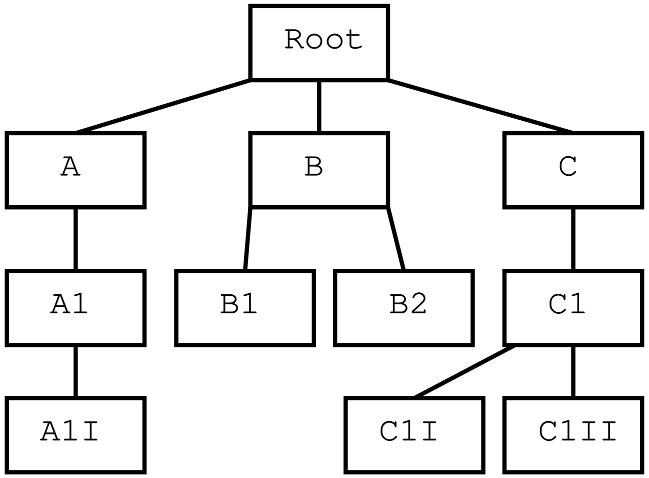

Abbildung 7.1: Unser Baum-Beispiel

7.4.3 Baumdarstellung mit Vater-Zeiger

Eine sehr beliebte Variante zur Baumdarstellung ist, die einzelnen Knoten mit einer eindeutigen ID zu versehen und dann bei jedem Knoten die jeweilige Vater-ID zu speichern. Da über die Vater-ID auf den Vater (übergeordneter Knoten) gezeigt wird, ergibt sich die Bezeichnung „Baumdarstellung mit Vater-Zeiger“. Konkret ergeben sich daraus die Werte in Tabelle 7.1, wobei eine Vater-ID mit 0 bedeutet, daß es die Root (engl.: Wurzel) ist.

ID	**Name**	**Vater-ID**
1	Root	0
2	A	1
3	B	1
4	C	1
5	A1	2
6	B1	3
7	B2	3
8	C1	4
9	A1I	5
10	C1I	8
11	C1II	8

Tabelle 7.1: Baumdarstellung mit Vater-Zeiger

Der Vorteil dieser Variante ist das relativ einfache Erstellen des Baumes. Auch ist er relativ robust gegen falsche Daten; eine falsche Vater-ID führt „nur“ zu einem falsch angehängten Ast, aber ansonsten ist noch alles OK. Auch das Umhängen von einzelnen Ästen ist relativ einfach. Das große Problem kommt allerdings z. B. bei der Ausgabe des Gesamtbaumes, bei der Bestimmung der maximalen Tiefe[4], oder beim Feststellen, ob ein Knoten noch Söhne hat. Siehe dazu auch die Übung 7.4.6.1

7.4.4 Nested Set Modell

Eine andere sehr elegante Methode für Baumstrukturen ist das Nested Set Model, das den Baum als Summe von Mengen betrachtet. Dies hat den Vorteil, daß mit relationalen Datenbanken sehr effizient damit gearbeitet werden kann, da Mengen Relationen sind.

Auf der Seite von Kristian Köhntopp (*http://www.koehntopp.de/*) habe ich diese Variante kennen gelernt. Die dort liegenden Folien sind jedoch zum Verstehen etwas knapp gehalten, so daß ich hier versuche, das etwas ausführlicher zu zeigen.

Wie aber funktioniert das nun? Wir betrachten den Baum als Summe sich überlappender Mengen, wobei immer ein Knoten mit allen Nachfolgern eine Menge bildet. D. h. das Element „Root“ bildet eine Menge, in der alle anderen Elemente enthalten sind; bei „A“ sind nur die Elemente „A1“ und „A1I“ enthalten usw.

Das Ganze ist in Abbildung 7.2 grafisch dargestellt.

Jetzt stellt sich nur noch die Frage, wie man dieses Mengenkonstrukt optimal in die Datenbank bringt. Nichts leichter als das: Jedes Element bekommt zwei zusätzliche Felder;

[4] Tiefe bei einem Baum heißt, wie viele Knoten hintereinander hängen

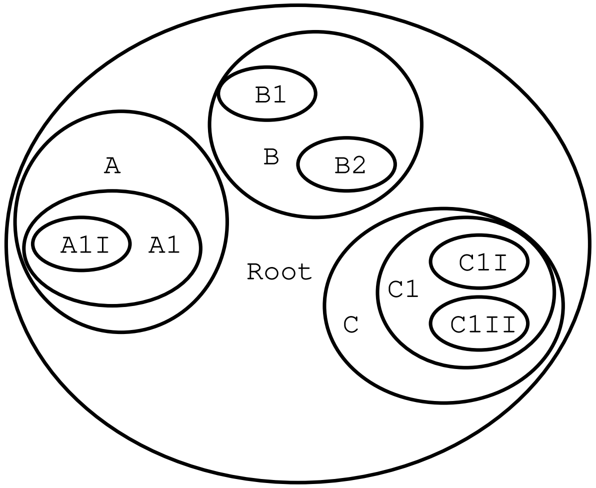

Abbildung 7.2: Baumdarstellung als Summe von Mengen

ich nenne sie einfach mal ‚l' und ‚r'. Dann nehme ich wieder die Abbildung 7.1, fange bei der Wurzel an und schreibe in das Feld ‚l' eine ‚1', gehe dann zu dem linken Nachfolger und schreibe dort in das Feld ‚l' eine ‚2' usw. bis ich unten angekommen bin. Dort schreibe ich in das Feld ‚r' eine ‚5' und gehe wieder zählend hoch. Wer dem jetzt nicht folgen konnte: Keine Angst, in Abbildung 7.3 ist das Ganze grafisch dargestellt und dann sollte es eigentlich klar werden.

Die an dieser Stelle wichtigste Abfrage, ob ein Element Nachfolger eines anderen ist (gleichbedeutend mit „in der Menge enthalten"), läßt sich mit `BETWEEN` erledigen.

Aber nun genug der Theorie, setzen wir das Ganze in SQL um. Dazu als erstes unser `CREATE TABLE` mit den `INSERT` Anweisungen:

```
CREATE TABLE NestedSet (
  ID     int not null primary key,
  Name   varchar(100),
  l      int,
  r      int
);

INSERT INTO NestedSet VALUES (1,'Root',1,22);
INSERT INTO NestedSet VALUES (2,'A',2,7);
INSERT INTO NestedSet VALUES (3,'B',8,13);
INSERT INTO NestedSet VALUES (4,'C',14,21);
```


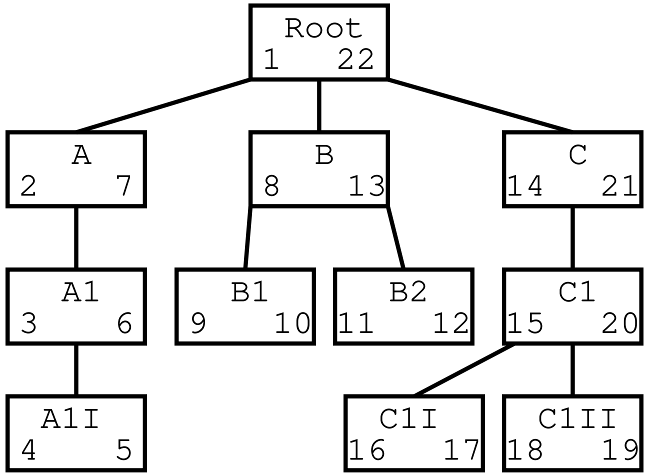

Abbildung 7.3: Baumdarstellung im Nested Set Modell

```
INSERT INTO NestedSet VALUES (5,'A1',3,6);
INSERT INTO NestedSet VALUES (6,'B1',9,10);
INSERT INTO NestedSet VALUES (7,'B2',11,12);
INSERT INTO NestedSet VALUES (8,'C1',15,20);
INSERT INTO NestedSet VALUES (9,'A1I',4,5);
INSERT INTO NestedSet VALUES (10,'C1I',16,17);
INSERT INTO NestedSet VALUES (11,'C1II',18,19);
```

Wie schön diese Struktur ist, zeigt folgende Abfrage:

```
mysql> SELECT v.Name AS Vater, s.Name AS Nachfolger
    ->   FROM NestedSet v, NestedSet s
    ->  WHERE s.l BETWEEN v.l AND v.r
    ->   ORDER BY s.l, v.l;
+-------+------------+
| Vater | Nachfolger |
+-------+------------+
| Root  | Root       |
| Root  | A          |
| A     | A          |
| Root  | A1         |
| A     | A1         |
| A1    | A1         |
```

```
| Root  | A1I        |
| A     | A1I        |
| A1    | A1I        |
| A1I   | A1I        |
| Root  | B          |
| B     | B          |
| Root  | B1         |
| B     | B1         |
| B1    | B1         |
| Root  | B2         |
| B     | B2         |
| B2    | B2         |
| Root  | C          |
| C     | C          |
| Root  | C1         |
| C     | C1         |
| C1    | C1         |
| Root  | C1I        |
| C     | C1I        |
| C1    | C1I        |
| C1I   | C1I        |
| Root  | C1II       |
| C     | C1II       |
| C1    | C1II       |
| C1II  | C1II       |
+-------+------------+
31 rows in set (0.01 sec)
```

Ich bekomme mit einer Abfrage das Ergebnis, wer alles Vorgänger zu einem bestimmten Knoten ist.

Durch eine entsprechende Eingrenzung kann man sich auch alle Vorgänger zu genau einem Knoten ausgeben lassen:

```
mysql> SELECT v.Name AS Vater, s.Name AS Nachfolger
    ->   FROM NestedSet v, NestedSet s
    ->  WHERE s.l BETWEEN v.l AND v.r
    ->    AND s.Name = 'C1I'
    ->   ORDER BY v.l;
+-------+------------+
| Vater | Nachfolger |
+-------+------------+
| Root  | C1I        |
| C     | C1I        |
| C1    | C1I        |
| C1I   | C1I        |
+-------+------------+
4 rows in set (0.00 sec)
```


Die Abfrage, wie viele Nachfolger ein Knoten hat, läßt sich ganz einfach über die Differenz von „l“ und „r“ feststellen.

```
mysql> SELECT (r-l-1)/2 AS Nachfolger, Name
    -> FROM NestedSet
    -> ORDER BY l;
+------------+------+
| Nachfolger | Name |
+------------+------+
|      10.00 | Root |
|       2.00 | A    |
|       1.00 | A1   |
|       0.00 | A1I  |
|       2.00 | B    |
|       0.00 | B1   |
|       0.00 | B2   |
|       3.00 | C    |
|       2.00 | C1   |
|       0.00 | C1I  |
|       0.00 | C1II |
+------------+------+
11 rows in set (0.00 sec)
```

Auch sehr interessant: alle Knoten, mit ihrer Tiefe:

```
mysql> SELECT s.Name, count(*) AS Level
    ->   FROM NestedSet v, NestedSet s
    ->  WHERE s.l BETWEEN v.l AND v.r
    ->  GROUP BY s.l;
+------+-------+
| Name | Level |
+------+-------+
| Root |     1 |
| A    |     2 |
| A1   |     3 |
| A1I  |     4 |
| B    |     2 |
| B1   |     3 |
| B2   |     3 |
| C    |     2 |
| C1   |     3 |
| C1I  |     4 |
| C1II |     4 |
+------+-------+
11 rows in set (0.00 sec)
```

Bevor wir den Baum gleich durch Löschen & Co verändern, gibt es auch hier Übungen

(7.4.6.2) zum Vertiefen. Keine Angst, falls das Ganze noch nicht klar geworden ist; mit der Zeit kommt das Verständnis für diese Struktur.

7.4.4.1 Löschen

So schön diese Methode auch zum Abfragen ist, so schwierig ist sie beim Ändern der Datenstruktur. Man muß immer aufpassen, keinen der Zeiger falsch zu belegen, weil sonst der ganze Baum durcheinander kommt.

Im Prinzip sind folgende Schritte notwendig:

- Sperren der Tabelle
- „l“ und „r“ von dem zu löschenden Element auslesen
- Element löschen
- „l“ und „r“ der Nachfolger um 1 reduzieren
- „l“ und „r“ des rechten Nachbarbaums um 2 reduzieren („r“ im bzw. in den Vaterknoten nicht vergessen!)
- Tabelle wieder freigeben

Um das zu verstehen, ist es hilfreich, sich einfach mal ein Blatt Papier und einen Stift zu nehmen und das exemplarisch durchzugehen.

Wir machen das jetzt hier in SQL, indem wir aus dem Beispiel den Knoten „B“ löschen. Die beiden Nachfolger „B1“ und „B2“ sollen dann direkt unter „Root“ hängen.

Als erstes verhindern wir, daß durch einen evtl. parallelen Löschvorgang die Tabelle durcheinander kommt, indem wir sie sperren.

```
mysql> LOCK TABLE NestedSet WRITE;
Query OK, 0 rows affected (0.00 sec)
```

Nun brauchen wir „l“ und „r“ von unserem Element:

```
mysql> SELECT l,r
    ->   FROM NestedSet
    ->  WHERE Name = 'B';
+------+------+
| l    | r    |
+------+------+
|    8 |   13 |
+------+------+
1 row in set (0.00 sec)
```

Diese müssen wir uns nun merken.

Den Knoten zu löschen, sollte kein großes Problem sein:

```
mysql> DELETE FROM NestedSet
    ->  WHERE Name = 'B';
Query OK, 1 row affected (0.01 sec)
```


Kommen wir nun zu den Nachfolgern (hier „B1“ und „B2“), zu erkennen an deren „l“ bzw. „r“, die zwischen „8“ und „13“ (siehe letzte Abfrage) liegen.

```
mysql> UPDATE NestedSet
    ->    SET l=l-1, r=r-1
    ->  WHERE l BETWEEN 8 AND 13;
Query OK, 2 rows affected (0.00 sec)
Rows matched: 2  Changed: 2  Warnings: 0
```

Nicht zu vergessen der rechte Nachbarbaum (in diesem Fall alle „C*“ Elemente).

```
mysql> UPDATE NestedSet
    ->    SET l=l-2
    ->  WHERE l > 13;
Query OK, 4 rows affected (0.00 sec)
Rows matched: 4  Changed: 4  Warnings: 0

mysql> UPDATE NestedSet
    ->    SET r=r-2
    ->  WHERE r > 13;
Query OK, 5 rows affected (0.00 sec)
Rows matched: 5  Changed: 5  Warnings: 0
```

Und als letzter wichtiger Schritt muß natürlich noch die Tabelle wieder freigegeben werden.

```
mysql> UNLOCK TABLES;
Query OK, 0 rows affected (0.00 sec)
```

Das war jetzt das Löschen eines Elements. Wer das Prinzip des Baumaufbaus verstanden hat, kann sich auch noch davon überzeugen, daß es tatsächlich funktioniert hat.

```
mysql> SELECT *
    ->   FROM NestedSet
    ->  ORDER BY l;
+----+------+------+------+
| ID | Name | l    | r    |
+----+------+------+------+
|  1 | Root |    1 |   20 |
|  2 | A    |    2 |    7 |
|  5 | A1   |    3 |    6 |
|  9 | A1I  |    4 |    5 |
|  6 | B1   |    8 |    9 |
|  7 | B2   |   10 |   11 |
|  4 | C    |   12 |   19 |
|  8 | C1   |   13 |   18 |
| 10 | C1I  |   14 |   15 |
| 11 | C1II |   16 |   17 |
+----+------+------+------+
10 rows in set (0.00 sec)
```

Löschen von ganzen Teilbäumen funktioniert analog. Schritt 3 kann dabei allerdings entfallen (es gibt ja keinen Nachfolger mehr) und bei Schritt 4 muß man nicht 2, sondern entsprechend der entfernten Elemente subtrahieren.

7.4.4.2 Einfügen

Das Einfügen funktioniert im Prinzip analog zum Löschen, man muß die Schritte nur in der umgekehrten Reihenfolge durchlaufen.

7.4.5 Performance vom Nested Set Modell

Wie wir gesehen haben, lassen sich die Basisoperationen relativ einfach ausführen. Aber wie sieht es mit der Geschwindigkeit aus? Bei der Abfrage dürfte es kaum ein schnelleres Modell geben. Eine `SELECT`-Abfrage mit einem Join und der ganze Baum ist ausgelesen. Bei Änderungen sieht es schon schlechter aus: Es müssen alle Datensätze, die unterhalb oder rechts von dem eingefügten/gelöschten Datensatz liegen, verändert werden. Normalerweise sind das im Durchschnitt die Hälfte der Datensätze.

7.4.6 Übungen

7.4.6.1 Vater-Zeiger

Um das Folgende machen zu können, muß erstmal die Tabelle 7.1 angelegt werden.

Nachdem die Tabelle nun existiert, kommen wir zu den häufigsten Abfragen (Lösung im Anhang B.1.1):

- Gib die Wurzel aus
- Wie viele Söhne hat die Wurzel?
- Welches sind die Söhne der Wurzel?
- Gib die Wurzel mit den Söhnen aus
- Gib den kompletten Baum aus, so daß etwa das Folgende herauskommt:

```
+-------+------+
| Tiefe | Name |
+-------+------+
|     0 | Root |
|     1 | A    |
|     2 | A1   |
|     3 | A1I  |
|     1 | B    |
```

  Also immer der Vaterknoten, gefolgt von den Söhnen mit Angabe der Tiefe. Die Abfrage soll allgemein sein, also auch bei größeren Tiefen funktionieren.

7.4.6.2 Nested Set

Etliche Abfragen wurden schon gezeigt, von daher kommt hier noch eine richtig schöne (und damit nicht ganz einfache). Aber zur Einstimmung erstmal was einfaches.

- Nicht immer braucht man den gesamten Baum. Gib alle Knoten, die unterhalb des Knotens „C“ liegen, mit ihrer Tiefe im Baum aus. Das Ergebnis sollte so aussehen:

```
+------+-------+
| Name | Level |
+------+-------+
| C    |     2 |
| C1   |     3 |
| C1I  |     4 |
| C1II |     4 |
+------+-------+
```

- Um einen Baum sauber in HTML ausgeben zu können, braucht man folgende Informationen: Name des Elements, Anzahl der Nachfolger (bzw. gibt es einen oder keinen), die Tiefe des Elements und ob es auf gleicher Tiefe noch ein Element gibt. Oder andersherum ausgedrückt, folgende Tabelle enthält alle notwendigen Informationen. Wie sieht die dazu passende Abfrage aus?

```
+------+------------+-------+--------+
| Name | Nachfolger | Tiefe | Bruder |
+------+------------+-------+--------+
| Root |      10.00 |     1 |      0 |
| A    |       2.00 |     2 |      1 |
| A1   |       1.00 |     3 |      0 |
| A1I  |       0.00 |     4 |      0 |
| B    |       2.00 |     2 |      1 |
| B1   |       0.00 |     3 |      1 |
| B2   |       0.00 |     3 |      0 |
| C    |       3.00 |     2 |      0 |
| C1   |       2.00 |     3 |      0 |
| C1I  |       0.00 |     4 |      1 |
| C1II |       0.00 |     4 |      0 |
+------+------------+-------+--------+
```

7.5 IF-Ausdrücke in SQL

Man kann – wie auch in PHP (siehe 8.2.14) – in MySQL if-Anweisungen benutzen. Gegeben sei folgende Tabelle:

```
mysql> select a,b from if_test1;
+-----+-----+
| a   | b   |
```

```
+-----+-----+
|   6 |   1 |
|   2 |  11 |
|   5 |   5 |
| 233 | 123 |
+-----+-----+
4 rows in set (0.00 sec)
```

Die Fragestellung lautet: Ist a größer oder gleich b?
Die 'traditionelle Methode' wäre jetzt folgende: man zieht sich alle Werte aus der Tabelle und geht dann mittels einer geeigneten Programmiersprache (z. B. PHP) daran, die Werte zu extrahieren – es geht aber auch einfacher: mittels der IF-Anweisung in MySQL kann man der Ergebnistabelle leicht Felder hinzufügen, die die gewünschten Informationen enthalten. Die IF-Anweisung ist folgendermaßen definiert:
`IF(Bedingung, wenn_wahr, wenn_falsch)`
`wenn_wahr` und `wenn_falsch` sind hierbei Zeichenketten, Spaltennamen, Funktionen auf Spalten oder andere IF-Anweisungen (siehe unten). Um die Fragestellung zu beantworten, ist also folgendes IF nötig:
`IF(a>=b,'a größer gleich b','a kleiner b')`.
Dieses IF müssen wir jetzt noch in eine SELECT-Anweisung verpacken, damit wir auch mit dem Ergebnis arbeiten können. Dem IF muß in der SELECT-Anweisung ein Alias (siehe 6.6.5.1) gegeben werden, da sonst der Wert nicht benutzt werden kann.

```
mysql> select a,b,
    -> IF(a>=b,'a groesser gleich b','a kleiner b') as vergleich
    -> from if_test1;
+-----+-----+---------------------+
| a   | b   | vergleich           |
+-----+-----+---------------------+
|   6 |   1 | a groesser gleich b |
|   2 |  11 | a kleiner b         |
|   5 |   5 | a groesser gleich b |
| 233 | 123 | a groesser gleich b |
+-----+-----+---------------------+
4 rows in set (0.00 sec)
```

Eine dreifach geschachtelte Aussage ist so jedoch nicht möglich. Wir können mit dieser Zwei-Wege-Form des IF nicht fragen: „Ist a größer, gleich oder kleiner als b?“. Um diese Frage zu beantworten, brauchen wir eine geschachtelte IF-Anweisung, modellhaft also etwas in dieser Art:

```
Ist a größer b?
  - ja
    --> 'a größer b'
  - nein
    Ist a gleich b?
      - ja
        --> 'a gleich b'
      - nein
        --> 'a kleiner b'
```


Wie oben gesehen ist eine IF-Anweisung aus einer Bedingung, einem Teil 'wenn wahr' und einem Teil 'wenn falsch' aufgebaut. Um das eben gezeigte Modell als IF nachzubauen, brauchen wir 2 geschachtelte IF-Anweisungen. Wir schreiben also (laut Modelldarstellung) in den 'wenn falsch'-Teil des äußeren IF ein neues IF, das die Frage „Ist a gleich b“ stellt.

```
IF(a>b,'a groesser b',
  IF(a=b,'a gleich b', 'a kleiner b')
)
```

MySQL liefert nun das gewünschte Resultat:

```
mysql> select a,b,
    -> IF(a>b,'a groesser b',
    ->   IF(a=b,'a gleich b', 'a kleiner b')
    -> ) as vergleich
    -> from if_test1;
+-----+-----+--------------+
| a   | b   | vergleich    |
+-----+-----+--------------+
|   6 |   1 | a groesser b |
|   2 |  11 | a kleiner b  |
|   5 |   5 | a gleich b   |
| 233 | 123 | a groesser b |
+-----+-----+--------------+
4 rows in set (0.05 sec)
```

7.5.1 Beispiele

7.5.1.1 Firma Ausbeuter & Co KG

Die Firma Ausbeuter & Co KG möchte geschlechtsspezifische Zulagen vergeben, da es ja allgemein bekannt sein dürfte, daß Männer schlechter arbeiten als Frauen[5]. Die Firma hat eine Mitarbeitertabelle, in der folgendes steht:

```
mysql> select * from mitarbeiter;
+--------------+----------+------------+--------+
| MitarbeiterNr | Name     | Geschlecht | Gehalt |
+--------------+----------+------------+--------+
|          351 | Meier    | m          |   2000 |
|          729 | Müller   | m          |   1142 |
|          921 | Guglhupf | w          |   4500 |
|          523 | Schmitt  | w          |   3488 |
|          331 | Reeg     | m          |      2 |
```

[5] Wer will, darf diesen Satz auch umdrehen.

```
+---------------+----------+------------+--------+
5 rows in set (0.00 sec)
```

Die Zulage für Frauen soll 250 EUR betragen, die für Männer 180 EUR. Da nur Mann und Frau unterschieden werden sollen, brauchen wir ein klassisches 2-Wege-IF:

`IF(Geschlecht = ’w’,250,180)`

Eingebunden in das SELECT ergibt sich dieses:

```
mysql> select MitarbeiterNr, Name, Geschlecht, Gehalt,
    -> IF(Geschlecht = 'w', 250, 180) as Zulage
    -> from mitarbeiter;
+---------------+----------+------------+--------+--------+
| MitarbeiterNr | Name     | Geschlecht | Gehalt | Zulage |
+---------------+----------+------------+--------+--------+
|           351 | Meier    | m          |   2000 |    180 |
|           729 | Müller   | m          |   1142 |    180 |
|           921 | Guglhupf | w          |   4500 |    250 |
|           523 | Schmitt  | w          |   3488 |    250 |
|           331 | Reeg     | m          |      2 |    180 |
+---------------+----------+------------+--------+--------+
5 rows in set (0.00 sec)
```

Teil III

Einführung PHP

8 PHP Grundlagen

8.1 Einleitung

> Programming today is a race between software engineers striving to buy bigger and better idiot-proof programs, and the Universe trying to produce bigger and better idiots. So far, the Universe is winning.
>
> Rich Cook

PHP[1] ist eine serverseitige, in HTML[2] eingebettete Scriptsprache - oder mit anderen Worten: PHP-Scripte werden auf dem Server ausgeführt, im Gegensatz z. B. zum üblichen JavaScript und Java[3]. Der Programmcode wird in die HTML-Quelldatei geschrieben und somit i. A. nicht in einer extra „PHP-Datei" abgelegt. Als Script werden Programme bezeichnet, die keine eigenständigen Programme sind, weil sie nicht kompliziert genug sind und andere Programme benötigen, um ausgeführt zu werden.

Im Gegensatz zu HTML und JavaScript erscheint der eigentliche PHP-Code i. A. nicht auf der Clientseite, d. h. der Quellcode, den man sich im Browser auch ansehen kann, enthält für gewöhnlich keinen PHP-Code[4]. Der HTML-Code wird beim Abruf der Webseite, wie bei normalen Seiten auch, 1:1 an den Client geschickt; der PHP-Code wird durch den Server ausgeführt und die Ausgabe dann, zusammen mit den nicht interpretierten HTML-Scriptanteilen, an den Client gesandt. Es ist auch möglich, PHP-Scripte ganz ohne (sichtbare) Ausgabe laufen zu lassen - was durchaus sinnvoll sein kann.

Immer wenn man sich fragt, ob etwas möglich ist, ist zu überlegen, ob dazu eine Aktion auf dem Server (wo die Webseite liegt) oder auf dem Client (wo die Webseite angezeigt wird) notwendig ist. Z. B.: Ist es möglich, mit einem PHP-Befehl die aktuelle Webseite auszudrucken? Die Antwort ist ganz einfach: Damit auf dem Client die Seite ausgedruckt wird, muß dem Browser ein Befehl übermittelt werden. Da PHP aber auf dem Server ausgeführt wird, kann es diesen Befehl folglich nicht selbst in die Tat umsetzen. Auf dem Client wird aber z. B. JavaScript ausgeführt, das einen Befehl anbietet, der die Seite ausdruckt (sofern JavaScript aktiviert ist). Für viele andere Aktionen ist aber nicht einmal JavaScript nötig.

Im vorigen Kapitel wurde SQL beschrieben, jetzt wird PHP erklärt. Dies sind zwei von-

1 PHP Hypertext Preprocessor
2 Hypertext Markup Language
3 Beides kann zwar auch serverseitig laufen, wobei sie dann PHP ersetzen. I. A. wird es aber clientseitig eingesetzt
4 Falls doch PHP-Code beim Client gelandet ist, hat der Server, der Serveradministrator oder der Scriptautor einen Fehler gemacht

einander unabhängige Sprachen, die erst einmal nichts miteinander zu tun haben[5]! Auch wenn es Funktionen in beiden Sprachen gibt, die ähnlich heißen, können sie sich doch deutlich unterscheiden. Im weiteren Verlauf wird gezeigt, wie man die beiden Programmiersprachen zusammenführt. Auch dann muß man sich weiter im Klaren darüber sein, was SQL und was PHP ist.

8.2 Grundbefehle

PHP wird einfach in den HTML-Quellcode geschrieben. Damit der Server weiß, in welcher Datei er nach PHP-Scripten suchen soll, müssen die Dateien die richtige Endung (Extension) haben. Bei PHP3 waren „.php3“ und „.phtml“ üblich, bei PHP4 ist dagegen „.php“ gebräuchlicher. Man kann natürlich den Webserver so konfigurieren, daß er jede beliebige Endung als PHP-Script identifiziert. Damit der Server darüber hinaus noch weiß, welche Ausdrücke er in der Datei interpretieren soll, müssen jeweils der Anfang und das Ende des PHP-Teils gekennzeichnet werden. Dafür gibt es drei Möglichkeiten:

```
1 <? echo "Hello world!"; ?>
2 <?php echo "Hello world!"; ?>
3 <script language="php">
    echo "Hello world!";
  </script>
```

Die erste Möglichkeit ist die kürzeste und damit bei vielen die beliebteste (wer ist nicht gerne faul?). Sie ist allerdings nicht XML[6]-konform, so daß später Probleme auf den Programmierer zukommen können, wenn man sie benutzt. Außerdem gibt es Server, die diese Variante nicht erkennen. Ich benutze immer die zweite Variante; sie ist kurz, aber dennoch XML-konform.

Die Sprache PHP ist hauptsächlich von C, aber auch von Java und Perl (die ihrerseits von C beeinflußt wurden) geprägt. Aber auch für Pascal/Delphi-Programmierer ist die Sprache nicht schwer zu erlernen.
Grundsätzlich gilt (merken!): Eine Anweisung wird immer mit einem ‘;‘ abgeschlossen und eine Funktion bzw. einen Funktionsaufruf erkennt man an runden Klammern ‘()‘.

8.2.1 Der echo-Befehl

Den wichtigsten Befehl haben wir oben schon verwendet: den echo-Befehl, der Strings ausgibt. Im obigen Beispiel wird jeweils „Hello world!“ ausgegeben. Der Text, der ausgegeben werden soll, muß natürlich in Anführungsstrichen stehen, da der Server sonst versucht, ihn als PHP-Befehl zu interpretieren. Dieses Vorgehen wird Quoten oder Quoting[7] genannt. Bei den Anführungsstrichen gibt es zwei verschiedene: einmal das einfache „’“ und das doppelte „"“. Es gibt auch einen Unterschied zwischen den beiden: Bei den doppelten

[5] SQL ist dem Sinn und Zweck nach viel weniger eine Programmiersprache als PHP
[6] Extensible Markup Language, eine HTML nicht unähnliche Ausdruckssprache zum einheitlichen und doch hochflexiblen Speichern von Daten aller Art
[7] von engl. to quote: zitieren

Anführungsstrichen versucht der Server, den Text zu interpretieren; bei den einfachen hingegen behandelt er ihn nicht speziell, sondern gibt ihn z. B. direkt aus. Weitere Erklärungen zu Anführungszeichen finden sich in Kapitel 8.2.6.4.

```
$var = 123;

echo 'Die Variable $var hat den Wert 123!\n';
echo "Die Variable $var hat den Wert 123!\n";
```

Das erste echo gibt „Die Variable $var hat den Wert 123!\n“ aus, das zweite hingegen „Die Variable 123 hat den Wert 123!“ mit folgendem Zeilenumbruch.

```
echo "Say \"Hello World\" my friend";
```

Die Ausgabe bei diesem `echo` ist “Say “Hello World!“ my friend“. Wie man sieht, müssen doppelte Anführungsstriche, die ausgegeben werden sollen, besonders gekennzeichnet werden. Dieses Vorgehen nennt man Escapen[8]. Es ist insbesondere für das Ausgeben von HTML-Quelltext in Verbindung mit `echo` und `print` nötig und kann u. U. zu Problemen führen, wenn man vergißt, in allen Teilstrings zu escapen. Siehe auch Kapitel 13 (Fehlersuche).

8.2.2 Der print-Befehl

Neben dem `echo`- gibt es auch den `print`-Befehl. Im Endeffekt leisten beide dasselbe: Sie geben Text aus. `echo` ist ein internes Sprachkonstrukt, wohingegen `print` ein Ausdruck (Expression) ist. `echo` kann mehrere Argumente haben, die nicht in Klammern stehen dürfen. `print` kann nur genau ein Argument haben.
Alle folgenden Anweisungen sind zulässig und geben dasselbe aus:

```
$var1 = "Hallo";
$var2 = "Welt!";

echo $var1," ",$var2;

echo $var1." ".$var2;
print ($var1." ".$var2);

$res = print ($var1." ".$var2);
```

8.2.3 Zuweisungen

Wenn man der Variablen $a den Wert der Variablen $b zuweisen will, muß man dies mithilfe des Gleichheitszeichens machen. Das bedeutet aber auch, daß man Vergleiche in PHP nicht mit dem einfachen Gleichheitszeichen machen kann; wie man dies erreicht, erfahren wir daher noch später.

[8] engl. to escape: entkommen. Hintergrund: Das escapete Zeichen „entkommt“ der vorgesehenen Interpretierung durch die Sprache, in diesem Fall PHP.

```
$a = $b;
```

8.2.4 Operatoren

Nur irgendwelche Werte in irgendwelche Variablen zu schreiben, wird irgendwann langweilig. Deshalb gibt es auch ein paar Operatoren. Dabei muß man zwischen den arithmetischen (Zahlen), String- (Text), Bit-, logischen (booleschen) und Vergleichs-Operatoren unterscheiden.

8.2.4.1 Arithmetische Operatoren

Beispiel	Name	Ergebnis
$a + $b	Addition	Summe von $a und $b
$a - $b	Subtraktion	Differenz von $a und $b
$a * $b	Multiplikation	Produkt von $a und $b
$a / $b	Division	Quotient von $a und $b
$a % $b	Modulo	Rest der Division von $a und $b

Tabelle 8.1: Arithmetische Operatoren in PHP

Wenn beide Operanden bei der Division vom Typ *integer* (ganzzahliger Wert) sind, ist das Ergebnis ebenfalls *integer*. Ist ein Operand eine Kommazahl, wird das Ergebnis auch eine Kommazahl.

Befehle wie '$a = $a + 5' kann man etwas abkürzen:

```
$a += 5;            // entspricht $a = $a + 5;
$a *= 2;            // entspricht $a = $a * 2;
$i++;               // entspricht $i = $i + 1;
$i--:               // entspricht $i = $i - 1;
```

8.2.4.2 String-Operatoren

Es gibt nur einen echten String-Operator: den Verbindungsoperator ('.').

```
$a = "Hello ";
$b = $a . "World!"; // jetzt ist $b = "Hello World!"
```

Auch hier lassen sich Befehle der Form '$a = $a . "noch etwas Text"' abkürzen:

```
$a = "Hello ";
$a .= "World!";
```

8.2.4.3 Bit-Operatoren

Bitweise Operatoren erlauben es, bestimmte Bits in einer Integervariablen zu setzen.

Beispiel	Name	Ergebnis
$a & $b	UND	Bits, die in $a und $b gesetzt sind, werden gesetzt
$a | $b	ODER	Bits, die in $a oder $b gesetzt sind, werden gesetzt
~$a	NICHT	Bits, die in $a gesetzt sind, werden nicht gesetzt und umgekehrt

Tabelle 8.2: Bit-Operatoren in PHP

8.2.4.4 Logische Operatoren

Logische bzw. boolesche Operatoren werden zum Beispiel zum Verknüpfen von mehreren Vergleichen bei einer Bedingung benötigt. „true“ ist übrigens der Wahrheitswert; dessen Verneinung lautet „false“.

Beispiel	Name	Ergebnis
$a and $b	UND	true, wenn beide ($a und $b) true sind
$a or $b	ODER	true, wenn mind. einer ($a oder $b) true ist
$a xor $b	Exklusiv-ODER	true, wenn genau einer ($a oder $b) true ist
!$a	NICHT	true, wenn $a false ist
$a && $b	UND	true, wenn beide ($a und $b) true sind
$a || $b	ODER	true, wenn mind. einer ($a oder $b) true ist

Tabelle 8.3: Logische Operatoren in PHP

Der Unterschied zwischen den beiden UND und ODER liegt in deren Priorität verglichen mit anderen Operatoren.

8.2.4.5 Kurzschlußlogik

Wichtig ist auch zu wissen, daß logische Verknüpfungen mittels der angegebenen Operatoren immer von links nach rechts evaluiert (ausgewertet) werden; d. h. bei einer UND-Verknüpfung (`&&`) werden alle booleschen Ausdrücke (alles, was zu *true* oder *false* evaluieren kann), die rechts von einem stehen, der zu *false* evaluiert, gar nicht erst abgefragt.

Der Fachbegriff hierfür lautet **Kurzschlußlogik**, weil die Sprache (in diesem Fall PHP) erkennt, daß der boolesche Ausdruck nicht mehr wahr werden kann und deshalb gleich abbricht (nur die Evaluierung, nicht das Skript!).

Das ist wichtig zu wissen, denn so kann man bestimmte Voraussetzungen abfragen, bevor man z. B. eine Funktion aufruft oder eine Abfrage startet, die von einem Parameter abhängt – und das alles z. B. in einem Ausdruck!

Ganz analog verhält es sich bei ODER-Verknüpfungen: Wird hier einer der booleschen Ausdrücke wahr, werden die anderen nicht mehr bearbeitet. Diesen Umstand kann man z. B. ausnutzen, indem man häufiger wahr werdende oder weniger Laufzeit in Anspruch nehmende Ausdrücke weiter vorne (links) hinschreibt.

Zu beachten ist, daß diese Kurzschlußlogik **nicht** bei den bitweisen Operatoren & und | zum Einsatz kommt.

8.2.5 Kommentare

Für Kommentare gibt es zwei Möglichkeiten der Schreibweise:

```
echo "Noch kein Kommentar!";

/* Dies ist ein Kommentar,
der auch ueber mehrere Zeilen gehen kann */

// Dies ist wieder ein Kommentar,
// der jeweils bis zum Ende der Zeile geht

echo "Kein Kommentar mehr!";
```

Die erste und die letzte Zeile sind Befehle, der Rest sind Kommentare.

8.2.6 Variablen

God is real, unless declared integer

Alle Variablen werden durch ein vorangestelltes '$' gekennzeichnet. Die Variablen müssen nicht vorher definiert oder deklariert werden. PHP verwendet den Typ, den es für richtig hält. Der Variablentyp kann auch bei der ersten Benutzung festgelegt werden, indem er davor in Klammern angegeben wird. In Tabelle 8.4 sind die verfügbaren Typen aufgelistet. Neben Variablen gibt es noch Konstanten, die in Kapitel 8.2.7 beschrieben werden.

Tabelle 8.4: Typen in PHP

int, integer	Integer
real, double, float	Double
boolean	Boolean
string	String
array	Array
object	Objekt

8.2.6.1 Integer

Eine Integer-Variable kann (auf 32-Bit-Maschinen) alle ganzen Zahlen im Bereich von -2.147.482.648 bis +2.147.482.647 (entspricht $-2^{31} - 1$ bis $+2^{31}$) als Wert annehmen.

Wird einer Integervariablen ein Wert außerhalb des oben genannten Wertebereichs zugewiesen, erfolgt die Umwandlung in den Typ ‚double'.

Man kann Zahlen nicht nur in dem uns geläufigen Dezimalsystem (Basis: 10) eingeben. Es gibt auch noch das hexadezimale System (Basis: 16) und Oktalsystem (Basis: 8). Damit PHP weiß, was wir meinen, wird bei Hexadezimalzahlen ein „`0x`" und bei Oktalzahlen eine „`0`" vorangestellt. Diese Zahlensysteme werden häufig wegen ihrer stärkeren Hardware-Nähe benutzt. Für uns sind sie im Moment eher weniger interessant.

8.2.6.2 Double/Float

Für reelle Zahlen gibt es den Datentyp ‚Double' bzw. ‚Float'. Der Wertebereich geht (auf 32-Bit-Maschinen) von ca. -1,7E308 bis ca. 1,7E308 (entspricht $-2^{1024} - 1$ bis $2^{1024} - 1$) mit einer Genauigkeit von grob 14 Stellen.

8.2.6.3 Boolean

Mit PHP4 ist auch der Datentyp ‚Boolean' eingeführt worden. PHP3 hat den booleschen Wert „true" als Integer mit dem Wert ‚1' interpretiert.

8.2.6.4 String

In Strings werden Buchstaben-/Zeichenketten gespeichert. Wenn man Strings definiert, muß ihr Inhalt in Anführungszeichen geschrieben werden (siehe auch Kapitel 8.2.1).

Betrachten wir als erstes die doppelten Anführungszeichen ("). Um z. B. innerhalb der Anführungszeichen eines `echo`- oder `print`-Befehls ein Anführungszeichen zu schreiben (so, daß es ausgegeben wird), muß dieses mit einem Backslash (\) versehen werden, weil es sonst den String beenden würde. Buchstaben mit einem vorangestellten Backslash werden als „escaped characters[9]" bezeichnet. Eine Übersicht der escaped characters gibt es in Tabelle 8.5.

\n	line feed: neue Zeile (Abk. LF, ASCII-Code 0x0A)
\r	carriage return (Abk. CR, ASCII-Code 0x0D)
\t	horizontaler Tabulator (Abk. HT, ASCII-Code 0x09)
\\	Backslash
\$	Dollar-Zeichen
\"	doppeltes Anführungszeichen

Tabelle 8.5: escaped characters

Auch bei den einfachen Anführungszeichen gibt es escaped characters: Es können genau zwei verwendet werden, nämlich \' (einfaches Anführungszeichen) und \\ (Backslash).

Ein String ist im Prinzip ein Array aus Zeichen. Dadurch kann man problemlos auf einzelne Zeichen zugreifen. Z. B. wird mit `$string[n]` auf das n-te Zeichen im String zugegriffen. Erfordert eine Funktion als Parameter nun unbedingt ein richtiges Array, muß der String erst konvertiert werden. Die einfachste mir bekannte Möglichkeit ist mit Hilfe einer kleinen for-Schleife (hier in eine Funktion verpackt):

```
/**
 * Die Funktion wandelt einen String in ein Array um
 * und gibt dieses zurück
 *
 * @param    string   Der Text, der umgewandelt werden soll
 * @return   array    Array mit den einzelnen Buchstaben
 *                    des Textes
```

[9] engl. character: Buchstabe (wird häufig auch als char abgekürzt)

```
 */
function str2array($text){
  $ar = array();
  for ($i=0; $i<strlen($text); $i++){
    $ar[] = $text[$i];
  }
  return $ar;
}
```

Die bisher unbekannten Befehle sollen erstmal nicht weiter stören; sie werden weiter unten erklärt. Mehr zu Funktionen im Kapitel 8.7.

8.2.6.5 Beispiel

Ein kleines Beispiel zu den bis hierher beschriebenen Datentypen:
Die Funktion `int floor(float number)` schneidet die Nachkommastellen ab und gibt einen Integer zurück.

```
$a = 1234;           // $a ist ein Integer
$b = (double) $a;    // $b ist ein Double mit dem Wert 1234
$a = 0123;           // Oktalzahl (entspricht 83 dezimal)
$a = 0xbad;          // Hexadezimalzahl
                     // (entspricht 2989 dezimal)
echo floor((0.1+0.7)*10); // Ausgabe ist 7
```

Die ersten Werte sind noch einfach nachzuvollziehen. Beim letzten ist es allerdings schon schwieriger. Eigentlich würde man als Ausgabe doch eher eine „8“ erwarten. Wieso aber eine „7“ ausgegeben wird, kann man in Worte gefaßt einfach Rundungsfehler nennen. Die längere Version: Computer rechnen intern im Dualsystem; bei ganzen Zahlen ergeben sich keine Rundungsfehler beim Wandeln von Dezimalsystem in das Dualsystem. Bei der Darstellung als Fließkommazahl (`float`) kommt es aber gelegentlich schon zu Rundungsfehlern. In diesem Beispiel tritt gerade so einer auf.

Wenn es zu keinen Rundungsfehlern kommen darf, muß man Integer als Datentyp nehmen. In der Regel ist das auch kein großes Problem, zum Beispiel kann man Preise einfach in Euro Cent abspeichern und schon braucht man keine Nachkommastellen mehr.

Das soll uns aber nicht weiter stören. Ich habe dieses Beispiel hier nur zur allgemeinen Verwirrung eingefügt ;-) .

8.2.6.6 Array

Ein Array ist eine n-dimensionale Liste. Was soll das heißen? Nehmen wir folgendes Beispiel:

```
$monat[1] = "Januar";
$monat[2] = "Februar";
$monat[3] = "Maerz";
$monat[4] = "April";
$monat[5] = "Mai";
```



```
$monat[6] = "Juni";
$monat[7] = "Juli";
$monat[8] = "August";
$monat[9] = "September";
$monat[10] = "Oktober";
$monat[11] = "November";
$monat[12] = "Dezember";
```

Oder als Tabelle:

Zeile	Name
1	Januar
2	Februar
3	Maerz
4	April
5	Mai
6	Juni
7	Juli
8	August
9	September
10	Oktober
11	November
12	Dezember

Bei dieser Tabelle spricht man von einer 1-dimensionalen Tabelle, weil eine Koordinate (die Zeilennummer) ausreicht, um jedes Feld eindeutig zu bestimmen. Der Index (=Zeilennummer) wird in eckigen Klammern hinter dem Array-Namen angegeben.

Wenn man neben dem Monatsnamen auch die Anzahl der Tage[10] im jeweiligen Monat abspeichern will, braucht man eine 2-dimensionale Tabelle:

Zeile	Name	Tage
1	Januar	31
2	Februar	28
3	Maerz	31
4	April	30
5	Mai	31
6	Juni	30
7	Juli	31
8	August	31
9	September	30
10	Oktober	31
11	November	30
12	Dezember	31

Und das ganze in PHP:

[10] Schaltjahre werden der Einfachheit einfach mal nicht beachtet

```
$monat[1]["Name"] = "Januar";        $monat[1]["Tage"] = 31;
$monat[2]["Name"] = "Februar";       $monat[2]["Tage"] = 28;
$monat[3]["Name"] = "Maerz";         $monat[3]["Tage"] = 31;
$monat[4]["Name"] = "April";         $monat[4]["Tage"] = 30;
$monat[5]["Name"] = "Mai";           $monat[5]["Tage"] = 31;
$monat[6]["Name"] = "Juni";          $monat[6]["Tage"] = 30;
$monat[7]["Name"] = "Juli";          $monat[7]["Tage"] = 31;
$monat[8]["Name"] = "August";        $monat[8]["Tage"] = 31;
$monat[9]["Name"] = "September";     $monat[9]["Tage"] = 30;
$monat[10]["Name"] = "Oktober";      $monat[10]["Tage"] = 31;
$monat[11]["Name"] = "November";     $monat[11]["Tage"] = 30;
$monat[12]["Name"] = "Dezember";     $monat[12]["Tage"] = 31;
```

In diesem Beispiel sehen wir zwei wichtige Eigenschaften von Arrays in PHP: Zum einen werden bei mehreren Dimensionen die Indizes einzeln in eckigen Klammern hinter dem Array-Namen angegeben. Zum anderen kann man bei PHP, im Gegensatz zu gewöhnlichen Programmiersprachen, für die Indizes beliebige skalare[11] Datentypen verwenden. Dadurch werden sogenannte assoziative Arrays möglich.

Wie kann man sich n-dimensionale Arrays vorstellen? Bei 3 Dimensionen ist es noch relativ einfach: Nimmt man mehrere 2-dimensionale Arrays und legt diese aufeinander, hat man die 3. Dimension. Ab der 4. Dimension wird das schon etwas schwerer. Aber im Zweifelsfall muß man sich die Arrays nicht vorstellen, sondern nur Daten in ihnen speichern.

8.2.7 Konstanten

In PHP gibt es neben Variablen auch Konstanten. Bei Konstanten ist, wie der Name schon sagt, der Wert nicht veränderlich, außerdem sind sie überall abrufbar. Weitere Unterschiede zu Variablen sind, daß Konstanten nur Werte von Primitivtypen[12] annehmen können und überhaupt grundsätzlich von Variablen unterschieden werden müssen: Konstantennamen bestehen nur aus alphanumerischen Zeichen, werden also **nicht** mit einem Dollarzeichen eingeleitet, und sind völlig unabhängig von Variablen gleichen Namens. Um die Unterscheidung deutlich zu machen, nimmt man daher für die Namen von Konstanten häufig nur Großbuchstaben.

Konstanten werden auch grundsätzlich anders definiert, nämlich mittels einer PHP-Funktion namens `define()`. Als ersten Parameter übergibt man dieser Funktion den Namen der zu definierenden Konstante und als zweiten deren Wert.
Folgendes Beispiel sollte das Gesagte verdeutlichen:

```
1 $var = "variabler Wert";
2 define("CONST", "konstanter Wert");
3 $var = CONST;
4 echo $var;
```

[11] Skalare Datentypen sind u. a.: integer, string
[12] Primitivtypen sind z. B. String oder Integer, im Gegensatz z. B. zu Arrays oder Objektreferenzen, dazu später mehr


```
5 $CONST = "Problem?";
```

Der Beispielcode würde zu *keinem* Problem führen, denn `CONST` hat weiterhin den Wert, der der Konstante mittels `define()` zugewiesen wurde; gleichzeitig hat durch die Zuweisung in Zeile 3 auch `$var` den Wert der Konstante angenommen[13].

Durch die letzte Zeile wird einfach eine neue *Variable* angelegt, die denselben Namen hat wie die Konstante. So sollte man jedoch ausdrücklich **nicht** programmieren! Es sollte nicht allzu schwierig sein, einen anderen Namen zu finden ...

Zu einem Fehler würde es erst dann kommen, wenn man in der letzten Zeile das Dollarzeichen entfernte. Dann nämlich würde etwas versucht, was per definitionem nicht erlaubt ist: einer Konstanten einen neuen Wert zuweisen. Da in PHP für die Definition einer Konstanten jedoch eine Funktion benutzt wird, kommt man auch nicht so schnell durcheinander.

8.2.8 Variable Variablen

Eine einerseits sehr vorteilhafte, andererseits aber auch leicht verwirrende Möglichkeit, in PHP zu programmieren, sind variable Variablen. Bei diesem Konzept nutzt man die Eigenschaft des „Wortcharakters" von Variablennamen aus: Der Name einer Variablen ist ja eigentlich auch nur ein String, dem durch das Dollarzeichen eine besondere Bedeutung zugewiesen wird. Somit liegt es gar nicht so fern, diesen Variablennamen selbst als einfachen String aufzufassen. In diesem Fall wäre es doch eine schöne Sache, wenn man diesen String wieder mit Hilfe anderer Variablen zusammenzusetzen... Das dachten sich sicher auch die PHP-Erfinder, denn in unserer schönen Skriptsprache ist das gar kein Problem:

```
$objekt = "Auto";
$$objekt = "Cabrio";     // $Auto = "Cabrio"
${$objekt} = "Cabrio";   // alternative Syntax
```

Natürlich darf man bei diesem Vorgehen nicht gegen die Regeln für die Vergabe von Variablennamen verstoßen, d. h. der String, den man als Namen einer anderen Variable benutzen möchte, darf z. B. nicht mit einer Zahl beginnen, da der Name der neuen Variable dann ungültig wäre.

Wirklich interessant wird es aber erst, wenn man den Wert bestehender Variablen mit festen Strings kombiniert um damit einen neuen Variablennamen zu erzeugen:[14]

```
$objekt = "Auto";
${"mein$objekt"} = "Cabrio";   // $meinAuto = "Cabrio";
```

Innerhalb der geschweiften Klammern macht man hier von dem Umstand Gebrauch, daß Variablen in Strings, die in doppelten Anführungsstrichen stehen, ausgewertet werden. Natürlich kann man auch mit dem Stringoperator `.` arbeiten.

Bei Arrays ist schließlich noch zu beachten, daß `$$array[0]` vielleicht nicht immer das macht, was man erwartet – will man nun den ersten Eintrag aus `$array` als Namen für eine andere Variable benutzen oder nicht vielmehr den ersten Eintrag aus dem Array

[13] Das heißt aber nicht, daß nun die Variable wirklich einen konstanten Wert hätte ... ;-)
[14] Man kann tatsächlich sogar soweit gehen, die Namen von Funktionen bzw. Methoden (in der OOP) mittels Variablen zusammenzusetzen. Das wollen wir aber nicht vertiefen ...

betrachten, dessen Name derselbe ist wie der Wert von `$array`? Diese Problematik läßt sich leicht umschiffen, indem man die alternative Syntax mit den geschweiften Klammern benutzt, d. h. entweder die eckigen Klammern mitsamt der Indexangabe (z. B. `[0]`) *in* die geschweiften Klammern schreiben oder *dahinter*.

8.2.9 Vergleichsoperatoren

Weiter oben (8.1) wurden bereits die gänigen Operatoren für Zuweisungen behandelt. Für das folgende Kapitel, in dem zum ersten Mal Vergleiche vorkommen, benötigt man jedoch zusätzlich Vergleichsoperatoren:

==	Gleich
!=	Ungleich
>	Größer
<	Kleiner
>=	Größer gleich
<=	Kleiner gleich

Tabelle 8.6: Vergleichsoperatoren in PHP

8.2.9.1 Typensichere Vergleiche

PHP ist von sich aus erst einmal eine nicht streng typisierte Sprache. Das bedeutet im Zusammenhang mit Vergleichen, daß Variablen verschiedener Typen i. A. schon dann gleich im Sinne von `==` sind, wenn ihr Inhalt gleich „aussieht“. Zwei Variablen sind demnach z. B. gleich bezüglich `==`, wenn sie dieselbe Zahl als Wert abgespeichert haben – egal, ob einmal als Integer und einmal als String oder beidesmal z. B. als Integer.

So kommt es auch, daß *jeder* String bezüglich `==` gleich der Integerzahl 0 ist, auch der Leerstring „“. Daß das zu unerwünschten Effekten führen kann (Intentionsfehler), zeigt ein Codebeispiel in Kapitel 9.4.1.

Umgehen läßt sich dieses Problem, indem man zusätzlich auf Typgleichheit überprüft. Dazu gibt es in PHP die Operatoren `===` (gleicher Wert und Typ) und `!==` (weder gleicher Wert noch Typ).

8.2.10 IF

Die `IF`-Abfrage ist eines der wichtigsten Elemente der Programmierung; die Syntax ist identisch zu C:

```
if (expr)
  statement
```

‘expr‘ ist Platzhalter für die Bedingung und ‘statement‘ für den Befehl, der bei erfüllter Bedingung ausgeführt werden soll. Als Beispiel:

```
if ($a>$b)
  print "a ist groesser als b";
```


Falls man mehr als einen Befehl hat, der ausgeführt werden soll, so ist auch das möglich. Man muß die Befehle nur in geschweifte Klammern einschließen:

```
if ($a>$b) {
  print "a ist groesser als b";
  $b = $a;
}
```

Man kann natürlich auch nur einzelne Befehle in geschweifte Klammern einschließen (und sei es nur, um einheitlich zu programmieren, oder Fehler beim Erweitern zu vermeiden).

8.2.11 ELSE

Wenn man zwei Anweisungen hat, die abhängig von einer Bedingung alternativ ausgeführt werden sollen, so kann man entweder zwei `IF`-Abfragen mit gegensätzlichen Bedingungen nehmen. . .

```
if ($a>$b)
  print "a ist groesser als b";
if ($a<=$b)
  print "a ist nicht groesser als b";
```

. . . oder aber den `ELSE`-Zweig verwenden:

```
if ($a>$b)
  print "a ist groesser als b";
 else
  print "a ist nicht groesser als b";
```

Analog zu `IF` kann man auch hier mehrere Befehle pro Zweig (IF, ELSE) angegeben; in diesem Fall müssen diese Anweisungen in geschweiften Klammern geschrieben werden. Kombinationen (z. B. geschweifte Klammern bei IF, aber keine bei ELSE) sind möglich.

8.2.12 ELSEIF

ELSEIF ist eine Kombination aus `ELSE` und `IF`. Am Beispiel wird dies hoffentlich deutlich:

```
if ($a > $b) {
    print "a ist groesser als b";
} elseif ($a == $b) {
    print "a ist gleich b";
} else {
    print "a ist kleiner als b";
}
```

8.2.13 Alternative Syntax für IF: IF(): ... ENDIF;

Falls man in dem Fall, daß eine bestimmte Bedingung erfüllt ist, ganze HTML-Blöcke ausgeben will, bietet sich eine alternative Syntax an:

```
<?php if ($a<$b): ?>
<h1>a ist kleiner als b</h1>
<?php endif; ?>
```

Der HTML-Text wird nur dann ausgegeben, wenn die Bedingung „A kleiner B“ erfüllt ist. Es können auch mehrere HTML-Zeilen benutzt werden. Hier machen geschweifte Klammern natürlich keinen Sinn, wie auch im folgenden Fall.

8.2.14 Alternative Syntax für IF: (?:)

Wenn man je nach Bedingung bestimmte Werte haben will, kann man auch die Kurzsyntax verwenden. Allgemein:

`(Bedingung?Rückgabewert wenn true:Rückgabewert wenn false)`

An einem konkreten Beispiel:

```
<?php
  echo ($a < $b?"a ist kleiner als b":"");
?>
```

Es gibt Stellen, wo sich diese Syntax sehr gut eignet (z. B. bei der Funktion `printf`, die in Kapitel 8.8.1 beschrieben wird). Zu häufige Anwendung führt aber oft auch zu sehr schlecht lesbarem Code.

8.2.15 WHILE

`WHILE`-Schleifen sind die einfachsten Schleifen in PHP. Die Grundform der `WHILE`-Schleife ist die folgende:

`WHILE (expr) statement`

Die Bedeutung der `WHILE`-Schleife ist einfach: Solange die Bedingung ‘expr‘ erfüllt ist, wird die Anweisung ‘statement‘ ausgeführt. Falls die Bedingung von Anfang an nicht erfüllt ist, wird die Anweisung überhaupt nicht ausgeführt. Analog zu `IF` müssen mehrere Anweisungen, die zur selben `WHILE`-Schleife gehören, in geschweifte Klammern eingeschlossen werden.

Man kann auch die alternative Syntax nehmen:

`WHILE (expr) : statement ... ENDWHILE;`

Das Ergebnis der folgenden Beispiele ist identisch; beide geben die Zahlen von 1 bis 10 aus.


```
/* Beispiel 1 */
$i=1;
while ($i<=10) {
    print $i++;  // $i wird erst ausgegeben und
                 // dann inkrementiert
                 // Inkrement = Nachfolger
                 //           = um eins erhoeht (bei Zahlen)
}

/* Beispiel 2 */
$i=1;
while ($i<=10):
    print $i;
    $i++;
endwhile;
```

8.2.16 DO ... WHILE

`DO ...WHILE`-Schleifen sind den `WHILE`-Schleifen ähnlich, es werden allerdings erst die Anweisungen ausgeführt und dann wird die Bedingung überprüft. Ein kleines Beispiel:

```
$i=0;
do {
    print $i;
} while ($i>0);    // $i wird genau einmal ausgegeben
```

Für Pascal/Delphi-Kenner:
Die `DO ...WHILE`-Schleife ist vom Prinzip her identisch mit `REPEAT UNTIL`.

8.2.17 FOR

`FOR`-Schleifen[15] sind die kompliziertesten Schleifen in PHP. Die Syntax ist identisch mit der in C:

`FOR (expr1; expr2; expr3) statement`

Der erste Ausdruck ‘expr1‘ wird genau einmal, am Anfang, ausgeführt. Damit initialisiert man in der Regel die Variable.
Der zweite Ausdruck ‘expr2‘ wird am Anfang jedes Schleifendurchlaufs überprüft. Wenn die Bedingung erfüllt ist, wird die Schleife ausgeführt; wenn nicht, wird abgebrochen. Das ist die Laufbedingung.
Am Ende jedes Schleifendurchlaufs wird der dritte Ausdruck ‘expr3‘ ausgeführt.
Jeder Ausdruck kann auch leergelassen werden.

[15] Wer sich über den Namen wundert: das englische „for“ hat neben dem allgemein bekannten „für“ auch noch eine andere, zeitliche Bedeutung und wird dann z. B. mit „für die Dauer von“ übersetzt!

Analog zu `IF` müssen mehrere Anweisungen, die zur selben `FOR`-Schleife gehören, in geschweifte Klammern eingeschlossen werden.

Die `FOR`-Schleife kann im Prinzip auch mit einer `WHILE`-Schleife nachgebildet werden. Allgemein ausgedrückt (mit den oben verwendeten Bezeichnern) sähe das dann so aus:

```
expr1;
while (expr2){
  statement
  expr3;
}
```

Die folgenden Beispiele geben jeweils die Zahlen von 1 bis 10 aus:

```
/* Beispiel 1 */
for ($i=1; $i<=10; $i++) {
    print $i;
}

/* Beispiel 2 */
for ($i=1;;$i++) {
    if ($i > 10) {
      break;
    }
    print $i;
}

/* Beispiel 3 */
$i=1;
for (;;) {
    if ($i>10) {
      break;
    }
    print $i;
    $i++;
}

/* Beispiel 4 */
$i=1;
while ($i<=10){
  print $i;
  $i++;
}
```

Das erste Beispiel ist natürlich das geschickteste. Im vierten Beispiel ist die `FOR`-Schleife mit Hilfe von `WHILE` nachgebildet worden. Wir sehen als erstes die Initialisierung (`$i` wird auf 1 gesetzt). Die `WHILE`-Schleife läuft so lange, wie die Laufbedingung erfüllt ist (`$i` muß kleiner/gleich 10 sein). Dann kommt die eigentliche Anweisung (Ausgabe der

Variablenwerte) und als letztes wird `$i` inkrementiert[16].

Mit `break` wird die aktuelle Schleife verlassen.

8.2.18 SWITCH

Die Funktion der `SWITCH`-Anweisung ist identisch mit der der `CASE OF`-Anweisung in Pascal/Delphi. Die Anweisung ist ähnlich einer Serie von `IF`-Anweisungen mit denselben Ausdrücken. In vielen Situationen will man eine Variable oder einen Ausdruck mit vielen Werten vergleichen und abhängig von den Werten der Variablen unterschiedliche Befehle ausführen. Genau das erreicht eine `SWITCH`-Anweisung.

Hier zwei Beispiele, die dasselbe Ergebnis ausgeben - einmal mit einer Reihe von `IF`-Anweisungen und einmal mit einer `SWITCH`-Anweisung gelöst:

```
/* Beispiel 1 */
if ($i == 0) {
  print "i ist gleich 0";
}
if ($i == 1) {
  print "i ist gleich 1";
}

/* Beispiel 2 */
switch ($i) {
  case 0:
    print "i ist gleich 0";
    break;
  case 1:
    print "i ist gleich 1";
    break;
}
```

Es ist wichtig zu wissen, wie die `SWITCH`-Anweisung arbeitet, um Fehler zu vermeiden. Bei der `SWITCH`-Anweisung wird Zeile für Zeile (wirklich, Anweisung für Anweisung!) abgearbeitet. Am Anfang wird kein Code ausgeführt. Nur dann, wenn eine `CASE`-Anweisung mit einem Wert gefunden wird, der gleich dem Wert des `SWITCH`-Ausdruckes ist, fängt PHP an, die Anweisungen auszuführen. PHP fährt fort, die Anweisungen bis an das Ende des `SWITCH`-Blockes auszuführen oder bis es das erste Mal auf eine `BREAK`-Anweisung stößt. Wenn man keine `BREAK`-Anweisung an das Ende einer `CASE`-Anweisung schreibt, fährt PHP fort, Anweisungen über den folgenden Fall auszuführen. Z. B.:

```
/*Beispiel 3 */
switch ($i) {
  case 0:
    print "i ist gleich 0";
  case 1:
    print "i ist gleich 1";
```

[16] Um eins erhöht

```
}
```

Falls $i gleich 0 sein sollte, würden beide Anweisungen ausgegeben, was in diesem Fall nicht erwünscht wäre.

Dieses Verhalten kann aber auch bewußt genutzt werden, wie man in den folgenden Fällen sieht. Solcher Code wird allerdings sehr schnell unleserlich, weil er sich anders verhält, als man es auf den ersten Blick erwartet. Von daher sollte man das entweder vermeiden oder ein paar Kommentare einfügen.

```
/* Beispiel 4 */
switch($i) {
  case 0:
    print "i ist gleich 0";
    break;
  case 1:
  case 2:
    print "i ist gleich 1 oder 2";
}

/* Beispiel 5 */
switch($i) {
  case 0:
    print "i ist gleich 0";
  case 1:
  case 2:
    print "i ist gleich 0 oder 1 oder 2";
}
```

Ein spezieller Fall ist der default-Fall. Dieser Fall trifft auf alles zu, was nicht von den anderen Fällen abgedeckt wird.

```
/* Beispiel 6 */
switch ($i) {
  case 0:
    print "i ist gleich 0";
    break;
  case 1:
    print "i ist gleich 1";
    break;
  default:
    print "i ist ungleich 0, 1";
}
```

Gibt man hinter einem `break` eine Zahl (int) größer Null an, wird versucht, diese Anzahl an Ebenen hinauf (hinaus) zu springen. Anwendung findet dies bei verschachtelten Schleifen, wie z. B. im folgenden:

```
$i = 0;
```



```
while($i < 10) {
  $i++;
  $j = 0;
  while ($j < 10) {
    $j += 2;
    if ($i > $j)
      break 2;
  }
}
```

Das Beispiel stoppt mit den Werten 3 und 2 in `$i` und `$j`, weil `$j` in der inneren Schleife immer mindestens den Wert 2 hat.

8.2.19 foreach

Wenn man alle Einträge eines Arrays durchgehen will, nimmt man dazu meist eine `for`-Schleife. Handelt es sich um ein assoziatives Array oder spielt der Index keine Rolle bei der Auswertung der Daten, bietet sich eine alternative Schleife an, die sog. `foreach`-Schleife. Eine Gegenüberstellung der beiden zeigt, wie viel einfacher letztere zu handhaben ist:

```
$tage = array("Montag", "Dienstag", "Mittwoch",
  "Donnerstag", "Freitag", "Samstag", "Sonntag");

// mit for
for ($i=0;$i<count($tage);$i++)
  echo $tage[$i];

// mit foreach
foreach ($tage as $tag)
  echo $tag;
```

Obiges Skript gibt nacheinander und ohne Leerzeichen oder Zeilenumbrüche zweimal alle Wochentage aus.

Wie man leicht sieht, weist man bei `foreach` nacheinander die Einträge eines Arrays (erste Variable) einer lokalen Variable zu, die in der Syntax hinter dem Schlüsselwort `as` angegeben wird. Bei jedem Schleifendurchlauf enthält diese lokale Variable **eine Kopie** (!) der Daten des jeweiligen Array-Eintrags; hierbei kann es sich z. B. auch wieder um ein Array handeln (wobei das eigentliche Array dann multidimensional wäre).

Besonders problematisch ist die erwähnte Kopiersemantik im Zusammenhang mit Objektreferenzen, siehe auch 20.2.1.3.

Zu beachten ist außerdem, daß `foreach` ein Array als ersten Eingabe erwartet, sonst gibt es eine Fehlermeldung. Um sicherzustellen, daß die Eingabe immer ein Array ist, kann man die Variable im Zweifelsfall initialisieren (auch auf Funktionen mit ähnlicher Problematik, wie z. B. `in_array`, anwendbar):

```
if (!isset($myArray))
  $myArray = array();
```

```
foreach ($myArray as $key=>$val)
  echo "$key: $val\n";
```

Richtig interessant wird es aber erst mit der erweiterten Syntax, die es erlaubt, auch den Index (auch `key`, engl.: Schlüssel) des Arrays einer lokalen Variable zuzuweisen – das geht nämlich mit einer `for`-Schleife nicht so ohne weiteres:

```
$produkte = array("Auto"=>"22399",
  "Fahrrad"=>"349", "Skates"=>"229");

$guthaben = 1325;
foreach ($produkte as $typ=>$preis) {
  printf("%s: %s EUR\n", $typ,
    number_format($preis, 2, ',', '.'));
  if ($preis<=$guthaben) {
    $guthaben -= $preis;
    $gekauft[$typ] = true;
  }
}
```

Die obige Funktion „kauft“ selbständig so lange Produkte, bis entweder die Liste durchlaufen ist – wie im Beispiel – oder das Guthaben nicht mehr reicht. Wenn man es nur an dieser Stelle verwendet, könnte man `$produkte` auch direkt als Eingabe für `foreach` hinschreiben, sich also die Variable ersparen. Im konkreten Fall liest sich die Ausgabe wie folgt:

```
Auto: 22.399,00 EUR
Fahrrad: 349,00 EUR
Skates: 229,00 EUR
```

Wenn man sich vorstellt, daß Arrays nicht nur statisch wie im Beispiel, sondern auch als Resultat einer Datenbank-Abfrage oder durch sonstige Berechnungen (z. B. POST-/GET-Übergabe) gefüllt werden können, macht der Zugriff auf den Index gleich mehr Sinn; bei komplexeren Arrays wird auch das Programmieren wesentlich übersichtlicher, wenn Namen statt Indizes benutzt werden, um mehrdimensionale Arrays zu strukturieren (vgl. auch `mysql_fetch_array()`: 10.1.6 und `mysql_fetch_row()`: 10.1.7).

8.2.20 continue

Für alle oben beschriebenen Schleifen (do, while, switch, for und foreach) gibt es analog zu `break` die Möglichkeit, die Schleife fortzusetzen, also zum Anbfrageteil der Schleife (den sog. Kopf) zu springen. Beispiel:

```
$i = 10;
while ($i > 0) {
  $i--;
  if (!($i%2))
    continue;
```



```
  echo $i;
}
```

Die obige Schleife gibt genau die ungeraden Zahlen bis 10 aus, in absteigender Reihenfolge.

Optional kann hinter `continue` ein Integerwert angegeben werden, der analog zu `break` die Anzahl der „hinauszugehenden“ Ebenen angibt.

8.3 include

Der Befehl

`include("dateiname");`

fügt an dieser Stelle den Inhalt der Datei 'dateiname' ein. Dadurch ist es möglich, Quellcode, der in mehreren Dateien benötigt wird, zentral zu halten, so daß Änderungen einfacher werden.

Die Datei, die eingefügt wird, wird als HTML-Code interpretiert, deshalb muß, wenn in der Datei nur PHP-Code steht, diese Datei mit `<?php` anfangen und mit `?>` aufhören (bzw. mit anderen PHP-Code-Markierungen, siehe Kapitel 8.2).

Wenn `include()` in Verbindung mit Bedingungen oder Schleifen eingesetzt wird, muß es immer in geschweiften Klammern geschrieben werden.

```
/* So ist es falsch */
if ($Bedingung)
  include("Datei.inc");

/* So ist es richtig */
if ($Bedingung){
  include("Datei.inc");
}
```

8.4 require

Ganz analog zu `include()` funktioniert `require()`. Es wird aber von PHP3 etwas anders behandelt. Seit PHP4 gibt es keinen Unterschied mehr.

Bei PHP3 wird der `require()`-Ausdruck beim ersten Aufruf durch die Datei ersetzt. Wie bei `include()` wird erst einmal aus dem PHP-Modus gesprungen. Es gibt drei wesentliche Unterschiede zu `include()`: Zum einen wird `require()` immer ausgeführt, also auch dann, wenn es eigentlich abhängig von einer IF-Bedingung nicht ausgeführt werden dürfte; zum anderen wird es innerhalb einer Schleife (FOR, WHILE) nur ein einziges Mal ausgeführt - egal, wie oft die Schleife durchlaufen wird (trotzdem wird der Inhalt der eingefügten Datei mehrmals abgearbeitet). Der dritte Unterschied liegt in der Reaktion auf nicht vorhandene Dateien: `include()` gibt nur ein „Warning“ aus und PHP läuft weiter, bei `require()` bricht PHP mit einem „Fatal error:“ ab.

8.5 Beispiele zu include und require

Es gibt im selbem Verzeichnis, in dem das PHP-Script liegt, noch folgende Dateien:
test.txt

```
Dies ist ein einfacher Text.<br>
Die Variable i hat den Wert <?php echo $i; ?>.<br>
```

test1.txt

```
Datei test1.txt
```

test2.txt

```
Datei test2.txt
```

Aber jetzt zu den Beispielen:

```
echo "Erstmal ein einfaches include/require<br>\n";
$i = 5;
include("test.txt");
require("test.txt");
```

Die Ausgabe des Scripts ist, wie zu erwarten, folgende:

```
Erstmal ein einfaches include/require<br>
Dies ist ein einfacher Text.<br>
Die Variable i hat den Wert 5.<br>
Dies ist ein einfacher Text.<br>
Die Variable i hat den Wert 5.<br>
```

Auch in Schleifen können `include()` und `require()` verwendet werden.

```
echo "\nKleine for-Schleife fuer include<br>\n";
for ($i=1;$i<=3;$i++){
  include("test.txt");
}
echo "\nKleine for-Schleife fuer require<br>\n";
for ($i=1;$i<=3;$i++){
  require("test.txt");
}
```

Auch hier ist die Ausgabe, wie zu erwarten:

```
Kleine for-Schleife für include
Dies ist ein einfacher Text.
Die Variable i hat den Wert 1.
Dies ist ein einfacher Text.
Die Variable i hat den Wert 2.
Dies ist ein einfacher Text.
Die Variable i hat den Wert 3.
```



```
Kleine for-Schleife für require
Dies ist ein einfacher Text.
Die Variable i hat den Wert 1.
Dies ist ein einfacher Text.
Die Variable i hat den Wert 2.
Dies ist ein einfacher Text.
Die Variable i hat den Wert 3.
```

Als letztes ein Beispiel, wo man einen Unterschied zwischen `include()` und `require()` sehen kann:

```
echo "\nKleine for-Schleife fuer include<br>\n";
for ($i=1;$i<=3;$i++){
  include("test$i.txt");
}
echo "\nKleine for-Schleife fuer require<br>\n";
for ($i=1;$i<=3;$i++){
  require("test$i.txt");
}
```

Hier gibt es einen Unterschied zwischen PHP3 und PHP4. Die Ausgabe bei PHP3 ist folgende:

```
Kleine for-Schleife für include<br>
Datei test1.txt
Datei test2.txt
<br>
<b>Warning</b>:  Failed opening 'test3.txt' for inclusion in <b>....

Kleine for-Schleife für require<br>
Datei test1.txt
Datei test1.txt
Datei test1.txt
```

Die Ausgabe von PHP4:

```
Kleine for-Schleife für include<br>
Datei test1.txt
Datei test2.txt
<br>
<b>Warning</b>:  Failed opening 'test3.txt' for inclusion (include_....

Kleine for-Schleife für require<br>
Datei test1.txt
Datei test2.txt
<br>
<b>Fatal error</b>:  Failed opening required 'test3.txt' (include_....
```

Die `include()`-Anweisung wird bei PHP3 und PHP4 identisch behandelt. In beiden Fällen wird versucht, die Dateien „test1.txt“„test2.txt“ und „test3.txt“ einzufügen, wobei letzteres mangels vorhandener Datei nicht funktioniert.

Bei `require()` sieht das ganze etwas anders aus. PHP3 verhält sich, wie ich es erwartet habe. Die Datei wird einmal eingefügt und dann wird der Inhalt der Datei mehrmals aufgerufen. Bei PHP4 verhält sich die `require()` wie die `include()` Anweisung.

8.6 include_once, require_once

Seit PHP4 gibt es neben den Funktionen `include()` und `require()` auch noch die Funktionen `include_once()` und `require_once()`. Der Name zeigt schon, wo der Unterschied ist. Bei den `*_once()` Funktionen wird die Datei nur einmal eingefügt, unabhängig davon, wie häufig man versucht, sie einzufügen.

Der Sinn ist einfach: Bei umfangreichen Webseiten gibt es häufig eine Datei, die die zentralen Funktionen enthält. Da diese in den Webseiten benötigt werden, fügt man sie immer am Anfang ein. Soweit kein Problem. Sobald aber mehrere zentrale Funktionsdateien existieren, die sich auch untereinander bedingen, wird es schwierig, weil jede nur einmal eingefügt werden darf.

8.7 Funktionen

Funktionen dienen dem Zusammenfassen mehrerer Befehle zu einem Aufruf. Dadurch werden Programme einfacher lesbar, weil klar ist, wozu ein Befehlsblock dient.

Bei einigen Programmiersprachen findet eine Unterscheidung zwischen Funktionen statt, die einen Wert zurückgeben und solchen, die keinen Wert zurückgeben. Z. B. in Pascal/-Delphi gibt es neben den sog. Funktionen, die einen Wert zurückgeben, sog. Prozeduren, die keinen Wert zurückgeben. PHP macht hier, genau wie C und C++, keinen Unterschied.

Die Syntax lautet wie folgt:

```
function foo($arg_1, $arg_2, ..., $arg_n) {
  echo "Example function.\n";
  return $retval;
}
```

Die Funktion bekommt die Argumente ‘Arg_1‘ bis ‘Arg_n‘ übergeben und gibt den Wert der Variablen ‘retval‘ zurück. Wird kein ‘return‘ in der Funktion benutzt, hat man dasselbe Verhalten wie bei einer Prozedur in Pascal/Delphi. Rückgabewerte müssen (im Gegensatz zu Pascal/Delphi) nicht abgefragt werden.

Ein kleines Beispiel:

```
function my_sqr($num) {   // gibt das Quadrat von $num zurueck
    return $num * $num;
}
echo my_sqr(4);           // gibt 16 aus
my_sqr(4);                // ruft die Funktion auf,
                          // es passiert aber nichts
                          // (der Rueckgabewert wird ignoriert)
```


8.7.1 Variablenparameter

Normalerweise werden in PHP Werteparameter[17] übergeben[18]. Will man jedoch die Änderungen der Parameter in der Funktion auch in der aufrufenden Funktion haben, muß man mit Variablenparametern[19] bzw. Referenzparametern[20] arbeiten[21].

Variablenparameter werden mit einem ‘&‘ im Funktionskopf gekennzeichnet.

Ein kleines Beispiel:

```
function foo1 ($st) {
  $st .= ' und etwas mehr.';
  // gleichbedeutend mit $st = $st.' und etwas mehr.';
}

function foo2 (&$st) {
  $st .= ' und etwas mehr.';
}

$str = 'Dies ist ein String';
echo $str;          //Ausgabe: Dies ist ein String
foo1 ($str);
echo $str;          //Ausgabe: Dies ist ein String
foo2 ($str);
echo $str;          //Ausgabe:
                    //Dies ist ein String und etwas mehr.
```

8.8 Formatierte Textausgabe

8.8.1 printf

`printf()` ist eine ganz spezielle Funktion, denn die Anzahl ihrer Parameter ist variabel. Das zeigt auch die Syntax:

```
int printf (string format [, mixed args...])
```

Auf den ersten Blick mag das sehr verwirren und zur Frage führen, was daran denn einfacher sein soll. Bevor ich das beantworten kann, muß ich aber erst einmal vollständig erklären, was man mit `printf()` machen kann.

Im einfachsten Fall, wenn der zweite Parameterteil entfällt, verhält sich `printf()` genauso wie `print()`. Anders jedoch, wenn mehr als nur ein String übergeben wird: Dann spielt die Funktion ihre Stärken aus. Im ersten Parameter, dem String, können nämlich Platzhalter eingebaut werden, die durch das ersetzt werden, was hinter dem String in

[17] In der Funktion wird mit einer Kopie der Variablen gearbeitet
[18] wird auch als „Call by value“bezeichnet
[19] Es wird mit den Originalvariablen gearbeitet, weil nur die Adresse übergeben wird
[20] Zwei Namen für dasselbe
[21] wird auch als „Call by reference“ genannt

Form weiterer Argumente angegeben wird. Im Normalfall[22] gibt es also genau so viele zusätzliche Argumente, wie Platzhalter in den String eingebaut wurden – der übrigens auch vollständig in einer Variable enthalten sein kann.

Die genannten Platzhalter werden grundsätzlich mit einem Prozentzeichen ‚%' eingeleitet, alles andere wird 1:1 ausgegeben – ausgenommen natürlich Ausgabe-Formatierungszeichen wie `\n`. Will man nun das Prozentzeichen selbst ausgeben, muß man es durch zwei Prozentzeichen ausdrücken[23]. Reines Ersetzen allein ist jedoch noch lange nicht alles, was uns diese Funktion bietet. Vielmehr kann man mithilfe der Platzhalter genau vorgeben, wie der Wert, der an entsprechender Stelle eingefügt wird, formatiert werden soll. Hierbei reichen die Möglichkeiten von der Festlegung auf einen bestimmten Typ wie Zahlen zu verschiedenen Basen bis hin zum Zurechtschneiden, Ausrichten und Auffüllen von Daten.

Die beiden wichtigsten Platzhalter sind übrigens `%s` und `%d`. Während erstgenannter einen beliebigen String als Wert akzeptiert, wird bei letzterem der Wert als Integer interpretiert und als Dezimalzahl ausgegeben. Alle Werte, die kein Integer sind, werden kurzerhand in eine 0 verwandelt. Auf diese Weise kann man also sicher stellen, daß an einer bestimmten Stelle im auszugebenden String auch wirklich eine Zahl steht. Die Tabelle „Platzhalter" listet alle erlaubten Platzhalter auf.

Platzhalter	Behandlung	Darstellung
%b	Integer	Binärzahl
%c	Integer	Zeichen mit entsprechendem ASCII-Code
%d	Integer	Dezimalzahl, ggf. mit Vorzeichen
%u	Integer	Dezimalzahl ohne Vorzeichen
%f	Double	Fließkommazahl
%o	Integer	Oktalzahl
%s	String	String
%x	Integer	Hexadezimalzahl mit Kleinbuchstaben
%X	Integer	Hexadezimalzahl mit Großbuchstaben

Tabelle 8.7: printf: Platzhalter

Zwischen dem einen Platzhalter einleitenden Prozentzeichen und dem Buchstaben, der die Typisierung des Platzhalters bestimmt, können noch weitere Angaben zur Formatierung gemacht werden. Diese sind allesamt optional und müssen, wenn sie angegeben werden, in folgender Reihenfolge auftreten:

1. Das Füllzeichen. Dieses wird benutzt, um den eingefügten Wert auf der rechten Seite bis zur weiter unten angegebenen Länge aufzufüllen. Standardmäßig wird hierbei mit Leerzeichen aufgefüllt, es kann aber auch eine 0 (Null) oder ein beliebig anderes Zeichen angegeben werden, wobei letzteres dann mit einem einfachen Hochkomma (') gekennzeichnet werden muß.

2. Die Ausrichtung. Normalerweise wird rechtsbündig ausgereichtet; stellt man den folgenden Angaben jedoch ein Minuszeichen (-) voran, wird linksbündig formatiert.

[22] Ausnahmen bestätigen die Regel, aber dazu weiter unten mehr
[23] Im Gegensatz zum sonst in PHP üblichen Escapen mithilfe des Backslashes

3. Die Länge. Die Zahl, die man hier angibt, bestimmt die Anzahl der Zeichen, die die formatierte Ausgabe des jeweiligen Wertes mindestens umfassen soll.

4. Die Nachkommastellen. Hier gibt eine Zahl, gefolgt von einem Dezimalpunkt, die Anzahl der Dezimalstellen für Fließkommazahlen an. Ist der gewählte Typ für die Darstellung nicht `double`, so hat diese Angabe keine Bedeutung. Für komplexere Zahlenformatierungen sollte die Funktion `number_format()` (Kapitel 8.8.3) genutzt werden.

Insgesamt ergibt sich somit folgende Syntax für Platzhalter:

`%[Füllzeichen][Ausrichtung][Länge][Nachkommastellen]Typisierung`

Im Allgemeinen braucht man die optionalen Angaben aber recht selten, so daß man dann ja immer noch mal nachlesen kann und es sich nicht wirklich merken muß.

Folgende Beispiele demonstrieren, wie sich das eben gelernte nun tatsächlich einsetzen läßt:

```
$tag = 13;
$monat = 5;
$jahr = 2009;
$format = "%02d.%02d.%04d\n";
printf($format, $tag, $monat, $jahr);
printf($format, 2*$tag, $monat, $jahr+2);
```

Gibt „13.05.2009" und ein anderes wichtiges Datum aus.

```
$betrag1 = 12.95;
$betrag2 = 57.75;
$betrag = $betrag1 + $betrag2;
echo $betrag." ";
printf("%01.2f", $betrag);
```

Gibt „70.7 70.70" aus.

Ist nun eine der Variablen für die Platzhalter im Formatierungs-String von anderen abhängig, müßte normalerweise eine `if`-Abfrage vor dem Funktionsaufruf getätigt werden. Ist diese Abfrage jedoch simpel, so kann man praktischerweise auch die alternative Kurzvariante benutzen, die in Kapitel 8.2.14 vorgestellt wurde. Um klar zu machen, daß es sich um eine solche handelt – und ggf. auch, um Ergänzungen mittels des Punkt-Operators zu erlauben[24] —, sollte man diese in runden Klammern schreiben.

Das folgende Beispiel gibt für die Zahlen von 0 bis 9 jeweils aus, ob sie gerade oder ungerade ist:

```
for ($i=0;$i<10;$i++)
  printf("%d ist %sgerade.\n",
    $i,
```

[24] U. U. kann es ja sein, daß man den Formatierungs-String nicht verändern möchte, z. B. weil er schon in Form einer Variablen vorliegt

```
    ($i%2==0 ? "" : "un")
  );
```

An dieser Stelle möchte ich die Gelegenheit nutzen, darauf hinzuweisen, daß die `printf`-Syntax nur dann als „guter Stil“ bezeichnet werden kann, wenn man sie auch sinnvoll einsetzt. Kommt z. B. eine einzelne Variable am Ende eines Strings vor, kann man sie einfach direkt mit dem Punktoperator anhängen und muß nicht gleich zu `printf` greifen. Ähnlich verhält es sich bei Variablen, die selbst vom Typ String sind und deren Name aus einem einzelnen Wort besteht. Dann kann man nämlich von der Eigenschaft der doppelt gequoteten Strings Gebrauch machen, daß Variablen in ihnen ersetzt werden.

8.8.1.1 Argumente vertauschen/numerieren

Seit PHP-Version 4.06 kann man die Argumente (zur Erinnerung: das sind die zusätzlichen Parameter, deren Werte anstelle der Platzhalter ausgegeben werden) durchnumerieren und entsprechend referenzieren. Dadurch ergibt sich nicht nur die Möglichkeit, Argumente in der Reihenfolge ihres Auftretens bei der Ersetzung zu vertauschen, sondern auch bestimmte Argumente mehrfach zu plazieren, dabei jedoch ggf. verschieden zu formatieren. Eine Referenz auf ein Argument definiert man nun mittels folgender Syntax:

`%[Argumentnummer\$][sonstige Formatierung]Typisierung`

Folgendes Beispiel einer Sprach-angepaßten Datumsausgabe verdeutlicht die Vorteile[25]:

```
// Angenommen, in $lang stehe die Sprache
if ($lang == "de") {
  $text = "Heute ist der %1\$d.%2\$d.%3\$04d.";
}
elseif ($lang == "en") {
  $text = "Today's date is %3\$04d-%2\$02d-%1\$02d.";
}
else {
  // unbekannte Sprache -> wir geben nur die Zahlen aus
  $text = "%3\$04d-%2\$02d-%1\$02d";
}
$tag = 13;
$monat = 5;
$jahr = 2009;
// Ausgabe je nach Sprache:
// de -> Heute ist der 13.5.2009
// en -> Today's date is 2009-05-13
printf($text,$tag,$monat,$jahr);
```

Aber nocheinmal der Hinweis: Gerade durch die Möglichkeit des Numerierens kann die Komplexität des Codes schnell ansteigen (und das in wenigen Zeilen Code) und dadurch, ähnlich wie bei regulären Ausdrücken, die später noch behandelt werden, das Verständnis

[25] Man hätte hier auch an Stelle der ganzen `if` und `elseif` auch eine `switch`-Anweisung nehmen können

erheblich erschwert werden. Zugunsten der Lesbarkeit sollte man also auf allzu verspielte Konstruktionen verzichten und dann, wenn es doch einmal der Übersichtlichkeit dienen sollte, zumindest gut kommentieren.

Ein Beispiel für schlechten Stil wäre es demnach, schon beim Vorkommen eines einzelnen doppelten Arguments mit der Kanone Referenzierung auf den Spatz String zu schießen…

8.8.2 sprintf

Im Prinzip arbeitet `sprintf` genauso wie `printf` und folgt exakt derselben Parametrisierung. Verwendung findet es im Gegensatz zu letzterem jedoch besonders gerne bei Zuweisungen oder als Parameter für andere Funktionsaufrufe – `sprintf` liefert ja den erzeugten String zurück, anstatt ihn direkt auszugeben.

`sprintf()` läßt sich besonders gut im Zusammenhang mit SQL-Abfragen (siehe auch nächstes Kapitel) besonders gut einsetzen, weil man damit den eigentlichen Abfrage-String sauber von den Werten trennen kann, die in PHP-Variablen stecken. Das Prinzip ist immer das gleiche: Der eigentlichen Abfragefunktion `mysql_query()` übergibt man als einzigen Parameter das `sprintf()`-Konstrukt, das den SQL-String zusammenbaut und zurückgibt (der Empfänger ist, bedingt durch die Schachtelung, die Abfragefunktion).
Innerhalb dieses Konstruktes herrscht eine einfache Zweiteilung: der erste Parameter definiert das String-Grundgerüst, also i. A. alle SQL-Befehle und Vergleichsoperatoren; die restlichen Parameter geben die Quellen (Variablen) für die Werte an, die die im String-Grundgerüst einzubauenden Platzhalter ersetzen. Für diese Variablen gilt übrigens, daß man sie mittels `addslashes()` behandeln sollte[26], falls ihre Werte möglicherweise Hochkommata enthalten könnten – diese werden durch die erwähnte Funktion mittels Backslash escaped. Für die Rückwandlung bei der Ausgabe an anderer Stelle steht die Funktion `stripslashes()` zur Verfügung, die mit `htmlentities()` kombiniert gerade bei der HTML-Ausgabe nützlich ist.

Eine schöne Aufgabe zum Gesagten findet sich in Kapitel 10.3.3.

8.8.3 number_format

Mit der Funktion `number_format()` bietet PHP die Möglichkeit, eine Zahl zu formatieren. Insbesondere im Zusammenhang mit Internationalisierung läßt sich diese Funktion nutzen, um Fließkommazahlen, die in PHP standardmäßig bekanntlich entsprechend der amerikanischen Normen[27] formatiert werden, anderen Formaten entsprechend umzuwandeln, z. B. dem deutschen.

Die Syntax ist wie folgt, wobei wahlweise ein, zwei oder vier Parameter angegeben werden können (nicht drei!):

```
string number_format (float number [, int decimals
                      [, string dec_point , string thousands_sep]])
```

Wird nur die zu formatierende Zahl als einziger Parameter übergeben, so wird diese mit einem Komma (,) als Trennzeichen zwischen Tausenden und ohne Nachkommastellen ausgegeben.

[26] Das allerdings nur auf Servern, bei denen PHP dies nicht schon implizit macht!
[27] Komma als Tausender-Trennzeichen und Punkt als Dezimaltrenner

Bei zwei Parametern wird die Zahl mit `decimals` Stellen hinter dem Komma ausgegeben. Auch hier trennt wieder ein Komma jede Tausenderstelle; die Nachkommastellen werden vom ganzzahligen Wert durch einen Punkt getrennt.

Die Ausgabe bei Angabe aller vier Parameter schließlich unterscheidet sich durch die bei nur zweien dadurch, daß sie die Zeichen für den Dezimaltrenner und das Tausender-Trennzeichen (in dieser Reihenfolge) verwendet.

Achtung: Es wird nur das erste Zeichen des Tausender-Trennzeichens für die Ausgabe benutzt!

Das Beispiel verdeutlicht, wofür diese Funktion vornehmlich benutzt werden kann:

```
function Euro ($preis) {
  return sprintf("%s EUR\n",
    number_format($preis, 2, ',', '.')
  );
}
echo Euro(17392.48365); // Ausgabe: 17.392,48 EUR
```

8.9 Guter Stil

> Real programmers don't use comments because the code is obvious. It was hard to write, it should be hard to understand.

Programmieren ist eine Sache. So zu programmieren, daß nicht nur man selbst, sondern auch andere den Code später in möglichst kurzer Zeit verstehen und daraufhin auch erweitern können, eine andere. Man darf auch nicht vergessen, daß es passieren kann, daß man selbst irgendwann mal – ein Jahr später oder so – noch etwas ändern will oder muß. Dabei ist es im Prinzip gar nicht so schwer: Das Ziel läßt sich erreichen, indem man strukturiert, kommentiert und abstrahiert (d. h. einen Codeabschnitt im Kontext betrachtet) sowie auf Wiederverwendbarkeit achtet. Für letzteres hat sich die objektorientierte Denkweise als hilfreich erwiesen und auch in PHP Einzug gehalten (siehe Kapitel 18). Selbst Kommentieren will gelernt sein – hier bietet sich PHPDOC (Kapitel 14) an.

Wenn es jedoch zur Struktur kommt, fragt sich mancher angesichts ellenlanger Codekonstrukte schnell, wie man dem beikommen soll. Da wird geschachtelt, was das Zeug hält, Zeichen werden wild escaped und Parameter derart in Strings eingebettet, daß selbst der Autor des Scripts sich nicht mehr an eine Änderung heranwagen möchte. Viele Autoren scheinen einfach nicht zu wissen, welch überaus hilfreiche und strukturierende Funktionen PHP von Hause aus bietet. Vielleicht tragen ja diese Zeilen dazu bei, daß sich das ändert – ich hoffe es jedenfalls.

Ähnlich wie in der Sprache C, von der PHP bekanntlich stark beeinflußt ist, bietet unsere Scriptsprache die Funktionen `printf()` und `sprintf()`. Während `printf()` von `print()` abgeleitet ist, also letztlich Text direkt ausgibt, dient `sprintf()` dazu, die Ausgabe der

Funktion als Argument (Parameter) einer weiteren Funktion oder einfach als Wert einer Zuweisung zu benutzen[28].

8.9.1 Warn-/Fehlermeldungen anzeigen

Bei Fehlermeldungen unterscheidet PHP verschiedene Stufen, wo die Fehler je nach Schwere eingeordnet sind. So ist z. B. ein Syntax-Fehler schlimmer als der lesende Zugriff auf eine nicht initialisierte Variable[29]. Beim Syntax-Fehler kann der PHP-Interpreter nicht weiter arbeiten (weil er nicht weiß, was er machen soll), bei der nicht initialisierte Variable hingegen nimmt er einfach den Standardwert (und gibt im Normalfall keine Meldung aus).

An dieser Stelle kann jetzt ein Glaubenskrieg anfangen, ob es ein Fehler ist, von einer nicht initialisierten Variable zu lesen, oder nicht. Meiner Meinung nach ist es kein Fehler, aber wenn man es nicht erlaubt, schließt man einige Fehlerquellen.

```
<?php
$variable1 = "Hallo Welt!";

// ganz viel Code

echo $varaible1;
?>
```

Frage: Was wird ausgegeben? Normalerweise nichts. Beim `echo` habe ich mich verschrieben und die Variable `varaible1` ist nicht initialisiert, also leer. Normalerweise beginnt hier die große Suche.

Jetzt schalten wir (fast) alle Fehlermeldungen an.

```
<?php
error_reporting(E_ALL);

$variable1 = "Hallo Welt!";

// ganz viel Code

echo $varaible1;
?>
```

Jetzt bekommen wir eine Warnung:

```
Notice: Undefined variable: varaible1 in [...]/test.php5 on line 8
```

Mit Hilfe von `error_reporting` kannst du PHP sagen, wie viele Fehlermeldungen du bekommen willst. Die genauen Stufen kannst du in der PHP-Dokumentation nachschlagen, ich zeige hier nur ein paar Beispiele:

```
// Fehlermeldungen ganz abschalten
error_reporting(0);
```

[28] Worauf schon das ‚s' hindeutet: Es steht für „silent", zu deutsch: still.
[29] Eine Variable, der noch kein Wert zugewiesen wurde, ist nicht initialisiert

```
// Einfache Laufzeitfehler melden
error_reporting(E_ERROR | E_WARNING | E_PARSE);

// Alle Fehler ausser E_NOTICE melden
// Dies ist die Standardeinstellung in php.ini
error_reporting(E_ALL ^ E_NOTICE);

// Alle PHP-Fehler melden
// IMHO zum Entwickeln empfohlen
error_reporting(E_ALL);

// Seit PHP5 gibt es noch eine Steigerung
error_reporting(E_ALL | E_STRICT);
```

Als Parameter nimmt man einfach die gewünschten Fehler und verknüpft sie mit einem bitweisen ODER.

8.10 Rekursion

Um Rekursion verstehen zu können, muß man erst Rekursion verstanden haben.

Rekursion ist ein beliebtes Mittel der Programmierpraxis, um Probleme zu lösen, die an mindestens einer Stelle zu einem weiteren Problem führen, das aber in Wirklichkeit nichts anderes ist als das ursprüngliche Problem, ggf. mit leicht veränderten Voraussetzungen. Das klingt kompliziert, ist im Prinzip aber ganz einfach – wie das folgende beliebte Beispiel zeigt.

In der Mathematik gibt es den Begriff der Fakultät. Die Fakultät einer natürlichen Zahl ist definiert als das Produkt aller Zahlen von 1 bis zu dieser Zahl, also z. B. $5! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 = 120$. Mittels Rekursion kann man das wie folgt ausdrücken[30]: $1! = 1, n!_{n \neq 1} = n \cdot (n-1)!$, d. h. die Fakultät von 1 wird definiert als 1 und für $n \neq 1$ ist die Fakultät von n gleich n mal der Fakultät von $n-1$. Um das Ergebnis zu berechnen, muß man also erst die Fakultät von $n-1$ berechnen. Diese ist aber definiert als $n-1$ mal die Fakultät von $(n-1)-1 = n-2$ usw. Irgendwann wird man dahin kommen, die Fakultät von $n-x$ zu berechnen, wobei dieser Ausdruck den Wert 1 haben wird. Dann ist es mit der Rekursion vorbei und die erste Definition greift: Die Fakultät von 1 ist gleich 1. An dieser Stelle hat man also ein erstes Teilergebnis, das mit all den „gemerkten“ Faktoren (der erste war n) multipliziert werden muß – man geht also den ganzen „Weg“ wieder zurück und „sammelt“ dabei das auf, was man „liegen gelassen“ hat. Am Ende hat man auf diese Weise die eigentliche Fakultät berechnet. Das folgende kleine Beispiel sollte das Prinzip verdeutlichen:

$$
\begin{aligned}
1! &= 1 \\
2! &= 2 \cdot 1 && = 2 \cdot (1) && = 2 \cdot 1! \\
3! &= 3 \cdot 2 \cdot 1 && = 3 \cdot (2 \cdot 1) && = 3 \cdot 2! \\
4! &= 4 \cdot 3 \cdot 2 \cdot 1 && = 4 \cdot (3 \cdot 2 \cdot 1) && = 4 \cdot 3! \\
5! &= 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 && = 5 \cdot (4 \cdot 3 \cdot 2 \cdot 1) && = 5 \cdot 4! \\
6! &= 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 && = 6 \cdot (5 \cdot 4 \cdot 3 \cdot 2 \cdot 1) && = 6 \cdot 5!
\end{aligned}
$$

Die Frage, welchen Vorteil diese Methode gegenüber dem einfachen Zählen und Multiplizieren hat, läßt sich an diesem Beispiel noch nicht klären; mehr dazu später. Doch nun zu *dem* Paradebeispiel für Rekursion schlechthin:

8.10.1 Die Türme von Hanoi

Man stelle sich ein fernöstliches Kloster vor. Die dortigen Mönche wollen nun viele Kreisscheiben verschiedenen Durchmessers von einem Stapel auf einen anderen befördern. [31]

[30] Da auch definiert wurde, das $0! = 1$, kann man die Rekursion auch bis $n = 0$ laufen lassen.
[31] Von der Vorgeschichte scheint es Dutzende von Abwandlungen zu geben. Letztlich ist sie aber eh nur erfunden …

Erschwert wird das Ganze dadurch, daß immer nur eine Scheibe gleichzeitig bewegt werden und außerdem nie eine größere auf einer kleineren Scheibe liegen darf. Der Legende nach geht die Welt unter, wenn die Mönche ihr Werk vollendet haben ...

Schnell wird klar, daß diese Aufgabe mit nur zwei Stapeln nicht lösbar ist: Ein dritter Stapel muß her. Mit diesen Vorgaben ist es nun möglich, die Aufgabe theoretisch für jede beliebige Anzahl von Scheiben zu lösen. Der zugehörige Algorithmus (das Problem wurde von einem Mathematiker definiert ...) lautet nun in Pseudo-Syntax:

```
Hanoi(n, Start, Temp, Ziel) {
  Wenn n=0, dann Ende
  Hanoi(n-1, Start, Ziel, Temp)
  Bewege von Start nach Ziel
  Hanoi(n-1, Temp, Start, Ziel)
}
```

Oder als normaler Text: Um einen Turm der Höhe `n` vom `Start`-Stapel auf den `Ziel`-Stapel zu bewegen, bewegt man erstmal einen Turm der Höhe `n-1` (also alles bis auf die letzte Scheibe) auf einen `Temp`-Stapel, legt dann die letzte Scheibe auf den `Ziel`-Stapel und schichten danach den Turm vom `Temp`-Stapel auch noch auf den `Ziel`-Stapel.

Abgesehen davon, daß so eine Scheibe bei entsprechender Höhe des Stapels doch recht schwer werden dürfte (nach unten hin wird's ja immer größer!) gibt es noch ein Problem ganz anderer Natur: Das Umschichten der Stapel kostet Zeit. Gehen wir einmal gemeinsam den Algorithmus durch für drei Scheiben (vgl. Bild 8.1):

1. Bewege die kleine Scheibe von Stapel 1 nach Stapel 3.
2. Bewege die mittlere Scheibe von Stapel 1 nach Stapel 2.
3. Bewege die kleine Scheibe von Stapel 3 nach Stapel 2.
4. Bewege die große Scheibe von Stapel 1 nach Stapel 3.
5. Bewege die kleine Scheibe von Stapel 2 nach Stapel 1.
6. Bewege die mittlere Scheibe von Stapel 2 nach Stapel 3.
7. Bewege die kleine Scheibe von Stapel 1 nach Stapel 3.

Man könnte meinen, daß das doch noch akzeptabel wäre. Das Problem wird aber schnell deutlich, wenn man die Fälle vier, fünf oder sechs Scheiben durchspielt. Dabei ergibt sich nämlich ein Aufwand von 15, 31 und 63 Schritten. Wer sich jetzt an eine gewisses Schachbrett[32] erinnert, der liegt richtig: Der Aufwand wächst exponentiell mit der Anzahl der Scheiben. Genauer gesagt sind $2^n - 1$ Schritte notwendig, um n Scheiben vom Start-

[32] Der Legende nach versprach der König dem Erfinder des Schachspiels ihm einen Wunsch zu erfüllen. Dieser dachte lange nach und wünschte sich dann: Er wollte Reis auf ein Schachbrett. Und zwar auf das erste Feld 1 Korn, auf das zweite Feld 2 Körner, auf das dritte Feld 4 Körner, auf das vierte Feld 8 Körner usw. bis zum 64. Feld. Der König lachte und versprach dem Erfinder den Wunsch zu erfüllen. Trotz aller Anstrengungen war es ihm aber unmöglich. Der König hätte mehr als die 800fache Jahresproduktion an Reis von 1994 gebraucht.

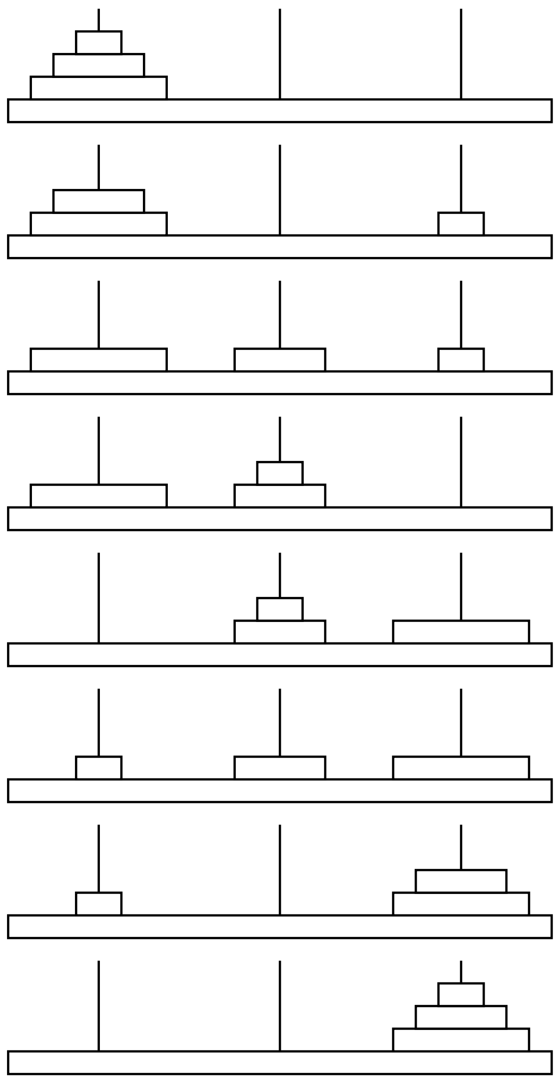

Abbildung 8.1: Umschichten der Türme von Hanoi

auf den Zielstapel zu bewegen. Schon bei nur 20 Scheiben sind $2^{20} - 1 = 1048575$ Schritte notwendig! Die Mönche wären also für hundert Scheiben ihr Leben lang und darüber hinaus beschäftigt ...

Doch was mag das nun mit Rekursion zu tun haben? Nun, wie man sich denken kann, machen die Mönche im Prinzip immer wieder dasselbe: Sie bewegen Scheiben von einem Stapel zum anderen. Das einzige, was sich dabei ändert, ist die jeweilige Ausgangssituation, d. h. es liegen jedesmal weniger Scheiben auf dem Startstapel und mehr auf dem Zielstapel. Sieht man sich nun einmal den Algorithmus näher an, sieht man, daß die Funktion sich selbst aufruft. Das mag auf den ersten Blick komisch aussehen, aber wenn man sicherstellt, daß die Funktion sich irgendwann nicht mehr aufruft, sondern statt dessen beendet, wird das Prinzip dahinter klar: Rekursion bedeutet, daß eine Funktion sich selbst eine bestimmte, zuvor meist nicht bekannte Anzahl mal selbst aufruft und nach etlichen Aufrufen irgendwann beendet. Damit ist dann zwar diese Funktion beendet, aber diese wurde ja von einer anderen Funktion aufgerufen, die nun an der Stelle nach dem Funktionsaufruf fortgesetzt wird. Das Besondere ist nun, daß beide Funktionen eigentlich gleich sind. Eigentlich deshalb, weil sie sich doch unterscheiden, und zwar nur in den Werten ihrer lokalen, selbstdefinierten Variablen und ggf. in der jeweiligen Stelle, wo die jeweilige Funktion fortgesetzt wird, wenn die von ihr aufgerufene Funktion sich beendet. Anstatt „jeweilige Funktion" sagt man übrigens *Rekursionsstufe*, um zu verdeutlichen, daß es sich um einen wiederholten Aufruf derselben Funktion mit bestimmten Eingabedaten handelt.

Damit sind wir auch schon, ähnlich wie bei Schleifen, bei den beiden zentralen Dingen, die ein Programmierer sicherstellen muß, wenn er Rekursion benutzt: Durch die Anwendung der Rekursion muß sich die Situation derart ändern, daß in endlicher Zeit (d. h. irgendwann einmal, in der Praxis natürlich je früher desto besser) der Algorithmus für jede Aufgabenstellung terminiert, d. h. sinnvoll beendet wird – man kann auch sagen, die Abbruchbedingung muß irgendwann erfüllt werden. Das Beispiel der Türme von Hanoi[33] zeigt eindrucksvoll, daß rekursiv definierte Funktionen ein z. T. sehr hohes Wachstum auch schon für niedrige Eingabewerte haben.

8.10.2 Speicherverbrauch und Stack

Für den Computer bedeutet eine große Zahl von Rekursionsstufen, d. h. wie tief die Funktion verschachtelt aufgerufen wird, aber immer auch einen hohen Speicherverbrauch. Irgendwo muß sich der Computer schließlich auch merken, an welcher Stelle im Programm er weitermachen muß, wenn eine Rekursionsstufe durchlaufen wurde und damit verlassen wird. Insbesondere auch die Daten jeder Rekursionsstufe („lokale" Variablen) kosten viel Speicherplatz: Das Einführen einer zusätzlichen Variable geht dabei mit einer Vervielfachung des Speicherbedarfs für die Rekursion einher.

Die genannten Informationen werden intern auf dem sog. Stack verwaltet, auch Stapel genannt. Von diesem kann man immer nur den obersten Eintrag sehen, der dem letzten hinzugefügten Eintrag entspricht (das sog. LIFO[34]-Prinzip). Die Größe des Stacks ist außerdem beschränkt, da er sich ja den Hauptspeicher mit dem Programm selbst sowie seinen Daten teilen muß[35], im Gegensatz zu diesen Speicherbereichen aber i. A. vom Speicherende

[33] Und mehr noch die Ackermann-Funktion, s. u.
[34] Last in first out
[35] Dies ist Teil der sog. *von Neumann*-Architektur

her wächst.

8.10.3 Vorteile der Rekursion

Rekursion hat aber nicht nur Nachteile. Ein großer Vorteil dieser Methode ist, daß man in jeder Rekursionsstufe wieder neue „lokale" Variablen hat, d. h. außer dem Rückgabewert können alle Variablen nach Lust und Laune verändert werden, ohne daß das Auswirkungen auf die anderen, noch abzuarbeitenden Rekursionsstufen hätte[36]. Verwendet man beispielsweise eine Variable n, so kann diese in jeder Rekursionsstufe einen anderen Wert haben und natürlich von den Parametern des Funktionsaufrufs abhängen, die ja innerhalb der Rekursion meist selbst aus den Werten von lokalen Variablen und Parametern des vorigen Aufrufs berechnet werden.

8.10.4 Rekursion und Iteration im Vergleich

Man könnte meinen, es gäbe viele Problemstellungen, die man nur mit Rekursion lösen kann, nicht aber mit Iteration. In Wahrheit sind beide Formen der Programmierung aber annähernd gleich mächtig: Nimmt man zur klassischen Iteration nämlich noch die Möglichkeit, einen Stack (Stapel, Keller, s. o.) zu benutzen, hinzu, so kann man viele mittels Rekursion implementierte Funktionen auch iterativ programmieren. Lediglich mehrfach rekursive Funktionen wie die Ackermann-Funktion bilden hier die Ausnahme – im Alltag trifft man diese aber wohl nicht allzu häufig an...

Nur der Vollständigkeit halber gebe ich hier doch nochmal die genannte Ackermann-Funktion an:

$$ack(n,m) = \begin{cases} \text{m + 1} & ,\ \text{falls n = 0} \\ \text{ack(n - 1, 1)} & ,\ \text{n > 0, m = 0} \\ \text{ack(n - 1, ack(n, m - 1))} & \text{sonst} \end{cases}$$

Die Formel liest sich dabei wie folgt: Der Ackermann-Wert zweier natürlicher Zahlen n und m ist $m+1$, falls n Null ist. Ist dagegen n größer Null und gleichzeitig m gleich Null, so berechnet sich das Ergebnis rekursiv mit $n' = n-1$ und $m' = 1$. In jedem anderen Fall, also letztlich nur wenn n und m größer Null sind, muß zuerst die Rekursion mit $n' = n$ und $m' = m-1$ berechnet werden und das Ergebnis als m'' für die Rekursion mit $n'' = n-1$ eingesetzt werden.

Beispiel: $ack(4,3)$.

1. $ack(4,3) = ack(3, ack(4,2))$
2. $ack(4,2) = ack(3, ack(4,1))$
3. $ack(4,1) = ack(3, ack(4,0))$
4. $ack(4,0) = ack(3,1)$
5. $ack(3,1) = ack(2, ack(3,0))$
6. $ack(3,0) = ack(2,1)$
7. $ack(2,1) = ack(1, ack(2,0))$

[36] Davon ausgenommen sind natürlich Klassenvariablen und Referenzen auf Objekte in der OOP

8. $ack(2,0) = ack(1,1)$
9. $ack(1,1) = ack(0, ack(1,0))$
10. $ack(1,0) = ack(0,1)$
11. $ack(0,1) = 1 + 1 = 2 = ack(1,0)$
12. $ack(0,2) = 2 + 1 = 3 = ack(1,1) = ack(2,0)$
13. $ack(1,3) = ack(0, ack(1,2))$
14. $ack(1,2) = ack(0, ack(1,1))$
15. ...

8.10.5 Ein Beispiel aus der Praxis: MIME parsen

Bei solch an sich so theoretischen Dingen wie Rekursion fragt sich vielleicht mancher, was man damit in der Praxis – hier meine ich damit das Programmieren in PHP – anfangen kann. An dieser Stelle möchte ich deshalb die Gelegenheit nutzen, einen solchen Praxisbezug aufzuweisen.

Angenommen, du möchtest ein Webmail-System bauen. Die PHP-IMAP-Funktionen bieten schon recht viel dessen, was man für das Backend braucht, aber an einer zentralen Stelle muß man doch selbst etwas konstruieren: Dann nämlich, wenn es darum geht, E-Mails zu entschlüsseln. Da eine E-Mail in Wirklichkeit nichts anderes als ein speziell gegliederter ASCII-Text ist, kann man das Erkennen von Attachments (Mail-Anhängen) oder anderen speziellen Teilen, wie z. B. dem HTML-Anteil einer Mail, auf das Erkennen von bestimmten Strings zurückführen. Grundsätzlich enthält eine „moderne“ Mail oft mehrere Teile, MIME[37]-Parts genannt. Diese Parts sind in einem Baum (siehe auch Kapitel 7.4) organisiert, der sich beliebig verzweigen kann. Durch diesen Aufbau ist es z. B. möglich, eine E-Mail mit Attachments weiterzuleiten, dabei neue Attachments anzuhängen und trotzdem die Ursprungsmail mit ihren Attachments theoretisch problemlos extrahieren zu können. Wenn du diesen Satz in seiner ganzen Komplexität genau gelesen hast, hast du es vielleicht schon gemerkt: Sowohl das Anlegen wie auch das Auslesen bzw. Interpretieren eines solchen Baumes läßt sich am besten rekursiv lösen. Ein iterativer Ansatz ist bestimmt auch möglich, aber sicher sehr viel weniger intuitiv, als einfach Rekursion zu benutzen.

Benutzt man nun tatsächlich die PHP-IMAP-Funktionen, dann wird man schnell feststellen, daß die für das Extrahieren bestimmter Parts nötige Funktion `imap_fetchbody()` einen String als Parameter erwartet, der einen Part identifiziert. Dieser String setzt sich aus durch Punkte getrennten Zahlen zusammen und muß leider – wie oben schon angedeutet – selbst zusammengezimmert werden. Hierzu bietet es sich an, die Funktion `imap_fetchstructure()` zu benutzen, um die Struktur für die jeweilige E-Mail zu bekommen (in diesem Fall in Form eines verschachtelten Objekts).

Der eigentlichen, rekursiv definierten, neu zu schreibenden Funktion übergibt man nun das Parts-Objekt, das `imap_fetchstructure()` als Unterobjekt der Hauptstruktur zurückliefert. Die rekursive Funktion geht nun durch den kompletten Parts-Baum und stellt

[37] Multi Purpose Internet Mail Extensions, Mehrzweck-Internet-E-Mailerweiterungen

für jedes Blatt den Part-Typ fest. Im Falle eines Attachments wird einfach dessen Struktur analysiert und die relevanten Daten wie Name und Kodierung vermerkt. Handelt es sich jedoch um einen anzuzeigenden Text-Part, wird dieser an den String angehängt, der später als Nachrichtentext ausgegeben wird. Zur Extraktion des Textes aus einem solchen Part muß wie gesagt der spezielle Identifizierungsstring bekannt sein. Diesen String erhält man wie folgt:

1. Beim erstmaligen Aufruf der Funktion ist der ID-String im Array `$data` noch nicht gesetzt, d. h. leer.
2. Innerhalb einer Rekursionsstufe setzt sich der lokale ID-String aus dem ID-String im Array `$data` sowie der Nummer des aktuellen Parts zusammen, getrennt durch einen Punkt, außer, der ID-String im Array ist leer (erste Rekursionsstufe). Die gesuchte Part-Nummer ist dabei immer um eins größer als der Array-Zähler.
3. Vor dem Starten einer neuen Rekursionsstufe wird der lokale ID-String statt des ID-Strings, der sich im Array `$data` befindet, übergeben.
4. Bei der Rückkehr aus jeder Rekursionsstufe ist der ID-String im Array `$data` unverändert.

Der folgend dargestellte Aufbau einer komplexen Mail sollte das verdeutlichen (übersetzte Version von *http://www.ietf.org/rfc/rfc2060.txt*):

```
HEADER     ([RFC-822] Header der Nachricht)
TEXT       MULTIPART/MIXED
1          TEXT/PLAIN
2          APPLICATION/OCTET-STREAM
3          MESSAGE/RFC822
3.HEADER   ([RFC-822] Header der Nachricht)
3.TEXT     ([RFC-822] Textkörper der Nachricht)
3.1        TEXT/PLAIN
3.2        APPLICATION/OCTET-STREAM
4          MULTIPART/MIXED
4.1        IMAGE/GIF
4.1.MIME   ([MIME-IMB] Header des IMAGE/GIF)
4.2        MESSAGE/RFC822
4.2.HEADER ([RFC-822] Header der Nachricht)
4.2.TEXT   ([RFC-822] Textkörper der Nachricht)
4.2.1      TEXT/PLAIN
4.2.2      MULTIPART/ALTERNATIVE
4.2.2.1    TEXT/PLAIN
4.2.2.2    TEXT/RICHTEXT
```

Solch komplexe Mails können z. B. dann entstehen, wenn man eine Nachricht, die bereits Attachments enthält, komplett eingebettet weiterleitet und dann noch eigene Attachments anhängt. Außerdem werden auch HTML-Mails intern mit MIME-Parts realisiert, da sie neben dem HTML-Teil meist auch einen reinen ASCII-Textteil enthalten.

Das folgende Code-Beispiel zeigt den Rahmen einer entsprechenden Implementierung innerhalb einer geeigneten Klasse (siehe Kapitel 20), die bereits einen Mailbox-Stream und die Message-ID bereitstellt. Die Funktion `imap_fetchbody` extrahiert hierbei den Nachrichtenkörper, also den eigentlichen Text. Sie erwartet als Parameter einen geöffneten Mailbox-Stream, eine Message-ID, eine Part-ID sowie (optional) ein Flag zum Benutzen von eindeutigen Message-IDs. Auf ihren Rückgabewert werden dann noch einmal einige Funktionen angewendet, um Leerzeichen vorne und hinten sowie Escapezeichen zu entfernen sowie Sonderzeichen (Umlaute, ...) zu konvertieren. Die Funktion `imap_fetchstructure` schließlich liefert ein Objekt zurück, das Aufschluß über die Struktur der E-Mail gibt und u. a. alle MIME-Parts in verschachtelten Arrays enthält.

```
function parseParts($parts, $data) {
  // Schleife über alle Parts der aktuellen Ebene
  for ($i=0;$i<count($parts);$i++) {
    // ID-String: <ID-String>.<Partnummer>
    $idstr = sprintf("%s%s%d", $data["idstr"],
                     ($data["idstr"]!='' ? '.' : ''), $i+1);
    if (is_array($parts[$i]->parts)) {
      // Rekursion, falls aktueller Part Unterparts enthält
      $tmp = $data["idstr"];
      $data["idstr"] = $idstr;
      $data = $this->parseParts($parts[$i]->parts, $data);
      $data["idstr"] = $tmp;
    }
    ... // Code für Attachment-Erkennung
    if (!$is_attachment) {
      // kein Attachment -> Body
      $msgbody = htmlspecialchars(stripslashes(trim(
        imap_fetchbody($this->mbox, $this->msgid,
          $idstr, FT_UID))));
      ... // weitere Behandlung
    }
  }
  return $data;
}

$structure = imap_fetchstructure($this->mbox,
  $this->msgid, FT_UID);
$data = $this->parseParts($structure->parts, $data);
... // Daten-Auswertung
```

In diesem Beispiel ist die Abbruchbedingung in Form der `return`-Anweisung noch recht einfach erkennbar. Wichtig ist dabei weniger, daß es eine Anweisung zur Beendigung von Rekursionsstufen gibt, sondern vielmehr, daß eine solche Anweisung im Verlauf der Rekursion immer in endlicher Zeit erreicht werden muß. Andernfalls gibt es schnell eine sich unendlich weiterverschachtelnde Rekursion: das Pendant zur Endlosschleife. Da bei einer

Rekursion rechnerintern die Rücksprungadressen immer auf dem Stack gespeichert werden (s. o.) und dieser irgendwann den normalen Speicher überschreiben würde, gibt es dann mehr oder weniger schnell den gefürchteten *stack overflow*, d. h. der Rechner riegelt ab, bevor schlimmeres passiert. Da PHP nur eine Scriptsprache ist, wird aber schlimmstenfalls nur das Script unsanft beendet. Vermeiden sollte man sowas aber trotzdem auf jeden Fall!

9 PHP & HTML

9.1 Formulare

Mit PHP können Formulardaten relativ einfach eingelesen werden. Man muß im Formular als **action** den Dateinamen der das Formular auswertenden PHP-Datei angeben und als **method** empfiehlt sich häufig 'post'. Bei 'get' kann man sonst die Werte der Eingabe in der URL wiederfinden[1].

In bestimmten Fällen macht es auch Sinn, *kein* `action` anzugeben, nämlich dann, wenn das Zielscript dasselbe wie das aktuelle ist. Zu beachten ist dabei, daß in diesem Fall **alle GET-Parameter** des Scriptaufrufs, der zum Anzeigen des Formulars führte, implizit wieder übergeben werden – wenn das Formular also mit GET arbeitet, hat man nach dem Absenden sowohl die ursprünglichen als auch die neuen GET-Parameter!

Die Felder eines verschickten Formulares[2] werden beim Laden der Empfängerseite (angegeben im ‚action' des Formulars) automatisch in Variablen gleichen Namens verwandelt, auf die man im Verlauf des Script-Hauptteils direkt zugreifen kann.[3] Will man auf solche Variablen auch in Funktionen zugreifen, ohne sie global definieren zu müssen, kann man ab PHP 4.1 die „superglobalen" Arrays `$_GET` und `$_POST`, je nach Übergabeart, verwenden. In älteren PHP-Versionen heißen diese Arrays `$HTTP_GET_VARS` bzw. `$HTTP_POST_VARS` und müssen außerhalb des Script-Hauptteils explizit als global deklariert werden (`global $var;`).

Nun ein kleines Beispiel. Die Eingaben, die auf der Seite 'eingabe.html' eingetragen wurden, werden durch Drücken von OK an die Datei 'ausgabe.php' übermittelt. Durch diese werden sie dann ausgegeben.
Quelltext der Datei 'eingabe.html':

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<html>
<head>
 <title>Eingabe</title>
</head>
<body>
<div align="center">
<form action="ausgabe.php" method="post">
  Feld1: <input name="feld1" size="60" maxlength="60"><br>
  Feld2: <input name="feld2" size="60" maxlength="60"><br>
  <input type="submit" value="OK">
```

[1] Im sog. URL-encoded Format, s. u.
[2] Gilt allgemein für alle POST/GET-Übergaben!
[3] Das Gesagte gilt nur, wenn in der PHP-Konfiguration `register_globals` aktiviert wurde, was standardmäßig bis Version 4.1.X der Fall ist. Danach ist es standardmäßig ausgeschaltet (und das ist auch gut so).


```
  <input type="reset" value="Abbrechen">
</form>
</div>
</body>
</html>
```

Quelltext der Datei 'ausgabe.php':

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<html>
<head>
 <title>Ausgabe</title>
</head><body>
<?php
printf("Feld 1:%s<br>Feld 2:%s",
       $_POST["feld1"],
       $_POST["feld2"]);
?>
</body>
</html>
```

Natürlich kann die Seite zum Senden und die zum Auswerten auch genau dieselbe sein; in diesem Fall muß sie logischerweise eine PHP-Endung haben.

Diese Variante (Formular und Auswertung in einer Seite) hat sich in vielen Situationen als praktisch erwiesen und wird deshalb auch häufig empfohlen. Siehe auch Kapitel 9.4.

9.2 Werte übergeben

Es gibt auch Situationen, da möchte man dem PHP-Script Werte übergeben, ohne ein Formular zu verwenden. Dies ist natürlich auch möglich. Ein normaler HTML-Link sieht folgendermaßen aus:

```
<a href="datei.php">Linktext</a>
```

Wenn man jetzt der Datei die Werte 'Wert1' und '2' in den Variablen 'VAR1' und 'VAR2' übergeben will, sieht der Link folgendermaßen aus:

```
<a href="datei.php?var1=Wert1&var2=2">Linktext</a>
```

Allgemeiner formuliert: An das Verweisziel (in unserem Fall 'datei.php') wird mit einem '?' beginnend der Variablenname und mit einem Gleichheitszeichen der Wert angehängt; weitere Werte mit einem '&' statt '?'. Es dürfen keine Leerzeichen dabei entstehen. Sonderzeichen in Werten, wie Umlaute, Leerzeichen, das Kaufmannsund- oder Fragezeichen, müssen nach einem bestimmten Schema kodiert werden, das URL-encoded genannt wird. PHP bietet mit den Funktionen `urlencode` und `urldecode` die Möglichkeit, Strings von und in dieses Format zu wandeln; beim Verschicken von Formularen wird quasi für jedes Feld `urlencode` aufgerufen und beim Bereitstellen der POST/GET-Daten in Dateien

Zeichen	Code	Zeichen	Code	Zeichen	Code
Leerzeichen	+	/	%2F	{	%7B
!	%21	:	%3A	|	%7C
„	%22	;	%3B	}	%7D
#	%23	<	%3C	~	%7E
$	%24	=	%3D	ä	%E4
%	%25	>	%3E	ö	%F6
&	%26	?	%3F	ü	%FC
'	%27	@	@	Ä	%C4
(	%28	[	%5B	Ö	%D6
)	%29	\	%5C	Ü	%DC
*	*	]	%5D	ß	%DF
+	%2B	^	%5E	—	
,	%2C	_	_	—	
-	-	`	%60	—	
.	.	—			

Tabelle 9.1: urlencoded

umgekehrt `urldecode`. Tabelle ‚urlencoded' listet die wichtigsten Sonderzeichen und ihre Kodierung auf.

Wenn man die übergebenen Werte verwendet, darf man nicht vergessen, daß **jeder**[4] die Werte beim Aufruf verändern kann. Deshalb sollten die Variablen vor der Weiterverarbeitung auf korrekte Werte hin überprüft werden.

9.3 Dynamische Formulare

Formulare zur Übermittlung von Daten (per POST oder GET) an sich sind schon ganz brauchbar, aber für wirklich interaktive, dynamische Formulare braucht es noch etwas mehr Verständnis von HTML, insbesondere den Formularelementen. Grundsätzlich hat jedes solcher Elemente einen Namen (`name`) und einen Wert (`value`). Der Name wird beim Verschicken des Formulars zum Namen der Variable bzw. des Indexes im Array, die den Wert des entsprechenden Formularelements annimmt. Im Wesentlichen sind folgende Elemente zu unterscheiden:

1. INPUT-Felder gliedern sich in folgende Typen (`type`):

   a) TEXT: Normales Texteingabefeld

   b) PASSWORD: Paßworteingabefeld (Anzeige von Sternchen statt Zeichen) — **Achtung:** Die Übermittlung erfolgt i. A. ungesichert!

   c) SUBMIT: Button, der das Formular abschickt. Wird ein Name angegeben, so hat dieser der HTML-Syntax entsprechend den Wert des Buttontitels.

[4] Über einfaches Kopieren und Verändern der URL

   d) RADIO: Radiobutton. Mehrere Radiobuttons mit demselben Namen aber unterschiedlichem Wert werden zu einer Gruppe zusammengefaßt, aus der nur ein (oder am Anfang ggf. kein) „Knopf" gleichzeitig gedrückt sein kann. Der standardmäßig gewählte Knopf wird mit dem Attribut `checked` (bei HTML ohne Wert) definiert.

   e) CHECKBOX: Checkbox. Soll sie standardmäßig gesetzt sein, wird dies mit dem Attribut `checked` (bei HTML ohne Wert) angegeben. Haben mehrere Checkboxen den gleichen Namen und endet dieser mit den Zeichen `[]`, so werden sie bei der Umwandlung durch PHP in *ein* Array umgewandelt, das genau die Einträge (mit den jeweils entsprechenden Werten) enthält, deren zugehörige Checkboxen beim Verschicken gesetzt waren. Auf diese Weise lassen sich besonders gut logisch zusammengehörende Mehrfachauswahlen realisieren.

   f) HIDDEN: verstecktes Feld. Dient der Übermittlung von Informationen, die weder veränderbar noch sichtbar sein sollen [5].

2. SELECT-Boxen erlauben die Auswahl eines Eintrags aus einer Dropdown-Liste. Ist das `value`-Attribut bei den `option`-Tags nicht vorhanden, so dienen die Einträge selbst als Werte.

Das folgende Beispiel zeigt, wie die genannten Formularelemente eingesetzt werden können und wie ihre Vorbelegung implementiert wird. Interessant ist dabei u.a. die Verwendung zweier neuer PHP-Funktionen: `is_array()` prüft, ob eine Variable ein Array ist[6], und mit `in_array()` kann man feststellen, ob ein Wert (Parameter 1) in einem Array (Parameter 2) enthalten ist.[7]

Anstelle der statischen Arrays in den beiden Funktionen für die Ausgabe der Optionen können, in Verbindung mit einer Datenbank-Anbindung, natürlich auch Arrays, die aus einer SQL-Abfrage resultieren, verwendet werden. Solche Arrays erstellt man i. A. mittels einer `while`-Schleife.

functions.inc:

```
<?php
  /**
   * Beruf-Optionen ausgeben
   *
   * @param $beruf Bisheriger Wert
   */
  function print_beruf_options($beruf=0) {
    $berufe = array("Angestellter",
                    "Oedie",
                    "Student/Schueler");
    for ($i=0; $i<count($berufe); $i++) {
      printf("<option value=\"%d\"%s>%s</option>\n",
```

[5] Hindert natürlich keinen, sie trotzdem zu verändern
[6] Es gibt z. B. auch `is_numeric()` für Zahlen
[7] Als dritten, booleschen Parameter kann man optional noch angeben, ob auch auf Typgleichheit geprüft werden soll.

```
        ($i+1), ($beruf==($i+1) ? " selected" : ""),
        htmlentities($berufe[$i])
      );
    }
  }

  /**
   * Hobby-Optionen ausgeben
   *
   * @param $hobby Array bisheriger Werte
   */
  function print_hobby_options($hobby) {
    if (!is_array($hobby))
      $hobby = array();
    $hobbies = array("Lesen", "Radfahren", "Schwimmen");
    for ($i=0;$i<count($hobbies);$i++) {
      printf("<input type=\"checkbox\" name=\"hobby[]\" ".
             "value=\"%s\"%s> %s\n",
        htmlentities($hobbies[$i]),
        (in_array($hobbies[$i],$hobby) ? " checked" : ""),
        $hobbies[$i]
      );
    }
  }
?>
```

daten.php:

```
<?php include("./functions.inc"); ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<html>
<head>
 <title>Pers&ouml;nliche Daten</title>
</head>
<body>
<?php
  // lokale Variablen setzen
  $vars = array("sender", "geschl", "vorname",
                "name", "beruf", "hobby");
  foreach ($vars as $var)
    $$var = $_POST[$var];

  if ($sender)
    printf("Die Daten wurden von %s aus verschickt.",
           htmlentities($sender));
  if (!isset($geschl))
    $geschl = 'm';
```



```
?>
<form action="daten.php" method="post">
<input type="hidden" name="sender" value="daten.php">
  Vorname: <input name="vorname" size="25" maxlength="60"
    value="<?php
      printf("%s", htmlentities($vorname));
    ?>"><br>
  Name: <input name="name" size="25" maxlength="60"
    value="<?php
      printf("%s", htmlentities($name));
    ?>"><br>
  Geschlecht: <input type="radio" name="geschl"
    value="m"<?php
      printf("%s", ($geschl=='m' ? " checked" : ""));
    ?>> m&auml;nnlich  
  <input type="radio" name="geschl"
    value="w"<?php
      printf("%s", ($geschl=='w' ? " checked" : ""));
    ?>> weiblich<br>
  Beruf: <select name="beruf">
  <option value="0"<?php
    echo (!isset($beruf) ? " selected" : "");
  ?>>--- Bitte w&auml;hlen ---</option>
  <?php
    print_beruf_options($beruf);
  ?>
  </select><br>
  Hobbies: <?php
    print_hobby_options($hobby);
  ?><br>
  <input type="submit" value="Weiter">
  <input type="reset" value="Zur&uuml;cksetzen">
</form>
</body>
</html>
```

Der Effekt dieser ganzen Abfragen und Zusatzangaben ist letztlich nicht nur, daß die Daten verschickt werden, sondern daß auch das Formular nach dem Verschicken wieder genau so aussieht, wie zuvor. In diesem Beispiel werden die verschickten Daten, bis auf das Ausfüllen des Formulars, nicht weiterverarbeitet. Anbieten würde sich für persönliche Daten z. B. das Anlegen oder Aktualisieren von Datensätzen einer entsprechenden Datenbank. Da im obigen Beispiel die Hobbies über Checkboxen realisiert wurden, die eine Auswahl ermöglichen, die aus mehreren Werten besteht, müßte dieses Feld datenbanktechnisch wohl relational definiert werden, also über eine Zusatztabelle, die die Beziehung zwischen persönlichen Daten und einer Hobbytabelle herstellt:

pers_daten	pd_hob	hobbies
ID	← PID, HID →	ID

Tabelle 9.2: Datenbank-Realisierung der Hobbies-Mehrfachauswahl

9.4 Normalform von Formularen

Die Verarbeitung von Daten, die über ein Formular verschickt werden, folgt allgemein einem bestimmten Schema, das sich einfach daraus ergibt, daß die Daten nicht nur verifiziert und damit ggf. auf neue, korrigierte Werte gesetzt werden müssen (nebst Erstellung einer Fehlermeldung, die zusammen mit allen anderen an zentraler Stelle ausgegeben werden kann), sondern im Verlauf der Datenverarbeitung auch der Fall auftreten kann, daß man HTTP-Header, wie für eine Weiterleitung notwendig, schicken will. Wie wir aus Kapitel 11.1 wissen, darf jedoch keine direkte Ausgabe, also z. B. mittels `echo`, `print(f)` oder bloßer Text außerhalb der PHP-Marken, erfolgen, bevor nicht alle Header ausgegeben worden sind. Dies wiederum bedeutet, daß alle Codeabschnitte, die unter irgendwelchen Bedingungen HTTP-Header ausgeben könnten, vor der ersten direkten Ausgabe stehen müssen.

Es bietet sich also an, die gesamte Datenverarbeitung an den Beginn eines PHP-Scriptes zu stellen und darauf zu achten, daß kein einziges anderes Zeichen vor der ersten PHP-Marke (am Dateianfang) steht. Eine Weiterleitung über HTTP-Header macht v. a. dann Sinn, wenn das Formular nach erfolgtem Eintragen und Verschicken von korrekten Daten nicht mehr gebraucht wird[8]. In diesem Fall hat dies auch den Vorteil, daß ein „Reload“ der Seite keinen Effekt hat. Auch sonst sollte man aber immer überprüfen, ob eine doppelte Eintragung durch falsche bzw. mißbräuchliche Browserbenutzung möglich ist und entsprechende Vorkehrungen bei der Konstruktion der Datenverarbeitung treffen (z. B. nach gleichen Einträgen suchen, bevor ein INSERT versucht wird).

Die Erkennung, ob Daten verschickt wurden oder nicht, macht man i. A. davon abhängig, ob die Variable, die den gleichen Namen hat wie der Submit-Button des Formulars, das die Daten verschickt, gesetzt ist oder nicht. Zur Erinnerung: Ein solcher Submit-Button trägt dann, wenn er einen Namen hat, den Wert, der als `value`-Attribut angegeben wurde – welcher in diesem Fall gleichbedeutend ist mit Button-Titel. In PHP spielt es dabei meist eine untergeordnete Rolle, ob man für die Überprüfung die Funktion `isset(var)` benutzt, die auf Existenz einer Variable testet, oder einfach direkt die Variable abfragt, denn in PHP ist jeder String außer dem leeren und „0“ gleich dem booleschen Wert `TRUE`.

Im folgenden Schema sei `$submitted` *true*, falls das (hier nicht angegebene) Formular abgeschickt wurde. Das könnte man z. B. durch Abfragen des Submit-Buttons erreichen, dessen Beschriftung bekanntlich gleich seinem Wert ist. Wie alle anderen Formularwerte findet sich auch der des Submit-Buttons im passenden superglobalen, assoziativen Array[9], genau gesagt an der Indexposition mit Namen des name-Attributs des Submit-Buttons.

Der gesamte Datenverarbeitungsblock kann natürlich auch über Funktions- oder Methodenaufrufe (bei OOP, siehe Kapitel 20) erledigt werden, dann ist jedoch unbedingt darauf zu achten, eine saubere Parameterübergabe zu verwenden; hierbei bieten sich besonders

[8] Sind die Daten nicht korrekt, kann eine Fehlermeldung generiert und das Formular ggf. mit den korrigierten Daten erneut angezeigt werden, anstatt weiterzuleiten

[9] `$_POST` bzw. `$_GET`, je nach Übermittlungsart

assoziative Arrays an[10].

```
<?php
// ggf. Funktions-Include
if ($submitted) {
  // Daten prüfen, ggf. Fehlermeldungen erzeugen
  if (<Daten OK>) {
    // eigentliche Verarbeitung:
    // - neuen Datensatz anlegen
    // - bestehenden Datensatz aktualisieren
    // - bestehenden Datensatz löschen
    // und ggf. weiterleiten
  } else {
    // Fehlermeldung erzeugen
    // (in Variable schreiben)
  }
} elseif ($fillform) {
  // alte Daten lesen und in Variablen speichern
}
// Frühestmögliche Ausgabe: ggf. HTML-Header-Include;
// Fehler ausgeben
?>
Hier das Formular (mit Werten) anzeigen
<?php
// ggf. HTML-Footer-Include
?>
```

Mit den Includes sind mögliche Einbindungen von Funktions- und Klassendefinitionsdateien gemeint bzw. HTML-Ausgaben, die weitgehend unabhängig von der Datenverarbeitung sind (HTML-Header und -Footer).
Falls Daten geändert werden sollen (im Beispiel angedeutet durch Abfragen der booleschen Variable `$fillform`), muß natürlich das Formular ausgefüllt werden, d. h. entsprechende DB-Abfragen müssen gestartet und ausgewertet werden, so daß danach die Variablen, die die Formularfelder vorbelegen, sinnvolle Werte haben.

Ein schönes Beispiel für die Anwendung des hier gelesenen findet sich übrigens in Kapitel 16.

9.4.1 Wieso ist das Null?!

In obiger Formular-Normalform wird an der Stelle `if (<Daten OK>)` überprüft, ob die übergebenen Werte korrekt sind. Was genau an dieser Stelle zu tun ist, hängt ganz vom jeweiligen Fall ab. Ein häufiger Denkfehler (im Englischen nennt man das übrigens „common pitfall“) dabei ist, daß Werte, die per GET oder POST übermittelt werden, mit dem Integerwert 0 verglichen werden und dabei angenommen wird, daß dies nur wahr wird,

[10] Namensvergabe im Formular: einheitlicher Name, direkt gefolgt von dem eigentlichen Variablenname in eckigen Klammern, z. B. daten[vorname]

wenn die fragliche Variable auch tatsächlich den Wert Null (nicht `null`) hat. **Falsch gedacht!** Wer das aktuelle DSP bis zu dieser Stelle aufmerksam gelesen hat, weiß, warum: In Kapitel 8.2.9.1 wurde gesagt, daß *jeder* String bei normalem Vergleich gleich dem Integerwert Null ist.

```
if ($_POST['stueckmenge']==0) {
  // Formular für Bestellung ausgeben
} else {
  // Bestellung aufnehmen
}
```

Da hier nicht mittels `===` auf Typgleichheit überprüft wurde, wird das Formular **immer** ausgegeben werden (d. h. die Abfrage ist in dieser Form nutzlos) – POST- und GET-Werte sind immer Strings. . .

10 PHP & MySQL

Jetzt kommen wir zu dem Punkt, auf den wir schon die ganze Zeit hingearbeitet haben: **Die Verbindung von PHP und MySQL**. In diesem Kapitel werden die Befehle beschrieben. Weiter unten werden noch Befehlsfolgen für gewisse Aufgaben aufgelistet.

Hier werden jetzt Befehle von PHP und MySQL verwendet. Zwischen diesen beiden Sprachen ist streng zu trennen!

10.1 Syntax

10.1.1 allgemein

Zum Beschreiben der Syntax von PHP-Befehlen hat sich ein gewisser Standard entwickelt. Dieser hat große Ähnlichkeit mit dem anderer Programmiersprachen.

Was muß in der Syntax erklärt werden? Als erstes interessiert natürlich der Funktionsname. Da Funktionen immer was zurückgeben, interessiert auch der Rückgabetyp und last but not least muß man auch wissen, welche Parameter die Funktion erwartet (und welchen Typ sie haben müssen). Diese Informationen sind in der Syntax versteckt. Nehmen wir als Beispiel die Funktion `mysql_connect()`. Laut Anleitung ist die Syntax wie folgt (wegen Platzproblemen in zwei Zeilen geschrieben, eigentlich gehört alles hintereinander):

```
resource mysql_connect
              ( [string server [, string username [, string password ]]])
```

Schauen wir uns die Syntax mal von vorne an. Das Wort `resource` gibt den Rückgabetyp der Funkion `mysql_connect` an. In den Klammern stehen dann die Parameter, wobei das `string` den erwarteten Typ angibt, d. h. alle drei Parameter müssen in diesem Fall vom Typ `string` sein. Eckige Klammern definieren jeweils einen optionalen Teil, der nur bei Bedarf angegeben werden muß. Wenn der jeweilige Parameter nicht angegeben wird, wird ein Standardwert genommen. Die genaue Beschreibung der Parameter und welche Standardwerte diese haben, wenn sie optional sind, steht in der anschließenden Beschreibung.

Die Typangabe und die eckigen Klammern werden später selbstverständlich nicht mit eingegeben. Mit allen Parametern würde der Aufruf des Befehls z. B. folgendermaßen aussehen:

```
$link = mysql_connect('localhost','MeinName','MeinPasswort');
```

10.1.2 mysql_connect

Syntax:

```
resource mysql_connect
              ( [string server [, string username [, string password ]]])
```

Mit **mysql_connect()** wird eine Verbindung zum Server geöffnet.

Wenn der zurückgegebene Wert nicht ‚FALSE‘ ist, verlief die Verbindung erfolgreich. Bei einem zweiten Aufruf mit denselben Parametern wird keine neue Verbindung erstellt, aber es wird der ‘link_identifier‘ zurückgegeben. Der ‘link_identifier‘ wird benötigt, um bei mehreren Verbindungen eine eindeutig bestimmen zu können.

Als ‘hostname‘ ist in unserem Fall **immer** localhost zu nehmen. Für ‘username‘ und ‘password‘ sind die von uns bzw. von dir selbst zugewiesenen Werte zu nehmen.

Die Verbindung wird automatisch bei Beenden des Scripts geschlossen; sie kann aber auch mit **mysql_close()** explizit geschlossen werden.

10.1.3 mysql_close

Syntax:

```
bool mysql_close ( [resource link_identifier])
```

mysql_close schließt eine bestehende Verbindung zum Server.

Es wird entweder ‘TRUE‘ bei Erfolg oder ‘FALSE‘ bei einem Fehler zurückgegeben.

Mit ‘link_identifier‘ kann explizit angegeben werden, welche Verbindung geschlossen werden soll. Wenn nichts angegeben wurde, wird die zuletzt geöffnete Verbindung geschlossen.

10.1.4 mysql_select_db

Syntax:

```
bool mysql_select_db ( string database_name [, resource link_identifier])
```

Mit **mysql_select_db** wird die Datenbank ausgewählt, auf die sich die Anfragen beziehen soll.

Es wird entweder ‘TRUE‘ bei Erfolg oder ‘FALSE‘ bei einem Fehler zurückgegeben.

Wenn kein ‘link_identifier‘ angegeben wurde, wird die zuletzt geöffnete Verbindung zum Server benutzt.

10.1.5 mysql_query

Syntax:

```
resource mysql_query ( string query [, resource link_identifier])
```

mysql_query sendet die SQL-Befehlsfolge ‘query‘ an den Server mit der DB, die durch den ‘link_identifier‘ festgelegt ist. Wird kein ‘link_identifier‘ angegeben, wird die zuletzt geöffnete Verbindung genutzt.

Es wird ein sogenannter Zeiger auf das Ergebnis (result pointer) zurückgegeben. Wenn dieser (Result-)Zeiger den Wert ‚FALSE‘ hat, gibt es einen Fehler. Mit dem Zeiger an sich kann man aber nicht viel anfangen; vielmehr benötigt man ihn, um die folgenden Funktionen nutzen zu können. Dort wird er als Parameter `result` übergeben.

10.1.6 mysql_fetch_array

Syntax:

```
array mysql_fetch_array(resource result [, int resulttype])
```

Dies ist eine erweiterte Version von ‘mysql_fetch_row‘ (siehe nächste Funktion). Die Indizes des Arrays werden nicht von 0 ausgehend durchgezählt, sondern nach den Spaltennamen benannt.

Die Funktion ist in der Ausführung nur unwesentlich langsamer als `mysql_fetch_row`, obwohl es deutlich angenehmer zu programmieren ist.

Wenn beim Join zweier Tabellen zwei Spalten denselben Namen haben, müssen ihnen mit Hilfe der `SELECT`-Anweisung andere Namen gegeben werden. Beispiel:

```
SELECT t1.s1 AS foo, t2.s1 AS bar FROM t1, t2
```

Die Spalte `s1` der Tabelle `t1` hat nun den Namen *foo* und `s1` aus `t2` hat den Namen *bar*.

Das optionale zweite Argument ***`result_type`*** in `mysql_fetch_array` ist eine Konstante und kann die folgenden Werte annehmen: `MYSQL_ASSOC`, `MYSQL_NUM` und `MYSQL_BOTH`.

Bei `MYSQL_NUM` bekommt das zurück gegebene Array einen durchnumerierten Index, d. h. `mysql_fetch_array` verhält sich genauso wie `mysql_fetch_row`. Bei `MYSQL_ASSOC` werden die Spaltennamen als Index des Arrays genutzt und bei `MYSQL_BOTH` (was auch der Standard ist) gibt es beide Indizes.

```
<?php
mysql_connect($host,$user,$password);
mysql_select_db("database");
$result = mysql_query("SELECT user_id, fullname FROM users");
while($row = mysql_fetch_array($result)) {
  echo $row["user_id"];
  echo $row["fullname"];
}
?>
```

10.1.7 mysql_fetch_row

Syntax:

```
array mysql_fetch_row(resource result)
```

Holt eine Zeile aus der DB-Abfrage, gibt diese als Array zurück und setzt den Result-Zeiger auf die nächste Zeile. Für ‘result‘ muß der Rückgabewert (=Zeiger) der ‘mysql_query‘-Funktion genommen werden.

Dasselbe Beispiel wie bei `mysql_fetch_array`:

```
<?php
mysql_connect($host,$user,$password);
mysql_select_db("database");
$result = mysql_query("SELECT user_id, fullname FROM users");
```

```
while($row = mysql_fetch_row($result)) {
  echo $row[0];
  echo $row[1];
}
?>
```

10.1.8 mysql_error

Syntax:

`string mysql_error([resource link_identifier])`

Gibt die Fehlermeldung des letzten SQL-Befehls zurück. Wenn es keinen Fehler gab, wird nichts zurück gegeben. Wird kein 'link_identifier' angegeben, wird die zuletzt geöffnete Verbindung genutzt.

10.1.9 mysql_errno

Syntax:

`int mysql_errno([resource link_identifier])`

Gibt die Fehlernummer des letzten SQL-Befehls zurück. Wenn es keinen Fehler gab, wird 0 zurückgegeben. Wird kein 'link_identifier' angegeben, wird die zuletzt geöffnete Verbindung genutzt.

10.1.10 mysql_insert_id

Syntax:

`int mysql_insert_id([resource link_identifier])`

Gibt die Nummer zurück, die beim letzten `INSERT` dem Feld mit `AUTO_INCREMENT` zugewiesen wurde.

10.1.11 mysql_num_rows

Syntax:

`int mysql_num_rows(resource result);`

Gibt die Anzahl der Zeilen im Ergebnis zurück.

10.2 Tips und Tricks

Gerade beim Zusammenspiel von PHP und MySQL gibt es einige Stellen, an denen man es sich einfach machen kann, wenn man weiß, wie.

Im Folgenden sei bereits jeweils eine Datenbankverbindung hergestellt.

10.2.1 Abfragen mit IN

Gegeben sei ein Array von IDs, die im IN-Teil einer Abfrage auftauchen sollen. Mithilfe der PHP-Funktion `implode` spart man sich das händische Zusammenbauen des entsprechenden Strings.

Denkbar wäre etwa folgende Abfrage zum Löschen aller Datensätze, deren IDs im übergebenen Array vorkommen. Dieses Array, nennen wir es einmal `delIDs`, könnte z. B. von einem Formular mit Checkboxen wie in 9.3 beschrieben stammen: Auf einer Übersichtseite, auf die hier nicht näher eingegangen wird, könnten alle oder bestimmte Datensätze in Tabellenform nebst je einer Checkbox angezeigt werden, wobei die `VALUE`-Attribute der Checkboxen genau den IDs der Datensätze entsprächen.

```
mysql_query("DELETE FROM tabelle
             WHERE ID IN (".implode(',', $delIDs).")");
```

Wenn die Werte keine Integers sondern Strings sind, kommt man aber wohl kaum um einen kompletten Durchlauf des Arrays herum:

```
foreach ($ids as $key=>$id)
  $ids[$key] = "'$id'";
```

10.3 Übung

10.3.1 Ergebnis-Tabelle ausgeben I

Als Vertiefung und Übung zu dem bisher gesagten eine kleine Aufgabe:
Es soll eine Funktion geschrieben werden, die das Ergebnis einer SQL-Abfrage tabellarisch darstellt. Die Ausgabe soll im Prinzip die Tabelle, die man beim MySQL-Prompt bekommt, in HTML umsetzen (inkl. Spaltennamen).

Wenn man zum Beispiel am MySQL-Prompt alle Mitarbeiter abfragen will, kommt folgende Ausgabe zustande:

```
mysql> select * from Mitarbeiter;
+-----+------+-------+----------------+------------+--------------+
| MNr | VNr  | AbtNr | Name           | GebDat     | Telefon      |
+-----+------+-------+----------------+------------+--------------+
|   1 | NULL |     3 | Christoph Reeg | 1979-05-13 | NULL         |
|   2 |    1 |     1 | junetz.de      | 1998-03-05 | 069/764758   |
|   3 |    1 |     1 | Uli            | NULL       | NULL         |
|   4 |    3 |     1 | JCP            | NULL       | 069/764758   |
|   5 |    1 |     2 | Maier          | NULL       | 06196/671797 |
|   6 |    5 |     2 | Meier          | NULL       | 069/97640232 |
+-----+------+-------+----------------+------------+--------------+
6 rows in set (0.00 sec)

mysql>
```

Um das in einer HTML-Tabelle darzustellen, ist folgender (vereinfachter) HTML-Code notwendig:

```
<html>
<body>
<table>
  <tr>
    <th>MNr</th>
    <th>VNr</th>
    <th>AbtNr</th>
    <th>Name</th>
    <th>GebDat</th>
    <th>Telefon</th>
  </tr>
  <tr>
    <td>1</td>
    <td></td>
    <td>3</td>
    <td>Christoph Reeg</td>
    <td>1979-05-13</td>
    <td></td>
  </tr>
  <tr>
    <td>2</td>
    <td>1</td>
    <td>1</td>
    <td>junetz.de</td>
    <td>1998-03-05</td>
    <td>069/764758</td>
  </tr>
  <tr>
    <td>3</td>
    <td>1</td>
    <td>1</td>
    <td>Uli</td>
    <td></td>
    <td></td>
  </tr>
  <tr>
    <td>4</td>
    <td>3</td>
    <td>1</td>
    <td>JCP</td>
    <td></td>
    <td>069/764758</td>
  </tr>
```



```
  <tr>
    <td>5</td>
    <td>1</td>
    <td>2</td>
    <td>Maier</td>
    <td></td>
    <td>06196/671797</td>
  </tr>
  <tr>
    <td>6</td>
    <td>5</td>
    <td>2</td>
    <td>Meier</td>
    <td></td>
    <td>069/97640232</td>
  </tr>
</table>
</body>
</html>
```

Als kleine Hilfe hier das Rahmenprogramm für die Funktion:

```
<?php
$db_host   =      "localhost";
$db_user   =      "cr";
$db_pass   =      "123";

$datab  =         "cr";


function print_result_table($result){
// hier das richtige schreiben
}

// Hauptprogramm

/* Verbindung zur Datenbank aufbauen */
$db = @mysql_connect($db_host,$db_user,$db_pass)
      or die(mysql_error());
@mysql_select_db($datab,$db) or die(mysql_error());

/* HTML-Startcode ausgeben */
echo "<html>\n<body>\n";

/* SQL-Abfrage */
$result = @mysql_query("SELECT * FROM Mitarbeiter");
print_result_table($result);
```

```
/* HTML-Endcode ausgeben */
echo "</body>\n</html>\n";
?>
```

Tips zur Lösung

Die Aufgabe in mehreren Schritten lösen:

1. Nur in der 1. Zeile die 1. Spalte ausgeben.
2. Mit einer Schleife für alle Zeilen die 1. Spalte ausgeben.
3. Mit einer 2. Schleife alle Spalten ausgeben (vorher in der Dokumentation nachsehen, mit welcher Funktion man die Spaltenanzahl abfragen kann).
4. Als letztes noch die Spaltennamen ausgeben.

Eine Lösung befindet sich in Anhang B.2.

10.3.2 Ergebnis-Tabelle ausgeben II

Das Hauptprogramm aus der vorigen Aufgabe soll etwas erweitert werden. Bei meiner Vorgabe ist bis jetzt noch keine Fehlerprüfung enthalten. Was für Fehler können auftreten?

- Die SQL-Anweisung enthält Fehler.
- Die Abfrage enthält keine Datensätze.

Bei dem ersten Fehler soll die Fehlermeldung von MySQL ausgegeben werden, beim zweiten "Keine Datensätze gefunden". Das Hauptprogramm soll entsprechend erweitert werden.
Eine Lösung befindet sich in Anhang B.3.

10.3.3 Abfrage mit sprintf()

Auch in diesem Beispiel wird obige Tabelle benutzt. Die Zeile „/* SQL-Abfrage */" und die folgende werden jedoch nun wie folgt verändert[1].

```
...
$was = "Name";
$monat = 5;
$jahr = 1979;

/* SQL-Abfrage */
$result = @mysql_query(
  sprintf("SELECT %0\$s
           FROM Mitarbeiter
```

[1] Fehlerbehandlung wurde hier absichtlich außen vor gelassen, um die vorhergehenden Aufgaben nicht zu beeinträchtigen


```
           WHERE GebDat LIKE
           '%2\$02d-%3\02d-%'",
          $was, $monat, $jahr
  )
);
...
```

In diesem Programmabschnitt sind sechs Fehler versteckt. Wer findet sie? Auflösung in Anhang B.4.

10.3.4 Einfügen mit automatischer ID

Nun etwas neues: Es soll in zwei Tabellen eingefügt werden, wobei die ID der ersten Einfügeoperation für die zweite benötigt wird. Konkret soll der Mitarbeiter „Jens" eingefügt werden, der direkt Christoph Reeg unterstellt ist, in der Verwaltung arbeitet und am 26.5.1981 Geburtstag hat. Diesem Mitarbeiter unterstellen wir wiederum einen Helfer namens Kile, der in der EDV arbeitet. Zu beachten ist, daß beim Einfügen **keine** Mitarbeiter-Nummer vergeben werden sollen; das soll MySQL selbst machen (die MNr ist ein AUTO_INCREMENT-Feld). Alle hier nicht angegebenen Werte sollen NULL sein.

Eine mögliche Lösung findet sich in Anhang B.5. Die Verbindungsdaten für MySQL seien dieselben wie bei 10.3.1. Die vorgegebenen Tabellen finden sich unter A.2.

11 PHP & HTTP

11.1 Header

Neben der eigentlichen Seite schickt der Server an den Client (Browser) noch einige Zusatzinformationen. Diese werden vor der eigentlichen Seite im sog. Header gesendet. Mit diesen Informationen sagt der Server z. B., ob die angezeigte Seite wirklich die gewünschte Seite ist (Status '200 Found'), oder ob die Seite nicht gefunden werden konnte und deshalb eine Fehlerseite angezeigt wird (Status '404 Not Found'). Auch kann der Server dem Client mitteilen, daß die Seite sich unter einer anderen Adresse befindet (Status '301 Moved Permanently' oder '302 Found'). Es kann auch die Aufforderung geschickt werden, sich zu authentifizieren (Status '401 Unauthorized').

Zusätzlich zum Status einer Seite kann auch übermittelt werden, wann die Seite zum letzten Mal verändert wurde (Last-Modified), ob sie gecached werden darf (Cache-Control) und wenn ja wie lange (Expires) oder welchen Typ ihr Inhalt hat (Content-Type).

Normalerweise sendet der Webserver (in der Regel Apache) automatisch den richtigen Header. Mit PHP kann man den gesendeten Header allerdings beeinflussen. **Zu beachten ist, daß kein einziges Zeichen vor der `header`-Anweisung ausgegeben werden darf!** Ausgeben heißt in diesem Fall: Die Seite muß unbedingt mit PHP-Code (<?php) anfangen und darf vor dieser Codemarke nichts (nicht einmal ein Leerzeichen oder einen Zeilenumbruch) enthalten. Auch innerhalb der Codemarken dürfen Ausgaben mittels `echo`, `print` etc. erst nach dem Senden der Headerangaben gemacht werden.

Wenn PHP als CGI installiert ist, gibt es außerdem einige Einschränkungen, z. B. kann keine Authentifizierung gemacht werden (mehr dazu siehe weiter unten).

Wie der Header aussehen muß, ist in dem RFC[1] 2616 festgelegt. Er spezifiziert das HTTP/1.1 Protokoll. Im Folgenden zeige ich ein paar Möglichkeiten der Anwendung der `header`-Anweisung.

11.1.1 Weiterleiten

Wie bereits oben erwähnt, kann man, neben JavaScript und HTML[2], auch mit PHP den Client auf eine andere Seite weiterleiten. Dies geschieht mit folgender Anweisung:

```
header('Location: absolute_URL');
exit;
```

`absolute_URL` muß natürlich durch die gewünschte URL ersetzt werden. Es muß nach RFC die absolute URL angegeben werden, auch wenn fast alle Browser eine relative verstehen!

Das `exit` ist nicht unbedingt notwendig, allerdings würde es nichts bringen, nach dem `header` noch etwas auszugeben, da es sowieso nicht angezeigt wird.

[1] Request for Comments

[2] hier allerdings nicht konditional, also bedingungsabhängig, sondern nur nach einer festen Anzahl Sekunden

Bei dieser Anweisung sendet Apache automatisch den Statuscode 302.

11.1.2 Nicht gefunden

Wenn du Apache so konfiguriert hast, daß er als Fehlerseite eine PHP-Seite anzeigt, wird als Statuscode 200 (OK) gesendet. Da dies aber unpraktisch ist, weil so z. B. Suchmaschinen deine Fehlerseite in ihren Index aufnehmen, solltest du den Statuscode 404 (Not Found) senden, wodurch diese Seite als Fehlerseite erkannt wird. Die Anweisung dazu lautet wie folgt:

```
header('HTTP/1.0 404 Not Found');
```

11.1.3 Authentifizierung

Mit PHP besteht die Möglichkeit, den Browser ein Fenster öffnen zu lassen, in dem Name und Paßwort eingetragen werden müssen. Wenn PHP nicht als Modul, sondern als CGI läuft, funktioniert das allerdings **nicht**[3].

Es ist eigentlich ganz einfach, eine solche Datei muß vom Prinzip her so aussehen:

```
<?php
  if($PHP_AUTH_USER!="Christoph" OR $PHP_AUTH_PW!="Reeg") {
    Header('HTTP/1.1 401 Unauthorized');
    Header('WWW-Authenticate: Basic realm="Top Secret"');
    echo "Mit Abbrechen kommst Du hier nicht rein. ;-) \n";
    exit;
  }
?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<html>
<head>
 <title>Authentification</title>
</head>
<body>
<h1>Hier ist der Top-Secret Bereich</h1>
<h2><?php
  echo "Username: ".$PHP_AUTH_USER." Paßwort: ".$PHP_AUTH_PW;
?></h2>
</body>
</html>
```

Das Funktionsprinzip ist ganz einfach: Beim ersten Aufruf sind im Array „_SERVER" die beiden Stellen '„PHP_AUTH_USER' und 'PHP_AUTH_PW' nicht gesetzt. Dadurch wird der Bereich in der `IF`-Abfrage bearbeitet. Hier werden die beiden Header zurückgegeben, die den Browser veranlassen, nach Usernamen und Paßwort zu fragen. Diese beiden Zei-

[3] Übrigens: PHP gibt es auch als Modul für Windows, siehe auch PHP-FAQ[8] "Wo finde ich PHP als Modul für Windows?"

len müssen fast genau so übernommen werden, damit es funktioniert![4] Das einzige, was geändert werden darf, ist das 'Top Secret'. Der Text danach wird nur dann ausgegeben, wenn jemand bei der Paßwortabfrage auf ‚Abbrechen' klickt (oder, im Falle des Internet Explorers, drei Versuche, sich zu authentifizieren, mißlungen sind); dann springt der Webserver nach dem 'echo' aus der Datei und der Rest wird nicht mehr ausgegeben. Wenn jedoch jemand das richtige Paßwort mit dem richtigen Usernamen eingegeben hat, wird der Bereich in der `IF`-Abfrage nicht bearbeitet und der Rest der Datei wird abgearbeitet. In unserem Fall wird die Überschrift „Hier ist der Top-Secret Bereich" und die Zeile „Username: Christoph Paßwort: Reeg" im HTML-Format ausgegeben.

Es gibt noch ein kleines Sicherheitsproblem bei der ganzen Sache - der Browser behält sich nämlich den Usernamen und das Paßwort, so daß die Autoren derjenigen Seiten, die man nach der Paßworteingabe abruft, theoretisch das Paßwort abfragen könnten. Dies kann man jedoch ganz einfach verhindern, indem man den Browser komplett beendet.

Auf fast dieselbe Weise kann man sich natürlich auch direkt für den Zugriff auf eine Datenbank authentifizieren. Der folgende Quelltext zeigt, wie man dies erreicht:

```
<?php
if ($_SERVER["PHP_AUTH_USER"] == ""
      OR !@mysql_connect("localhost",
                         $_SERVER["PHP_AUTH_USER"],
                         $_SERVER["PHP_AUTH_PW"])) {
    Header('HTTP/1.0 401 Unauthorized');
    Header('WWW-Authenticate: Basic realm="Top Secret"');
    echo "Mit Abbrechen kommst Du hier nicht rein. ;-)\n";
    exit;
 }
?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<html>
<head>
 <title>Authentification</title>
</head>
<body>
<h1>Hier ist der Top-Secret Bereich</h1>
<h2><?php
  echo "Username: ".$_SERVER["PHP_AUTH_USER"];
  echo " Paßwort: ".$_SERVER["PHP_AUTH_PW"];
?></h2>
</body>
</html>
```

Das '@'-Zeichen vor dem `mysql_connect` hat nichts mit der `if`-Abfrage zu tun. Es sorgt dafür, daß keine Fehlermeldung beim Aufruf von `mysql_connect` ausgegeben wird. Die Fehlermeldung würde nicht nur stören, sondern sie würde auch die Paßwortabfrage zuver-

[4] Beachtet man z. B. die Groß-/Kleinschreibung nicht, kann das dazu führen, daß die Authentifizierung nicht mehr mit dem Internet Explorer, wohl aber weiterhin mit dem Netscape Navigator funktioniert!

lässig verhindern. Vor dem `header`-Aufruf darf nichts ausgegeben werden.

Der Bereich in der obigen `IF`-Abfrage wird genau dann nicht bearbeitet, wenn mittels Benutzername und Paßwort eine Verbindung zur Datenbank aufgebaut werden konnte. In jedem anderen Fall wird, wie im ersten Beispiel, abgebrochen und (in diesem Fall) der Text „Mit Abbrechen..." ausgegeben. Um sich Probleme zu ersparen, sollte man obige Bedingung der `IF`-Anweisung einfach 1:1 übernehmen, denn diese ist bestens erprobt! :-)

Noch eine Anmerkung zum Schluß: Anstatt der Zeichenkette "HTTP/1.0 401 Unauthorized" kann auch "Status: 401 Unauthorized" benutzt werden. Im Falle des o. g. PHP-CGI-Problems scheint es dann so, als ob die Authentifizierung funktionieren würde (es tritt kein Fehler 500 mehr auf); dies ist jedoch ein Trugschluß, denn trotz allem werden die beiden benötigten Authentifizierungs-Variablen nicht mit den Werten gefüllt, die der Browser nach der Eingabe durch den Benutzer im entsprechenden Dialog zurückliefert.

11.1.4 Download

In bestimmten Fällen wäre es schön, wenn ein PHP-Script die von ihm erzeugten Daten nicht einfach in Form einer HTML-Seite ausgeben, sondern diese an den Client senden könnte. Dieser sollte dann die Daten z. B. in Form einer Datei abspeichern oder auf sonstige Weise verarbeiten (an spezielle Applikationen übergeben).

Solche Situationen gibt es z. B. bei Anhängen (Attachments) in einem Webmail-System. Normalerweise wird die Ausgabe eines PHP-Scripts als HTML interpretiert, welches der Browser anzeigen soll. Damit der Browser die Datei aber direkt auf die Platte speichert (bzw. dem Benutzer überläßt, was er damit machen will), muß die Angabe über den Typ des Dateiinhalts für die Übertragung geändert werden. Das geschieht mit folgender Anweisung (siehe auch weiter unten):

```
header("Content-Type: application/octetstream");
```

Wenn nichts anderes angegeben wird, benutzt der Browser den Dateinamen des Scripts aus der URL als Dateinamen zum Abspeichern.

```
header("Content-Disposition: attachment; filename=datei_name.ext");
```

Mit diesem Header wird der Dateiname auf „datei_name.ext" gesetzt. Man beachte das Fehlen von Quoting-Zeichen wie etwa Hochkommata[5]. Grund hierfür ist, daß bestimmte Browser wie der IE sonst die Quoting-Zeichen als Teil des Dateinamens ansehen. Natürlich kann anstelle des hier beispielhaft eingetragenen jeder mögliche Dateiname stehen. Eventuelle Pfadangaben sollen explizit ignoriert werden. D. h. es ist möglich den Dateinamen festzulegen, aber nicht in welches Verzeichnis die Datei gespeichert werden sollte.

Microsoft liest die RFCs scheinbar anders als alle anderen (oder gar nicht?), so daß der IE 5.5[6] nur folgenden Header versteht:

```
header("Content-Disposition: filename=datei_name.ext");
```

Über die Variable `_SERVER["HTTP_USER_AGENT"]` können wir PHP auch entscheiden lassen, welche Variante wahrscheinlich die richtige ist.

[5] Nach RFC muß der Dateiname gequotet werden

[6] Wie es sich mit IE 6.x verhält, wissen wir erst, wenn die endgültige Version erschienen ist. Ältere Versionen dagegen scheinen den Fehler nicht aufzuweisen.

```
header("Content-Disposition: ".
       (strpos($_SERVER["HTTP_USER_AGENT"],"MSIE 5.5")?""
                                              :"attachment; ").
       "filename=datei_name.ext");
```

Die Variante, den Dateinamen über Header festzulegen, hat einen kleinen Nachteil: Wenn der Nutzer später im Browser nicht auf den Link klickt, um dann die Datei zu speichern, sondern direkt über „Save Link as“ speichern will, konnte noch kein Header gesendet werden, so daß der Browser den Dateinamen nicht kennt und wieder den Dateinamen des Scripts vorschlägt. Das kann nur umgangen werden, indem man dafür sorgt, daß der gewünschte Dateiname in der URL steht. Dies ist wiederum nur über Funktionen des Webservers möglich. Beim Apache sind das die Funktionen Rewrite und Redirect.

Die Erfahrung hat gezeigt, daß ein „Content-Transfer-Encoding“ Header die ganze Sache sicherer macht, auch wenn er laut RFC 2616 nicht benutzt wird.

```
header("Content-Transfer-Encoding: binary");
```

Die „großen“ Browser zeigen beim Download häufig einen Fortschrittsbalken an. Dies funktioniert allerdings nur dann, wenn der Browser weiß, wie groß die Datei ist. Die Größe der Datei in Bytes wird über den „Content-Length“ Header angegeben.

```
header("Content-Length: {Dateigröße}");
```

Zusammenfassend können wir nun folgenden Header benutzen, wenn die Ausgabe eines Scripts heruntergeladen werden soll:

```
// Dateityp, der immer abgespeichert wird
header("Content-Type: application/octetstream");
// Dateiname
// mit Sonderbehandlung des IE 5.5
header("Content-Disposition: ".
     (!strpos($HTTP_USER_AGENT,"MSIE 5.5")?"attachment; ":"").
     "filename=datei name.ext");
// eigentlich ueberfluessig, hat sich aber wohl bewaehrt
header("Content-Transfer-Encoding: binary");
// Zwischenspeichern auf Proxies verhindern
// (siehe weiter unten)
header("Cache-Control: post-check=0, pre-check=0");
// Dateigröße für Downloadzeit-Berechnung
header("Content-Length: {Dateigroesse}");
```

Diese Headerkombination sollte zuverlässig funktionieren. Bei der Vielzahl von Browsern, die sich nicht immer an die RFCs halten, ist jedoch nicht ausgeschlossen, daß das ganze angepaßt werden muß. Sollte jemand eine Kombination haben, die besser funktioniert, freue ich mich natürlich über eine Rückmeldung.

Ein letztes Wort noch zur Header-Kombination: Wie sich zeigte, funktioniert diese Download-Methode nicht mehr, wenn vor dem Senden o. g. Header schon bestimmte andere Header, wie die für das Nicht-Cachen (11.1.6), gesandt wurden. Man sollte also immer

auf die Reihenfolge der Header achten und sicherstellen, daß vor den Headern für den Download keine oder nur definitiv nicht störende Header verschickt werden.

11.1.5 Content-Type

Neben dem oben schon verwendeten Content-Type gibt es natürlich noch andere. So lange „nur“ „normale“ Webseiten ausgegeben werden, interessiert uns der Typ nicht. Wenn aber z. B. mit den PHP image functions ein Bild dynamisch erzeugt wird, was auch als Bild im Browser angezeigt wird, muß der Typ explizit angegeben werden. Die Tabelle 11.1 zeigt die wichtigsten Typen.

Bei statischen Dateien entscheidet der Webserver in der Regel anhand der Endung, welchen Typ er sendet. Normalerweise beachtet der Client den Typ, den der Server sendet. Es gibt jedoch IE Versionen, die der Meinung sind, anhand der Endung selbst besser entscheiden zu können, um was für eine Datei es sich handelt.

Bezeichnung	**Bedeutung**
application/octetstream	wird als Datei (Binärdaten) angesehen (wird gespeichert)
image/gif	GIF-Bild
image/jpeg	JPEG-Bild
image/png	PNG-Bild
text/html	wird als Webseite angesehen, HTML wird interpretiert
text/plain	wird als reiner Text angesehen, HTML wird nicht beachtet

Tabelle 11.1: Content-Type Typen

Die Bedeutung des Type können wir uns an den folgenden Dateien ansehen.

```
<?php

?>
<html>
<body>
<h1>Hallo Welt!</h1>
</body>
</html>
```

Die Datei ist im Endeffekt eine ganz normale Webseite, enthält zwar einen Bereich für PHP Anweisungen, der ist jedoch leer. Sie wird auch dem entsprechend im Browser angezeigt.

```
<?php
header("Content-Type: text/plain");
?>
<html>
<body>
<h1>Hallo Welt!</h1>
</body>
</html>
```

Nachdem das Script nun von sich behauptet, es wäre ein normaler Text, werden die HTML-Anweisung komplett mißachtet und der Text so wie er ist ausgegeben.

```
<?php
header("Content-Type: text/html");
?>
<html>
<body>
<h1>Hallo Welt!</h1>
</body>
</html>
```

Mit dem richtigen Typ ist die Welt aber wieder in Ordnung.

```
<?php
header("Content-Type: image/png");
?>
<html>
<body>
<h1>Hallo Welt!</h1>
</body>
</html>
```

Als Bild taugt der HTML-Code nun wirklich nicht und Netscape zeigt das Symbol für ein defektes Bild an.

```
<?php
header("Content-Type: application/octetstream");
?>
<html>
<body>
<h1>Hallo Welt!</h1>
</body>
</html>
```

Und den Octetstream will Netscape ordnungsgemäß als Datei abspeichern.

11.1.6 Cache

Über die richtigen Headerangaben kann auch das Verhalten der Proxies beeinflußt werden. So verhindert folgende Kombination zuverlässig das Zwischenspeichern.

```
header("Expires: -1");
header("Cache-Control: post-check=0, pre-check=0");
header("Pragma: no-cache");
header("Last-Modified: " . gmdate("D, d M Y H:i:s") . " GMT");
```

Zusätzlich braucht der IE ab Version 5 offenbar noch folgenden HTML-Code am **Ende** der Datei, d. h. zwischen `</body>` und `</html>`


```
<head>
<meta http-equiv="pragma" content="no-cache">
</head>
```

Das Nicht-Cachen sollte aber nur benutzt werden, wenn es wirklich notwendig ist, d. h. die Seite sich bei jedem Aufruf ändert. Es ist sonst nämlich zum einen für den Besucher nervig, da die Seite bei jedem Aufruf neu geladen werden muß - normalerweise kommt sie ja nach dem ersten Aufruf für eine gewisse Zeit aus dem Cache. Zum anderen belastet das Neuladen den Server unnötig. Davon abgesehen ignorieren viele Suchmaschinen Seiten, die nicht gecached werden dürfen, denn warum soll eine Seite aufgenommen werden, wenn die Seite von sich behauptet, daß sie sich ständig ändert? Von daher sollte die Seite, wenn sie in Suchmaschinen auftauchen soll, einen gescheiten `Last-Modified` und `Expires` Header senden.

Was tun bei einer Seite, die sich jede Minute ändert? Ganz einfach: Die entsprechenden Header richtig setzen. Die Tabelle 11.2 zeigt die entsprechenden Header. Es muß unterschieden werden zwischen den META-Tags, die im HTML stehen und den HTTP-Headern. Erstere werden nur vom Browser gelesen, letztere auch von Proxies.

Bezeichnung	**Bedeutung**
Cache-Control	Anweisungen für Proxies und Browser
Expires	Zeitpunkt, ab dem der Inhalt wahrscheinlich veraltet sein wird
Last-Modified	letzte Änderung des Inhalts

Tabelle 11.2: Cache Header

Cache-Control

Das RFC 2616 sagt zum Thema Syntax des `Cache-Control` (hier nur ein Ausschnitt):

```
Cache-Control   = "Cache-Control" ":" 1#cache-directive

    cache-directive = cache-request-directive
         | cache-response-directive

    cache-request-directive =
           "no-cache"                          ; Section 14.9.1
         | "no-store"                          ; Section 14.9.2

     cache-response-directive =
         | "no-cache" [ "=" <"> 1#field-name <"> ]; Section 14.9.1
         | "no-store"                             ; Section 14.9.2
```

Bei einem `Cache-Control: no-cache` Header muß der Proxy bei jeder Anfrage überprüfen, ob eine aktuellere Version vorliegt. Wenn dies nicht der Fall ist, kann er die gespeicherte Seite senden. Bei `Cache-Control: no-store` darf der Proxy die Seite nicht speichern, das heißt, sie muß bei jedem Aufruf neu übertragen werden.

Expires

Beim Expire-Header muß die Uhrzeit richtig formatiert sein und in Greenwich Mean Time[7]. Kleines Beispiel:

```
Expires: Thu, 01 Dec 1994 16:00:00 GMT
```

Diese Seite wäre bis zum 1. Dezember 1994 0 Uhr aktuell gewesen. Danach muß sie wieder auf Aktualität überprüft werden.

Last-Modified

Bei `Last-Modified` sollte die Zeit der letzten Änderung angegeben werden. Bei statischen Seiten wird dafür häufig das Änderungsdatum der Datei genommen. Bei dynamischen Seiten gestaltet sich die Sache etwas schwieriger.

Alternative: GET-Parameter

Eine Alternative zum Header ist, einen zufälligen GET-Parameter an die URL zu hängen. Zum Beispiel wird dann aus der URL *http://reeg.net/index.html* die Adresse *http://reeg.net/index.html?IRGENDETWAS*. Es muß natürlich gewährleistet sein, daß IRGENDETWAS immer etwas anderes ist.

11.2 URLs parsen

11.2.1 Beispiel: PHP-Manual

Niemand, der PHP programmiert, kennt alle Befehle und deren Syntax auswendig. Wozu auch - es gibt schließlich das PHP-Manual, ein kurzer Blick auf *http://php.net/* genügt. Die Betreiber dieser Site haben sich etwas besonders schlaues einfallen lassen, damit man sich nicht immer durch zig Unterseiten hangeln muß, bis man den Befehl gefunden hat, den man sucht: Man kann einfach den Namen des gesuchten Befehls hinter die o. g. URL hängen und landet automatisch auf der richtigen Seite des PHP-Manuals. Beispiel: `http://php.net/echo` zeigt die Beschreibung des Befehls `echo` an. Wie man sich denken kann, wurde diese Annehmlichkeit mit einem PHP-Script realisiert. Im Folgenden werde ich also erst einmal zeigen, wie dieser Trick funktioniert und im zweiten Schritt dann noch ein anderes Beispiel geben.

Doch nun los: Zuerst einmal muß man verstehen, was passiert, wenn der Webserver, der die HTML-Seiten (auch die durch PHP-Scripte dynamisch erzeugten!) an den Client (d. h. Browser) schickt, eine Seite nicht findet. Dann nämlich schickt er den Status-Code 404 (siehe 11.1.2) in Verbindung mit einer Fehlerseite, die dann im Browser angezeigt wird. Diese kann man abfangen und statt der Standard-Fehlerseite eine eigene angeben - und das ist es, was wir hier ausnutzen möchten.

Um einem Apache-Webserver mitzuteilen, daß er beim Status-Code 404 zu einer anderen als der Standard-Fehlerseite umleiten soll, erstellt man eine Datei namens `.htaccess` mit folgendem Inhalt:

[7] kurz GMT oder auch UTC (Universal Time)


```
ErrorDocument 404 /pfad/zur/alternativseite.php4
```

Auf diese Weise wird an Stelle der normalen Fehlerseite die Alternativseite aufgerufen, wobei der Benutzer davon nichts mitbekommt. In dieser Seite kann man dann in dem Fall, daß mit Status-Code 404 umgeleitet wurde, auf den gesuchten letzten Teil der URL wie folgt zugreifen:

```
<?php
  if ($_SERVER["REDIRECT_STATUS"]==404) {
    $keyword = substr($_SERVER["REDIRECT_URL"],
      strrpos($_SERVER["REDIRECT_URL"],"/")+1);
  }
?>
```

Mit etwas Erfahrung sieht man dem Script direkt an, daß es einfach alles abschneidet, was hinter dem letzten Slash `/` kommt, und es in die Variable `$keyword` schreibt. Letztere kann man nach Belieben im weiteren Scriptverlauf auswerten (und natürlich auch ganz anders nennen!). Im Falle des PHP-Manuals wird aus dieser Information eine neue URL zusammengestellt, was recht leicht zu bewerkstelligen ist, da die meisten Dateinamen der Seiten des Manuals die Form `function.FUNKTIONSNAME.html` haben - lediglich Underscores (`_`) werden durch das Minuszeichen ersetzt. Mit der neuen URL wird dann eine einfache Header-Weiterleitung (siehe 11.1.1) durchgeführt.

11.2.2 Anderes Beispiel: Akronyme

Angenommen, du hast eine recht bekannte Homepage, die auf einer serverseitigen Datenbank (z. B. mit MySQL) aufsetzt. Hier soll einmal eine Akronym-DB als Beispiel dienen. Nun möchtest du vielleicht den Besuchern dieser Seite die Möglichkeit geben, Akronyme schnell und einfach nachzuschlagen, indem sie ein gesuchtes Akronym einfach ans Ende der ihnen schon bekannten URL hängen. Mit http://akronyme.junetz.de/ als Basis-URL ergibt sich so z. B. die Möglichkeit, das Akronym `ROTFL` nachzuschlagen, indem man einfach `http://akronyme.junetz.de/rotfl` eintippt.

Doch wie soll das funktionieren? „Soll ich etwa für jede mögliche Eingabe eine eigene Seite bauen?“ höre ich euch schon rufen. Natürlich nicht, wofür dann eine Datenbank benutzen?! Das geht viel eleganter:

Erstelle eine Datei namens `.htaccess` mit folgendem Inhalt:

```
ErrorDocument 404 /akronyme/index.php4
```

Wobei hier `/akronyme/index.php4` der absolute Pfad (relativ zur Server-HTTP-Wurzel) zu unserem Akronyme-Skript ist. Die weitere Behandlung der `REDIRECT_URL` wurde bereits im vorigen Beispiel beschrieben.

12 Reguläre Ausdrücke

Was sind denn überhaupt reguläre Ausdrücke[1]? Es gibt Leute, die finden reguläre Ausdrücke fast so genial wie die Erfindung des geschnittenen Brotes: Im Prinzip sind reguläre Ausdrücke Suchmuster, die sich auf Strings (Zeichenketten) anwenden lassen. Auf den ersten Blick sehen sie etwas kryptisch aus, was sie durchaus auch sein können. Was sie aber so sinnvoll macht ist ihre Fähigkeit, komplexe Suchen durchzuführen - mehr dazu im folgenden.

Anfängern empfehle ich, dieses Kapitel erstmal zu überspringen. Alles, was man mit regulären Ausdrücken realisieren kann, kann auch mit normalen Befehlen programmiert werden, ist dann aber natürlich etwas länger. Mit regulären Ausdrücken ist es einfach eleganter.

In Unix-Programmen, vor allem in Perl[2], werden die regulären Ausdrücke sehr gerne benutzt, wenn normale Suchmuster nicht mehr ausreichen. Auch wenn einfache Suchmuster noch relativ einfach zu erstellen sind, kann das beliebig kompliziert werden. Nicht umsonst gibt es dicke Bücher, die sich nur mit diesem Thema beschäftigen.

Wann sollen denn nun die regulären Ausdrücke eingesetzt werden? Die Antwort ist ganz einfach: Wenn man nach Mustern suchen will, bei denen die normalen Funktionen nicht mehr ausreichen. Wenn hingegen die normalen String-Funktionen, wie z. B. str_replace() oder strpos(), ausreichen, sollten diese auch benutzt werden, weil sie schneller abgearbeitet werden.

PHP kennt zwei verschiedene Funktionsgruppen von regulären Ausdrücken, die ereg*- und die preg*-Funktionen. Erstere sind von Anfang an dabei, halten sich an die POSIX-Definition, sind aber langsamer und nicht so leistungsfähig wie die letzteren, die sich an Perl orientieren. Von daher sollten, wenn möglich, die preg*-Funktionen verwendet werden. In Konsequenz werde auch ich hier die Perl-kompatiblen Funktionen verwenden; diese werden auch häufig als PCRE[3]-Funktionen bezeichnet.

Mit regulären Ausdrücken ist es wie mit Programmen (und Dokus ;-)): Es ist eine Kunst, gut, verständlich, fehlerfrei und effizient zu schreiben, und mit der Zeit perfektioniert man sein Können. Von daher: Nicht verzweifeln, wenn es am Anfang nicht funktioniert oder man Minuten braucht, bis dieser 20 Zeichen lange Ausdruck steht - üben, probieren und nochmals üben. Irgendwann geht es.

12.1 einfache Suchmuster

Das Suchmuster muß bei den preg*-Funktionen von Begrenzungszeichen (sog. Delimiter) eingeschlossen werden. Häufig werden der Schrägstrich (Slash) / oder das Gleichheitszeichen = verwendet. Im Prinzip kann jedes Zeichen benutzt werden, solange es nachher nicht

1 Engl.: regular expressions
2 verbreitete Scriptsprache unter Unix
3 Perl Compatible Regular Expressions

im Suchmuster verwendet wird, bzw. immer mit einem Backslash \ escaped wird.

Z. B. sucht das Muster /<title>/ nach dem HTML-Tag <title>. Wenn man jetzt aber Header-Tags (<h1>,<h2>,...,<h6>) finden will, funktioniert das so nicht mehr. Hier braucht man einen Platzhalter. Das, was auf der Kommandozeile der Stern * und in SQL bei LIKE das Prozentzeichen % ist, ist hier der Punkt .. Das Muster /<h.>/ würde auf alle HTML-Tags zutreffen, bei denen auf das h ein beliebiges Zeichen folgt. Wir wollten aber nur die Zahlen 1-6. Mit \d steht ein Platzhalter für eine beliebige Zahl zur Verfügung. /<h\d>/ trifft nur noch die HTML-Tags <h0>,<h1>,<h2>,...,<h9>. Das ist zwar schon besser, aber auch noch nicht perfekt. Man bräuchte genau die Menge der Zahlen 1,2,3,4,5,6. Dies ist selbstverständlich auch möglich: Der Ausdruck /<h[123456]>/ bewirkt das Gewünschte, er läßt sich aber noch verschönern zu /<h[1-6]>/.

Wie wir gesehen haben, kann mit Hilfe der eckigen Klammern [] eine Menge von Zeichen definiert werden, indem man entweder alle aufzählt oder den Bereich angibt. Wenn man das Dach ^ als erstes Zeichen angibt, wird die Auswahl negiert.

.	ein beliebiges Zeichen (bis auf Newline \n)
[] z. B. [1-6]	die angegebenen Zeichen bzw. Bereiche die Zahlen 1, 2, 3, 4, 5 und 6
[^] z. B. [^1-6]	als erstes Zeichen wird die Auswahl negiert alle Zeichen bis auf die Zahlen 1-6
\d	Dezimalziffer
\D	alle Zeichen bis auf die Dezimalziffern
\s	Whitespace (Leerzeichen, Tabulator etc.)
\S	alle Zeichen außer Whitespace
\w	alle "Wort"-Zeichen (Buchstaben, Ziffern, Unterstrich)
\W	alle "Nicht-Wort"-Zeichen

Tabelle 12.1: Zeichenmengen

\n	Zeilenumbruch (LF)
\r	Wagenrücklauf (CR)
\\	Ein \
\t	Tabulator

Tabelle 12.2: Sonderzeichen bei den PCRE

Das Dach ^ hat auch im Muster eine besondere Bedeutung: es markiert den Stringanfang. Der Ausdruck /^Hallo/ paßt auf alle Strings, bei denen das Wort ‚Hallo' am Anfang steht, nicht jedoch in der Mitte. Neben dem Anfang ist natürlich auch das Ende interessant, das mit einem Dollar $ markiert wird. Das Suchmuster /Welt!$/ paßt demnach auf alle Strings, die mit dem Wort „Welt!" enden. Das läßt sich natürlich auch kombinieren: Mit /^Hallo Welt!$/ findet man genau die Strings „Hallo Welt!" .

Zur Vertiefung und Übung habe ich versucht, ein paar Beispiele zu erfinden. Diese befinden sich bei den Übungen am Ende des Kapitels.

12.2 Quantifizierer

Mit dem bisher Gesagten kann man zwar schon schöne Ausdrücke schreiben, aber irgendwie fehlt noch etwas. Wie die letzte Übung gezeigt hat, ist es doch relativ umständlich, mehrere aufeinander folgende Zeichen anzugeben. Selbstverständlich ist auch das möglich: hier hilft der Quantifizierer (auch Quantor genannt). Man unterscheidet die folgenden Typen:

Da wäre als erstes der altbekannte Stern `*`, der für „keinmal" oder „beliebig oft" steht. Das Suchmuster `/^\d*$/` würde z. B. alle Strings, die aus keiner oder beliebig vielen Ziffern bestehen, finden. Wie man an dem Beispiel sieht, muß der Quantifizierer immer hinter dem Zeichen bzw. der Zeichenmenge stehen. Neben dem Stern `*`, Allquantor genannt, gibt es noch das Fragezeichen `?`, das für keinmal oder einmal steht und Existenzquantor genannt wird, sowie das Pluszeichen `+` (mind. einmal).

`?`	kein-/einmal
`+`	mind. einmal
`*`	keinmal oder beliebig oft
`{n}`	genau n-mal
`{min,}`	mind. *min*-mal
`{,max}`	keinmal bis höchsten *max*-mal
`{min,max}`	mind. *min*, aber höchstens *max*-mal

Tabelle 12.3: Quantifizierer

Wenn man aber genau fünf Zahlen oder zwischen drei und sieben kleine Buchstaben haben will, reichen die bisherigen Quantifizierer dann doch noch nicht. Deshalb gibt es noch die geschweiften Klammern `{}`, in die man die gewünschten Zahlen eintragen kann. Das Suchmuster für genau fünf Zahlen sieht z. B. so aus: `/\d{5}/`; das für die drei bis sieben kleinen Buchstaben so: `/[a-z]{3,7}/`.

Auch hier gibt es zur Vertiefung Übungen am Ende des Kapitels.

12.3 Gruppierungen

Die runden Klammern `( )` haben auch eine besondere Bedeutung: Der Hauptzweck ist, den eingeklammerten Bereich zur späteren Verwendung zu markieren. Die Klammern kommen auch beim wiederholten Vorkommen eines Teilaudrucks hintereinander zum Einsatz. Z. B. würde ein Ausdruck, der auf drei zweistellige Zahlen, die mit einem Doppelpunkt anfangen, paßt, ohne Klammern so aussehen `/:\d{2}:\d{2}:\d{2}/` . Mit Klammer wird das ganze übersichtlicher: `/(:\d{2}){3}/` .

Die markierten Teilbereiche eines Ausdruckes werden auch Backreferences genannt. Sie werden von links nach rechts durchnumeriert. Innerhalb eines Ausdruckes kann man sich mit `\1` auf den Text, der von der ersten Klammer eingeschlossen wird, beziehen, mit `\2` auf den zweiten usw. Der Ausdruck `/(\w)\s\1/` paßt immer dann, wenn zweimal dasselbe Wort nur durch ein Leerzeichen getrennt vorkommt.

Zur allgemeinen Verwirrung gibt es auch hier am Ende des Kapitels Übungen.

12.4 Optionen

Die ganzen Ausdrücke können sich je nach angegebener Option auch noch anders verhalten. Die Optionen werden hinter dem letzten Delimiter angegeben. So bewirkt z. B. ein `i`, daß nicht auf Groß-/Kleinschreibung geachtet wird. Der Ausdruck `/hallo/i` paßt auf „Hallo", „hallo", „HALLO" oder jede andere Kombination von Groß-/Kleinbuchstaben. Eine Übersicht zu den Optionen gibt die Tabelle 12.4.

`i`	Es wird nicht auf Groß-/Kleinschreibung bei Buchstaben geachtet.
`m`	Normalerweise betrachtet PHP bei den PCRE, wie auch Perl, den String als eine Zeile, das heißt das Dach `^` paßt nur auf den Anfang des Strings und das Dollar `$` auf das Ende des Strings. Wenn diese Option gesetzt ist, paßt das `^` auf jeden Zeilenanfang, wie auch auf den String-Anfang. Beim `$` gilt das Ganze analog für das Ende. Wenn es keine Zeilenumbrüche in dem Text gibt oder im Ausdruck kein `^` bzw. `$` verwendet wird, hat diese Option logischerweise keine Funktion.
`s`	Ohne diese Option paßt der Punkt `.` auf alle Zeichen, außer den Zeilenumbruch `\n`. Mit gesetzter Option paßt der Punkt auf alle Zeichen, inklusive Zeilenumbruch. In negierten Mengen, wie z. B. `[^a]`, ist der Zeilenumbruch immer enthalten, unabhängig von der Option.
`e`	Mit dieser Option wird der zu ersetzende Text bei `preg_replace()` als PHP-Code aufgefaßt und entsprechend ausgewertet.
`A`	Bei Setzen dieser Option, wird der Ausdruck auf den Anfang des Strings angewandt. Dieses Verhalten kann auch durch entsprechende Konstrukte im Ausdruck erreicht werden.
`U`	Normalerweise sind die Quantifizierer greedy (gierig), das heißt, sie versuchen immer den größtmöglichen Text zu treffen. Mit Hilfe dieser Option wird auf ungreedy umgeschaltet, das heißt, es wird immer der kürzestmögliche Text genommen.

Tabelle 12.4: Optionen für reguläre Ausdrücke

12.5 Übungen

12.5.1 einfache Suchmuster

Passen die folgenden Suchmuster auf die beiden Texte oder nicht? Text1=„Hallo Welt!", Text2=„PHP ist einfach genial und die Regex sind noch genialer". Die Lösung befindet sich in Anhang B.6.1.

12.5.1.1

`/hallo/`

12.5.1.2

`/[^0-9A-Z]$/`

12.5.1.3

`/^[HP][Ha][lP]/`

12.5.1.4

`/\w\w\w\w\w/`

12.5.1.5

`/\w\w\w\w\w\w/`

12.5.2 Quantifizierer

Hier gilt dieselbe Aufgabenstellung wie in der vorigen Aufgabe

12.5.2.1

`/^\w* \w*$/`

12.5.2.2

`/^\w* \w*!$/`

12.5.2.3

`/^\w* [\w!]*$/`

12.5.2.4

`/\w{4} \w+/`

12.5.2.5

`/\w{4} \w+$/`

12.5.2.6

Diesmal darfst du selbst nachdenken: Schreibe einen Ausdruck, der überprüft, ob eine Log-Zeile dem Standard-Log Format des Apache-Servers entspricht. Eine Zeile kann z. B. so aussehen: `192.168.1.1 - - [01/Apr/2001:08:33:48 +0200] "GET /test.php4 HTTP/1.0" 200 3286` Die Bedeutung der Felder ist hier nicht wichtig, kann aber trotzdem interessant sein: Als erstes steht die IP-Adresse bzw. der Rechnername, von dem die Anfrage kam. Die nächsten beiden Felder sind der Username (vom identd bzw. von auth), beide Felder müssen aber nicht existieren. In den eckigen Klammern steht das genaue Datum mit Zeit und Zeitzone. In den Anführungszeichen steht die angeforderte Datei mit der Methode (kann auch POST sein) und dem Protokoll (kann auch HTTP/1.1 sein). Als vorletztes wird der Status angegeben (200 ist OK) und das letzte Feld sagt, wie viel Daten bei diesem Aufruf übertragen wurden.

12.5.3 Gruppierungen

12.5.3.1

Schreibe einen Ausdruck, der überprüft, ob bei einer Preisangabe der DM-Betrag gleich dem Pfennig-Betrag ist. Die Preisangabe sieht vom Format her so aus: `99,99DM`.

13 Fehlersuche

> Fehlschläge sind die Würze, die dem Erfolg sein Aroma geben.

Auch wenn man sehr gut programmieren kann und sowohl PHP als auch SQL „im Schlaf" beherrscht, kommt es vor, daß es „nicht so tut" wie man will. In solchen Fällen ist es praktisch, wenn man weiß, wie man auf Fehlersuche gehen kann.

Als erstes sollte man sich natürlich überlegen, in welcher Reihenfolge das Programm was wie tun sollte und dann überprüfen, bis zu welchem Punkt es funktioniert. Um das festzustellen, gibt es verschiedene Methoden; am sinnvollsten ist es i. d. R., zu überprüfen, ob die Variablen die Werte haben, die sie haben sollten. Dazu kann man sie z. B. einfach mit `echo` ausgeben. Bei Arrays kann die Funktion `var_dump($var)` sehr praktisch sein. `var_dump` gibt nämlich das gesamte Array mit Index und Werten aus. Da es aber in HTML vorkommen kann, daß Text lediglich im Quelltext der Seite sichtbar wird (innerhalb von Tabellenkonstrukten zum Beispiel), ist es sinnvoll, einfach noch einen festen String mit auszugeben; auf diese Weise weiß man nachher auch noch, welche Variable es genau war. Als Beispiel haben wir die Variablen ‘VAR1‘ und ‘VAR2‘:

```
echo $VAR1.'AAA';
echo $VAR2.'BBB';
```

Für reine Testzwecke hat es sich auch als hilfreich erwiesen, um die Testausgabe der Variablen herum einen HTML-PRE-Block zu setzen. Das bewirkt dann mit Sicherheit eine Darstellung der Ausgabe auch auf dem Bildschirm:

```
echo "<pre>";
echo $VAR1." ".$VAR2;
echo "</pre>";
```

Die dritte Möglichkeit schließlich findet dann Anwendung, wenn man mehrere Variablen, z. B. von POST/GET-Daten oder Servervariablen, im Zusammenhang sehen will und vielleicht die Namen einiger dieser Variablen gar nicht kennt. In diesem Fall kann man auf die PHP-interne Funktion `phpinfo()` zurückgreifen, die eine komplette HTML-Seite voll mit datenüberfüllten Tabellen ausgibt. Hierüber lassen sich dem System auch „intime Details" über die Serverkonfiguration, den User-Agent (Browser) uvm. entlocken.

Bei `IF`-Abfragen z. B. kann es auch passieren, daß Anweisungen gar nicht erst ausgeführt werden. Um das zu überprüfen, setzt man einfach ein ‘echo‘ in die Abfrage hinein. Zum Beispiel:

```
$i=1;
echo $i.'CCC';
if (i==1){
```



```
  echo 'DDD';
  ....
}
```

Bei dieser Abfrage würde man auf den ersten Blick davon ausgehen, daß ‘1CCCDDD‘ ausgegeben wird, da $i, durch das erste ‘echo‘ bewiesen, den Wert 1 hat. Da man nun aber weiß, daß etwas mit der `IF`-Abfrage nicht stimmt, kann man auch einen zweiten Blick riskieren und sieht dann, daß beim `i` das $ fehlt.

Ein anderes Problem tritt häufig auch im Zusammenhang mit Strings auf - warum wird wohl bei folgendem String nicht das gewünschte ausgegeben:

```
$text = 'Alternativtext';
echo "<img src=\"bild.gif\" width=\"10\" height=\"10\"";
echo " alt=\".$text.\">";
```

Oder warum bekommen wir hier eine Fehlermeldung:

```
$text = 'Alternativtext';
echo "<img src=\"bild.gif\" width=\"10\" height="10"";
echo " alt=\"".$text."\">";
```

Ganz einfach: Das Escapen stimmt nicht. Im oberen Beispiel sind hinter ‘alt=‘ zwar die doppelten Anführungsstriche escaped, aber da direkt im Anschluß daran mit Hilfe des Stringoperators Strings verbunden werden, müßten an dieser Stelle noch einmal doppelte Anführungsstriche stehen, genauso wie bei der zweiten Stringverbindung. Im unteren Beispiel wurden zwar die Strings korrekt beendet und wieder angefangen, dafür wurden die Anführungszeichen um die ‘10‘ nicht escaped. Komplett richtig muß es so aussehen:

```
$text = 'Alternativtext';
echo"<img src=\"bild.gif\" width=\"10\" height=\"10\"";
echo " alt=\"".$text."\">";
```

Solcherlei Vergeßlichkeiten sind nicht selten Grund genug für eine langwierige Fehlersuche. Man kann dies aber von vorne herein vermeiden, indem man gleich gut nachdenkt und dabei alle doppelten Anführungszeichen, die wirklich ausgegeben werden sollen, escaped. :-)

Häufig funktioniert zwar der komplette PHP-Teil, die SQL-Anweisungen jedoch nicht. In diesen Fällen gebe ich selbst einfach den Abfrage-String mit ‘echo‘ aus.[1] Dann sieht man nämlich schon häufig den Fehler – ein häufiger ist übrigens, daß ein Feld einen String erwartet (z. B. die SQL-Typen `TEXT` und `VARCHAR`), in der Abfrage jedoch die für Strings charakteristischen Hochkommata vergessen wurden. Wenn man ihn jedoch nicht findet, benutzt man einfach den String und gibt ihn direkt am SQL-Prompt[2] ein. Wenn es dann immer noch nicht funktioniert, kürzt man die Abfrage solange, bis sie funktioniert. Die andere Richtung funktioniert natürlich auch, d. h. man konstruiert sich die Abfrage Stück für Stück zusammen und probiert, wie lange sie noch funktioniert und hat dann hoffentlich den Fehler.

[1] Hat man die Tips für guten Stil aus Kapitel 8.8.2 angewandt und `sprintf()` benutzt, kann man hier einfach testweise das ‚s‘ entfernen. :-)

[2] Oder z. B. in PHPMyAdmin, siehe Kapitel 7.3

Damit wären wir auch schon beim nächsten Punkt der Fehlersuche angelangt: Wenn etwas nicht laufen will, ersetzt man die Variablen durch Konstanten und probiert. Auf diese Weise ist nämlich eine Fehlerquelle (falsche Werte bei den Variablen) ausgeschlossen. Bei großen PHP-Dateien ist es teilweise sinnvoll, den nicht funktionierenden Teil in eine Datei zu kopieren, denn nicht selten werden Anweisungen durch andere beeinflußt, die eigentlich gar nichts miteinander zu tun haben (sollten).

Wenn es immer noch nicht funktioniert, hilft es vielleicht, wenn ein Freund, der auch etwas programmieren kann (das muß nicht einmal PHP sein) das Ganze mal begutachtet - denn vier Augen sehen mehr als zwei.

Jeder hat natürlich seine eigene Art, eine Fehlersuche anzugehen. Ich selbst habe hier nur kurz zeigen wollen, wie ich es mache (und damit bisher immer zum Ziel gekommen bin!). Wahrscheinlich habe ich wieder die Hälfte vergessen, aber du kannst mich natürlich auch auf bessere Methoden aufmerksam machen (gilt übrigens für die gesamte Anleitung)!

13.1 Übungen

Man lernt die Fehlersuche nirgendwo so gut wie beim Selbermachen. Ich habe hier mal versucht, ein paar "typische" Fehler zu sammeln. Für alle Übungen gelten folgende Fragen:

- Was soll das Programm eigentlich tun? Was soll es ausgeben?
- Was macht das Programm?
- Gibt es eine Fehlermeldung, und wenn ja, welche?
- Wo liegt der Fehler?
- Kann man durch die Programmierweise diesen Fehler vermeiden?

Die Lösungen befinden sich im Anhang B.7.

13.1.1 Ein komisches IF

Teil 1

```
<?php
$i = 0;
if ($i = 1){
  echo "Die Variable $i hat den Wert 1";
}
else {
  echo "Die Variable hat nicht den Wert 1";
}
```


Teil 2

```
$i = 0;
if ($i = 0){
  echo "Die Variable $i hat den Wert 0";
}
else {
  echo "Die Variable hat nicht den Wert 0";
}
?>
```

13.1.2 Fehler in einer leeren Zeile?

Die Zeilennummern sind hier nur der Übersichtlichkeit halber eingefügt worden. Sie stehen natürlich nicht in der PHP-Datei.

```
<?php
for ($i=1; $i<=5; $i++){
echo "$i<br>\n";
echo "Das waren die Zahlen 1-5";
?>

```

13.1.3 Wieso fehlt ein ';' wo eins ist?

Wie im vorigen Beispiel sind die Zeilennummern wieder nur der Übersichtlichkeit halber eingefügt worden.

```
<?php
for ($i=1; $i<=5; $i++){
  echo "$i<br>\n;
}
echo "Das waren die Zahlen 1-5";
?>
```

14 PHPDOC

Beim Programmieren, unabhängig von der verwendeten Sprache, ist nicht nur die Funktionalität und Effizienz (Qualität) des Codes wichtig, sondern meist auch die Verständlichkeit und Wiederverwendbarkeit. Das liegt darin begründet, daß teilweise mehrere Programmierer an einem Projekt arbeiten, zu dem der jeweilige Quelltext gehört; auch wird oft der Code stetig erweitert und verbessert oder es müssen Fehler gefunden und behoben werden.

In jedem Fall ist es also notwendig, daß der Code übersichtlich ist. Darüber hinaus ist es jedoch unerläßlich, den Quelltext ausführlich zu kommentieren. I. A. ist es hier dem jeweiligen Programmierer überlassen, wie er dies anstellt. Es hat sich allerdings gezeigt, daß eine standardisierte Vorgehensweise beim Kommentieren von Vorteil sein kann: Jeder Programmierer braucht die Syntax dieses Standards nur einmal erlernen und kann ihn dann sowohl selbst anwenden als auch schnell wiedererkennen, wenn es gilt, Quelltexte anderer zu verstehen.

Mit der Programmiersprache Java wurde ein solcher Standard eingeführt und JavaDoc getauft. Dem JDK[1] liegt ein gleichnamiges Programm bei, das beliebige Klassen – Java ist eine durch und durch objektorientierte Sprache – durchparst und die enthaltenen standardisierten Kommentare so aufbereitet, daß eine HTML-Übersicht[2] entsteht, die ebenfalls eine einheitliche Form aufweist: Die offizielle Java-Dokumentation wird ebenfalls mit JavaDoc erstellt.

Einige PHP-Programmierer haben diese Idee aufgegriffen und für die Scriptsprache angepaßt. Dabei herausgekommen ist PHPDOC – eine standardisierte Methode, PHP-Funktionen und Klassen[3] zu kommentieren. Mit dem PHPDOC-Programm lassen sich dafür, ganz ähnlich wie mit JavaDoc, HTML-Übersichten erzeugen. Doch auch ohne das PHPDOC-Programm einzusetzen, wird der Code bei sachgemäßer Anwendung von PHPDOC sehr viel übersichtlicher. Es finden sich im Internet verschiedene PHPDOC-Pakete, wobei folgende Varianten zu unterscheiden sind:

- Der phpDocumentor von *http://www.phpdoc.org/* ist PHP-basiert und recht aktuell. Er scheint der einzige zu sein, der noch weitergepflegt wird.
- Unter *http://www.phpdoc.de/*, *http://sourceforge.net/projects/phpdoc/* und *http://wwpakis.free.fr/php3/phpdoc/index_en.html* gibt es noch eine weitere Varianten, diese werden aber offensichtlich schon lange nicht mehr weiterentwickelt.
- Es gibt von Christian Calloway eine Java-basierte Lösung, die den Vorteil hat, daß sie auf das Original-JavaDoc zurückgreift und somit dessen umfangreiche Ausgabemöglichkeiten bietet. Allerdings gibt es die Webseite nicht mehr und Google findet keine neue.

[1] Java Development Kit
[2] Es ist auch möglich, die Übersicht in anderen Formaten wie PS oder PDF auszugeben ...
[3] PHP erlaubt ja, in beschränktem Umfang, ebenfalls OOP [20]!

- In den neueren Versionen unterstützt auch Doxygen (*http://www.doxygen.org/*) PHP.

Jede dieser Lösungen ist leider etwas anders und funktioniert auch unterschiedlich gut. Auf die Unterschiede gehe ich weiter unten noch näher ein, wobei ich nur den phpDocumentor im folgenden betrachten werde.

Doch nun zur Vorgehensweise. Grundsätzlich stehen alle PHPDOC-Kommentare in C-Kommentarzeichen (`/* C-Kommentar */`); an das einleitende Zeichen wird jedoch in leichter Abwandlung ein zusätzlicher Stern angehängt: `/** PHPDOC-Kommentar */`

Das folgende, etwas komplexere Beispiel zeigt einen typischen PHPDOC-Kommentar für die ebenfalls exemplarische Funktion `foo`.

```
/**
 * foo: liefert -$bar zurück.
 *
 * @author      Jens Hatlak
 * @version     1.0
 * @param       $bar     Ein beliebiger Wert
 * @return      Der negative Eingabewert
 */
 function foo($bar) {
   return -$bar;
 }
```

Da erst `*/` den mit `/**` begonnen Kommentar wieder beendet, kann dazwischen alles andere stehen, auch einzelne Sterne `*` wie am Anfang jeder Folgezeile; sie dienen der besseren Abgrenzung des Kommentars vom Code. Üblicherweise beginnt der Kommentar dann auch in der zweiten Zeile mit der Beschreibung der Funktion/Klasse/Methode. Der erste Satz wird auch in der Übersicht genutzt und der Rest erscheint dann bei der eigentlichen Beschreibung. Diese kann sich über mehrere Zeilen hinziehen und sollte die Funktionalität des zu kommentierenden Folgecodes im Kern dokumentieren.

Nach einer zusätzlichen „Leerzeile[4]“ folgen dann die einzelnen Angaben zur Codebeschreibung. Hierzu werden Schlüsselworte benutzt, die grundsätzlich mit einem `@` eingeleitet werden und ansonsten aus der Tabelle 14.1 zu ersehen sind.

In JavaDoc gibt es übrigens noch die Schlüsselworte `@deprecated`, `@exception` und `@throws` sowie `@package` und `@subpackage`, die allerdings in PHPDOC in Ermangelung entsprechender Konstrukte in PHP nicht benötigt werden. Da PHP (zumindest in der aktuellen Version 4.x) keine Zugriffsrechte kennt, macht auch die Verwendung des `@access`-Schlüsselwortes entsprechend wenig Sinn.

14.1 phpDocumentor

Ein Aufruf des phpDocumentors, der derzeit nur unter Linux/Unix lauffähig ist, erfolgt z. B. mit `phpdoc -f script.php -d html` – das würde die Datei `script.php` im aktuellen

[4] Es sollte natürlich, wie oben beschrieben, doch ein Stern an passender Stelle stehen ...

Tabelle 14.1: PHPDOC-Schlüsselworte

Schlüsselwort	**Bedeutung**
@author	Autor des Codeabschnitts
@version	Version des Codeabschnitts
@since	eine Version oder ein Datum
@access	Zugriffsrechte: private oder public
@copyright	z. B. Name und Datum
@see	zu verlinkendes, ebenfalls dokumentierbares Element
@link	eine weiterführende URL
@param	Parameter (Wert und Typ, ggf. auch Angabe von „optional“) der Funktion bzw. Methode in der Reihenfolge der An- bzw. Übergabe
@return	Typ des Rückgabewerts der Funktion bzw. Methode

Verzeichnis parsen und die HTML-Übersicht im (ggf. neu zu erstellenden) Unterverzeichnis `html` erzeugen.

Teil IV

Beispiele

15 Einfaches Gästebuch

Dieses Beispiel-Gästebuch soll wirklich ganz einfach gehalten sein, es soll nur zeigen, wie ein einfaches Script aufgebaut ist. Es werden keinen Funktionen oder gar Objektorientierung benutzt. Das wäre hier Overkill. Entsprechende Beispiele kommen später noch.

Was muß so ein Gästebuch eigentlich können? Im Prinzip nur vorhandene Einträge anzeigen und neue Einträge entgegen nehmen. Das Speichern der Einträge machen wir hier der Einfachheit halber einmal in einer Datenbank.

Es stellt sich also zuerst die Frage, wie die Tabelle aussehen muß. Was für Informationen müssen denn gespeichert werden? Bei einem einfachen Gästebuch sollen mal der Name des Eintragenden und der eigentliche Text, sowie das Eintragedatum reichen. Daraus läßt sich eigentlich schon unser `create table` bestimmen.

```
CREATE TABLE meldung(
  id            int not null primary key auto_increment,
  datum         datetime,
  name          varchar(200),
  eintrag       text
);
```

Ich habe hier noch eine eindeutige ID eingefügt, damit man einen Eintrag auch später noch eindeutig identifizieren kann.

Nun stellt sich die Frage: Was programmiert man zuerst? Erst die Eingabe oder die Ausgabe? Ohne Daten kann man die Ausgabe nicht testen, aber ohne Ausgabe auch nicht vollständig die Eingabe. Wenn man die Eingabe (Formular und Eintragen in die Datenbank) programmiert, muß man irgendwie überprüfen, ob das Script die Daten auch richtig in die Datenbank schreibt. Das kann man z. B. per geeignetem `select` am MySQL-Prompt erledigen. Um die Ausgabe zu testen, braucht man natürlich irgendwelche Daten. Auch diese könnte man zum Testen von Hand per passendem `insert` in die Datenbank schreiben. Dies ist in der Regel jedoch aufwendiger, so daß es sich empfiehlt, erst die Eingabe und dann die Ausgabe zu programmieren.

Wer gut ist, kann natürlich auch beides gleichzeitig programmieren und damit beide Scripte auf einmal testen. Problematisch wird es allerdings im Fehlerfall. Bei unserem Gästebuch zum Beispiel: Nehmen wir an, die Ein-/Ausgabe ist soweit programmiert, aber bei der Ausgabe bleibt das Feld mit dem Kommentar leer. Wo liegt dann der Fehler? Einerseits kann er bei der Eingabe liegen (dann steht nichts in der Datenbank) oder bei der Ausgabe (dann wird der korrekt in der DB stehende Text nicht ausgegeben).

Nun aber zu unserem Script. Als erstes nun die Eingabe. Im Prinzip sollte es durch die Kommentare selbst erklärend sein. Eine nette Spielerei ist die Weiterleitung nach erfolgtem Eintrag. Wie wir jedoch seit Kapitel 11.1.1 wissen, muß die angegebene Adresse absolut sein. Hier werden die von PHP zur Verfügung gestellten Umgebungsvariablen genutzt, um automatisch die aktuelle Adresse zu ermitteln und dorthin weiterzuleiten. Die Alternative wäre gewesen, die Adresse „hart“ reinzuschreiben und dann hätte man Probleme, wenn sich die Adresse des Scripts ändert.

```
<?php
$DBHost    = "localhost";
$DBName    = "reeg";
$DBUser    = "reeg";
$DBPasswd = "super_geheim";

// Verbindug zu DB-Server herstellen
mysql_connect($DBHost, $DBUser, $DBPasswd)
    OR die("Konnte DB-Server nicht erreichen");
mysql_select_db($DBName);

if (isset($_GET["submit"])){
// Der Submit - Button wurde gedrückt
//             -> die Werte müssen überprüft
// und bei Gültigkeit in die DB eingefügt werden

  // wir gehen von der Gültigkeit der Daten aus
  $DatenOK = 1;

  // es gab noch keine Fehlermeldung
  $error = "";

  if (!isset($_GET["name"])){
    // es wurde kein Name eingegeben
    $DatenOK = 0;
    $error .= "Es muß ein Name eingegeben werden<br>\n";
    $daten["name"] = "";
  }
  else {
    $daten["name"] = $_GET["name"];
  }

  if (!isset($_GET["eintrag"])){
    // es wurde kein Komementar eingegeben
    $DatenOK = 0;
    $error .= "Ein Eintrag ohne Komemntar mach nicht viel";
    $error .= " Sinn, oder?<br>\n";
    $daten["eintrag"] = "";
  }
  else {
    $daten["eintrag"] = $_GET["eintrag"];
  }
```



```
  if ($DatenOK){
    // Daten waren OK -> also in DB eintragen
    mysql_query(sprintf('insert into meldung
                            (datum,name,eintrag)
                      VALUES (now(),"%s","%s")',
                     addslashes($daten["name"]),
                     addslashes($daten["eintrag"])));
    echo mysql_error();

    // Alles eingetragen -> zurück zur Übersicht
    header('Location: http://'.$_SERVER["HTTP_HOST"].
           substr($_SERVER["PHP_SELF"],0,
                  strrpos($_SERVER["PHP_SELF"],'/'))
           .'/');
    // und fertig...
    die();
  }

}
else {
  $daten["name"] = "";
  $daten["eintrag"] = "";
}
?>

<html>
<head>
<title>Neuer Eintrag in unser GB</title>
</head>
<body>
<?php
if ($submit && !$DatenOK){
  // Das Formular wurde schon abgeschickt aber die Daten
  //  waren nicht OK
  // -> Fehlermeldung ausgeben
  echo "<h2>Fehler:</h2>\n";
  echo $error;
}

// Formular anzeigen
?>
<form action="<?php echo $_SERVER["PHP_SELF"];?>" method="GET">
Name:
<input type="text" name="name" size="30" maxlength="200"
 value="<?php echo $daten["name"]; ?>">
```

```
<br>
Text:<br>
<textarea rows="10" cols="50" wrap="virtual" name="eintrag">
<?php echo $daten["eintrag"]; ?>
</textarea>
<br>
<input type="submit" name="submit" value="Absenden">

</body>
</html>
```

Die Ausgabe ist eigentlich noch einfacher als die Eingabe. Für den ‚realen' Einsatz sollte man allerdings z. B. das Datum schöner formatieren und die Tabelle ist auch nicht so toll.

```
<?php

$DBHost   = "localhost";
$DBName   = "reeg";
$DBUser   = "reeg_ro";
$DBPasswd = "streng_geheim";


// Verbindug zu DB-Server herstellen
mysql_connect($DBHost, $DBUser, $DBPasswd)
     OR die("Konnte DB-Server nicht erreichen");
mysql_select_db($DBName);

?>

<html>
<head>
<title>Die Eintr&auml;ge in unserem GB</title>
</head>
<body>
<?php

$res = mysql_query('select datum, name, eintrag
                    from meldung
                    order by datum desc');
echo mysql_error();

while ($row = mysql_fetch_array($res)){
    echo "<table border=\"1\" width=\"600\">\n";
    printf("<tr><td>Name:</td><td>%s</td></tr>\n",
           htmlentities($row["name"]));
    printf("<tr><td>Datum:</td><td>%s</td></tr>\n",
           $row["datum"]);
```



```
    printf("<tr><td>Eintrag:</td></tr>\n");
    printf("<tr><td colspan=\"2\">%s</td></tr>\n",
           nl2br(htmlentities($row["eintrag"])));
    echo "</table>\n";
}
?>
<hr>
<a href="eintrag.php4">neuen Eintrag hinzufügen</a>
</body>
</html>
```

Das Script in Aktion gibt es auf meiner Homepage *http://reeg.net/*. Die einzigen Änderungen wurden am Layout vorgenommen.

16 Spruch des Abrufs

Was soll das Script eigentlich tun? Nun, es gibt häufiger den Spruch des Tages oder ähnliches. So etwas mit PHP zu realisieren wird schwierig. Wenn man einen Spruch des Tages realisieren will, hat man zwei Teilprobleme: Erstens muß man für diesen Tag einen Spruch auswählen und diesen zweitens anzeigen. Ersteres könnte man theoretisch dadurch lösen, daß man um Mitternacht ein Script aufruft, das den gewählten Spruch irgendwo speichert, von wo er dann den ganzen Tag lang ausgegeben werden kann.

PHP bietet allerdings von sich aus keine Möglichkeit, zu gewissen Zeiten irgendetwas automatisch zu machen. Die einzige Möglichkeit ist, mit einem Hilfsprogramm, wie z. B. CRON unter Unix, zum gewünschten Zeitpunkt das Ganze anzustoßen. Es gibt auch noch die Bastellösung, bei der bei der Ausgabe immer abgeprüft wird, ob es der erste Aufruf an diesem Tag ist und wenn ja, wird ein neuer Spruch ausgewählt.

Um das Ganze etwas einfacher zu halten, wird bei jedem Aufruf ein Spruch ausgewählt, der dann ausgegeben wird.

Kommen wir nun aber zum Programmieren. Als erstes stellt sich, wie immer, die Frage nach der Tabelle. Dies sollte hier kein großes Problem sein.

```
CREATE TABLE spruch (
  SID        int not null primary key auto_increment,
  Spruch     text,
  anzeigen   bool
) TYPE=MYISAM;

CREATE unique INDEX spruch_idx_Spruch ON spruch (Spruch(200));
```

Neben dem eigentlichen Text und einem eindeutigen Schlüssel habe ich noch eine kleine Spalte ‚anzeigen' eingeführt, über die man später verhindern kann, daß ein Spruch angezeigt wird.

Viel interessanter ist die zweite Anweisung. Im Endeffekt will ich nur erreichen, daß ein Spruch nicht zweimal eingetragen werden kann. Im Prinzip kein Problem, ein `UNIQUE` im `CREATE TABLE` auf „Spruch" sorgt schon dafür, sollte man meinen. Dem ist aber leider nicht so, da „Spruch" vom Typ `TEXT` ist und dieser in Sachen Index etwas anders funktioniert als z. B. `VARCHAR` (d. h. er erlaubt keinen). Ab MySQL Version 3.23 ist es aber möglich, über diese zweite Anweisung einen Index auf eine `TEXT`-Spalte zu legen, wobei dann nur die ersten n Buchstaben genutzt werden (hier sind es 200).

Nachdem nun die Tabellenerstellung geklärt ist, kommen wir zu den eigentlichen PHP-Scripten. Hier stellt sich wieder die Frage, nämlich, in welcher Reihenfolge sie programmiert werden sollten. Ich habe mich für die folgende entschieden:

- neuen Spruch einfügen
- zufälligen Spruch ausgeben

- alle Sprüche ausgeben
- Spruch löschen
- Spruch ändern

Kurze Begründung:
Das Einfügen in die Datenbank kann über ein einfaches `SELECT` von Hand überprüft werden. Der zufällige Spruch ist ja das eigentliche Ziel des Ganzen; hier könnte man jetzt auch schon fast aufhören; zumindest kann man jetzt das Ganze in Betrieb nehmen. Um einen Spruch zu löschen, muß man ihn irgendwie auswählen; deshalb die Gesamtausgabe vor dem Löschen. Über Löschen und anschließendes Einfügen kann man im Prinzip auch ändern; um das aber etwas komfortabler zu machen, gibt es als Letztes auch noch ein Ändern.

16.1 Neuen Spruch einfügen

Das folgende Script ist im Prinzip genauso aufgebaut, wie in 9.4 schon beschrieben. Im Prinzip sollte es durch die Kommentare selbsterklärend sein. Trotzdem noch zwei kleine Anmerkungen.

Es erfolgt keinerlei Authentifizierung, d. h. jeder kann beliebige Daten in die Datenbank einfügen. Wie man eine entsprechende Paßwortüberprüfung einbauen könnte, steht im Kapitel 11.1.3.

In der Variable `$anzeigen` bzw. im Checkbox-Inputfeld wird nicht der Wert übergeben, der nachher in der Datenbank stehen soll (0 oder 1), sondern „checked" oder „". Dadurch wird zwar die Wiederanzeige des Formulars einfacher, aber das Einfügen in die Datenbank etwas schwieriger. Es ist also eigentlich egal, wie man es realisiert.

```
<?php

// Verbindungsdaten MySQL DB
$DB_HOST = "localhost";
$DB_USER = "cr";
$DB_PASS = "123";
$DB_NAME = "cr";

$DatenOK = true;
$Fehler  = "";

if (isset($_GET["submit"])){
  // Formular wurde abgeschickt -> Daten pruefen
  if ($_GET["Spruch"] == ""){
    $DatenOK = false;
    $Fehler .= "Bitte noch einen Text eingeben!<br>\n";
    $daten["Spruch"] = "";
  }
  else {
```

```
    // Daten OK
    $daten["Spruch"] = $_GET["Spruch"];
  }

  // anzeigen kann nur true oder false sein
  // bei true ist es gesetzt
  if (isset($_GET["anzeigen"])){
    $daten["anzeigen"] = 1;
  }
  else {
    $daten["anzeigen"] = 0;
  }

  if ($DatenOK){
    // Daten in DB eintragen
    mysql_connect($DB_HOST, $DB_USER, $DB_PASS)
      OR die("Konnte DB nicht erreichen!");
    mysql_select_db($DB_NAME)
      OR die("Konnte DB nicht erreichen");

    mysql_query(sprintf('INSERT INTO spruch
                              (Spruch, anzeigen)
                         VALUES ("%s" , %d)',
                        addslashes($daten["Spruch"]),
                        $daten["anzeigen"]));
    switch (mysql_errno()){
    case 0:
      // Alles OK
      header('Location: http://'.$_SERVER["HTTP_HOST"].
             substr($_SERVER["PHP_SELF"],0,
                    strrpos($_SERVER["PHP_SELF"],'/'))
             .'/show_all.php4');
      exit;
      continue;
    case 1062:
      // Spruch doppelt eingetragen
      $DatenOK = false;
      $Fehler .= "Den Spruch gibt es schon<br>\n";
      continue;
    default:
      // Sonstiger Fehler
      // -> Fehlermeldung ausgeben
      $DatenOK = false;
      $Fehler .= "MySQL: ".mysql_errno().": ".
                 mysql_error()."<br>\n";
    }
```



```
  }
}
else {
  // Werte vorbelegen
  $daten["Spruch"]   = "";
  $daten["anzeigen"] = 1;
}
?>
<html>
<head>
  <title>SDA eintragen</title>
</head>
<body>
<?php
if (!$DatenOK){
  echo "<h1>$Fehler</h1>";
}
?>
<form action="<?php
  echo $_SERVER["PHP_SELF"];
?>" method="GET">
<div align="center">

<p>
<textarea name="Spruch" rows="5" cols="80"><?php
echo htmlentities($daten["Spruch"]);
?></textarea>
</p>

Spruch anzeigen <input type="checkbox" name="anzeigen"
value="checked" <?php
  echo ($daten["anzeigen"] ? "checked" : "");
?>><p>

<input type="submit" name="submit" value=" Alles OK ">

</div>
</form>
</body>
```

16.2 Zufälligen Spruch ausgeben

Das ist jetzt fast der einfachste Teil des Ganzen. Wie man einen zufälligen Datensatz auswählt, ist bereits im Kapitel 7.1 erklärt, von daher spare ich mir hier die Erklärung.

Ansonsten sollte das Script eigentlich selbsterklärend sein. In der ersten Datei, die ich sda.php4 genannt habe, wird eine Funktion `get_spruch()` realisiert. Diese kann dann später aus beliebigen anderen Dateien per `include()` eingefügt werden.

Es gibt auch noch eine zweite Funktion `get_sprueche`, die ein Array mit mehreren Sprüchen zurück gibt. Diese Funktion ist praktisch, wenn du verhindern willst, das deine Besucher den Server mit ständigem Neuladen der Webseite überlasten, um an alle Sprüche zu kommen. ;-)

```
<?php

/**
 * Holt einen zufaelligen Spruch aus der Datenbank
 * und gibt ihn als String zurueck
 */
function get_spruch(){
  // Verbindungsdaten MySQL DB
  $DB_HOST = "localhost";
  $DB_USER = "cr";
  $DB_PASS = "123";
  $DB_NAME = "cr";

  mysql_connect($DB_HOST, $DB_USER, $DB_PASS)
    OR die("Konnte DB nicht erreichen!");
  mysql_select_db($DB_NAME)
    OR die("Konnte DB nicht erreichen");

  $res = mysql_query('SELECT Spruch,
                             SID*0+rand() AS sort
                        FROM spruch
                       WHERE anzeigen = 1
                       ORDER BY sort
                       LIMIT 1');

  if (!$row = mysql_fetch_array($res)){
    echo "Fehler im Script!";
  }

  return $row["Spruch"];
}


/**
 * Holt die gewuenschte Zahl an Spruechen aus der
 * Datenbank und gibt diese als Array zurueck.
 * Wird nichts angegeben, werden 3 Sprueche aus der
 * Datenbank geholt.
 */
```



```
function get_sprueche($anzahl=3){
  // Verbindungsdaten MySQL DB
  $DB_HOST = "localhost";
  $DB_USER = "cr";
  $DB_PASS = "123";
  $DB_NAME = "cr";

  mysql_connect($DB_HOST, $DB_USER, $DB_PASS)
    OR die("Konnte DB nicht erreichen!");
  mysql_select_db($DB_NAME)
    OR die("Konnte DB nicht erreichen");

  $res = mysql_query(sprintf('SELECT Spruch, SID*0+rand() AS sort
                                FROM spruch
                               WHERE anzeigen = 1
                               ORDER BY sort
                               LIMIT %d',
                             $anzahl));

  while($row = mysql_fetch_array($res)){
    $sprueche[] = $row["Spruch"];
  }

  return $sprueche;
}
?>
```

Und um zu sehen, daß die Funktion tatsächlich funktioniert, hier noch ein sehr kompliziertes Script.

```
<?php

  // Wir wollen sauber programmieren
error_reporting(E_ALL);

include('./sda.php4');
?>
<html>
<head>
</head>
<body>
<pre>
<?php echo get_spruch(); ?>
</pre>
</body>
```

Und hier das zweite Script, für mehrere Sprüche. Es liest aus einen GET-Parameter die gewünschte Anzahl der Sprüche.

```
<?php

  // Wir wollen sauber programmieren
error_reporting(E_ALL);

include('./sda.php4');

if (isset($_GET["anzahl"]) &&
    is_numeric($_GET["anzahl"]) &&
    $_GET["anzahl"] > 0) {
  $anzahl = $_GET["anzahl"];
}
else {
  $anzahl = 3;
}
?>
<html>
<head>
</head>
<body>
<?php
$sprueche = get_sprueche($anzahl);
foreach ($sprueche AS $spruch){
  printf("<pre>\n%s\n</pre>\n",
         $spruch);
}
?>
</body>
```

16.3 Alle Sprüche ausgeben

Wie schon anfangs geschrieben, wird zur Administration eine Übersicht über alle existierenden Sprüche benötigt. Dies soll hier realisiert werden. Dazu muß natürlich zu jedem Spruch noch ein Link zum Ändern bzw. Löschen angegeben werden. Die ID des zu ändernden bzw. löschenden Scripts wird per GET-Parameter an das dann noch zu schreibende Script übergeben.

Auch hier erfolgt keinerlei Paßwortabfrage oder ähnliches, das heißt jeder kann sich alle Sprüche ansehen.

Das Script ist nicht sehr elegant programmiert, aber es funktioniert.

```
<?php

// Verbindungsdaten MySQL DB
$DB_HOST = "localhost";
```



```
$DB_USER = "cr";
$DB_PASS = "123";
$DB_NAME = "cr";

mysql_connect($DB_HOST, $DB_USER, $DB_PASS)
     OR die("Konnte DB nicht erreichen!");
mysql_select_db($DB_NAME)
     OR die("Konnte DB nicht erreichen");

?>
<html>
<head>
  <title>SDA ansehen</title>
</head>
<body>
<center>
<a href="insertchange.php4">Neu einf&uuml;gen</a>
</center>
<ul>
<?php

$res = mysql_query('SELECT SID, Spruch
                    FROM spruch
                    ORDER BY SID');

while ($row = mysql_fetch_array($res)){
  printf('<li><pre>%s</pre>
<a href="insertchange.php4?SID=%d">&Auml;ndern</a>  
<a href="delete.php4?SID=%d">L&ouml;schen</a>'."\n",
         $row["Spruch"],
         $row["SID"],
         $row["SID"]);
}

?>
</ul>
</body>
```

16.4 Spruch löschen

Nachdem wir nun Sprüche eintragen und sogar ausgeben können, wäre es evtl. auch ganz praktisch, sie zu löschen[1]. Das Löschen ist eigentlich kein großes Problem; das Script braucht nur die ID des zu löschenden Spruchs und ruft dann ein `DELETE` auf. Den Aufruf

[1] Manche Leute behaupten, bei meinen Scripten würde nie die Löschenfunktion programmiert

für das Script haben wir ja eben schon geschrieben.

Nach dem Löschen kann man entweder eine Erfolgsmeldung ausgeben, oder einfach wieder auf z. B. die Übersichtsseite weiterleiten, was hier gemacht wird.

```
<?php

if (!isset($_GET["SID"]) || !is_numeric($_GET["SID"])){
  die("Fehler");
}

// Verbindungsdaten MySQL DB
$DB_HOST = "localhost";
$DB_USER = "cr";
$DB_PASS = "123";
$DB_NAME = "cr";

mysql_connect($DB_HOST, $DB_USER, $DB_PASS)
     OR die("Konnte DB nicht erreichen!");
mysql_select_db($DB_NAME)
     OR die("Konnte DB nicht erreichen");

mysql_query(sprintf('DELETE FROM spruch
                     WHERE SID=%d',
                    $_GET["SID"]));

header('Location: http://'.$_SERVER["HTTP_HOST"].
             substr($_SERVER["PHP_SELF"],0,
                    strrpos($_SERVER["PHP_SELF"],'/'))
       .'/show_all.php4');
?>
```

16.5 Spruch ändern

Und als letztes wäre es natürlich auch ganz nett, einen Spruch auch ändern zu können. Wer jetzt faul ist (und wer ist das nicht?) wird denken: Das HTML-Formular haben wir doch schon. Stimmt, deshalb benutzen wir das auch; allerdings nicht per Copy and Paste, sondern wir erweitern das Einfügen-Script ein wenig.

Die Änderungen sind eigentlich relativ einfach. Als erstes wird die Datenbankverbindung immer aufgebaut, da sie an mehreren Stellen benötigt wird (hätte man auch durch eine Funktion eleganter lösen können). Als nächstes muß bei der Datenprüfung nun auch die Spruch-ID geprüft werden. Hier beschränke ich mich auf die Prüfung, ob es eine Zahl ist. Die eigentliche SQL-Anweisung muß natürlich auch erweitert werden, die Fehlerabfrage kann dann jedoch wieder dieselbe sein. Schließlich müssen auch irgendwo die alten Werte geholt werden und als letztes wird das HTML-Formular so erweitert, daß es bei Bedarf auch noch die Spruch-ID wieder übergibt. Das waren dann auch schon alle Änderungen

und wir haben unser Änderungsscript fertig.

```
<?php

// Verbindungsdaten MySQL DB
$DB_HOST = "localhost";
$DB_USER = "cr";
$DB_PASS = "123";
$DB_NAME = "cr";

// erstmal ist alles OK
$DatenOK = true;
$Fehler  = "";

mysql_connect($DB_HOST, $DB_USER, $DB_PASS)
     OR die("Konnte DB nicht erreichen!");
mysql_select_db($DB_NAME)
     OR die("Konnte DB nicht erreichen");


if (isset($_GET["submit"])){
  // Formular wurde abgeschickt -> Daten pruefen
  if ($_GET["Spruch"] == ""){
    $DatenOK = false;
    $Fehler .= "Bitte noch einen Text eingeben!<br>\n";
    $daten["Spruch"] = "";
  }
  else {
    // Daten OK
    $daten["Spruch"] = $_GET["Spruch"];
  }

  // anzeigen kann nur true oder false sein
  // bei true ist es gesetzt
  if (isset($_GET["anzeigen"])){
    $daten["anzeigen"] = 1;
  }
  else {
    $daten["anzeigen"] = 0;
  }

  if (isset($_GET["SID"])){
    // Eine Spruch-ID wurde übergeben
    // -> sie muß geprüft werden
    //
    // Überprüfung, ob sie tatsächlich existiert,
```

```
    // könnte auch noch erfolgen
    if (!is_numeric($_GET["SID"])){
      $DatenOK = 0;
      $Fehler .= "Ungültige Spruch-ID &uuml;bergeben!<br>\n";
    }
    else {
      // Daten OK
      $daten["SID"] = $_GET["SID"];
    }
  }

  if ($DatenOK){
    // Daten in DB eintragen
    if (isset($daten["SID"])){
      // Spruch-ID wurde übergeben
      // -> Spruch wird geändert
      mysql_query(sprintf('UPDATE spruch
                              SET Spruch="%s", anzeigen=%d
                              WHERE SID=%d',
                             addslashes($daten["Spruch"]),
                             $daten["anzeigen"],
                             $daten["SID"]));
    }
    else {
      // neuer Spruch -> einfügen
      mysql_query(sprintf('INSERT INTO spruch
                                       (Spruch, anzeigen)
                                VALUES ("%s"  , %d)',
                             addslashes($daten["Spruch"]),
                             $daten["anzeigen"]));
    }

    // MySQL-Rückgabewert auswerten
    // Wenn OK -> Weiterleiten
    // sonst -> Fehlermeldung
    switch (mysql_errno()){
    case 0:
      // Alles OK
      header('Location: http://'.$_SERVER["HTTP_HOST"].
             substr($_SERVER["PHP_SELF"],0,
                    strrpos($_SERVER["PHP_SELF"],'/'))
             .'/show_all.php4');
      exit;
      continue;
    case 1062:
      // Spruch doppelt eingetragen
```



```
      $DatenOK = false;
      $Fehler .= "Den Spruch gibt es schon<br>\n";
      continue;
    default:
      // Sonstiger Fehler
      // -> Fehlermeldung ausgeben
      $DatenOK = false;
      $Fehler .= "MySQL: ".mysql_errno().": ".
                 mysql_error()."<br>\n";
    }
  }
}
elseif (isset($_GET["SID"])){
  // es soll der Spruch SID geändert werden
  // -> bisherige Werte laden
  //
  // SID muss an der Stelle nicht auf numerisch
  // getestet werden, da es nur mit %d im sprintf()
  // genutzt wird
  $res= mysql_query(sprintf('SELECT SID, Spruch, anzeigen
                               FROM spruch
                               WHERE SID=%d',
                             $_GET["SID"]));

  if (!$row = mysql_fetch_array($res)){
    $DatenOK = false;
    $Fehler .= "Konnte Spruch nicht laden";
  }
  $daten["SID"]      = $row["SID"];
  $daten["Spruch"]   = $row["Spruch"];
  $daten["anzeigen"] = $row["anzeigen"];
}
  else {
    // Werte vorbelegen
    $daten["Spruch"]   = "";
    $daten["anzeigen"] = 1;
  }
?>
<html>
<head>
  <title>SDA eintragen</title>
</head>
<body>
<?php
if (!$DatenOK){
  echo "<h1>$Fehler</h1>";
```

```
}
?>
<form action="<?php
  echo $_SERVER["PHP_SELF"];
?>" method="GET">
<div align="center">

<p>
<textarea name="Spruch" rows="5" cols="80"><?php
echo htmlentities($daten["Spruch"]);
?></textarea>
</p>

Spruch anzeigen <input type="checkbox" name="anzeigen"
value="checked" <?php
  echo ($daten["anzeigen"] ? "checked" : "");
?>><p>

<?php
if (isset($daten["SID"])){
  // zum Ändern auch die SID übergeben
  printf('<input type="hidden" name="SID" value="%d">',
         $daten["SID"]);
}
?>

<input type="submit" name="submit" value=" Alles OK ">

</div>
</form>
</body>
```

16.6 Schlußbemerkung

Dieses Beispiel erhebt keine Anspruch darauf, ein elegant programmiertes Script zu sein. So ist es z. B. ungeschickt, die Datenbankverbindungsdaten in jede Datei zu schreiben. Eine Änderung und man muß jede Datei ändern. Es bietet sich also an, diese Daten in eine zentrale Datei zu speichern. Bei der Gelegenheit könnte man dann auch gleich noch den ganzen Datenbankverbindungsaufbau mit auslagern. Noch geschickter wäre natürlich die Verwendung von OOP, wie sie in Kapitel 18 beschrieben wird.

Ähnliches gilt für das HTML-Layout der Adminseiten[2]. Hier steht es fest in jeder Datei; flexibler wären dann z. B. HTML-Header und HTML-Footer Dateien, die jeweils per `include()` eingebunden würden.

[2] Bei diesem Beispiel könnte man auch fragen: Welches Layout?

Ein Programmieren mit Copy and Paste geht zwar am Anfang schneller, aber sobald man etwas ändern muß, wird es aufwendig, weil jede Stelle geändert werden muß. Beim Ändern hätte man auch das Einfügescript kopieren und dann anpassen können, dann hätte man sich die Abfragen gespart. Allerdings wäre der Aufwand beim Einfügen eines weiteren Feldes größer gewesen. Bei dieser Lösung muß an 4 Stellen erweitert werden: bei der Datenprüfung, den beiden SQL-Anweisungen und dem HTML-Formular. Bei der Kopierlösung müßte man zwei mal 3 Stellen ändern.

16.7 Übung

Wieso habe ich bei der Auswahl von mehrere Sprüchen eine extra Funktion `get_sprueche` implementiert? Ich hätte doch auch mit Hilfe einer `for`-Schleife mehrmals die Funktion `get_spruch` aufrufen können.

17 Kleines Bannerscript

In diesem Kapitel zeige ich, wie ein Script zum Bannereinblenden und -zählen aussehen kann. Es erhebt keinen Anspruch darauf, perfekt zu sein oder alles zu können. Es soll nur ein paar Möglichkeiten von PHP demonstrieren.

Was soll das Script können?

- zufälliges Banner einblenden
- zählen, wie häufig welches Banner angezeigt wurde
- die zum Banner gehörige URL als Link anbieten
- zählen, wie häufig auf welches Banner geklickt wurde

Das Ganze habe ich in zwei Teile geteilt; das Script aus dem ersten Teil erfüllt die Anforderungen der ersten beiden Punkte, das zweite kann alles geforderte. Zu Demonstrationszwecken wurde das erste Script sowohl ohne als auch mit Datenbank realisiert, das zweite hingegen der Einfachheit halber nur mit Datenbank.

17.1 Erste Realisierung

Als erstes stellt sich die Frage, welche Daten gespeichert werden müssen. Im Moment sind das noch nicht viele:

- welche Banner gibt es?
- wie häufig wurde das Banner angezeigt?

Den ersten Punkt können wir dadurch abhaken, daß wir mit Hilfe der ‚directory functions' auslesen lassen, welche Dateien in einem Verzeichnis existieren. Für die Lösung ohne Datenbank habe ich dies auch gemacht, bei der zweiten Variante dagegen alle Dateinamen in eine Tabelle geschrieben und dann aus dieser einen zufälligen Eintrag auswählen lassen. Man könnte dies natürlich kombinieren, indem man ein Script schreibt, was alle möglichen Dateinamen aus dem Verzeichnis ausliest und dann in die Tabelle schreibt, um dann nachher nur mit der Tabelle zu arbeiten.

Die ganze Administrationsoberfläche habe ich bei diesem Beispiel ganz dezent ignoriert; d. h. die Dateinamen müssen von Hand in die SQL-Tabelle eingetragen werden, und wie häufig welches Banner angezeigt wurde, muß ebenfalls von Hand aus den Dateien ausgelesen bzw. aus der Tabelle abgefragt werden.

Bei dem folgenden Script erledigen die beiden Funktionen genau dasselbe - mit dem Unterschied, daß die erste Funktion ohne Datenbank arbeitet.

Noch ein kurzes Wort zu Installation: Beide Funktionen gehen davon aus, daß die Banner im Unterverzeichnis ‚banner' liegen. Die Dateisystemvariante benötigt zusätzlich noch ein Verzeichnis, in dem die Dateien zum Zählen liegen, das wird hier ‚rw' genannt.

Für die Datenbankversion wird auch eine entsprechende Tabelle benötigt, für die hier die entsprechende `create table`-Anweisung steht.

```
CREATE TABLE banner(
  BID     int not null primary key auto_increment,
  Name    varchar(50) not null,
  shown   int,
  unique(Name)
);
```

Das Script sollte durch die Kommentare eigentlich soweit selbsterklärend sein.

```
<?php

function file_get_name(){

  function get_file_array(){
    // gibt ein Array mit allen Dateinamen zurück
    $handle=opendir('banner');
    while ($file = readdir($handle)){
      if ($file != "." && $file != ".."){
        $ret[]=$file;
      }
    }
    return $ret;
  }

  // alle Dateinamen in Array schreiben
  $files = get_file_array();

  // Zufall initialisieren
  srand((double)microtime()*1000000);
  $nr = rand(0,count($files)-1);

  // zufällige Datei auswählen
  $banner = $files[$nr];

  //Zähler für Datei erhöhen

  // Dateiname für Zähler in $filename schreiben
  $filename = "rw/".$banner;
  if (file_exists($filename)){
    $fp = fopen($filename,"r+");
    flock($fp,2);
    $count = fgets($fp,1024);
    if (empty($count)){
      $count = 0;
```

```
    }
    $count++;
    fseek($fp,0);
  }
  else {
    $fp = fopen($filename,"w");
    $count = 1;
  }
  fwrite($fp,$count);
  fclose($fp);
  return $banner;
}


function db_get_name(){
  $DB_HOST   = "localhost";
  $DB_USER   = "cr";
  $DB_PASSWD = "123";
  $DB_NAME   = "cr";

  mysql_connect($DB_HOST, $DB_USER, $DB_PASSWD);
  mysql_select_db($DB_NAME);

  $res = mysql_query('select BID, Name, BID*0+rand() AS sort
                      from banner
                      order by sort
                      LIMIT 1;');

  if (!$res){
    // Fehler
    echo "Fehler in der Datenbank";
    return false;
  }

  // Ergebnis holen
  $row = mysql_fetch_array($res);

  // shown-Zähler um eins erhöhen
  mysql_query(sprintf('update banner
                       set shown = shown+1
                       where BID=%d',
                      $row["BID"]));

  // Bildname zurück geben
  return $row["Name"];
}
```



```
?>

<html>
<head>
<title>Banner</title>
</head>
<body>
<img src="banner/<?php echo file_get_name(); ?>">
<img src="banner/<?php echo db_get_name(); ?>">
</body>
</html>
```

17.2 Zweite Realisierung

Nachdem wir nun die Grundfunktionalität haben, heißt es, das Ganze noch etwas zu erweitern. Es fehlen nämlich noch zwei Anforderungspunkte:

- die zum Banner gehörige URL als Link anbieten
- zählen, wie häufig auf welches Banner geklickt wurde

Ersteres sollte kein Problem sein. An Stelle von ‚`<img src=...>`' geben wir einfach ein ‚`<a href=...><img src=...></a>`' aus. Dann haben wir allerdings ein Problem; wir können nicht zählen, wie häufig auf ein Banner geklickt wurde. Das Klicken findet auf Clientseite statt und dort haben wir mit PHP keine Möglichkeit, irgendetwas zu machen[1]. Also müssen wir uns etwas anderes überlegen.

Die Lösung ist relativ einfach. Wir geben nicht die eigentliche URL an, sondern ein Script auf unserem Server, was in Ruhe zählen kann und dann auf die richtige URL weiterleitet. Ich habe das hier in einem Script erledigt.

Aber erstmal zu unserer Tabelle in der Datenbank. Diese muß jetzt natürlich auch nochmal geringfügig erweitert werden. Es fehlen nämlich die Felder für die Klicks und für die URL. Im Endeffekt ergibt sich daraus folgendes `create table`:

```
CREATE TABLE banner(
  BID     int not null primary key auto_increment,
  Name    varchar(50) not null,
  URL     varchar(100) not null,
  shown   int,
  clicked int,
  unique(Name)
);
```

Nun aber zum Script. Der erste Teil (die Funktion ‚show_banner') ist fast identisch mit der Funktion aus der ersten Realisierung. Sie gibt nun allerdings nicht mehr den Bildnamen, sondern das Ganze als Link zurück.

[1] Viele Leute sagen: Zum Glück

Die größte Änderung befindet sich im Hauptprogramm. Hier wird nun nicht mehr nur die Funktion aufgerufen, sondern es gibt eine Fallunterscheidung. Wird das Ganze ohne ‚bid' aufgerufen, so wird das Banner zurückgegeben, ansonsten wird der Klick gezählt und zur URL weitergeleitet.

Die Kombination in einem Script mag für diesen Fall einfacher sein, weil man alles zusammen hat; häufig ist es aber wahrscheinlich besser, diese beiden Funktionen in zwei verschiedene Scripte zu packen.

```
<?php

$DB_HOST    = "localhost";
$DB_USER    = "cr";
$DB_PASSWD = "123";
$DB_NAME    = "cr";

mysql_connect($DB_HOST, $DB_USER, $DB_PASSWD);
mysql_select_db($DB_NAME);

function show_banner(){
  // zufälliges Banner aus DB holen
  $res = mysql_query('select BID, Name, BID*0+rand() AS sort
                      from banner
                      order by sort
                      LIMIT 1;');

  if (!$res){
    // Fehler
    echo "Fehler in der Datenbank";
    return false;
  }

  // Ergebnis holen
  $row = mysql_fetch_array($res);

  // shown-Zähler um eins erhöhen
  mysql_query(sprintf('update banner
                       set shown = shown+1
                       where BID=%d',
                      $row["BID"]));

  // Banner mit Link ausgeben
  printf('<a href="%s?bid=%d">'.
         '<img src="banner/%s" border="0"></a>',
         $_SERVER["PHP_SELF"],
         $row["BID"],
         $row["Name"]);
```



```
}


/************* Hauptprogramm ************/

if (isset($_GET["bid"])){
  // bid gesetzt -> es wurde auf eine Banner geklick
  // -> User weiterleiten

  if (!is_numeric($_GET["bid"])){
    // bid muß immer numerisch sein
    die('Keine gültige ID übergeben');
  }

  // URL aus DB holen
  $res = mysql_query(sprintf('select URL
                              from banner
                              where bid=%d',
                             $_GET["bid"]));

  if (!$row = mysql_fetch_array($res)){
    // Fehler
    die("Fehler in der Datenbank");
  }

  // click-Zaehler um eins erhöhen
  mysql_query(sprintf('update banner
                       set clicked = clicked+1
                       where BID=%d',
                      $_GET["bid"]));

  // User auf die richtige Seite weiterleiten
  header('Location: '.$row["URL"]);
  exit;
}
else {
  // Banner anzeigen
?>
<html>
<head>
<title>Banner</title>
</head>
<body>
<?php show_banner(); ?>
</body>
</html>
```

```
<?php
}
?>
```

Teil V

Objektorientierung theorethisch

18 Bedeutung von Objektorientierung

> Das Problem zu erkennen ist wichtiger, als die Lösung zu erkennen, denn die genaue Darstellung des Problems führt zur Lösung.
>
> Albert Einstein

In diesem Teil soll die Objektorientierte Programmierung (kurz OO bzw. OOP) allgemein erklärt werden. Im nächsten Teil kommt dann die Realisierung in PHP.

Die ersten Beispielprogramme, die im Verlauf dieses Manuals besprochen wurden, hatten keine große Struktur: In ihnen wird einfach am Anfang des Programmes angefangen und am Ende aufgehört. Teilweise wird mit if & Co. ein wenig hin und her gesprungen. Das Ganze wird allerdings schnell unübersichtlich und Teile lassen sich nur durch Cut'n'Paste wieder verwenden.

Bei der Programmierung mit Funktionen wird das ganze schon etwas übersichtlicher. Es gibt Funktionen, die bei einem Aufruf mit den übergebenen Werten etwas machen. Diese Funktionen lassen sich in diverse Dateien ausgliedern und können dadurch in mehreren Programmen wieder verwendet werden. Allerdings können die Funktionen untereinander (über den eigentlichen Funktionsaufruf hinaus) keine Werte austauschen, das heißt die Werte müssen alle über das aufrufende Programm gehen.

Bei der Objektorientierung ist man einen Schritt weiter gegangen und hat die Daten und die Funktionen zusammengefaßt, so daß man sie nun als eine Einheit betrachtet. Im Prinzip hat man damit versucht, die reale Welt abzubilden. Konkret heißt das, daß in einer solchen Einheit, Klasse genannt, die Funktionen (hier Methoden genannt) auf gemeinsame Daten (Attribute, Klassen- oder Instanzvariablen genannt) zugreifen können.

18.1 Ein kleines Beispiel: Funkwecker

Jeder kennt digitale Funkwecker. Diese besitzen ein Display, um die aktuelle Uhrzeit, das Datum und die Weckzeit anzuzeigen, und über Tasten können letztere auch eingestellt werden. Wie die Uhr die einzelnen Werte speichert, können wir von außen nicht feststellen[1]. Wir können auch nicht direkt die Uhrzeit ändern, sondern müssen die Tasten benutzen und können dadurch z. B. auch nicht 25:00 Uhr einstellen. Das heißt, die Uhrzeit ist gekapselt und man kommt nur über die entsprechenden Funktionen an sie heran. Die Gesamtheit aller Tasten in diesem Beispiel nennt man in der OOP übrigens Schnittstelle, und die Kapselung ist ein zentraler Bestandteil der Objektorientierung[2].

Wenn wir uns jetzt vorstellen, das Ganze mit Funktionen nachzubauen, hätten wir ein Problem: Wir könnten zwar Funktionen wie `setTime()` oder `setAlarm()` programmieren,

[1] Normalerweise interessiert uns das auch gar nicht
[2] Wenn auch einer, der in PHP bisher nicht konsequent umgesetzt wurde – mehr dazu später

könnten die Werte aber nicht speichern (auf globale Variablen wollen wir unter anderem des guten Programmierstils wegen verzichten). Bei einem Objekt gibt es das Problem nicht, denn dieses besitzt das Attribut Uhrzeit und kann diese mit Hilfe der Methode `setTime()` nach einer Plausibilätsprüfung setzen. Das Beispiel Funkwecker hinkt natürlich etwas, da Objekte im Hintergrund nicht einfach weiterlaufen und damit z. B. die Weckfunktion nicht funktionieren würde.

18.2 Jedem sein Auto

Ein weiteres anschauliches Beispiel, das oft verwendet wird, ist das einer Autofabrik. Stellen wir uns vor, wir würden ein Auto der Marke DSP[3] fahren, das in den DSP-Werken hergestellt wird. Wir bekommen – entweder direkt von der Fabrik oder über einen Händler – ein Exemplar eines nagelneuen DSPs geliefert und der gehört dann uns. Gleichzeitig gibt es aber noch Millionen anderer DSPs, die allesamt aus dem DSP-Werk stammen. Wenn wir vereinfachend annehmen, daß die DSP-Werke nur ein Modell „CR" herstellen (das aber in millionenfacher Auflage), dann haben wir hier ein prima Beispiel für die Beziehungen der Objektorientierung:

Während die DSP-Werke die Pläne für die Konstruktion von Autos des Modells DSP CR haben, haben wir als Besitzer eines DSP CR nur genau ein Exemplar, können also selbst keine weiteren Exemplare herstellen. Uns interessiert auch gar nicht, wie der Fertigungsprozeß aussieht – wir wollen ja nur mit dem Auto fahren, es also benutzen. Alles, was wir dazu kennen müssen, ist die Schnittstelle des DSP CR: Wie startet man den Wagen, wie beschleunigt, bremst, schaltet und lenkt man?

An dieser Stelle kann man schon etwas sehr grundlegendes der Objektorientierung feststellen: Durch die Kapselung der Funktionalität erreicht man, daß man diese problemlos optimieren kann, solange man nur die Schnittstelle unverändert läßt und natürlich am zu erwartenden Ergebnis einer Funktion nichts verändert (wenn man bisher immer beschleunigt hat, wenn man aufs Gaspedal getreten hat, wäre man sicher mehr als überrascht, wenn die DSP-Werke in der nächsten Modellgeneration einfach die Bremse mit dem Gaspedal verbinden würden ...). Auch sollten Eigenschaften grundsätzlich nur über die Schnittstelle abruf- und veränderbar sein, damit etwaige Änderungen an der Art, *wie* etwas gelöst wurde, jederzeit vorgenommen werden können, ohne weitere Abhängigkeiten verfolgen zu müssen.

Doch zurück zum Autofahren: Wir erwarten also von unserem Wagen, daß er sich wie ein ganz normales Auto verhält. Den DSP-Werken überlassen wir es, die gewünschte Funktionalität zur Verfügung zu stellen (d. h. einzubauen). Wir wollen aber vielleicht nicht nur einen Wagen haben, sondern doch lieber zwei – z. B. einen für jeden Ehepartner. :-) Die beiden Wagen sollen aber natürlich nicht genau gleich sein, das wäre ja langweilig. Wir sagen den DSP-Werken also beim Bestellen des Zweitwagens gleich, daß wir diesmal keinen Standard-Silbermetallic-CR haben wollen, sondern lieber einen dunkelgrünen CR in der Family-Ausführung. Innerhalb der DSP-Werke ist für diese Bestellung der Konstruktor zuständig: Er nimmt die Daten auf und sorgt dafür, daß diese allen daran interessierten Fabrikationsstationen des Werkes in der einen oder anderen Form zur Verfügung gestellt werden.

[3] Könnte z. B. für Donner-schneller Pfeil stehen ;-)

Nun stellen die DSP-Werke im Wesentlichen natürlich DSP-Modelle her. Sicher gibt es aber auch den einen oder anderen Extra-Dienst, der nicht direkt abhängig von einem Wagen ist, jedoch sinnvollerweise von den DSP-Werken angeboten wird. Auf diesen Extra-Dienst soll jeder Außenstehende zugreifen können – egal, ob er einen DSP CR hat oder nicht. Vielleicht will der DSP-Konzern aber auch einfach nur eine Sonderaktion starten und für begrenzte Zeit ein Logo auf jeden gefertigten DSP CR sprühen lassen. Die beiden Fälle scheinen auf den ersten Blick völlig unterschiedliche Problematiken zu beschreiben, aber sie haben eine Gemeinsamkeit: Sie sollen beide auch indirekt von jedem Besitzer eines DSP CR aufruf- bzw. veränderbar sein. Logischerweise dürfen Extra-Dienste der DSP-Werke auf keine Eigenschaften von DSP-Modellen zugreifen, denn wenn ein solcher Dienst von jemandem erbeten wird, der keinen DSP CR besitzt, dann kann auch an keinem DSP CR eine Veränderung durchgeführt werden – nicht einmal die *Abfrage* von Daten eines DSP-Modells ist erlaubt[4]!

18.3 Von Autos zu Objekten

Nun haben wir hoffentlich alle verstanden, welche Beziehungen zwischen den Autofahrern und den DSP-Werken bestehen – es wird Zeit, zur Objektorientierung zu kommen. Doch Halt! Das war alles schon objektorientiert! :-) Was noch fehlt, ist eigentlich nur noch die Erklärung. Wenn man von obigem Beispiel zur OOP kommen will, muß man einige neue Begriffe einführen. In der folgenden Tabelle werden daher die Begriffe des Beispiels denen der tatsächlichen Objektorientierung gegenübergestellt.

Welt der Autos	**Welt der Objekte**
DSP-Werk	Klasse DSP
mein DSP CR	Objekt der Klasse DSP
Eigenschaften des DSP CR	Attribute der Klasse DSP
Funktionalität des DSP CR	Methoden der Klasse DSP
Konstruktor der DSP-Werke	Konstruktor der Klasse DSP
Zweitwagen	anderes Objekt der Klasse DSP
Extra-Dienste der DSP-Werke	`static`-Methoden der Klasse DSP
Sonderaktion der DSP-Werke	`static`-Attribut der Klasse DSP

Tabelle 18.1: Von Autos zu Objekten

18.4 Vererbung

Wer hat nicht schon einmal von Vererbung gehört? Beim Programmieren und insbesondere bei Objekten geht es natürlich nicht um den biologischen Begriff. Vererbung ist der zweite fundamentale Begriff der OOP und bedeutet, daß jedes Objekt einer Klasse B dieselben Methoden beherrscht wie ein Objekt der Klasse A, von der Klasse B abgeleitet wurde, d. h. geerbt hat. Zusätzlich kann es noch eigene Methoden definieren oder vorhandene Methoden neu definieren (überschreiben).

[4] Außer, ein solches Datum ist wiederum vom gleichen Charakter wie die Sonderaktion

Bezogen auf das obige Auto-Beispiel könnten z. B. die DSP-Werke ein neues Modell DSP CR-JH herausbringen wollen. Der Einfachheit halber nehmen sie dazu die Pläne des DSP CR, ergänzen hier und ändern da etwas – das neue Modell basiert ja auf dem erfolgreichen DSP CR. Ebenso ist der DSP CR im Grunde auch nur ein Auto, also warum für jedes Modell das Rad neu erfinden? ;-) Denkbar ist doch, daß irgendwo in den Tiefen des DSP-Archivs schon ein Plan für ein Ur-Auto liegt, der wieder herausgekramt werden kann, wenn alle Ideen des Modells CR verworfen werden müßten – niemals aber die Grundidee des Autos!

18.4.1 Image-Beispiel

Ein Beispiel aus der Programmier-Praxis zeigt es noch deutlicher:
Wir haben eine Klasse „Image" (Bild), die mit Hilfe der Image-Funktionen der Programmiersprache ein leeres Bild mit definierbarer Höhe und Breite erzeugt. Diese Klasse kennt die Methode `show()`, die das Bild anzeigt. Ein leeres Bild ist aber langweilig, also benötigen wir eine neue Klasse „Point", durch die es möglich sein soll, einen Punkt in diesem Bild zu malen. Für diesen Zweck gibt es die Methode `set(x, y)`. Da die Methode `show()` mehr oder weniger unverändert übernommen werden kann, nehmen wir „Point" (Punkt) als Erweiterung von „Image", das heißt „Point" erbt alle Methoden und Attribute von „Image". Auf dieselbe Weise kann man von „Point" wiederum „Circle" (Kreis) und „Rectangle" (Rechteck) ableiten, wobei letzteres durch die Spezialform „Square" (Quadrat) erweitert werden kann. Das ganze wird später als Übung noch etwas vertieft werden.

18.5 Konstruktor

Wie schon in der Erklärung des Auto-Beispiels gesehen, braucht man zum Erzeugen eines Objektes einen sogenannten Konstruktor. Bei unserem Image-Beispiel haben wir die Methode `init()`, die bei jedem Objekt als erstes einmal aufgerufen werden muß, damit verschiedene Attribute initialisiert werden. Wird dies nicht gemacht, kann es passieren, daß die Klasse nicht richtig funktioniert[5]. Damit aber eine solche Initialisierungsfunktion immer einmal aufgerufen wird, wurde der Konstruktor eingeführt. In vielen Programmiersprachen – so auch in PHP – heißt der Konstruktor so wie die Klasse. Der Konstruktor ist im Grunde nichts anderes als eine spezielle Methode, die keine Return-Anweisung enthalten darf, weil der Konstruktor grundsätzlich und implizit ein Objekt der Klasse zurückliefert.

Es gibt die Unterscheidung zwischen dem Standardkonstruktor und dem Allgemeinen Konstruktor. Während ersterer parameterlos und dadurch eindeutig bestimmt ist, erwartet letzterer mindestens einen Parameter und ist dadurch allgemein brauchbar (daher die Bezeichnung). „Allgemeiner Konstruktor" ist natürlich nur ein Oberbegriff für alle Konstruktoren außer dem Standardkonstruktor.

18.6 Destruktor

Manchmal braucht man neben einem Konstruktor auch eine Funktion, die am Ende die Aufräumarbeiten erledigt. Auch dafür wurde eine Möglichkeit geschaffen, Destruktor genannt. Destruktoren können keine Parameter haben.

[5] Die oben erwähnten `static`-Methoden müssen natürlich immer funktionieren

PHP kennt allerdings bisher leider keine Destruktoren. Man kann sich über die Funktion `register_shutdown_function()` einen Behelf zusammenbauen oder sich PHP 5 anschauen. ;-)

18.7 Klasse oder Objekt

Wer das erste Mal von Klassen und Objekten hört, wird sich unweigerlich fragen, was diese Begriffe bedeuten und was der Unterschied zwischen beiden ist. Die Erklärung zum Auto-Beispiel sollte diese Frage eigentlich schon geklärt haben, doch allgemein könnte man sagen, ein Objekt ist einfach ein konkretes Exemplar einer Klasse. Die Klasse wiederum ist die abstrakte Definition der Eigenschaften eines Objektes. Nehmen wir als Beispiel die Klasse „Lebewesen". Welche Eigenschaften ein Exemplar dieser Klasse haben muß, kann dir ein Biologe sagen. Ich kann dir nur sagen, daß du ein Objekt der Klasse „Lebewesen" bist: du bist ein konkretes Exemplar der abstrakten Gattung Lebewesen.

18.8 Zugriffsrechte: public, private oder doch protected?

Bei Klassen gibt es Methoden, die jedes Objekt einer solchen Klasse benutzen können soll. In unserem Beispiel sind das `init()`, `show()` und `set()`. Auf der anderen Seite sollen Daten ja gekapselt sein, d. h. man kann nicht direkt darauf zugreifen. Für genau diese Unterscheidung gibt es die drei Schlüsselwörter „public" (auf die Methode / das Attribut darf von überall aus zugegriffen werden), „private" (nur aus **genau der definierenden Klasse** darf auf die Methode / das Attribut zugegriffen werden) und „protected" (aus der Klasse und allen abgeleiteten Klassen darf auf die Methode / das Attribut zugegriffen werden). Zusätzlich dürfen alle Objekte einer Klasse auf *alle* Eigenschaften eines Objekts derselben Klasse zugreifen (also auch private-definierte).

Bei unserem obigen Image-Beispiel müssen die genannten Methoden public sein (sonst würden sie keinen Sinn machen). Die Attribute wie z. B. `width` oder `height`, die über `init()` gesetzt werden, sollten nicht auf public gesetzt werden, damit nicht jeder an sie heran kommt. Wenn sie nun in der Klasse „Image" auf private gesetzt würden, hätten wir keine Möglichkeit, aus der Klasse „Point" darauf zuzugreifen. Wenn dies gewünscht ist, müssen sie auf protected gesetzt werden.

Wem dies jetzt zu viel war, der kann sich beruhigt zurücklehnen und aufatmen: PHP 4 kennt noch keine Unterscheidung zwischen public, private und protected. Es ist einfach alles public. Da sich das jedoch mit PHP 5 definitiv ändern wird (siehe 20.5), sollte man sich zumindest grundlegende, grobe Gedanken machen und z. B. alle Methoden, die ausschließlich von der Klasse selbst (oder erbenden Klassen) und nur intern benutzt werden, an das Ende der Klasse verfrachten und für private oder protected vormerken, z. B. durch Kennzeichnung mittels einem (keinesfalls zwei!) vorangestelltem Unterstrich – gilt auch für Attribute.

19 Bezeichnungen

Im Rahmen der Objektorientierung gibt es eigene Vokabeln wie Objekte, Klassen, Methoden, Vererbung etc. Ich versuche, sie hier zu erklären.

19.1 Instanz

Eine Instanz ist ein konkretes Exemplar, das zu genau einer Klasse gehört, also von ihr erzeugt wurde.

19.2 Objekt

Ein Objekt ist eine Instanz einer Klasse, der man einen Namen gegeben hat, also eine ganz bestimmte Instanz. Im Endeffekt wird nur mit den Objekten gearbeitet.

Vom Prinzip her kann man Objekte mit Variablen und Klassen mit Datentypen vergleichen.

19.3 Klasse

Eine Klasse ist die Definition, welche Attribute und Fähigkeiten ein Objekt später haben soll. Also quasi ein Bauplan für Objekte.

19.3.1 Basisklasse

Die Klasse, von der eine Klasse erbt, also quasi ihr direkter Vorfahre.

19.4 Methode

In der OOP werden die Funktionen einer Klasse als Methoden bezeichnet.

19.5 Attribut

Die Variablen einer Klasse heißen Attribute. Attribute werden oft auch Eigenschaften eines Objekts genannt, zusammen mit den Methoden (wobei die Attribute Eigenschaften wie Farbe oder Größe sein können – Attribute eben – und Methoden z. B. Öffnen oder Lesen und Schreiben, also eine gewisse Fähigkeit).

19.6 Konstruktor

Methode, die beim Instanzieren[1] eines Objektes einmal automatisch aufgerufen wird. Diese wird in der Regel genutzt, um Initialisierungen vorzunehmen.

19.7 Destruktor

Das Gegenteil vom Konstruktor: Die Methode wird beim Löschen des Objektes automatisch aufgerufen.

PHP 4 kennt keine Destruktoren, erst die kommende Version 5 (siehe 20.5).

[1] Erstellen einer Instanz

Teil VI

Objektorientierung praktisch

20 Objektorientierung in PHP

> Objektorientiert:
> Den Code habe ich von meinem
> Vorgänger geerbt.
>
> Kristian Köhntopp

PHP, ursprünglich eine prozedurale Skriptsprache, hat erst mit Version 4 brauchbare objektorientierte Züge angenommen. In diesem Kapitel will ich versuchen zu zeigen, wie man objektorientiert in PHP programmiert und auch ein paar Beispiele geben, um zu sehen, wie man von der Problemstellung zur OO-Umsetzung kommt und auch, daß sich das bißchen Mehraufwand wirklich lohnt (Stichwort Übersichtlichkeit/Wiederverwendbarkeit). Das folgende Kapitel gilt der Einfachheit halber nur für PHP 4.06 aufwärts, exklusive PHP 5.

20.1 Klassen in PHP

Klassen beginnen in PHP mit dem Schlüsselwort `class`, gefolgt von einem Bezeichner, dem Klassennamen. Wie bei Funktionen wird der Klassenrumpf von geschweiften Klammern umschlossen. Alle Attribute bzw. Klassen-/Instanzvariablen sowie Methoden (einschließlich der Konstruktoren) müssen sich in diesem Block befinden, um als zur Klasse gehörig erkannt zu werden[1].

Zu beachten ist, daß der Name „stdClass" in PHP4 reserviert ist, ebenso wie alle Methodennamen, die mit zwei Underscores (z. B. `__sleep`, `__wakeup`) beginnen.

Beispiel:

```
class DSP_CR {
  function Auto() {
    ...
  }

  ...
}
```

20.1.1 Konstruktoren

Konstruktoren in PHP heißen grundsätzlich genau so wie die Klasse, zu der sie gehören. Anders als etwa in Java kann es in PHP immer nur genau einen Konstruktor pro Klasse geben. Durch die Eigenschaft von PHP-Funktionen, Parameter im Funktionskopf auf

[1] Vorsicht: In PHP3 gilt eine Methode mit dem Namen einer Klasse auch dann als Konstruktor dieser Klasse, wenn sich die betreffende Methode gar nicht innerhalb dieser Klasse befindet!

Defaultwerte setzen zu können, ergibt sich jedoch die Möglichkeit, durch `if`-Abfragen oder `switch`-Blöcke die meisten Konstellationen von Parameterübergaben abzudecken.

Eine Besonderheit beim Umgang mit Konstruktoren ist schließlich noch der implizite Aufruf des Konstruktors der Basisklasse in dem Fall, daß eine Klasse keinen Konstruktor besitzt. Die Unfähigkeit von PHP3, dies zu meistern, ist auch der Grund, warum nach Möglichkeit PHP4 eingesetzt werden sollte.

20.1.2 Vererbung in PHP

Vererbung wird mittels des Schlüsselwortes `extends` signalisiert, dem der Name der Klasse, von der geerbt werden soll, folgen muß. Der gesamte Vererbungsausdruck ist jedoch optional.

Beispiel:

```
class Auto {
  ...
}

class DSP_CR extends Auto {
  ...
}
```

20.1.3 Attribute in PHP

Attribute bzw. Klassen-/Instanzvariablen werden in PHP mit dem Schlüsselwort `var` definiert und können optional mit beliebigen statischen Werten initialisiert werden. Funktionsaufrufe, die Verwendung des Punktoperators oder Variablenzuweisungen sind bei ihrer Definition nicht erlaubt, hierfür sollte der Konstruktor verwendet werden. Die Wertzuweisung von Attributen direkt bei ihrer Deklaration ist allerdings nicht empfohlen[2]; hierfür sollte genauso der Konstruktor verwendet werden wie für Konstanten, die in PHP ja mittels der `define()`-Funktion (8.2.7) deklariert werden.

Beispiel:

```
class Auto {
  var $baujahr;
  var $farbe;
  ...
}

class DSP_CR extends Auto {
  // hier in Ermangelung von Methoden noch
  // nicht über den Konstruktor gesetzt
  var $farbe = "metallic";
  ...
}
```

[2] Noch ist es möglich, es wird aber wahrscheinlich irgendwann nicht mehr erlaubt sein

Zur Referenzierung von Attributen mehr weiter unten (20.2.1).

20.1.4 Methoden in PHP

Da Methoden nichts anderes sind als Funktionen innerhalb einer Klasse, werden diese auch genau wie Funktionen definiert. Sie können also Parameter unterschiedlicher Anzahl übergeben bekommen und Werte mittels `return` zurückliefern.

Beispiel:

```
class Auto {
  var $baujahr;
  var $farbe;

  function Auto() {
    ...
  }
  ...
}

class DSP_CR extends Auto {
  // Da DSP_CR von Auto erbt, müssen hier keine Variablen
  // definiert werden. Die geerbten Attribute (und auch
  // Methoden) der Basisklasse kann man natürlich nutzen.

  function DSP_CR($farbe) {
    if (empty($farbe))
      $this->farbe = "metallic";
    else
      $this->farbe = $farbe;
  }

  function setzeFarbe($farbe) {
    ...
  }
  ...
}
```

Zur Referenzierung von Methoden mehr weiter unten (20.2.1).

20.1.5 Klassen dokumentieren

Eine äußerst sinnvolle Methode, Klassen zu dokumentieren und zu kommentieren stellt PHPDoc dar (14). Hierbei wird die Klasse selbst ebenso berücksichtigt wie Konstruktoren, Attribute und Methoden samt ihrer spezifischen Eigenschaften (Copyright, Parameter, Rückgabewerte; ggf. sogar Sichtbarkeit). Mehr dazu im angegebenen Kapitel.

20.2 Objekte und Referenzierung in PHP

Wie wir bereits aus dem Kapitel „Bedeutung von Objektorientierung“ (18) wissen, sind Objekte konkrete Exemplare einer Klasse. In PHP definiert man Objekte genau wie Variablen mit dem Dollarzeichen. Die Erzeugung eines Objektes geschieht grundsätzlich außerhalb der Klasse, zu der es gehört (außer bei rekursiven Definitionen). Trotzdem kann eine Klasse natürlich beliebige Objekte anderer Klassen erzeugen und auch als eigene Attribute verwalten.

Folgende Syntax, in der das Schlüsselwort `new` neu eingeführt wird, erzeugt in PHP ein Objekt:

```
$meinWagen = new DSP_CR();
```

Das Objekt `meinWagen` wird hierbei von der Klasse `DSP_CR` erzeugt, die dazu ihren parameterlosen Konstruktor benutzt.[3] Der Quelltext der Klasse muß natürlich bereits vom PHP-Parser gelesen worden sein, bevor eine solche Zuweisung erfolgen kann. Üblicherweise lagert man Klassendefinitionen in Include-Dateien aus, z. B. nach dem Schema `<KlassenName>.inc`, und importiert sie mit `include()` oder `require()`.

Wird anstelle des parameterlosen Konstruktor einer verwendet, der Parameter erwartet, erfolgt die Objektinitialisierung ganz analog, indem man wie bei Funktionsaufrufen die Parameter in entsprechender Reihenfolge innerhalb der runden Klammern angibt.

Auf das so erzeugte Objekt kann nun so lange zugegriffen werden, bis es explizit auf NULL gesetzt wird oder das Skript beendet wird. Allgemein kann man mit Objektvariablen übrigens ganz analog zu normalen Primitivtyp-Variablen Objekt-Zuweisungen durchführen. Folgender Code ist somit gültig:

```
$meinWagen = new DSP_CR();
$meinZweitwagen = new DSP_CR();
$meinWagen = $meinZweitwagen;
```

Achtung: Bis vor der Zuweisung waren `meinWagen` und `meinZweitwagen` noch völlig unterschiedliche Objekte! Danach enthalten die beiden Objektvariablen zwar auch noch unterschiedliche Referenzen (zeigen also nicht auf das gleiche Objekt), aber der Inhalt der beiden Objekte ist nun so lange gleich, bis eines der Objekte geändert wird.[4]

Innerhalb einer Klasse gehören die Attribute und Methoden der Klasse immer dem gerade zu betrachtenden Objekt, das wie gesagt nur außerhalb der Klasse existiert. Da die Klasse quasi den „Plan“ für ein Objekt darstellt, liegt der Gedanke nahe, daß dieser Plan in dem Moment, wo man ein spezielles Objekt betrachtet, die Eigenschaften des Objekts annimmt. Wenn also die Klasse eigentlich nur der Bauplan für Objekte ist, so verwandelt sie sich doch in dem Moment, wo man ein einzelnes Objekt betrachtet, in das Objekt selbst! Dadurch hat man die Möglichkeit, ein Objekt abhängig von seinen ganz „persönlichen“ Eigenschaften zu verändern, ohne es beim Erstellen der Klasse schon zu kennen! :-)

[3] Vorsicht: Die bisherigen Beispiele waren noch unvollständig!
[4] Wenn das auch an dieser Stelle etwas verwirrend erscheint, so sollte es doch im Folgenden klar werden.

20.2.1 Referenzierung

Um eine Methode eines Objekts bzw. einer Klasse aufzurufen, muß man quasi den „Umweg“ über das Objekt bzw. die Klasse gehen, denn PHP muß schließlich wissen, wessen Methode aufgerufen werden soll (unterschiedliche Klassen können natürlich Methoden mit genau demselben Namen definieren!).

Will man also Attribute oder Methoden referenzieren, beginnt man mit dem für PHP typischen Dollarzeichen, gefolgt vom Namen des zu referenzierenden Objektes, einem Strichpfeil (`->`) und dem Namen des Attributs bzw. der Methode[5]. Will man innerhalb einer Klasse auf die Attribute oder Methoden derselben Klasse zugreifen (die wie gesagt in dem Moment, wo man ein Objekt betrachtet, das Objekt selbst darstellt), muß man das Schlüsselwort `this` anstelle des Objektnamens verwenden.[6]

Eine kleine Anmerkung noch: Wie vielleicht bekannt sein dürfte, werden in in doppelten Anführungsstrichen geschriebenen Strings Variablen ersetzt. Das gilt aber nur für solche Variablen, deren Namen nur aus Buchstaben bestehen. Insbesondere werden Referenzen nicht erkannt, da sie notwendigerweise den Strichpfeil enthalten. Hier ist also ein Unterbrechen des Strings unter Zuhilfenahme des Punkt-Stringoperators geboten.

20.2.1.1 Kopier- vs. Referenzsemantik

Beim Aufruf einer Methode kann man dieser – vorausgesetzt, diese erlaubt das – Parameter übergeben. Ist einer dieser Parameter ein Objekt, genauer: eine Referenz auf ein Objekt, so findet in PHP bei der Übergabe dasselbe Vorgehen statt wie bei gewöhnlichen Primitivtypen (Integer, Boolean, String): Es wird eine Kopie des ursprünglichen Datums, bei Objekten also eine Kopie der Referenz, erzeugt und übergeben. Da eine Referenz auf ein Objekt ja immer nur das Objekt bezeichnet, es aber nicht selbst darstellt, kann folglich keine Methode das ursprüngliche Objekt (z. B. mit NULL) überschreiben. Auch in Java, wo bei Objekten statt der Kopiersemantik die sog. Referenzsemantik zum Einsatz kommt[7], funktioniert dies nicht. Der Unterschied zwischen Java und PHP in der Behandlung von Objekten kommt erst dann zum Tragen, wenn versucht wird, den Wert eines Objekt-Attributes zu ändern. Während Java innerhalb einer aufgerufenen Methode nämlich tatsächlich das Attribut sowohl der übergebenen als auch der ursprünglichen Referenz auf das Objekt ändert, erfolgt dies bei PHP standardmäßig nur auf der Referenzkopie, die lokal in der Methode sichtbar ist.

Grundsätzlich ist es natürlich immer möglich, mittels einer `return`-Anweisung die lokale Referenz zurückzugeben und der ursprünglichen Objektvariable zuzuweisen.

20.2.1.2 Ausweg aus dem Referenzdilemma

Wie also kann man sicherstellen, daß eine Referenz auch als solche behandelt wird? In PHP gibt es dafür spezielle Syntax, bei der das Kaufmanns-Und den Referenz-Charakter einer Zuweisung oder einer Variablen-Betrachtung anzeigt:

[5] Im gesamten Ausdruck also nur ganz vorne ein Dollarzeichen. Steht hinter dem Pfeil auch ein Dollar, wird dieser Teil als variabel aufgefaßt und zuerst ersetzt!
[6] Eine implizite Referenzierung ohne Angabe eines Objekts wie in Java gibt es in PHP nicht.
[7] man spricht daher auch von „Call by value“ bzw. „Call by reference“

```
$a = &$b;
$c =& $a;
$c = 42;
```

Die beiden obigen Zuweisung bewirken gleichermaßen, daß jeweils beide Variablen nicht nur denselben Wert haben, sondern auch eine Wertänderung einer der Variablen automatisch die exakt selbe Wertänderung der anderen Variablen bewirkt. Die letzte Anweisung bewirkt folglich, daß alle drei Variablen den Wert 42 annehmen.

Auch bei Funktions- und Methodenaufrufen wird normalerweise Kopiersemantik angewandt. Um explizit Referenzsemantik zu fordern, stellt man hierbei bei einer Funktions-/Methodendefinition einer zu referenzierenden Parameter-Variablen ein Kaufmanns-Und voran:

```
function Plus($var) { $var++; }
function Minus(&$var) { $var--; }
$var = 42;
Plus($var);
Minus($var);
echo $var;
```

Die Ausgabe obigen Codes ist natürlich 41, da nur die Funktion `Minus` den Wert der Variablen `$var` ändert. In Java wäre das nicht so offensichtlich, da hier u. U. eine Klassenvariable namens `var` existieren könnte, die implizit gemeint sein könnte. In PHP müßte man dann aber `$this->var` schreiben.

Unter 20.7.2 findet sich noch eine Übung zu komplexeren Anwendungen der Referenzsemantik.

20.2.1.3 foreach mit Objektreferenzen

Wie bereits im `foreach`-Kapitel 8.2.19 beschrieben, benutzt diese Schleifenvariante[8] stets Kopien der Arrayinhalte. Hat man nun ein Array von Objektreferenzen, kann dies leicht zu unerwünschten Effekten führen. Deshalb macht es hier Sinn, nur den jeweiligen Array-Schlüssel in Erfahrung zu bringen und diesen zu nutzen, um auf das Original-Objekt zuzugreifen:

```
$car = new Auto();
$dsp = new DSP_CR();
$my_cars = array($car, $dsp);

// mit Kopiersemantik
foreach ($my_cars as $object_copy)
  echo $object_copy->farbe."\n";

// Referenzsemantik I
foreach ($my_cars as $key=>$object)
```

[8] Dieser Begriff beschreibt zwar auch das, was sich von einem Schleifendurchlauf zum nächsten ändert; das ist hier aber nicht gemeint


```
  echo $my_cars[$key]->farbe."\n";

// Referenzsemantik II
foreach (array_keys($my_cars) as $key)
  echo $my_cars[$key]->farbe."\n";
```

In diesem Beispiel habe ich direkt auf die Klassenvariable `$farbe` zugegriffen, weil die Klasse Auto außer dem Konstruktor keine bekannten Methoden hat. Das ist in PHP4 auch gar kein Problem; in PHP5 wird das je nach Sichtbarkeitsdefinition aber nicht mehr möglich sein. Außerdem sollte man sich grundsätzlich angewöhnen, auf Attribute nur per Getter- und Settermethoden zuzugreifen, da in diesen dann (ggf. auch erst nachträglich) geeignete Plausibilitätstests gemacht und diese bei Bedarf in erbenden Klassen überschrieben werden können.

20.3 Methoden-Aufrufe

20.3.1 static

Manchmal ist es nötig, eine Methode einer Klasse aufzurufen, ohne ein Objekt derselben erzeugt zu haben. In diesem Fall kann man einen sogenannten „static"-Aufruf machen. Zu beachten ist dabei, daß eine so aufgerufene „Klassenmethode" nirgends auf die Attribute der Klasse, in der sie sich befindet, zugreifen darf, da diese ja nur im Kontext einer Instanz (Objekt) vorhanden sind. Lokale und globale Variablen dürfen dagegen natürlich benutzt werden.

Ein primitives Beispiel:

```
class Auto {
...
  function Erfinder() {
    return "Etienne Lenoir, Carl Benz";
  }
...
}
echo Auto::Erfinder();
```

20.3.2 parent

Im Falle von Zugriffen auf geerbte, aber überschriebene Methoden sollte man statt des Klassennamens das Schlüsselwort `parent` benutzen. Dadurch taucht der Name der beerbten Klasse nur hinter `extends` auf, was das nachträgliche Vornehmen von Änderungen z. T. stark erleichtert. Außerdem wird dadurch auch der Unterschied zwischen „static" (kein Objekt vorhanden) und normalem Instanz-Methodenaufruf deutlich.

Beispiel: Angenommen, wir hätten in der Klasse `DSP_CR` die Methode `lenke(drehung)` der Klasse `Auto` überschrieben, von der `DSP_CR` erbt. Will man nun innerhalb von `DSP_CR` auf die gleichnamige Methode der Basisklasse `Auto` zugreifen, kommt folgende Syntax zum Einsatz:

```
...
  function lenke($drehung) {
    // Servo aktivieren
    $drehung = $this->servo($drehung);
    // Methode der Basisklasse aufrufen
    parent::lenke($drehung);
  }
...
```

20.4 Das fertige Beispiel

```
/**
 * Auto Klasse
 *
 * @author  DSP
 * @version 1.0
 */
class Auto {

  /**
   * Baujahr
   * @type int
   */
  var $baujahr;

  /**
   * Farbe
   * @type string
   */
  var $farbe;

  /**
   * Konstruktor
   * @param     $farbe  gewünschte Farbe
   */
  function Auto($farbe) {
    $this->farbe = $farbe;
    $this->baujahr = date();
  }

  /**
   * Ändert die Wagenfarbe
   * @param $farbe       gewünschte Farbe
   */
```



```
  function setzeFarbe($farbe) {
    $this->farbe = $farbe;
  }

  /**
   * Welches Baujahr?
   *
   * @return Das Baujahr
   * @returns string
   */
  function Baujahr() {
    return $this->baujahr;
  }
}

/**
 * DSP CR Klasse
 * Konstruktor der Basisklasse wird implizit aufgerufen
 *
 * @author  DSP
 * @version 1.0
 */
class DSP_CR extends Auto {

  /**
   * Konstruktor
   * eigentlich wäre gar keiner notwendig,
   * aber wir wollen ja die Farbe initialisieren...
   */
  function DSP_CR($farbe) {
    if (empty($farbe))
      $farbe = "metallic";

    parent::Auto($farbe);
  }
}
```

Alternativ zum letzten Codeteil, in dem der Konstruktor der Basisklasse aufgerufen wird, kann man auch folgende Syntax verwenden, die unabhängig vom Namen der Basisklasse ist - in Java würde man das übrigens mit einem einfachen `super(parameter)` machen.

```
class A extends B {
  function A($param) {
    $parent = get_parent_class($this);
    $this->$parent($param);
  }
```

```
}
```

20.5 Ausblick: PHP5

Mit PHP 5 bricht ein neues Zeitalter für die Skriptsprache an. Endlich halten die zentralen Bestandteile der OOP Einzug und ermöglichen es damit, auch in dieser Sprache tatsächlich objektorientiert zu programmieren. Natürlich ist die neue Zend-Engine auch um einiges schneller, aber ihr größter Vorteil ist sicherlich, daß sie Objekte grundsätzlich mit Referenz-Semantik behandelt[9] statt wie bisher standardmäßig mit Kopiersemantik (wie es bei Primitivtypen wie `integer` auch weiterhin der Fall ist). Eine echte Kopie stellt man dann mittels der nun jeder Klasse inhärent zugehörigen „magic" Methode `__clone` her, also z. B. `$myObject->__clone()`. Diese neue Spezialmethode ermöglicht aber z. B. auch das gleichzeitige, teilweise Modifizieren der Kopie.

Neu ist auch die Unterstützung eines der zentralen Grundsätze der OOP: Sichtbarkeit. Mit PHP5 kann man Attribute und Methoden als *public*, *private* oder *protected* deklarieren und damit einschränken, wer darauf von wo zugreifen darf. Wem das Java vorkommt, dem werden auch `static` und `final` sowie `const` etwas sagen: Ersteres ist vom Prinzip her weiter oben beschrieben (20.3.1, im Unterschied dazu wird einfach ein neues Schlüsselwort eingeführt, das bewirkt, daß `$this` nicht verwendet werden darf); `final` beschreibt nicht in erbenden Klassen überschreibbare Methoden bzw. Attribute und `const` konnte man bisher mittels `define()` simulieren: konstante, also nach dem Setzen nicht mehr veränderbare Instanzvariablen.

Konstruktoren heißen jetzt standardmäßig `__construct`, womit der Konstruktor der Basisklasse mittels `parent::__construct()` eindeutig bestimmt ist. Analog dazu heißen die neu eingeführten Destruktoren, die immer dann automatisch aufgerufen werden, wenn die letzte Referenz auf ein Objekt gelöscht wird, `__destruct`. Mit ihnen kann man z. B. Debuginformationen sammeln oder Datenbankverbindungen sauber beenden.

Auch von Java bekannt sind abstrakte Klassen und Interfaces, also einerseits nichtinstanzierbare Klassen mit Methoden, die von erbenden Klassen überschrieben werden **müssen** und andererseits Schnittstellenbeschreibungen, die definieren, welche Funktionalität (Methoden) eine Klasse, die ein bestimmtes Interface *implementiert*[10], bieten muß.

Schließlich gibt es nun auch ein ausgeklügeltes System zum Abfangen von Exceptions (Ausnahmen, das sind behandelbare Laufzeitfehler im Unterschied zu nicht abfangbaren, z. B. dem Absturz der PHP-Engine). Wen wundert's noch, daß sich dieses try-catch nennt?[11]

Wem PHPDoc nicht genug ist, der kann nun auch ganz explizit die erwarteten Klassenzugehörigkeiten vor die Variablen im Parameterteil von Methodendeklarationen schreiben, also z. B.:

```
...
  function setMetallic(Auto $car) {
    $car->setzeFarbe("metallic");
```

[9] Also nicht nur bei Zuweisungen, sondern auch bei Parameterübergaben an Funktionen und Methoden!
[10] Implementieren, engl. to implement, heißt in diesem Zusammenhang genau das Beschriebene
[11] Richtig, auch hier läßt wieder Java grüßen! :-)


```
  }
...
```

Diese Methode würde also alle Objekte akzeptieren, die von Auto oder einer davon erbenden Klasse (z. B. DSP_CR) instanziert worden sind. Falls ein Objekt von anderem Typ oder gar ein Primitivtyp übergeben wird, wird eine Fehlermeldung ausgegeben.

Nett ist auch, daß nun von Methoden zurückgelieferte Objekte direkt angesprochen, sprich dereferenziert, werden können. In PHP4 muß der Rückgabewert eines Funktionsaufrufs immer erst in einer (Objekt-) Variablen abgespeichert werden, auf die man dann weitere Aufrufe anwenden kann. Beispiel für die neue Freiheit:

```
...
  class MyTest {
    var $test;
    function __construct() {
      $this->test = "42 is the answer.";
    }
    function print() {
      echo $this->test;
    }
  }
  MyTest()->print();
...
```

Wer schon einmal herausfinden wollte, von welcher Klasse ein Objekt ist (zur Laufzeit, sonst ist das ja trivial), wird sicherlich `is_a` und Konsorten bemüht haben. Mit PHP5 gibt es nun ganz analog zu Java die Infix-Notation `instanceof`, die dasselbe ermöglicht (im Gegensatz zu `is_a` wird der Klassenname aber nicht als PHP-String aufgefaßt, sprich gequotet).

Die neue `__autoload()` Funktion erleichtert das Einbinden von Klassen, indem sie immer dann automatisch aufgerufen wird, wenn eine Klasse nicht gefunden wird. Mit ihrer Hilfe kann man z. B. vom Namen der Klasse abhängig verschiedene includes definieren.

Ganz besonders schick ist die extra geschaffene Möglichkeit, Methodenaufrufe sowie das Setzen und Abfragen von Variablen zu überschreiben (mittels Methoden `__call`, `__get` und `__set`). Die Setter-Funktionen werden dabei konsequenterweise nicht nur bei Zuweisungen, sondern auch auch beim Inkrementieren und Dekrementieren aufgerufen. Mit diesen Eigenschaften eignen sich diese Funktionen natürlich besonders gut zum sauberen Debuggen.

Bleibt noch anzumerken, daß all dies rückwärtskompatibel implementiert wurde, d. h. PHP4-Skripts sollten mit evtl. leichten Anpassungen auch unter PHP5 laufen. Das ist wohl der Punkt, wo sich die Geister scheiden und Lager bilden – OOP-pur-Verfechter auf der einen und PHP3-Hacker auf der anderen Seite . . .

Zu den weiteren Neuerungen in PHP5 zählen die rundum erneuerte XML-Unterstützung (libxml2-basiert), SQLite-Integration (dafür ist MySQL-Support nicht mehr direkt eingebaut) sowie verbesserte Handhabung von Streams[12].

[12] Wobei diese sicher trotzdem keine so zentrale Rolle spielen werden wie etwa in C++

Wer
sich selbst ein Bild machen will, sollte sich mal *http://www.php.net/zend-engine-2.php* ansehen.

20.6 Konzeptionelle Beispiele

Für Neulinge der Objektorientierung stellt sich anfangs oft die Frage, wie man eine Problemstellung am besten abstrahiert und „objektorientiert“ denkt. Eine generell optimale Lösung gibt es wie immer nicht, aber vielleicht helfen dir ja die folgenden Beispiele aus meiner Programmiererfahrung.

Zuerst jedoch die Methode, die Christoph bevorzugt:
Man abstrahiert von der Problemstellung soweit, daß man Subjekte, also Teile des Ganzen mit individuellen Eigenschaften, identifizieren kann. In der Programmierpraxis sind das z. B. eigenständige Prozesse wie Datei-Ein-/Ausgabe oder die jeweils zentrale Informationseinheit, deren Daten z. B. in einer Datenbank festgehalten werden[13]. Hat man diese Abstraktion erst einmal geschafft, fällt es leicht, diese Subjekte mit Objekten zu assoziieren, die zu einer zu erstellenden Klasse gehören. Die Eigenschaften des Subjekts setzt man der OO-Logik folgend in Form von Attributen (Klassenvariablen) um. Schließlich muß man sich noch über die Schnittstelle Gedanken machen – i. A. sind Konstruktor, Lese- und Schreibzugriff für die Eigenschaften und einige zusätzliche Funktionen nötig, die allesamt als Methoden der Klasse implementiert werden.

Und nun mein Ansatz:
Als erstes sollte man das Problem auf die zentrale Funktionalität reduzieren, denn diese bildet i. A. die Schnittstelle der späteren Klasse, also die benötigten Funktionen. Darüber hinaus werden natürlich meist noch weitere Funktionen benötigt, diese sind aber intern und könnten daher ‚private‘ bzw. ‚protected‘ deklariert werden.

Zu beachten ist auch, daß man in erster Linie die Funktionalität in eine Klasse steckt, die gemeinsame Daten (die späteren Attribute) verwendet. Es folgen einige Beispiele – wer noch Anschauliches hat, kann diese gerne an Christoph schicken, der sich mit Objektorientierung in PHP mindestens so gut auskennt wie ich. ;-)

20.7 Übung

20.7.1 OOP

Das Beispiel aus Abschnitt 18.4.1 bietet sich an, um OOP verstehen zu lernen. Folgende Aufgabe ist zu lösen:

Schreibe eine Klasse „Image“, die mit Hilfe der PHP-Image-Funktionen[14] ein leeres Bild erzeugt und die Methode `show()` implementiert. Leite von dieser Klasse eine neue namens „Point“ab mit der Methode `set(x, y)`, die einen Punkt malt. Programmiere schließlich die Klassen „Circle“ (Kreis), „Rectangle“ (Rechteck) als Erben von „Point“ sowie die Klasse

[13] Siehe auch die unten folgenden Beispiele
[14] Spätestens hier solltest du einen Blick in das offizielle PHP-Manual werfen; dort gibt es ein Liste aller Image-Funktionen samt Syntax.

Gemeinsame Daten	Zentrale Funktionalität

• IMAP-Session/Mailbox
• ID der aktuellen Nachricht
• String mit Meldungen
• Sortierungsdaten
• Anzahl Nachrichten in der Mailbox
	• Mailbox-Übersicht ausgeben
	  – einzelne Daten aus den Mail-Headern lesen
	  – Sortierung ändern
	• einzelne Mail lesen
	  – einzelne Daten aus den Mail-Headern lesen
	  – Umbruch/Formatierung
	• Mail verschicken
	• Mail(s) löschen
	• Attachmentmanagement
	• Meldungen/Fehler ausgeben

Tabelle 20.1: IMAP-Webmail-System

Gemeinsame Daten	Zentrale Funktionalität
zusätzlich zu Webmail (Vererbung!) • Newsgroupdaten	zusätzlich zu Webmail (Vererbung!) • Newsgroup-Liste ausgeben • Newsgroup wechseln • Mail verschicken • Mail(s) löschen • Meldungen/Fehler ausgeben

Tabelle 20.2: IMAP-Webnews-System

Gemeinsame Daten	Zentrale Funktionalität
• Benutzer • String mit Meldungen	• Adressen hinzufügen • Adressen aktualisieren • Adressen löschen • Adressauswahlliste • Suchfunktion • Adreßliste • Ausführliche Anzeige • Schnittstelle zu Webmail • Exportmöglichkeit

Tabelle 20.3: Mehrbenutzer-Adreßbuch

Gemeinsame Daten	Zentrale Funktionalität
• Benutzer • String mit Meldungen	• Bookmarks hinzufügen • Bookmarks aktualisieren • Bookmarks löschen • Bookmarkauswahlliste • Gruppennamen ändern • Gruppen löschen • Gruppenauswahlliste • Anzeige/Ausgabe

Tabelle 20.4: Mehrbenutzer-Bookmarkverwaltung

„Square“ (Quadrat), die „Rectangle“ erweitert. Erstelle von jeder malfähigen Klasse ein Objekt und benutze es jeweils, um das jeweilige Symbol auszugeben.

Eine mögliche Lösung findet sich in Abschnitt B.9.

20.7.2 Referenzsemantik

Wenn man von anderen objektorientierten Sprachen wie Java her gewohnt ist, bestimmte Dinge mit Referenzen machen zu können, wird man von PHP 4 zumindest anfangs enttäuscht sein.[15] Wie sich zeigt, kann man aber auch in PHP 4 so programmieren, daß Referenzen auch als solche behandelt werden. In dieser Übung soll deshalb eine bestehende Implementierung derart abgeändert werden, daß sie alle Referenzen auch als solche behandelt, d. h. z. B. bei Zuweisungen statt Kopier- Referenzsemantik anzuwenden.

Vorgegeben ist folgende „naive“ Implementierung, die die besonderen Erfordernisse der Verwendung von Referenzen schlicht ignoriert und damit sicher nicht das Gewünschte leistet. Außerdem steht unten noch das erwartete Ergebnis, also das, welches eine korrekte Implementierung erzeugen würde – und deine Lösung ist das doch hoffentlich, oder? :-)

Hinweis: Eine korrekte Implementierung kann schon erreicht werden, wenn pro Zeile maximal ein Zeichen (welches wohl?) hinzugefügt wird.

```
class MyArray {
  var $array;
  function MyArray() {}
  function add($val) {
    $this->array[] = $val;
  }
}

class Test {
  var $var;
  function Test($var, $array) {
    $this->var = $var;
    $array->add($this);
  }
  function setVar($var) {
    $this->var = $var;
  }
  function getVar() {
    return $this->var;
  }
}
```

```
$array = new MyArray();
$test  = new Test(42, $array);
$test2 = $test;
```

[15] Mit PHP 5 wird standardmäßige Referenzsemantik für Objekte eingeführt, so daß sich dann das folgende zumindest teilweise erübrigt

```
$test->setVar(0);
echo $test->getVar()." ".$test2->getVar()."\n";
var_dump($array);
```

Das Beispiel soll zuerst „0 0“ ausgeben. Bei falscher Implementierung (wie hier) wird statt dessen „0 42“ ausgegeben. Der Variablen-Dump schließlich soll bestätigen, daß tatsächlich eine Referenz auf das Objekt `$test` im Array `$array` abgelegt wurde. Ist dies nicht der Fall, ist der Wert der Variablen `var` 42 statt 0 und es steht dann kein Kaufmanns-Und vor `object`.

```
object(myarray)(1) {
  ["array"]=>
  array(1) {
    [0]=>
    &object(test)(1) {
      ["var"]=>
      int(0)
    }
  }
}
```

Eine mögliche Lösung findet sich in Kapitel B.10.

Teil VII

PHP Erweitert

21 Sessions

21.1 Was sind Sessions?

Mit „Session“ (engl.: Sitzung) bezeichnet man alle Daten, die dem Server einer Site (z. B. einem personalisierten System) während dem Verweilen eines Benutzers auf derselben bekannt sind. Verschiedene Benutzer werden somit auch durch verschiedene Sessions identifiziert.[1]

Technisch kann man sich die über einen Nutzer bekannten Daten als einen „serverseitigen Cookie[2]“ vorstellen. Die Daten können über einzelne Seiten hinweg benutzt und manipuliert werden, ohne dabei den Bezug zum jeweiligen Benutzer bzw. Abrufer zu verlieren. Sie werden aber – im Gegensatz zum normalen Cookie – nicht beim Client, sondern auf dem Server gespeichert. Das klingt natürlich erstmal fantastisch, da man sich so keine Gedanken über Browser-spezifische Probleme[3] machen müßte. Doch gleich die schlechte Nachricht: So einfach funktioniert das nicht, denn der Server kann bei einer normalen Anfrage nicht herausfinden, welche gespeicherte Session er dem Client zuweisen soll. Deswegen muß diese Aufgabe wiederum der Client übernehmen: Er muß eine sog. Session-ID zur Identifikation selbst speichern, was er normalerweise wieder mit Cookies macht – ein Teufelskreis.

Keine Arme, keine Kekse ... Es gibt auch die Möglichkeit, die Session-ID per URL weiterzugeben. Es muß dabei – egal ob POST- oder GET-Anfrage – die 32-stellige ID[4] immer übergeben werden. Das passiert bei einer POST-Anfrage (Formular) über versteckte Felder[5]:

```
<form>
<input type="hidden"
       name="PHPSESSID"
       value="edb0e8zz5et4e9042fe0176a89cbde16" />
</form>
```

und bei GET über die URL:

```
index.php?PHPSESSID=edb0e8zz5et4e9042fe0176a89cbde16
```

Einen Teil dieser Arbeit kann PHP selbst übernehmen, wenn man die entsprechenden Stellen in der Konfigurationsdatei `php.ini` richtig eingestellt hat:

```
url_rewriter.tags = "a=href,area=href,frame=src,input=src,form=fakeentry"
```

[1] Für komplexere Systeme gibt es zudem die Möglichkeit, Sessions zu gruppieren, indem verschiedene Sessionnamen vergeben werden. Mehr dazu weiter unten.
[2] Cookies sind Variablen mit Werten, die zwischen Server und Client (Browser) ausgetauscht werden.
[3] Manche Browser (oder deren User) nehmen keine Cookies an
[4] Die Session-ID ist ein md5-verschlüsselter Zufallswert. Die Chance, ihn zu erraten, ist $1 : 2^{128}$
[5] Im Beispiel heißt die Session „PHPSESSID“, das muß aber nicht immer so sein

Dies veranlaßt PHP dazu, die entsprechenden Erweiterungen bei den HTML-Tags <a>, <area>, <frame>, <input> und <form> selbst einzutragen.

Außerdem muß der Pfad, unter dem die Sessions auf dem Server gespeichert werden sollen, bei Nicht-UNIX-Systemen[6] noch angepaßt werden:

```
session.save_path = c:\temp\
```

21.2 Praxis

21.2.1 Datei 1 - index.php

```
<?php
session_start();

$string = "Test";
$integer = 1;
$array = array("Wert1", "Wert2");

session_register("string", "integer", "array");
?>
<html>
<head>
<title>Beginn</title>
</head>
<body>
<a href="index2.php">Weiter zu Index2</a>
</body>
</html>
```

21.2.2 Datei 2 - index2.php

```
<?php
session_start();
?>
<html>
<head>
<title>Beginn</title>
</head>
<body>
<pre>
<?php
print "String: ".$_SESSION['string']."<br />";
print "Integer: ".$_SESSION['integer']."<br />";
```

[6] Standard: /tmp/


```
print "Array:<br />";
print_r($_SESSION['array']);

// In der Session gespeicherten Wert verändern
$_SESSION['integer']++;

?>
</pre>
<a href="index2.php">Erneut aufrufen</a>
</body>
</html>
```

Rufen wir die Datei index.php auf, so wird zuerst eine Session durch session_start gestartet. Danach füllen wir 3 Variablen mit verschiedenen Werten, die dann durch session_register [7] für die Session registriert (also gespeichert) werden. Wenn man danach über den Link die 2. Seite aufruft, bekommt man die eben gespeicherten Werte angezeigt. Wir haben in diesem Fall keinen Versuch unternommen, die Session-ID manuell weiterzugeben, sondern überlassen diese Aufgabe PHP, was nicht unbedingt vorteilhaft ist, da PHP den Browser dazu veranlaßt, die Session-ID als Cookie zu speichern und gleichzeitig teilweise die ID der URL hinzufügt. Wenn wir eine Cookie-unabhängige Seite programmieren wollen, müssen wir diese Aufgabe selbst übernehmen und an jeden Link in der HTML-Ausgabe den String „*?SESSION-NAME=SESSION-ID*“ anhängen. Der Standard-Sessionname ist PHPSESSID. Wenn wir sichergehen wollen, benutzen wir die Funktion session_name(), die den Namen als String zurückgibt[8]. Das gleiche gilt für die Session-ID, hier verwenden wir die Funktion session_id() oder alternativ die Konstante SID. Beispiel:

```
<?php
// Unterbinden von Cookies:
ini_set("session.use_cookies", "0");

// Da wir die Session-ID automatisch anfügen,
// muß PHP diese Aufgabe nicht mehr übernehmen:
ini_set("url_rewriter.tags", "");

...

printf('<a href="index2.php?%s=%s">Weiter ...</a>',
       session_name(),
       session_id()
      );
?>
```

[7] Nimmt beliebig viele Parameter entgegen

[8] Setzen kann man den Sessionnamen, indem man derselben Funktion einen String als Parameter übergibt. Ein solcher Aufruf muß logischerweise immer vor dem ersten Benutzen/Manipulieren der Session gemacht werden.

21.2.3 Logout - logout.php

Irgendwann will man eine Session auch mal beenden. Das geht so:

```
session_unset();
session_destroy();
unset($mySessVar1, $mySessVar2, ...);
// jetzt sind die Variablen auch lokal nicht mehr definiert
header("Location: index.php?logout=true");
exit;
```

Aufzurufen per Link:

```
<a href="logout.php">Logout</a>
```

oder Button:

```
<form action="logout.php">
<input type="submit" value="Logout">
</form>
```

21.3 Beispiel Warenkorb

Ein gutes Beispiel für den Gebrauch von Sessions ist eine Warenkorbfunktion, wie man sie oft in Onlineshops sieht. Unser Warenkorb soll so einfach wie möglich sein:

- Übersichtsseite mit Produkten, die über einen Link bestellt werden können und dadurch in den Warenkorb gelegt werden.
- Übersicht der bereits bestellten Produkte mit der Möglichkeit, diese wieder zu entfernen.

21.3.1 Waren

Die Datei `waren.php` enthält ein Array[9] mit verfügbaren Produkten. Da diese Seite von allen anderen eingebunden wird, startet sie auch die Session und registriert die Variable `warenkorb` als Session-Variable.

```
<?php
ini_set("session.use_cookies", "0");
ini_set("url_rewriter.tags",   "");

session_start();
session_register('warenkorb');

$waren = array(
  '1' => array(
```

[9] Bei einem richtigen Onlineshop würden die Produkte mit Sicherheit aus einer Datenbank kommen


```
    'titel' => 'DSP Taschenbuch',
    'preis' => 9.90,
    'text' => 'Das Standardwerk zum Thema Datenbank,
               SQL und PHP als handliches Taschenbuch.'
  ),
  '2' => array(
    'titel' => 'DSP auf CD',
    'preis' => 19.90,
    'text' => 'Wenn Ihnen die Downloadzeit zu wertvoll ist,
               können sie DSP auch als CD bestellen.'
  ),
  '3' => array(
    'titel' => 'DSP Hörspiel',
    'preis' => 29.90,
    'text' => 'Das Standardwerk zum Thema Datenbank,
               SQL und PHP für Analphabeten.'
  )
);
?>
<html>
<head>
<title>Warenkorb</title>
</head>
<body>
```

21.3.2 Übersicht

Die Seite `index.php` soll einen Überblick über die angebotenen Produkte liefern. Weiter unten erhält man einen Überblick darüber, was man bereits bestellt hat. Über einen Entfernen-Link kann man seine bisherige Bestellung außerdem noch korrigieren.

```
<?php
include_once("waren.php");

foreach ($waren as $id => $produkt) {
    printf('<p>
            <b>%s</b><br />
            Preis: <b>Euro %01.2f</b><br />
            Beschreibung: <b>%s</b><br />
            <a href="bestellen.php?id=%d&%s">Bestellen</a>
            </p>',
           $produkt['titel'],
           $produkt['preis'],
           htmlentities($produkt['text']),
           $id,
           SID
```

```
        );
}

if (isset($_SESSION['warenkorb']) &&
   !empty($_SESSION['warenkorb'])) {

    print "Sie haben folgende Waren bereits ausgew&auml;hlt:";
    print "<ul>";
    foreach ($_SESSION['warenkorb'] as $id) {
        printf('<li>
                %s
                (<a href="entfernen.php?id=%d&%s">
                    Entfernen
                </a>)
                </li>',
               htmlentities($waren[$id]['titel']),
               $id,
               SID
              );
    }
    print "</ul>";
}
?>
</body>
</html>
```

21.3.3 Bestellen

Die Datei `bestellen.php` legt Produkte in den Warenkorb

```
<?php
include_once("waren.php");

if (!isset($_GET['id'])) {
    die("Kein Produkt ausgew&auml;hlt.");
}

// Wenn das Produkt noch nicht gekauft wurde...
if (!isset($_SESSION['warenkorb']) ||
    !in_array($_GET['id'], $_SESSION['warenkorb'])) {

    // In den Warenkorb legen:
    $_SESSION['warenkorb'][] = $_GET['id'];
}
?>
<p>
```



```
  Das Produkt wurde Ihrem Warenkorb hinzugef&uuml;gt.<br />
  <a href="index.php?<?php print SID; ?>">
    Zur&uuml;ck zur Startseite
  </a>
</p>
</body>
</html>
```

21.3.4 Entfernen

Mit der Seite `entfernen.php` kann man schließlich noch einzelne Produkte aus dem Warenkorb löschen.

```
<?php
include_once("waren.php");

// Wenn kein Produkt ausgewählt wurde, oder
// das Produkt nicht im Warenkorb ist...
if (!isset($_GET['id']) ||
    !in_array($_GET['id'], $_SESSION['warenkorb'])) {

  print("Sie haben dieses Produkt noch nicht bestellt,
         oder kein Produkt ausgew&auml;hlt.");
}
else {
  foreach ($_SESSION['warenkorb'] as $id => $produkt) {
    if ($produkt == $_GET['id']) {
      unset($_SESSION['warenkorb'][$id]);
    }
  }
  print "Das Produkt wurde aus Ihrem Warenkorb gel&ouml;scht."
}
?>
<br />
<a href="index.php?<?php print SID; ?>">
  Zur&uuml;ck zur Startseite
</a>
</body>
</html>
```

22 XML-Dokumente parsen

In diesem Abschnitt geht es um das Parsen von XML-Dokumenten[1]

22.1 Was ist XML?

Die Extensible Markup Language (XML) ist eine Methode, um strukturierte Daten in einer Textdatei abzulegen. Auf den ersten Blick scheint XML wie eine Weiterentwicklung von HTML. In Wirklichkeit basieren aber sowohl XML als auch HTML auf der zwar mächtigeren aber auch sehr schwer verständlichen Sprache SGML. Darüber hinaus wurde HTML nach Version 4 in XML umgesetzt und sinngemäßerweise in XHTML umgetauft.[2] Ein wichtiger Aspekt von XML ist die vollständige Plattformunabhängigkeit, da es sich nur um reine Textdaten handelt.

22.2 Parsen von XML-Daten

Unter PHP gibt es zu diesem Thema elegante Lösungen, wie zum Beispiel Expat (*http://www.jclark.com/xml/*). Expat ist ein SAX[3]-ähnlicher, ereignisorientierter XML-Parser.

Der Ablauf eines Parsevorgangs ist einfach:
Es werden zuerst sog. Handler („Behandler") für bestimmte Ereignisse definiert. Solche Ereignisse können sein:

- Startendes Element

  `<abschnitt titel="XML-Dokumente">`

- Abschließendes Element

  `</abschnitt>`

- CDATA (Normaler Text)
- PIs (Processing Instructions, Verarbeitungs-Instruktionen)

  `<?php echo "hello world!"; ?>`

- Kommentare

[1] *http://www.w3.org/XML/*
[2] In dieser neue Sprach- „Version" muß man sich übrigens strikt an die XML-Regeln halten, also alle Tags schließen, ihre Namen klein schreiben und Attributwerte in doppelte Anführungszeichen einschließen!
[3] Simple API for XML

  `<!-- eXpat-abschnitt-Ende -->`

- DTD-Tags

  `<![CDATA[ (\$a <= \$b \&\& \$a >= \$c) ]]>`

Findet der Parser ein startendes Element, wie

`<abschnitt titel="XML-Dokumente">`

, so ruft er die passende, vordefinierte Funktion auf und übergibt dieser den Namen des Elements(abschnitt) sowie die Attribute (titel = XML-Dokumente).

22.3 Beispiel 1

22.3.1 Die XML-Datei (tutorial.xml)

```
<?xml version="1.0"?>
<tutorial titel="expat">
    <abschnitt titel="Einfuehrung">
        <index keyword="expat" />
        Text...
    </abschnitt>
    <abschnitt titel="Beispiele">
        <index keyword="beispiele" />
        Wie im <ref id="expat">1. Abschnitt</ref> gezeigt, ...
    </abschnitt>
</tutorial>
```

22.3.2 Das PHP-Skript

```
// Zuerst definieren wir die Funktionen, die später auf
// die diversen Ereignisse reagieren sollen

/**
 * Diese Funktion behandelt ein öffnendes Element.
 * Alle Parameter werden automatisch vom Parser übergeben
 *
 * @param    parser    Object    Parserobjekt
 * @param    name      string    Name des öffnenden Elements
 * @param    atts      array     Array mit Attributen
 */
function startElement($parser, $name, $atts) {
  global $html;
```

```
  // Die XML-Namen werden in Großbuchstaben übergeben.
  // Deshalb wandeln wir sie mit strtolower() in Klein-
  // buchstaben um.
  switch (strtolower($name)) {
  case "tutorial":
    // Wir fügen der globalen Variable eine Überschrift hinzu:
    $html .= "<h1>".$atts["TITEL"]."</h1>";
    break;
  case "abschnitt";
    $html .= "<h2>".$atts["TITEL"]."</h2>";
    break;
  case "index":
    // Einen HTML-Anker erzeugen:
    $html .= "<a name=\"".$atts["KEYWORD"]."\"></a>";
    break;
  case "ref":
    // Verweis auf einen HTML-Anker:
    $html .= "<a href=\"#".$atts["ID"]."\">";
    break;
  default:
    // Ein ungültiges Element ist vorgekommen.
    $error = "Undefiniertes Element <".$name.">";
    die($error . " in Zeile " .
        xml_get_current_line_number($parser));
    break;
  }
}
```

Wenn im XML-Dokument ein öffnendes Element gefunden wird, passiert folgendes:

- startElement() wird mit den entsprechenden Parametern aufgerufen.
- in der switch()-Schleife wird überprüft, welches Element vorliegt
  - tutorial: Eine Überschrift wird an das Ausgabe-Dokument(`$html`) angehängt.
  - abschnitt: Eine kleinere Überschrift wird erzeugt.
  - index: Eine HTML-Referenz wird erstellt.
  - ref: Ein Verweis auf eine gesetzte Referenz wird erstellt.
  - Falls ein anderes Element übergeben wird, gibt das Script einen Fehler aus.
- Dabei werden die in `$atts` gespeicherten Elemente verwendet.

```
/**
 * Diese Funktion behandelt ein abschließendes Element
 * Alle Parameter werden automatisch vom Parser übergeben
 *
```



```
 * @param  parser   Object   Parserobjekt
 * @param  name     string   Name des schließenden Elements
 */
function endElement($parser, $name) {
  global $html;

  switch (strtolower($name)) {
  case "ref":
    // Den HTML-Link schließen:
    $html .= "</a>";
    break;
  }
}
```

Diese Funktion schließt einen eventuell offenen HTML-Link.

```
/**
 * Diese Funktion behandelt normalen Text
 * Alle Parameter werden automatisch vom Parser übergeben
 *
 * @param   parser   Object   Parserobjekt
 * @param   text     string   Der Text
 */
function cdata($parser, $text) {
  global $html;

  // Der normale Text wird einfach an $html angehängt:
  $html .= $text;
}
```

Die Funktion cdata() wird aufgerufen, wenn normaler Text im XML-Dokument gefunden wird. In diesem Fall wird dieser einfach an die Ausgabe angehängt.

```
// Die XML-Datei wird in die Variable $xmlFile eingelesen
$xmlFile = implode("", file("tutorial.xml"));

// Der Parser wird erstellt
$parser = xml_parser_create();
// Setzen der Handler
xml_set_element_handler($parser,"startElement","endElement");
// Setzen des CDATA-Handlers
xml_set_character_data_handler($parser, "cdata");
// Parsen
xml_parse($parser, $xmlFile);
// Gibt alle verbrauchten Ressourcen wieder frei.
xml_parser_free($parser);
```

```
// Ausgabe der globalen Variable $html.
print $html;
```

Wenn man das XML-Dokument etwas abändert und ein fehlerhaftes Element an einer beliebigen Stelle einfügt, gibt das Script einen Fehler aus und zeigt an, in welcher Zeile das falsche Element gefunden wurde.

22.4 Nachteile

Das erste Beispiel hat den Nachteil, daß im Dokument keine Umlaute oder XML-spezifischen Sonderzeichen verwendet werden können. Man löst dieses Problem meistens durch Entities:

```
<?xml version="1.0"?>
<!DOCTYPE tutorial [
    <!ENTITY auml "&amp;auml;">
    <!ENTITY ouml "&amp;ouml;">
    <!ENTITY uuml "&amp;uuml;">
    <!ENTITY Auml "&amp;Auml;">
    <!ENTITY Ouml "&amp;Ouml;">
    <!ENTITY Uuml "&amp;Uuml;">
    <!ENTITY szlig "&amp;szlig;">
]>
<tutorial titel="expat">
    <abschnitt titel="Beispiele">
        <index keyword="beispiele" />
        Im normalen Text k&ouml;nnen wir
        jetzt Umlaute verwenden:
        &Auml;
        &auml;
        &Uuml;
        &uuml;
        &Ouml;
        &ouml;
        Und auch ein scharfes S:
        &szlig;
    </abschnitt>
</tutorial>
```

Dieses Dokument ist etwas kompliziert. `ä` (ein ä) wird zu `ä` wenn man & wiederum auflöst, erhält man `ä wird &auml`. Also den Ausgangszustand. Das ist aber gewünscht! Die Ausgabe soll nämlich in HTML gewandelt werden. Und dort verwenden wir wieder ein Entity...

Allerdings hat auch diese Methode ihre Nachteile. Zum Beispiel müssen &-Zeichen immer mit & geschrieben werden. Das ist ziemlich lästig, zum Beispiel bei Code-Beispielen:


```
...
<tutorial ...>
    ...
        <code language="php">
if ($a &amp;&amp; $b) {
    print "\$a und \$b sind nicht '0'";
}
        </code>
    ...
</tutorial>
```

In diesem Fall kann man aber auch `<![CDATA[ ... ]]>` verwenden:

```
...
<tutorial ...>
    ...
        <code language="php">
        <![CDATA[
if ($a && $b) {
    print "\$a und \$b sind nicht '0'";
}
        ]]>
        </code>
    ...
</tutorial>
```

Das Auflösen von Entities und CDATA-Abschnitten übernimmt expat.

22.5 Beispiel 2

Erweitern wir unser Parser-Script. Es soll jetzt zusätzlich auch noch:

- PHP-Code ausführen können und
- Codeabschnitte formatiert darstellen.

Außerdem sollen keine globalen Funktionen/Variablen mehr benutzt werden.

Zunächst benötigen wir ein neues XML-Dokument:

```
<?xml version="1.0"?>
<!DOCTYPE tutorial [
    <!ENTITY auml "&amp;auml;">
    <!ENTITY ouml "&amp;ouml;">
    <!ENTITY uuml "&amp;uuml;">
    <!ENTITY Auml "&amp;Auml;">
    <!ENTITY Ouml "&amp;Ouml;">
    <!ENTITY Uuml "&amp;Uuml;">
```

```
    <!ENTITY szlig "&amp;szlig;">
]>
<tutorial titel="Boolesche Werte">
    <abschnitt titel="Beispiele">
        Nachfolgend ein paar Beispiele zu booleschen Abfragen.
        <code language="php">
        <![CDATA[
$a = 3;
$b = 5;

if ($a && $b) {
    print '$a und $b sind nicht "0"<br />';
}

if ($a < $b) {
    print '$a ist kleiner als $b<br />';
}
        ]]>
        </code>
        Beim Ausf&uuml;ren erh&auml;lt man folgende Ausgabe.
        <?php
$a = 3;
$b = 5;

if ($a && $b) {
    print '$a und $b sind nicht "0"<br />';
}

if ($a < $b) {
    print '$a ist kleiner als $b<br />';
}
        ?>
    </abschnitt>
</tutorial>
```

Um globale Funktionen/Variablen zu vermeiden, bleibt uns als einziger Ausweg die Verwendung einer Klasse.

```
class TutorialParser {
  var $html; // Erstellter HTML-Code

  function TutorialParser($file) {
    // Überprüfen, ob die Datei vorhanden ist:
    if (!file_exists($file)) {
      $this->error('Datei '.$file.
                   ' kann nicht gefunden werden!');
```



```
    }
    else {
      $buffer = implode('', file($file));
      $p = xml_parser_create();

      // Durch das Setzen dieser Option werden nicht alle
      // Element- und Attributnamen in Großbuchstaben
      // umgewandelt.
      xml_parser_set_option($p, XML_OPTION_CASE_FOLDING, 0);
      // Wichtig, daß der Parser nicht globale Funktionen
      // aufruft, sondern Methoden dieser Klasse.
      xml_set_object($p, $this);

      xml_set_element_handler($p, 'startElement',
                              'closeElement');
      xml_set_character_data_handler($p, 'cdataHandler');
      // Setzt den Handler für Processing Instructions.
      xml_set_processing_instruction_handler($p, 'piHandler');
      xml_parse($p, $buffer);
      xml_parser_free($p);
    }
  }

  function startElement($parser, $name, $a) {
    // Wie im ersten Beispiel, allerdings wird die Klassen-
    // Variable $html anstelle der globalen benutzt.
    switch ($name) {
    case 'tutorial':
      $this->html .= '<h1>'.$a['titel'].'</h1>';
      break;
    case 'abschnitt';
      $this->html .= '<h2>'.$a['titel'].'</h2>';
      break;
    case 'index':
      $this->html .= '<a name="'.$a['keyword'].'"></a>';
      break;
    case 'ref':
      $this->html .= '<a href="#'.$a['id'].'">';
      break;
    case 'code':
      $this->html .= '<pre>';
      break;
    default:
      $error = 'Undefiniertes Element &lt;'.$name.'&gt;';
      $line = xml_get_current_line_number($parser);
      $this->error($error.' in Zeile '.$line);
```

```
    }
  }

  function closeElement($parser, $name) {
    switch ($name) {
    case 'ref':
      $this->html .= '</a>';
      break;
    case 'code':
      $this->html .= '</pre>';
      break;
    }
  }

  function cdataHandler($parser, $cdata) {
    $this->html .= $cdata;
  }

  function piHandler($parser, $target, $data) {
    switch ($target) {
    case 'php':
      // Es wurde eine Codestelle gefunden, die
      // ausgeführt werden soll. Zuerst starten
      // wir den Ausgabepuffer, damit die gesamte
      // Ausgabe eingefangen werden kann.
      ob_start();
      // Ausführen des PHP-Codes
      eval($data);
      // "Einsammeln" der Ausgabe
      $output = ob_get_contents();
      // Ausgabe verwerfen
      ob_end_clean();
      // Anhängen der Ausgabe an $html:
      $this->html .= '<b>Ausgabe:</b><br />';
      $this->html .= $output;
      break;
    }
  }

  function error($str) {
    // Ausgeben einer Fehlermeldung:
    die('<b>Fehler:</b> '.$str);
  }

  function get() {
    // Gibt den erzeugten HTML-Code zurück.
```



```
    return $this->html;
  }

  function show() {
    // Gibt den erzeugten HTML-Code aus.
    print $this->get();
  }
}
```

Zuletzt brauchen wir noch ein kleines Script, das die Klassenfunktionen benutzt.

```
include_once "tp.php"; // Einbinden der Klasse

$tp = new TutorialParser('tutorial.xml');
$tp->show();
```

23 Templates

Templates sind Layout/Design-Vorlagen, die meistens aus HTML-Code bestehen. Der Hauptvorteil von Templates besteht in der strikten Trennung von PHP-Code und HTML. Somit wird eine Website nicht nur übersichtlicher (für den Webmaster). Auch die Arbeit an der Homepage kann besser zwischen Scripter und Designer aufgeteilt werden.

Der Designer kann in Ruhe ein HTML-Dokument erstellen. An den Stellen, wo später der dynamische Inhalt eingesetzt werden soll, fügt er Variablen/Platzhalter ein. Der Programmierer setzt für diese Variablen Werte ein, und muß sich somit nicht um das Design kümmern. Wenn sich in der HTML-Datei etwas ändert, muß das Script nicht geändert werden – und umgekehrt.

23.1 Beispiel

```
<html>
  <head>
    <title>{titel}</title>
  </head>
  <body>
    <h1>{titel}</h1>
    <p>
      {inhalt}
    </p>
  </body>
</html>
```

So könnte ein einfaches Template zum Beispiel aussehen. Als Gegenstück dazu muß vom Programmierer ein Script geschrieben werden, das die Variablen „titel“ und „inhalt“ mit entsprechenden Werten füllt.

Da wir uns nicht mit regulären Ausdrücken herumschlagen wollen, verwenden wir ein Templatemodul. Ich will in den weiteren Abschnitten zwei solcher Module vorstellen.

23.2 PEAR::IT[X]

Integrated Template (Extension) ist ein Templatemodul, das mit PHP (ab Version 4) mitgeliefert wird. Es ist Bestandteil des Code-Repositorys PEAR[1] und ist somit bei korrekter Installation von PEAR per `include "HTML/ITX.php"` verfügbar.

```
<?php
include_once "HTML/ITX.php";
```

[1] PHP Extension and Application Repository


```
// ITX erwartet den Pfad zu den Templates als Parameter
$tpl = new IntegratedTemplateExtension("./");
$tpl->loadTemplateFile("template.html");

// Setzen der Variablen
$tpl->setVariable("titel", "Test-Template");
$tpl->setVariable("inhalt", "Just a test...");

// Ausgeben
$tpl->show();
?>
```

23.2.1 Block-API

IT[X] bietet auch noch weitere interessante Features an. Mit Hilfe der Block-API ist der Programmierer in der Lage, Teile des Templates mehrmals auszugeben (mit unterschiedlichen Werten). Als Beispiel könnte man zum Beispiel ein HTML-Selectfeld nehmen. Es hat mehrere Optionen zum auswählen. Wenn diese Optionen dynamisch generiert werden, kommt man mit den herkömmlichen Platzhaltern nicht mehr weit.

```
<html>
  <body>
    <select>
      <!-- BEGIN option_loop -->
      <option>{wert}</option>
      <!-- END option_loop -->
    </select>
  </body>
</html>
```

Das PHP-Script soll in diesem Fall für jede einzelne Option den gesamten Block einfügen, und für „wert“ etwas einfügen.

```
<?php
include_once "HTML/ITX.php";

// ITX erwartet als Parameter den Pfad zu den Template
$tpl = new IntegratedTemplateExtension("./");
$tpl->loadTemplateFile("template.html");


// Array aus Optionen
$optionen = array(
    "Option 1",
    "Option 2",
    "Noch eine Option",
```

```
    "Wieder eine",
    "Und die letzte"
);

// Auswählen des Blocks
$tpl->setCurrentBlock("option_loop");
foreach ($optionen as $wert) {
    $tpl->setVariable("wert", $wert);
    // Zuletzt muß der Block nach dem Setzen
    // der Variable(n) geparsed werden
    $tpl->parseCurrentBlock();
}

// Ausgeben
$tpl->show();
?>
```

23.3 Smarty

Smarty wird nicht mit PHP mitgeliefert, sondern muß von *http://smarty.php.net/* heruntergeladen werden. Die Smarty-Templates sind um einiges flexibler, aber auch komplizierter. Smarty bietet viele Features, die hier nicht alle aufgelistet werden können. Trotzdem ein kleines Beispiel:

```
<html>
  <head>
    <title>{$titel}</title>
  </head>
  <body>
    <select>
      {html_options output=$optionen}
    </select>
  </body>
</html>
```

Ich habe hier eine kombinierte Version der beiden oberen Beispiele benutzt. Wie man sieht, hat Smarty eine leicht veränderte Syntax für Variablen. Außerdem gibt es schon vorgefertigte Funktionen, wie zum Beispiel „html_options“, welche eine Select-Liste erstellt.

```
<?php
include_once "Smarty.class.php";

$smarty = new Smarty;

$smarty->assign("titel", "Smarty-Test");
$smarty->assign("optionen", array("Option 1",
```



```
                                  "Option 2",
                                  "Noch eine Option",
                                  "Wieder eine",
                                  "Und die letzte"));

$smarty->display("test.tpl"); // Templates werden bei
                              // Smarty als .tpl gespeichert
?>
```

Der Titel wird ähnlich wie bei IT[X] in das Template eingefügt. Aus dem Array mit dem Optionen erstellt Smarty direkt eine Select-Liste, was etwas Tipp-Arbeit erspart.

Smarty bietet noch viele weitere Funktionen, wie zum Beispiel eine Art Debugger, der außer einer Angabe, wieviel Zeit zum Parsen benötigt wurde, auch ausgibt, welche Variablen mit welchen Werten belegt wurden. Auch If-Else-Abfragen, Foreach-Schleifen und Include-Direktiven sind kein Problem. Allerdings stellt man sich bei der Fülle an Funktionen manchmal die Frage, ob man nicht gleich den PHP-Code in die HTML-Datei einbetten sollte, da die Templates oft sehr unübersichtlich werden.

Teil VIII

Anhang

A Unser Beispiel in SQL

A.1 Zu erstellende Tabellen

```
CREATE TABLE Abteilung (
   AbtNr      INT NOT NULL AUTO_INCREMENT,
   Name       VARCHAR(30),
   PRIMARY KEY(AbtNr)
);

CREATE TABLE Mitarbeiter (
   MNr        INT NOT NULL AUTO_INCREMENT,
   VNr        INT,
   AbtNr      INT NOT NULL,
   Name       VARCHAR(30) NOT NULL,
   GebDat     DATE,
   Telefon    VARCHAR(30),
   PRIMARY KEY(MNr),
   FOREIGN KEY(VNr) REFERENCES Mitarbeiter(MNr),
   FOREIGN KEY(AbtNr) REFERENCES Abteilung(AbtNr)
);

CREATE TABLE PKW (
   PKWNr      INT NOT NULL AUTO_INCREMENT,
   AbtNr      INT NOT NULL,
   Kennzeichen VARCHAR(10) NOT NULL,
   Typ        VARCHAR(30),
   PRIMARY KEY(PKWNr),
   UNIQUE(Kennzeichen),
   FOREIGN KEY(AbtNr) REFERENCES Abteilung(AbtNr)
);

CREATE TABLE Fahrbuch (
   MNr        INT NOT NULL,
   PKWNr      INT NOT NULL,
   Datum      DATETIME NOT NULL,
   PRIMARY KEY (MNr,PKWNr,Datum),
   FOREIGN KEY(MNr) REFERENCES Mitarbeiter(MNr),
   FOREIGN KEY(PKWNr) REFERENCES PKW(PKWNr)
);
```

A.2 Daten einfügen

Um ein paar Daten zu haben, mit denen wir arbeiten können, sollen folgende Werte in unsere DB eingefügt werden.

Mitarbeiter					
MNr	**VNr**	**AbtNr**	**Name**	**GebDat**	**Telefon**
1	NULL	3	Christoph Reeg	13.5.1979	NULL
2	1	1	junetz.de	5.3.1998	069/764758
3	1	1	Uli	NULL	NULL
4	3	1	JCP	NULL	069/764758
5	1	2	Maier	NULL	06196/671797
6	5	2	Meier	NULL	069/97640232

Abteilung	
AbtNr	**Name**
1	EDV
2	Verwaltung
3	Chefetage

Fahrtenbuch		
MNr	**PKWNr**	**Datum**
1	1	13.05.2000
1	2	03.05.1998

PKW			
PKWNr	**AbtNr**	**Kennzeichen**	**Typ**
1	3	MTK-CR 1	RR
2	1	F-JN 1	VW-Golf

Das ganze in SQL:

```
INSERT INTO Abteilung (AbtNr,Name)
  VALUES (1,'EDV');
INSERT INTO Abteilung (AbtNr,Name)
  VALUES (2,'Verwaltung');
INSERT INTO Abteilung (AbtNr,Name)
  VALUES (3,'Chefetage');

INSERT INTO Mitarbeiter (MNr,AbtNr,Name,GebDat)
  VALUES (1,3,'Christoph Reeg','1979-5-13');
INSERT INTO Mitarbeiter (MNr,VNr,AbtNr,Name,GebDat,Telefon)
  VALUES (2,1,1,'junetz.de','1998-3-5','069/764758');
INSERT INTO Mitarbeiter (MNr,VNr,AbtNr,Name)
  VALUES (3,1,1,'Uli');
INSERT INTO Mitarbeiter (MNr,VNr,AbtNr,Name,Telefon)
  VALUES (4,3,1,'JCP','069/764758');
INSERT INTO Mitarbeiter (MNr,VNr,AbtNr,Name,Telefon)
  VALUES (5,1,2,'Maier','06196/671797');
INSERT INTO Mitarbeiter (MNr,VNr,AbtNr,Name,Telefon)
  VALUES (6,5,2,'Meier','069/97640232');
```



```
INSERT INTO PKW (PKWNr,AbtNr,Kennzeichen,Typ)
  VALUES (1,3,'MTK-CR 1','RR');
INSERT INTO PKW (PKWNr,AbtNr,Kennzeichen,Typ)
  VALUES (2,1,'F-JN 1','VW-Golf');

INSERT INTO Fahrbuch (MNr,PKWNr,Datum)
  VALUES (1,1,'2000-5-13');
INSERT INTO Fahrbuch (MNr,PKWNr,Datum)
  VALUES (2,2,'1998-5-3');
```

B Lösungen

B.1 Lösung zu Baumdarstellungen

B.1.1 Vater-Zeiger

Die `CREATE TABLE` mit anschließender `INSERT` Anweisung sollte nicht das Problem sein:

```
CREATE TABLE vaterzeiger (
  ID     int not null primary key,
  Name   varchar(100),
  VID    int
);

INSERT INTO vaterzeiger VALUES (1,'Root',0);
INSERT INTO vaterzeiger VALUES (2,'A',1);
INSERT INTO vaterzeiger VALUES (3,'B',1);
INSERT INTO vaterzeiger VALUES (4,'C',1);
INSERT INTO vaterzeiger VALUES (5,'A1',2);
INSERT INTO vaterzeiger VALUES (6,'B1',3);
INSERT INTO vaterzeiger VALUES (7,'B2',3);
INSERT INTO vaterzeiger VALUES (8,'C1',4);
INSERT INTO vaterzeiger VALUES (9,'A1I',5);
INSERT INTO vaterzeiger VALUES (10,'C1I',8);
INSERT INTO vaterzeiger VALUES (11,'C1II',8);
```

Nachdem nun die Tabelle existiert, die einzelnen Abfragen:

•

```
mysql> SELECT * FROM vaterzeiger
    -> WHERE VID = 0;
+----+------+------+
| ID | Name | VID  |
+----+------+------+
|  1 | Root |    0 |
+----+------+------+
1 row in set (0.00 sec)
```

- Ich bin hier einfach mal davon ausgegangen, daß man die ID der Wurzel kennt.

```
mysql> SELECT count(*) FROM vaterzeiger
    -> WHERE VID = 1;
+----------+
| count(*) |
+----------+
```



```
|        3 |
+----------+
1 row in set (0.00 sec)
```

Wenn man die ID der Wurzel nicht kennt, wird die Abfrage etwas länger:

```
mysql> SELECT count(*)
    -> FROM vaterzeiger V, vaterzeiger S
    -> WHERE V.ID = S.VID
    ->   AND V.VID = 0;
+----------+
| count(*) |
+----------+
|        3 |
+----------+
1 row in set (0.01 sec)
```

Hier wird ein Self-Join gemacht, da MySQL keine Unterabfragen unterstützt.

- Dies ist im Prinzip dieselbe Abfrage wie gerade eben, nur das `count()` wird weggelassen.

- Eine Möglichkeit wäre

```
mysql> SELECT V.Name, ' ist Vater von ', S.Name
    -> FROM vaterzeiger V, vaterzeiger S
    -> WHERE V.ID = S.VID
    ->   AND V.VID = 0;
+------+-----------------+------+
| Name |  ist Vater von  | Name |
+------+-----------------+------+
| Root |  ist Vater von  | A    |
| Root |  ist Vater von  | B    |
| Root |  ist Vater von  | C    |
+------+-----------------+------+
3 rows in set (0.00 sec)
```

Viel schöner finde ich aber folgende Ausgabe:

```
mysql> SELECT S.VID != 0 AS Tiefe, S.Name
    ->   FROM vaterzeiger V, vaterzeiger S
    ->  WHERE (V.ID = S.VID OR S.VID = 0)
    ->    AND V.VID = 0;
+-------+------+
| Tiefe | Name |
+-------+------+
|     0 | Root |
```

```
|     1 | A    |
|     1 | B    |
|     1 | C    |
+-------+------+
4 rows in set (0.00 sec)
```

- Über entsprechend viele Self-Joins (hier 4) ist dies möglich, allerdings wird dadurch die maximale Tiefe des Baums bestimmt und die Abfrage ist nicht mehr wirklich schön und performant.

B.1.2 Nested Set

- Die einfachere Variante ist die folgende mit zwei Abfragen:

```
mysql> SELECT l,r
    ->   FROM NestedSet
    ->  WHERE Name = "C";
+------+------+
| l    | r    |
+------+------+
|   14 |   21 |
+------+------+
1 row in set (0.00 sec)

mysql> SELECT s.Name, count(*) AS Level
    -> FROM NestedSet v, NestedSet s
    -> WHERE s.l BETWEEN v.l AND v.r
    ->   AND s.l BETWEEN 14 AND 21
    -> GROUP BY s.l;
+------+-------+
| Name | Level |
+------+-------+
| C    |     2 |
| C1   |     3 |
| C1I  |     4 |
| C1II |     4 |
+------+-------+
4 rows in set (0.00 sec)
```

Mit Hilfe der ersten Abfrage holen wir uns die Werte für `l` und `r`, die wir dann bei der eigentlichen Abfrage verwenden.

Wenn du nur eine Anweisung haben willst, sieht eine mögliche Abfrage so aus:

```
mysql> SELECT s.Name, count(*) AS Level
    -> FROM NestedSet v, NestedSet s, NestedSet s_id
    -> WHERE s.l BETWEEN v.l AND v.r
```



```
    ->   AND s_id.Name = "C"
    ->   AND s.l BETWEEN s_id.l AND s_id.r
    -> GROUP BY s.l;
+------+-------+
| Name | Level |
+------+-------+
| C    |     2 |
| C1   |     3 |
| C1I  |     4 |
| C1II |     4 |
+------+-------+
4 rows in set (0.00 sec)
```

- Die Abfrage lautet:

```
mysql> SELECT s.Name,
    ->        (s.r-s.l-1)/2 AS Nachfolger,
    ->        count(*)+(s.l>1) AS Tiefe,
    ->        ((min(v.r)-s.r-(s.l>1))/2) > 0 AS Bruder
    -> FROM NestedSet v, NestedSet s
    -> WHERE s.l BETWEEN v.l AND v.r
    ->   AND (v.Name != s.Name OR s.l = 1)
    -> GROUP BY s.Name
    -> ORDER BY s.l;
+------+------------+-------+--------+
| Name | Nachfolger | Tiefe | Bruder |
+------+------------+-------+--------+
| Root |      10.00 |     1 |      0 |
| A    |       2.00 |     2 |      1 |
| A1   |       1.00 |     3 |      0 |
| A1I  |       0.00 |     4 |      0 |
| B    |       2.00 |     2 |      1 |
| B1   |       0.00 |     3 |      1 |
| B2   |       0.00 |     3 |      0 |
| C    |       3.00 |     2 |      0 |
| C1   |       2.00 |     3 |      0 |
| C1I  |       0.00 |     4 |      1 |
| C1II |       0.00 |     4 |      0 |
+------+------------+-------+--------+
11 rows in set (0.00 sec)
```

Eigentlich gar nicht so schwierig, oder? ;-)
Ich gebe zu, ich habe auch einen Moment für diese Abfrage gebraucht und mußte etwas basteln, bis es endlich funktioniert hat.

B.2 Lösung für „Ergebnis-Tabelle ausgeben I“

1. Teilschritt

Ich habe bei der Lösung `mysql_fetch_row()` verwendet, weil es einfacher ist, mit einer Schleife einen numerischen Index durchzugehen. Das wird im 3. Teilschritt deutlich.

```
function print_result_table($result){
  // Tabellenanfang
  echo "<table>\n";

  // Tabellenzeilen-Anfang
  echo "  <tr>\n";

  // Zeile aus DB-Anfrage holen
  $row = mysql_fetch_row($result);
  // erstes Feld der Zeile ausgeben
  echo "    <td>$row[0]</td>\n";

  // Tabellenzeilen-Ende
  echo "  </tr>\n";

  // Tabellenende
  echo "</table>\n";
}
```

2. Teilschritt

```
function print_result_table($result){
  // Tabellenanfang
  echo "<table>\n";

  // Alle Ergebniszeilen durchgehen
  while ($row = mysql_fetch_row($result)){
    // Tabellenzeilen-Anfang
    echo "  <tr>\n";

    // erstes Feld der Zeile ausgeben
    echo "    <td>$row[0]</td>\n";

    // Tabellenzeilen-Ende
    echo "  </tr>\n";
  }

  // Tabellenende
  echo "</table>\n";
```



```
}
```

3. Teilschritt

Mit `int mysql_num_fields(int result)` kann man abfragen, wie viele Spalten bei der Abfrage zurückgegeben wurden. Mit einer `for`-Schleife werden einfach alle Felder ausgegeben.

```
function print_result_table($result){
  // Tabellenanfang
  echo "<table>\n";

  // Alle Ergebniszeilen durchgehen
  while ($row = mysql_fetch_row($result)){
    // Tabellenzeilen-Anfang
    echo "  <tr>\n";

    // Alle Spalten durchgehen
    for ($i = 0; $i < mysql_num_fields($result); $i++){
      echo "    <td>$row[$i]</td>\n";
    }

    // Tabellenzeilen-Ende
    echo "  </tr>\n";
  }

  // Tabellenende
  echo "</table>\n";
}
```

4. Teilschritt

Als letztes müssen wir noch abfragen, wie die Spalten heißen. Das macht die Funktion `string mysql_field_name(int result, int field_index)`.

```
function print_result_table($result){
  // Tabellenanfang
  echo "<table>\n";
  // 1. Tabellenzeile Anfang
  echo "  <tr>\n";
  for ($i = 0; $i < mysql_num_fields($result); $i++){
    echo "    <th>".mysql_field_name($result,$i)."</th>\n";
  }
  // 1. Tabellenzeile Ende
  echo "  </tr>\n";
```

```
  // Alle Ergebniszeilen durchgehen
  while ($row = mysql_fetch_row($result)){
    // Tabellenzeilen-Anfang
    echo "  <tr>\n";

    // Alle Spalten durchgehen
    for ($i = 0; $i < mysql_num_fields($result); $i++){
      echo "    <td>$row[$i]</td>\n";
    }

    // Tabellenzeilen-Ende
    echo "  </tr>\n";
  }

  // Tabellenende
  echo "</table>\n";
}
```

B.3 Lösung für „Ergebnis-Tabelle ausgeben II“

Kleiner Tipp vorweg: Um zu testen, ob die eine oder andere Abfrage sauber funktioniert, kann man einfach mal einen Fehler provozieren, also z. B. einen falschen Namen bei `$db_user` eintragen oder die Abfrage zu einem „`SELECT * FROM Mitarbeiter WHERE MNr = 0`“ erweitern, damit keine Mitarbeiter gefunden werden.

```
<?php
$db_host   =      "localhost";
$db_user   =      "cr";
$db_pass   =      "123";

$datab  =         "cr";


function print_result_table($result){
  // s.o.
}

// Hauptprogramm

/* Verbindung zur Datenbank aufbauen */
$db = @mysql_connect($db_host,$db_user,$db_pass)
      OR die(mysql_error());
@mysql_select_db($datab,$db)
      OR die(mysql_error());
```



```
/* HTML-Startcode ausgeben */
echo "<html>\n<body>\n";

/* SQL-Abfrage */
$result = @mysql_query("SELECT * FROM Mitarbeiter");
/* Wenn die Fehlernummer != 0 ist, dann gab es einen Fehler
   => Fehlermeldung ausgeben */
if (mysql_errno() != 0){
  echo mysql_error();
}
// es gab keine Fehler => Ergebnis ausgeben
else {
  // Wie viele Datensätze wurden gefunden?
  // Bei 0 Meldung ausgeben
  if (mysql_num_rows($result) == 0){
    echo "Keine Datens&auml;tze gefunden!";
  }
  // sonst die Funktion aufrufen
  else{
    print_result_table($result);
  }
}

/* HTML-Endcode ausgeben */
echo "</body>\n</html>\n";
?>
```

B.4 Lösung für „Abfrage mit sprintf()“

Die vier Fehler sind im Einzelnen:

1. „`SELECT %0\$s`“: Es gibt kein nulltes Argument. Nie.
2. „`%2\$02d`“: Im ISO-Datumsformat steht das Jahr vorne – es müßte also das dritte Argument ausgewertet werden, nicht das zweite. Außerdem ist die Jahresangabe in Standard-Notation vierstellig (also zwei Fehler).
3. „`%3\02d`“: Ein Backslash allein macht noch keine Argument-Referenz ...
4. „`-%'`“: Soll ein einzelnes Prozentzeichen wirklich ausgegeben werden, muß man es escapen. In diesem Fall sind also zwei Prozentzeichen nötig: Eins zum Escapen und das Prozentzeichen an sich für die Abfrage mit LIKE als Allquantor.

B.5 Lösung für Einfügen mit automatischer ID

Um das gewünschte zu erreichen, sind zwei Tricks nötig: Zum einen muß das Erzeugen der MNr MySQL überlassen werden und zum anderen muß diese ID abgefragt werden, um die

VNr für die zweite Einfügeoperation zu haben. Ersteres erreicht man durch Weglassen der MNr bei der Liste der Felder (oder bei Angabe der MNr Setzen des Wertes auf NULL), letzteres durch `mysql_insert_id()`.

Folgender Code implementiert dies:

```
$db_host   =       "localhost";
$db_user   =       "cr";
$db_pass   =       "123";

$datab  =          "cr";

/* Verbindung zur Datenbank aufbauen */
$db = @mysql_connect($db_host,$db_user,$db_pass)
      OR die(mysql_error());
@mysql_select_db($datab,$db)
      OR die(mysql_error());

// Jens einfügen
mysql_query("INSERT INTO Mitarbeiter (VNr,AbtNr,Name,GebDat)
             VALUES (1, 2, 'Jens', '1981-05-26')");
echo mysql_error();

$vnr = mysql_insert_id();

// Kile einfügen und Jens unterstellen
mysql_query("INSERT INTO Mitarbeiter (VNr,AbtNr,Name)
             VALUES ($vnr, 1, 'Kile')");
echo mysql_error();
```

B.6 Lösungen zu Regulären Ausdrücken

B.6.1 Lösung zu „einfache Suchmuster“

B.6.1.1

Das Ergebnis ist zweimal nein. Im zweiten Text kommt das Wort „hallo“ gar nicht vor und in dem ersten ist es groß geschrieben. Es wird auf Groß-/Kleinschreibung geachtet.

B.6.1.2

Diesmal passen beide Ausdrücke; in beiden Fällen ist das letzte Zeichen kein großer Buchstabe oder eine Zahl.

B.6.1.3

Wieder zwei mal ja.

B.6.1.4

Und nochmal passen beide. In beiden Strings kommt ein 5 Zeichen langes Wort vor.

B.6.1.5

Auf den ersten Text paßt er nicht, dafür auf den zweiten.

B.6.2 Lösung zu „Quantifizierer“

B.6.2.1

Der Ausdruck paßt bei keinem der beiden Texte. Der erste besteht zwar aus zwei Wörtern, aber das Ausrufezeichen `!` ist nicht in `\w` enthalten.

B.6.2.2

Auf den ersten Text paßt jetzt der Ausdruck, auf den zweiten immer noch nicht.

B.6.2.3

Dieselbe Antwort wie bei der vorigen Frage.

B.6.2.4

Zweimal ja. In beiden Texten gibt es ein Wort aus vier Buchstaben, gefolgt von einem Leerzeichen und einem zweiten Wort. Im ersten Text lautet das 4-stellige Wort „allo“, wo noch ein „H“ davor hängt. Es ist keine Aussage gemacht, was vor dem Wort stehen darf, oder nicht.

B.6.2.5

Ja, nein.

B.6.2.6

Der folgende Ausdruck (eigentlich gehört alles in eine Zeile, nur aus Platzgründen ist er umbrochen) paßt auf die angegebene Log-Zeile. Er ist noch nicht optimal (siehe auch die Erklärung), aber da ich bei den regulären Ausdrücken auch noch am Lernen bin, ist mir bis jetzt noch nichts besseres eingefallen. Wer einen besseren Ausdruck weiß, kann ihn mir gerne schicken.

```
/^[\w.-]+ [-\w]+ [-\w]+ \[\d{2}\/\w{3}\/\d{4}:\d{2}:\d{2}:\d{2}
[+-]\d{4}\] "\w+ [\/.\w]+ HTTP\/\d\.\d" [1-5]\d{2} [-\d]*$/
```

Was macht der Ausdruck? Als erstes soll er die gesamte Zeile überprüfen, deshalb das Dach `^` am Anfang und das Dollarzeichen `$` am Ende. Am Anfang der Zeile soll der Name bzw. die IP des Anfragenden kommen. Das habe ich an dieser Stelle mit einem `[\w.-]+` erschlagen. Da der Punkt und der Bindestrich vorkommen darf, aber nicht bei den “Wort“-Zeichen enthalten ist, müssen sie hier extra aufgeführt werden. Dieser Teil ist nicht optimal, weil „1.1“, „keine.gueltiger.name“ oder „rechnername“ keine gültigen Namen oder IP-Adressen sind, trotzdem aber auf den Ausdruck passen.

Die nächsten beiden Felder sind einfacher, weil hier entweder normale Wörter oder ein Minus für „nicht vorhanden“ stehen. Die eckigen Klammern müssen mit einem Backslash escaped werden, weil ich hier keine Menge von Zeichen definieren will, sondern die Zeichen „eckige Klammer auf“ und „eckige Klammer zu“ haben will. Ähnliches gilt für den Slash: Wenn ich ihn nicht mit einem Backslash escaped hätte, wäre der Ausdruck an der Stelle zu Ende.

Der Ausdruck in den eckigen Klammern müßte eigentlich klar sein. Da die Anzahl der einzelnen Zeichen feststehen, habe ich sie hier explizit angegeben.

In den Anführungszeichen steht das nächste nicht-Optimum. Das erste Wort in den Anführungszeichen ist entweder „GET“ oder „POST“; trotzdem muß ich beliebige Wörter erlauben, weil ich noch nicht geschrieben habe, wie man Oder realisiert. Der Punkt bei `HTTP` muß auch mit einem Backslash escaped werden, damit nur der Punkt paßt und nicht ein beliebiges Zeichen.

Da der Statuscode immer eine dreistellige Zahl zwischen 100 und 505 ist, muß auch hier explizit die Anzahl der Ziffern angegeben. Ist auch nicht ganz optimal, weil nicht weit genug geprüft wird. Die Größenangabe (das letzte Feld) kann auch nichts (d. h. ein Minus `-`) enthalten, wenn es keinen Sinn machen würde. Daher dieser Ausdruck.

B.6.3 Gruppierungen

B.6.3.1

```
/(\d*),\1DM/
```

B.7 Lösungen zur Fehlersuche

B.7.1 Lösung für „ein komisches IF“

Teil 1

Der Sinn des Programms sollte eigentlich auf den ersten Blick klar sein. Zu Beginn wird die Variable `$i` auf einen Wert gesetzt (hier 0) und dann überprüft, ob sie den Wert 1 hat. Wenn ja, soll der Text „Die Variable $i hat den Wert 1“ ausgegeben werden, sonst „Die Variable hat nicht den Wert 1“.

Das Programm gibt folgendes ohne Fehlermeldung aus: „Die Variable 1 hat den Wert 1“. Der erste ist ein beliebter Fehler: statt des Vergleichsoperators `==` wurde der Zuweisungsoperator `=` benutzt. Damit wurde nicht die Variable mit dem Wert verglichen, sondern der Variablen wurde der Wert „1“ zugewiesen. Anschließend wurde dieser Wert genommen, so daß alle Werte, die nicht „0“ waren, als „true“ interpretiert wurden. Somit wurde in der IF-Anweisung der 1. Anweisungsblock ausgeführt.

Der zweite Fehler steckt in der Ausgabe. Ausgegeben werden sollte „Die Variable $i …“ und nicht „Die Variable 1 …“. Es gibt zwei Lösungen für das Problem. Entweder benutzt man die einfachen Anführungszeichen und verhindert damit das Ersetzen von Variablennamen durch deren Werte oder man escaped das `$`, dazu muß man statt `$i` ein `\$i` schreiben.

Solche Fehler könnte man vermeiden, würde man statt `if ($i == 1)` ein `if (1 == $i)` schreiben. Dann erhielte man nämlich eine Fehlermeldung, wenn man nur `1 = $i` schriebe, weil man dem Wert 1 nicht den Wert der Variablen zuweisen kann.

Teil 2

Im Prinzip gilt das oben Gesagte. Hier wird allerdings der else-Block ausgeführt. Bei der if-Anweisung wird der Wert „0“ genommen und der gilt als „false“.

B.7.2 Lösung für „Fehler in einer leeren Zeile?“

Die Ausgabe des Programms sollte eigentlich folgende sein:

```
1
2
3
4
5
Das waren die Zahlen 1-5
```

Das Programm gibt aber unter PHP3 folgendes aus:

```
1
Das waren die Zahlen 1-5
Parse error: parse error in /home/httpd/html/test.php3 on line 6
```

Und bei PHP4 folgendes:

```
Parse error: parse error in /home/httpd/html/test.php4 on line 6
```

Die unterschiedlichen Ausgaben kommen vom unterschiedlichen Design von PHP3 und PHP4. Die Fehlermeldung ist aber in beiden Fällen dieselbe: In Zeile 6 gibt es einen parse error. Zeile 6 ist allerdings leer. Der Fehler ist, daß in Zeile 2 ein Anweisungsblock mit einer `{` angefangen, aber nicht mehr beendet wurde. PHP kann das erst am Ende der Datei merken, daher wird in der Fehlermeldung Zeile 6 angegeben.

Vermeiden kann man solche Fehler, indem man richtig einrückt. Bei folgender Schreibweise fällt leichter auf, daß die (geschweifte Klammer zu) fehlt. Auf jeden Fall läßt sich so der Fehler leichter finden. Vorteilhaft in solchen Situationen ist es auch, einen Editor zu benutzen, der zueinander gehörige Klammern farblich hervorhebt.

```
<?php
for ($i=1; $i<=5; $i++){
  echo "$i<br>\n";
echo "Das waren die Zahlen 1-5";
?>
```

Manche schreiben es noch deutlicher.

```
<?php
for ($i=1; $i<=5; $i++)
{
  echo "$i<br>\n";
echo "Das waren die Zahlen 1-5";
?>
```

B.7.3 Lösung zu „Wieso fehlt ein ‘;‘ wo eins ist?“

Das gewollte Ergebnis ist dasselbe wie in der vorigen Übung. Die Ausgabe unterscheidet sich wieder je nach PHP-Version.

Ausgabe von PHP3 (die letzten beiden Zeilen sind eigentlich eine):

```
1
; } echo
Parse error: parse error, expecting `','' or `';''
in /home/httpd/html/test.php3 on line 5
```

Ausgabe von PHP4:

```
Parse error: parse error, expecting `','' or `';''
in /home/httpd/html/test.php4 on line 5
```

Wie im obigen Fall ist auch hier die Fehlermeldung bei beiden Versionen dieselbe; PHP erwartet in Zeile 5 ein „;“. Bei genauerem Betrachten sieht diese aber richtig aus. Der Fehler steckt in Zeile 3. Dort werden die Anführungszeichen nicht wieder geschlossen. Daher auch die Ausgabe bei PHP3. Nach dem Anführungszeichen in Zeile 3 wird alles bis zum nächsten Anführungszeichen ausgegeben, in unserem Fall in Zeile 5. Damit wird aber das, was eigentlich in den Anführungszeichen stehen sollte, als Reihe von PHP-Befehlen interpretiert. Und mit einem „Das“ nach einer Ausgabe kann PHP eben nicht viel anfangen und will, daß wir erstmal mit einem „;“ die Anweisung beenden.

An diesem Beispiel sieht man, daß der Fehler nicht immer in der gemeldeten Zeile liegen muß. In diesem Fall hilft nur konzentriertes Programmieren und ein Editor mit Syntax-Highlighting[1].

B.8 Lösung zu Spruch des Abrufs

Es gibt zwei Gründe für die zusätzliche Funktion:

- Es ist einfach deutlich schneller, nur eine Abfrage mit mehreren Datensätzen als Ergebnis zu machen, als mehrere Abfragen mit nur jeweils einem Datensatz.
- Durch diese Lösung vermeide ich, daß ein Spruch mehrmals vorkommt. Bei mehreren Abfragen könnte es ja passieren, daß ein Spruch bei zwei Abfragen zufällig ausgewählt würde.

Wenn man die Frage umdrehen würde, wäre sie sinnvoll: Wozu die Funktion `get_spruch` implementieren, wenn es ein `get_sprueche` gibt?

[1] farbliche Hervorhebung in Abhängigkeit der Bedeutung. Z. B. Text in Anführungszeichen wird grün gefärbt, reservierte Wörter in dunkelrot.

B.9 Lösung zum Image-Beispiel

Der folgende Code erzeugt transparente Bilder im PNG-Format[2]. Wohlgemerkt, es werden Bilder erstellt, kein HTML!

Als erstes die Image-Klasse:

```
/**
 * Image Klasse
 * Legt ein Bild (default: 100x100) an
 * und gibt es bei Bedarf aus
 */
class Image {
 /** Image stream */
  var $im;

 /** Breite */
  var $width;

 /** Höhe */
  var $height;

 /** Malfarbe */
  var $dcolor;

  /**
   * Konstruktor
   * Bild anlegen und Standardfarben setzen
   *
   * @param int Breite
   * @param int Höhe
   */
  function Image($width=100, $height=100) {
    if ($width<1 || $height<1)
      $width = $height = 100;

    $this->width = $width;
    $this->height = $height;

    $this->im = @ImageCreate($width, $height)
      or die ("Konnte keinen Image stream anlegen");

    // transparenter Hintergrund, schwarzer "Stift"
    $whitecol = ImageColorAllocate($this->im, 0, 0, 0);
    ImageColorTransparent($this->im, $whitecol);
```

[2] Die GD-Lib, die von den Image-Funktionen benutzt wird, unterstützt das GIF-Format aus lizenzrechtlichen Gründen nicht mehr

```
    $this->dcolor = ImageColorAllocate($this->im, 0, 0, 0);
  }

  /**
   * Bild anzeigen (PNG)
   */
  function show() {
    Header("Content-type: image/png");
    ImagePNG($this->im);
  }

  /**
   * Breite des Bildes zurückliefern
   *
   * @return int Breite
   */
  function getWidth() {
    return $this->width;
  }

  /**
   * Höhe des Bildes zurückliefern
   *
   * @return int Höhe
   */
  function getHeight() {
    return $this->height;
  }
}
```

Nun die Punkt-Klasse:

```
/**
 * Point Klasse
 * Kann einen Punkt malen und ausgeben
 */
class Point extends Image {
 /** X-Koordinate */
  var $x = 0;

 /** Y-Koordinate */
  var $y = 0;

  /**
   * Koordinaten und sonstiges setzen
   *
   * @param int X-Koordinate
```



```
   * @param int Y-Koordinate
   */
  function set($x=0, $y=0) {
    $this->x = $x;
    $this->y = $y;
    ImageSetPixel($this->im,$this->x,$this->y,$this->dcolor);
  }
}
```

Und schließlich die Ausgabe:

```
$mypoint = new Point();
$mypoint->set(50, 50);
$mypoint->show();
```

Als nächstes der Kreis:

```
/**
 * Circle Klasse
 * Kann einen Kreis malen und ausgeben,
 * erfordert aber die GD-Lib 2.0
 */
class Circle extends Point {
 /** Radius */
  var $r = 0;

  /**
   * Koordinaten und Radius setzen
   *
   * @param int X-Koordinate
   * @param int Y-Koordinate
   * @param int Radius
   */
  function set($x=0, $y=0, $r=10) {
    parent::set($x, $y);
    $this->r = $r;
    ImageEllipse($this->im, $this->x, $this->y,
                 $this->r, $this->r, $this->dcolor);
  }
}
```

Kreis-Ausgabe (funktioniert nur mit GD-Lib 2.0!):

```
$mycircle = new Circle();
$mycircle->set(50, 50, 30);
$mycircle->show();
```

Da ich die GD-Lib 2.0 nicht habe, fehlt hier das entsprechende Bild …
Ein Rechteck ist auch nicht schwer zu erstellen:

```
/**
 * Rectangle Klasse
 * Kann ein Rechteck malen und ausgeben
 */
class Rectangle extends Point {
 /** zweite X-Koordinate */
  var $x2;

 /** zweite Y-Koordinate */
  var $y2;

  /**
   * Koordinaten und Ausmaße setzen;
   * ggf. zentrieren
   *
   * @param int X-Koordinate
   * @param int Y-Koordinate
   * @param int Breite
   * @param int Höhe
   * @param boolean Um X/Y-Koordinaten zentrieren
   */
  function set($x=0,$y=0,$width=10,$height=10,$center=false){
    if ($center) {
      parent::set($x - $width/2, $y - $height/2);
      $this->x2 = $x + $width/2;
      $this->y2 = $y + $height/2;
    } else {
      parent::set($x, $y);
      $this->x2 = $x + $width;
      $this->y2 = $y + $height;
    }

    // (x,y) ist links oben, (x2,y2) rechts unten
    ImageRectangle($this->im, $this->x, $this->y,
                   $this->x2, $this->y2, $this->dcolor);
  }
}
```

Beispiel-Rechteck:

```
$myrect = new Rectangle();
$myrect->set(50, 50, 30, 40, true);
$myrect->show();
```

Zum Schluß noch das Quadrat:

```
/**
```



```
 * Square Klasse
 * Kann ein Quadrat malen und ausgeben
 */
class Square extends Rectangle {
  /**
   * Koordinaten und Ausmaße setzen;
   * ggf. zentrieren
   *
   * @param int X-Koordinate
   * @param int Y-Koordinate
   * @param int Seitenlänge
   * @param boolean Um X/Y-Koordinaten zentrieren
   */
  function set($x=0, $y=0, $length=10, $center=false) {
    parent::set($x, $y, $length, $length, $center);
  }
}
```

Und die obligatorische Ausgabe:

```
$mysquare = new Square();
$mysquare->set(50, 50, 30);
$mysquare->show();
```

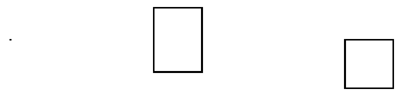

Abbildung B.1: erzeugte Bilder: Punkt, Rechteck und Quadrat

B.10 Lösung zur Referenzsemantik

Eine korrekte Implementierung sähe z. B. so aus:

```
class MyArray {
  var $array;
  function MyArray() {}
  function add(&$val) {
    $this->array[] =& $val;
  }
}
```

```
class Test {
  var $var;
  function Test($var, &$array) {
    $this->var = $var;
    $array->add($this);
  }
  function setVar($var) {
    $this->var = $var;
  }
  function getVar() {
    return $this->var;
  }
}
```

```
$array = new MyArray();
$test  =& new Test(42, $array);
$test2 =& $test;
$test->setVar(0);
echo $test->getVar()." ".$test2->getVar()."\n";
var_dump($array);
```


C Open Publication License

Draft v1.0, 8 June 1999

C.1 Englische Version

REQUIREMENTS ON BOTH UNMODIFIED AND MODIFIED VERSIONS

The Open Publication works may be reproduced and distributed in whole or in part, in any medium physical or electronic, provided that the terms of this license are adhered to, and that this license or an incorporation of it by reference (with any options elected by the author(s) and/or publisher) is displayed in the reproduction.

Proper form for an incorporation by reference is as follows:

Copyright (c) 2000 by Christoph Reeg. This material may be distributed only subject to the terms and conditions set forth in the Open Publication License, v1.0 or later (the latest version is presently available at http://www.opencontent.org/openpub/), not using any of the Open Publication License Options from section LICENSE OPTIONS.

Commercial redistribution of Open Publication-licensed material is permitted.

Any publication in standard (paper) book form shall require the citation of the original publisher and author. The publisher and author's names shall appear on all outer surfaces of the book. On all outer surfaces of the book the original publisher's name shall be as large as the title of the work and cited as possessive with respect to the title.

COPYRIGHT

The copyright to each Open Publication is owned by its author(s) or designee.

SCOPE OF LICENSE

The following license terms apply to all Open Publication works, unless otherwise explicitly stated in the document.

Mere aggregation of Open Publication works or a portion of an Open Publication work with other works or programs on the same media shall not cause this license to apply to those other works. The aggregate work shall contain a notice specifying the inclusion of the Open Publication material and appropriate copyright notice.

SEVERABILITY. If any part of this license is found to be unenforceable in any jurisdiction, the remaining portions of the license remain in force.

NO WARRANTY. Open Publication works are licensed and provided „as is“ without warranty of any kind, express or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose or a warranty of non-infringement.

REQUIREMENTS ON MODIFIED WORKS

All modified versions of documents covered by this license, including translations, anthologies, compilations and partial documents, must meet the following requirements:

- The modified version must be labeled as such.
- The person making the modifications must be identified and the modifications dated.
- Acknowledgement of the original author and publisher if applicable must be retained according to normal academic citation practices.
- The location of the original unmodified document must be identified.
- The original author's (or authors') name(s) may not be used to assert or imply endorsement of the resulting document without the original author's (or authors') permission.

GOOD-PRACTICE RECOMMENDATIONS

In addition to the requirements of this license, it is requested from and strongly recommended of redistributors that:

- If you are distributing Open Publication works on hardcopy or CD-ROM, you provide email notification to the authors of your intent to redistribute at least thirty days before your manuscript or media freeze, to give the authors time to provide updated documents. This notification should describe modifications, if any, made to the document.
- All substantive modifications (including deletions) be either clearly marked up in the document or else described in an attachment to the document.
- Finally, while it is not mandatory under this license, it is considered good form to offer a free copy of any hardcopy and CD-ROM expression of an Open Publication-licensed work to its author(s).

LICENSE OPTIONS

The author(s) and/or publisher of an Open Publication-licensed document may elect certain options by appending language to the reference to or copy of the license. These options are considered part of the license instance and must be included with the license (or its incorporation by reference) in derived works.

- To prohibit distribution of substantively modified versions without the explicit permission of the author(s). SSubstantive modificationïs defined as a change to the semantic content of the document, and excludes mere changes in format or typographical corrections. To accomplish this, add the phrase 'Distribution of substantively modified versions of this document is prohibited without the explicit permission of the copyright holder.' to the license reference or copy.

- To prohibit any publication of this work or derivative works in whole or in part in standard (paper) book form for commercial purposes is prohibited unless prior permission is obtained from the copyright holder. To accomplish this, add the phrase 'Distribution of the work or derivative of the work in any standard (paper) book form is prohibited unless prior permission is obtained from the copyright holder.' to the license reference or copy.

C.2 Deutsche Version

Inoffizielle deutsche Übersetzung des englischen Originals (von Stefan Meretz).

ERFORDERNISSE FÜR UNMODIFIZIERTE UND MODIFIZIERTE VERSIONEN

Open-Publication-Arbeiten dürfen als Ganzes oder in Teilen reproduziert und verteilt werden, in beliebigen Medien, physisch oder elektronisch, vorausgesetzt, die Bedingungen dieser Lizenz gehören dazu, und diese Lizenz oder ein Verweis auf diese Lizenz (mit jeder Option, die von dem Autor / den Autoren und/oder dem Herausgeber gewählt wurde) wird in der Reproduktion angezeigt.

Eine geeignete Form einer Aufnahme durch Verweis lautet wir folgt:

Copyright (c) 2000 by Christoph Reeg. Dieses Material darf nur gemäß der Regeln und Bedingungen wie sie von der Open Publication Licence, Version v1.0, festgelegt werden, verteilt werden (die letzte Version ist gegenwärtig verfügbar unter http://www.opencontent.org/openpub/). Diese Veröffentlichung macht von keiner der im Abschnitt LIZENZ-OPTIONEN genannten Optionen Gebrauch.

Die kommerzielle Weiterverbreitung von Open Publication lizensiertem Material ist untersagt.

Jegliche Publikation im Standard- (Papier-) Buch-Format erfordert die Zitierung der Original-Herausgeber und Autoren. Die Namen von Herausgeber und Autor/en sollen auf allen äußeren Deckflächen des Buchs erscheinen. Auf allen äußeren Deckflächen des Buchs soll der Name des Original-Herausgebers genauso groß sein wie der Titel der Arbeit und so einnehmend genannt werden im Hinblick auf den Titel.

COPYRIGHT

Das Copyright jeder Open Publication gehört dem Autor / den Autoren oder Zeichnungsberechtigten.

GÜLTIGKEITSBEREICH DER LIZENZ

Die nachfolgenden Lizenzregeln werden auf alle Open-Publication-Arbeiten angewendet, sofern nicht explizit anders lautend im Dokument erwähnt.

Die bloße Zusammenfassung von Open-Publication-Arbeiten oder eines Teils einer Open-Publication-Arbeit mit anderen Arbeiten oder Programmen auf dem selben Medium bewirkt nicht, daß die Lizenz auch auf diese anderen Arbeiten angewendet wird. Die zusammengefaßte Arbeit soll einen Hinweis enthalten, die die Aufnahme von Open-Publication-Material und eine geeignete Copyright-Notiz angibt.

ABTRENNBARKEIT. Wenn irgendein Teil dieser Lizenz durch irgendeine Rechtsprechung außer Kraft gesetzt werden, bleiben die verbleibenden Teile der Lizenz in Kraft.

KEINE GEWÄHRLEISTUNG. Open-Publication-Arbeiten werden lizensiert und verbreitet ”wie sie sindöhne Gewährleistung jeglicher Art, explizit oder implizit, einschließlich, aber nicht begrenzt auf, der impliziten Gewährleistung des Vertriebs und der Geignetheit für einen besonderen Zweck oder eine Gewähleistung einer non-infringement.

ERFORDERNISSE FÜR MODIFIZIERTE ARBEITEN

Alle modifizierten Versionen, die durch diese Lizenz abgedeckt werden, einschließlich von Übersetzungen, Anthologien, Zusammenstellungen und Teildokumenten, müssen die folgenden Erfordernisse erfüllen:

- Die modifizierte Version muß als solche gekennzeichnet werden.
- Die Person, die die Modifikationen vornimmt, muß genannt und die Modifikationen müssen datiert werden.
- Danksagungen der Original-Autors und -Herausgebers - sofern vorhanden - müssen in Übereinstimmung mit der normalen akademischen Zitierungspraxis erhalten bleiben.
- Der Ort des originalen unmodifizierten Dokuments muß benannt werden.
- Die Namen der Original-Autoren dürfen nicht benutzt werden ohne die Erlaubnis des Original-Autors / der Original-Autoren.

EMPFEHLUNGEN EINER GUTEN PRAXIS

In Ergänzung zu den Erfordernissen dieser Lizenz, wird von den Weiterverteilenden erwartet und ihnen stark empfohlen:

- Wenn Sie Open-Publication-Arbeiten als Hardcopy oder auf CD-ROM verteilen, schicken Sie eine E-Mail-Ankündigung Ihrer Absicht der Weiterverteilung mindestens dreißig Tage bevor Ihr Manuskript oder das Medium endgültig festgelegt ist, um den Autoren Zeit zu geben aktualisierte Dokumente anzubieten. Die Ankündigung sollte die Änderungen beschreiben, die gegebenenfalls am Dokument vorgenommen wurden.
- Alle substantiellen Modifikationen (einschließlich Löschungen) sind entweder im Dokument klar zu kennzeichnen oder sonst in einem Anhang zu beschreiben.

Schließlich, obwohl nicht erforderlich unter dieser Lizenz, ist es, eine vorgeschlagene gute Form eine kostenlose Kopie jedes Hardcopy- und CD-ROM-Ursprungs einer unter Open Publication lizensierten Arbeit dem Autor / den Autoren anzubieten.

LIZENZ-OPTIONEN

Der/die Autor/en und/oder der Herausgeber eines unter Open Publication lizensierten Dokuments darf bestimmte Optionen durch Anhängen von Regelungen an den Lizenz-Verweis oder die Lizenz-Kopie wählen. Diese Optionen sind empfohlener Teil der Lizenzbestimmungen und müssen in abgeleiteten Arbeiten in die Lizenz eingefügt werden.

- Verhindern der Verteilung von substantiell modifizierten Versionen ohne explizite Erlaubnis des Autors / der Autoren. „Substantielle Modifizierung“ ist definiert als eine Änderung des semantischen Inhalts des Dokuments und schließt bloße Format-Änderungen oder typographische Korrekturen aus.

  Zur Anwendung fügen Sie den Satz ‘Distribution of substantively modified versions of this document is prohibited without the explicit permission of the copyright holder‘ (Verbreitung von substantiell modifizierten Versionen dieses Dokuments ist ohne die explizite Erlaubnis des Copyright-Inhabers untersagt) dem Lizenz-Verweis oder der Lizenz-Kopie hinzu.

- Verhindern jeglicher Veröffentlichung dieser Arbeit oder abgeleiteter Arbeiten im Ganzen oder in Teilen in Standard- (Papier-) Buchform für kommerzielle Zwecke ohne vorherige Erlaubnis durch den Copyright-Inhaber.

  Zur Anwendung fügen Sie den Satz ‘Distribution of the work or derivative of the work in any standard (paper) book form is prohibited unless prior permission is obtained from the copyright holder‘(Verbreitung dieser Arbeit oder abgeleiteter Arbeiten in Teilen in Standard- (Papier-) Buchform für kommerzielle Zwecke ohne vorherige Erlaubnis durch den Copyright-Inhaber ist untersagt) dem Lizenz-Verweis oder der Lizenz-Kopie hinzu.

Literaturverzeichnis

C.3 zitierte Literatur

[1] Brockhaus Enzyklopädie, Band 6

[2] Duden Fremdwörterbuch

[3] IBM-Lehrgangsunterlagen „DB2 Familie - Datenbankdesign“

C.4 Weitere Informationsquellen

Dies soll keine Linksammlung werden - dazu gibt es einfach zu viele Webseiten über PHP und PHP-Portale im Internet. An dieser Stelle werden nur die wichtigsten Seite bzw. Portale aufgelistet. Von diesen Ausgangspunkten aus findet man fast alles.

[4] Die Herstellerseite von MySQL
http://www.mysql.com/

[5] Die Herstellerseite von MySQL (in deutsch)
http://www.mysql.de/

[6] Die Homesite von PHP
http://www.php.net/

[7] SELFHTML von Stefan Münz, DAS Nachschlagewerk in Sachen HTML/CSS
http:// de.selfhtml.org/

[8] Die FAQ zu PHP (sehr ausführlich und enthält auch MySQL)
http:// www.php-faq.de/

[9] Die FAQ zu MySQL (befindet sich noch im Aufbau)
http://mysql-faq.sourceforge.net/

[10] PHP-Center, ein deutsches PHP-Portal
http://www.php-center.de

[11] PHP-Homepage, ein weiteres deutsches PHP-Portal
http://www.php-homepage.de

[12] Webserver mit SSL-, PHP- und (über letzteres im Zusammenspiel mit einem MySQL-Server, s. o.) MySQL-Unterstützung
http://www.opensa.org/

Abbildungsverzeichnis

Tabellenverzeichnis

D Danksagungen

Bei so einem Werk dürfen die Danksagungen natürlich nicht fehlen.

Als erstes und wichtigstes muß ich natürlich allen Programmierern von Linux, Apache, MySQL, PHP, LaTeX, Ghostview, XFree86, xEmacs, CVS und allen Programmierern, deren Programme ich auch noch benutzt habe, danken. Denn erst durch sie ist die Grundlage für dieses Dokument möglich geworden.

Dann darf ich natürlich die ganzen Kritiker, allen voran Stefan Nagelschmitt, Jens Hatlak und Nikolai Stiehl mit Mutter(!), nicht vergessen, die dieses Dokument erst zu dem gemacht haben, was es jetzt hoffentlich ist. Ich hoffe, es kommen noch ein paar dazu.

Ein besonderes Dankeschön geht an die kreativen Leser, die einen Vorschlag für das neue Logo gemacht haben. Es waren viele, sehr interessante Logos dabei!

Und wie es immer so schön heißt: Ich bedanke mich noch bei allen, die ich vergessen habe.

E Versionen

Aufschieben ist die große Kunst, Dinge, zu denen man heute keine Lust hat, nächste Woche erst recht nicht zu tun.

E.1 19.06.2005

- neues Kapitel: Benötigte Software
- Rekursion überarbeitet
- MySQL: Nested Set etwas erweitert
- MySQL: IF Ausdrücke in MySQL
- PHP: foreach-Schleife; break x, continue
- PHP: Variable Variablen
- PHP: Typensichere Vergleiche
- PHP: Warn-/Fehlermeldungen anzeigen
- PHP/MySQL: Tips und Tricks (Abfragen mit IN)
- PHP/MySQL: Einfügen mit automatischer ID
- PHP/OO: Referenzsemantik mit Übung
- PHP/OO: Ausblick PHP5
- neues Kapitel: Sessions
- viel Kleinkram und Fehlerkorrekturen
- ein paar neue Übungen
- zusätzliche PS/PDF Versionen zum ausdrucken

E.2 13.05.2002

- PHP, SQL und HTML Code hat Syntaxhervorhebung
- neues Kapitel über Baumstrukturen in SQL
- neues Kapitel über PHPDOC
- neues Kapitel über das Parsen von URLs
- neues Kapitel über Objektorientierung
- neues Kapitel über XML
- neues Kapitel über Templates
- PHP: Funktionen (s)printf und number_format
- Beispielscripte Banner, SDA und Gästebuch hinzugefügt
- Kapitel über HTTP-Header ergänzt
- Kapitel Tips & Tricks erstellt
- HTML- und PDF-Versionen etwas verbessert
- Mailingliste für neue Versionen
- Autorenvorstellung
- viel Kleinkram

E.3 22.06.2001

- Verbesserungsvorschläge und Korrekturen von Markus Maier und Jens Teich umgesetzt

E.4 27.04.2001

- Kapitel zu regulären Ausdrücken hinzugeführt
- Vorwort erweitert
- Literaturverzeichnis komplettiert
- `require()`, `include()` besser erklärt
- verschiedene Kapitel etwas erweitert

E.5 05.12.2000

- Übungen zum Kapitel “Fehlersuche“ hinzugefügt
- ausführliche Erklärung der einzelnen Datentypen hinzugefügt
- kleine Detailverbesserungen

E.6 27.11.2000

- Funktion DATE_FORMAT hinzugefügt
- kleine Fehler korrigiert

E.7 19.10.2000

- Kapitel ‘PHP & HTTP‘ erweitert
- `print` erklärt
- Beispiel zur Erklärung der Tabellenstruktur hinzugefügt
- Erklärung zu Funktionen etwas überarbeitet

E.8 30.09.2000

- Übung “Ergebnis-Tabelle ausgeben I“ hinzugefügt
- Übung “Ergebnis-Tabelle ausgeben II“ hinzugefügt
- `ORDER BY`-Erklärung erweitert
- `mysql_num_rows` und `mysql_errno` mit aufgenommen
- Vorwort etwas überarbeitet

E.9 27.07.2000

- Grundzustand

Index