Datenbank, mySQL und PHP

Christoph Reeg

5. Dezember 2000

DSP - Datenbank SQL PHP

Inhaltsverzeichnis

1. Vorwort

Diese Anleitung erklärt die Grundlagen einer Datenbank in Verbindung mit der Abfragesprache SQL und der Scriptsprache PHP. Dafür wird anhand eines Beispiels ein Datenmodell entwickelt, welches nachher auch umgesetzt wird. Mit einigen Beispielen wird versucht, die Theorie etwas aufzulockern.

Die Kenntnis der Sprache HTML wird vorausgesetzt. Wer nicht alles darüber weiß, kann sich unter [6] noch ein Nachschlagewerk besorgen. In dieser Anleitung werden im Prinzip keine HTML-Befehle beschrieben.

Diese Anleitung wurde ursprünglich für Scripte 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 nach dem LAMP[1]-Prinzip arbeiten, solange dabei etwaige Server-spezifische Besonderheiten 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. Als Formate stehen PS, PDF und HTML (online und als ZIP) zum Herunterladen zur Verfügung.

Wenn Du Anregungen hast, was noch verbessert werden kann, würde ich mich über eine eMail an *dsp@reeg.net* sehr freuen.

1.2. Unterteilung

Dieses Dokument ist in verschiedene Teile unterteilt. Die ersten drei derselben können auch einzeln für sich gelesen werden.

Erster Teil

Im ersten Abschnitt dieses Teils der Anleitung soll geklärt werden, was eine Datenbank ist, sowie, 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, die die Datenstrukturen noch entwickeln müssen, 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 einmal Zeit, unsere Kenntnisse am Computer zu testen.

[1] Linux, Apache, mySQL, PHP

Als DBMS benutzen wir ‘mySQL‘, das als Abfragesprache SQL[2] benutzt. SQL ist eine nach bestimmten Normen festgelegte Sprache, die von der Mehrheit der Relationalen DBS[3] benutzt wird; dazu zählen unter anderem: 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 mußt.

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, kann sich unter [5] die gesamte Anleitung ansehen (in Englisch, sehr lang aber nicht ausführlich).

PHP gibt es seit kurzem in der Version 4. Bis alle umgestellt haben, wird jedoch sicherlich noch einige Zeit lang PHP3 sehr weit 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 ;-)).

Dieser Teil ist unbedingt notwendig, um in PHP programmieren zu können und sollte deshalb verstanden worden sein. Das soll nicht heißen, 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 eMail-Adresse kennst Du ja. ;-)

Ein paar der Beispiele sind auch unter *http://reeg.net/* in Aktion zu sehen.

1.3. Typographische Konventionen

Um die Übersichtlichkeit zu erhöhen, werden einige Sachen hervorgehoben. Die folgende Liste zeigt, auf welche Weise was hervorgehoben wird:

[2] Structured Query Language
[3] Datenbanksysteme, mehr dazu im ersten Teil

Wie	**Was**	**Beispiel**
kursiv	Internet-Adressen	Wenn Du Anregungen hast, dann schreib mir doch eine eMail an *dsp@reeg.net*
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. 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**.

Tabelle 1.1.: Typogr. Konventionen

Teil I.

theoretische Grundlagen

2. Datenbanksystem

2.1. Komponenten eines Datenbanksystems

Eine Datebank (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.

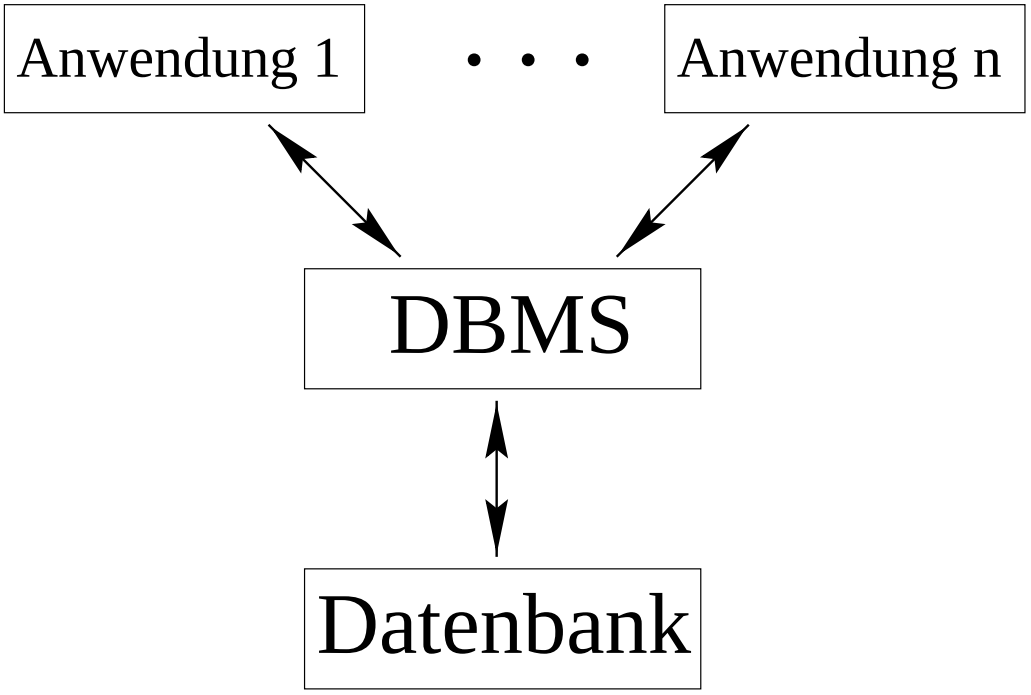

Abbildung 2.1.: Struktur eines Datenbanksystems

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.

Diese Funktionen 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.

2.2. Ebenen eines Datenbanksystems

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

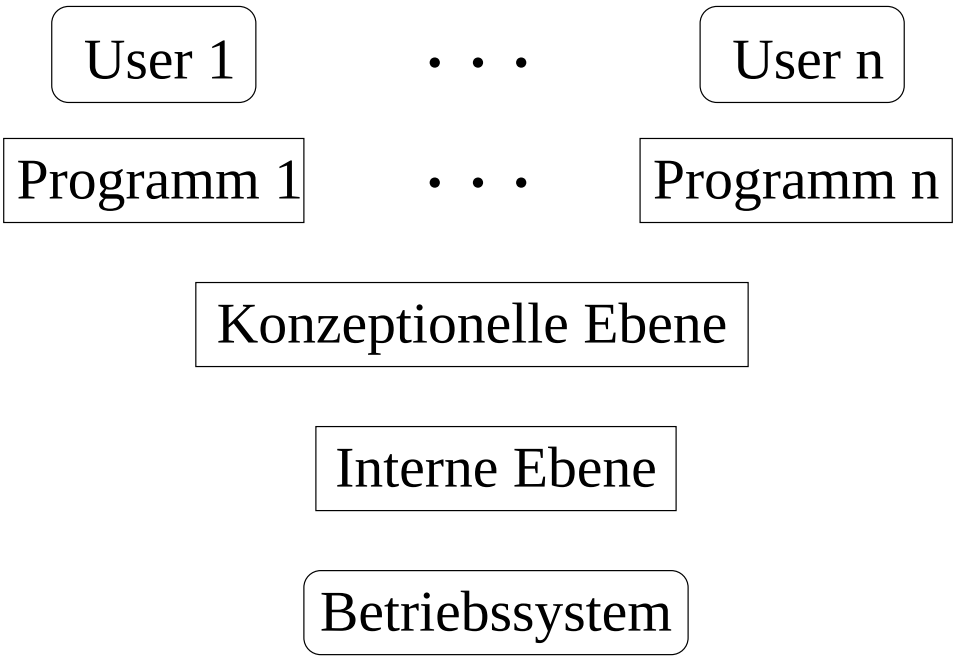

Abbildung 2.2.: Die vier Ebenen eines DBS

2.2.1. 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.

2.2.2. 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.

2.2.3. 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 und relationalen Datenmodellen. Wir verwenden im Folgenden lediglich das relationale Datenmodell, da es das modernste ist und sich durchgesetzt hat.

2.2.3.1. 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,

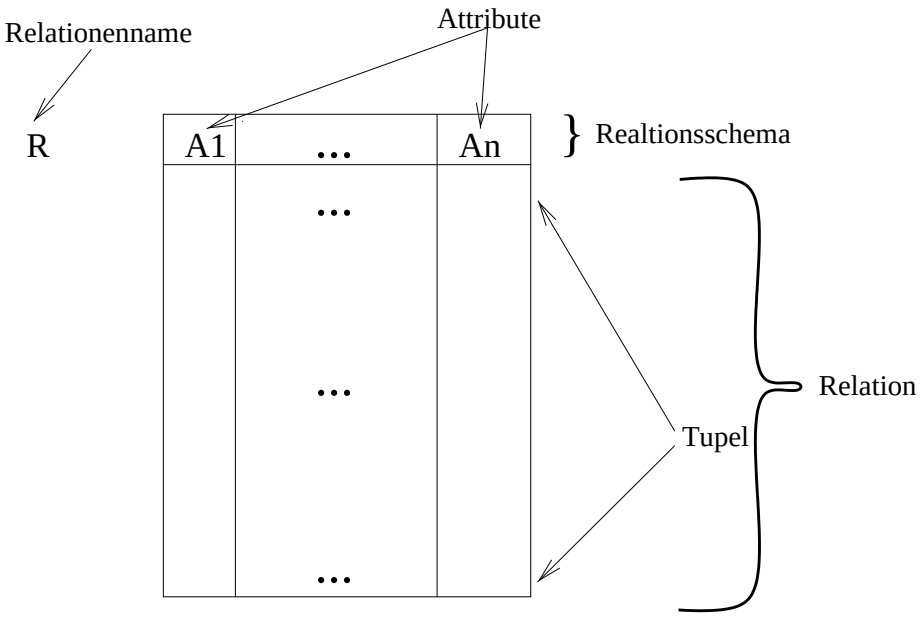

Abbildung 2.3.: Tabellenstruktur

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 2.1.: Beispiel für Tabellenstruktur

die Spaltenüberschriften als Attributnamen oder Attribute und alle Attributnamen zusammen werden als Relationsschema bezeichnet.

Allgemein wird in jeder Zeile eine Entität abgebildet.

In Abbildung 2.3 wurde versucht, die Zusammenhänge grafisch darzustellen.

Um das Ganze etwas konkreter zu machen, habe ich in Tabelle 2.1 ein kleines Beispiel dargestellt.

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.

2.2.3.2. 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 Schlüsselkanditat 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).

2.2.4. 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.

3. Datenbanken entwickeln

3.1. Vorgehensweise

Die Entwicklung einer DB vollzieht sich in mehreren Schritten. Zunächst ist festzustellen, welche Informationen die Anwender vom DBS erwarten. 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.

3.2. Grundsätze

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

3.2.1. 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.

3.2.2. 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.

3.2.3. 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.

3.3. Datenmodelle entwickeln

Es gibt mehrere Vorgehensweisen. Eine Möglichkeit ist, erst einmal darüber nachzudenken, was man eigentlich machen will, dann die entsprechenden Prozeduren entwickeln und dabei 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.

Bei der objekt-orientierten Programmierung geht man noch einmal anders vor. Hier überlegt man sich, welche Objekte benötigt werden, welche Daten sie speichern müssen und welche Methoden sie benötigen. Dazu später mehr in einem eigenen Kapitel.

3.3.1. Tabellen erstellen

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 3.4) oder etwas intuitiver über das ER-Modell (Kapitel 3.6), evtl. anschließend mit Kontrolle durch die fünf Normalformen (Kapitel 3.4).

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.

3.4. Die fünf Normalformen

3.4.1. 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. Siehe dazu auch folgende Tabelle ‘Auftrag‘.

Auftrag				
AuftragNr	**Datum**	**KundenNr**	**AritkelNr**	**Anzahl**
4711	3.10.1999	12345	4692	5
			0567	2
			5671	3
			0579	1
0815	1.3.1998	54321	8971	2
			5324	5
			0579	9

Um die Wiederholungsgruppen zu vermeiden, werden für diese gesonderte Tabellen erstellt. Dadurch würden sich die folgenden beiden Tabellen ergeben:

best. Artikel	
AritkelNr	**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 für ‘Auftrag‘. Nun fügen wir noch diese 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 9 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‘ bezeichnet.

Damit ergeben sich für unser Beispiel die folgenden beiden Relationen:

best. Artikel		
AufragNr	**AritkelNr**	**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

3.4.2. 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‘ abghängig. 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 folgende beiden Relationen entstehen:

best. Artikel		
AufragNr	**AritkelNr**	**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

3.4.3. 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 funktional abhängig von der HerstellerNr und nicht von der ArtikelNr.

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

3.4.4. 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.

3.4.5. 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? ;-)

3.4.6. 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 empfiehlt sich jedoch, aus Performancegründen wieder eine gewisse Denormalisierung herzustellen. Da wir aber keine so hohen Zugriffszahlen und Datenbestände haben, daß der Server überlasten 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 3.5) zu überprüfen, ob die Tabellen den Anforderungen der Vorgänge entsprechen.

3.5. 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 3.1 dargestellt.

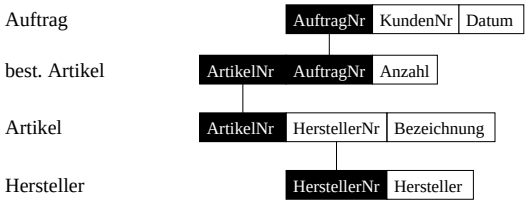

Abbildung 3.1.: Streifendiagramm

3.6. 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.

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. Ferner soll man einer einer Abteilung zugehörig sein können. Jeder Mitarbeiter hat einen Vorgesetzten. Außerdem verfügt jede Abteilung über einige oder keine PKWs aus dem Fuhrpark für die Mitarbeiter. Des weiteren soll es möglich sein zu sagen, wer wann mit welchem Wagen gefahren ist.

Diese Realtitätsbeschreibung legt drei Entitäten nahe: Mitarbeiter, Abteilung, PKW. Die Zeichnung 3.2 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 durchgezogener Strich bedeutet **genau einer**, der gestrichelte **einer oder keiner**. Der Krähenfuß auf der anderen Seite bedeutet **oder mehrere**. Die Wörter an den Strichen zeigen, was die Beziehung aussagt.

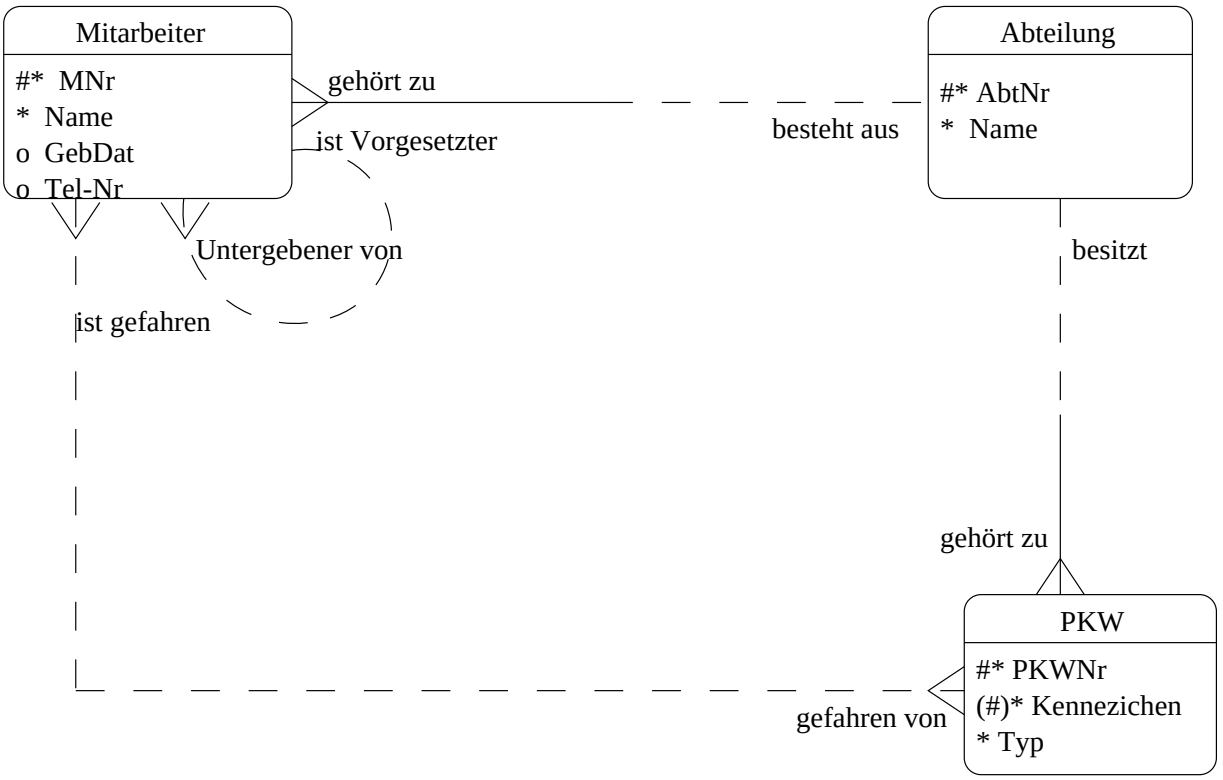

Abbildung 3.2.: ER-Beispiel 1

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?

In den Rechtecken stehen die wichtigsten Attribute. Primärschlüssel werden durch ein # gekennzeichnet, Primärschlüsselkanditaten 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 3.3 dargestellt.

3.7. 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
Fahrbuch	MNr, PKWNr, Datum
PKW	PKWNr

Die Relation ‘Fahrbuch‘ hat einen zusammengesetzten Primärschlüssel. Die ‘MNr‘ oder auch

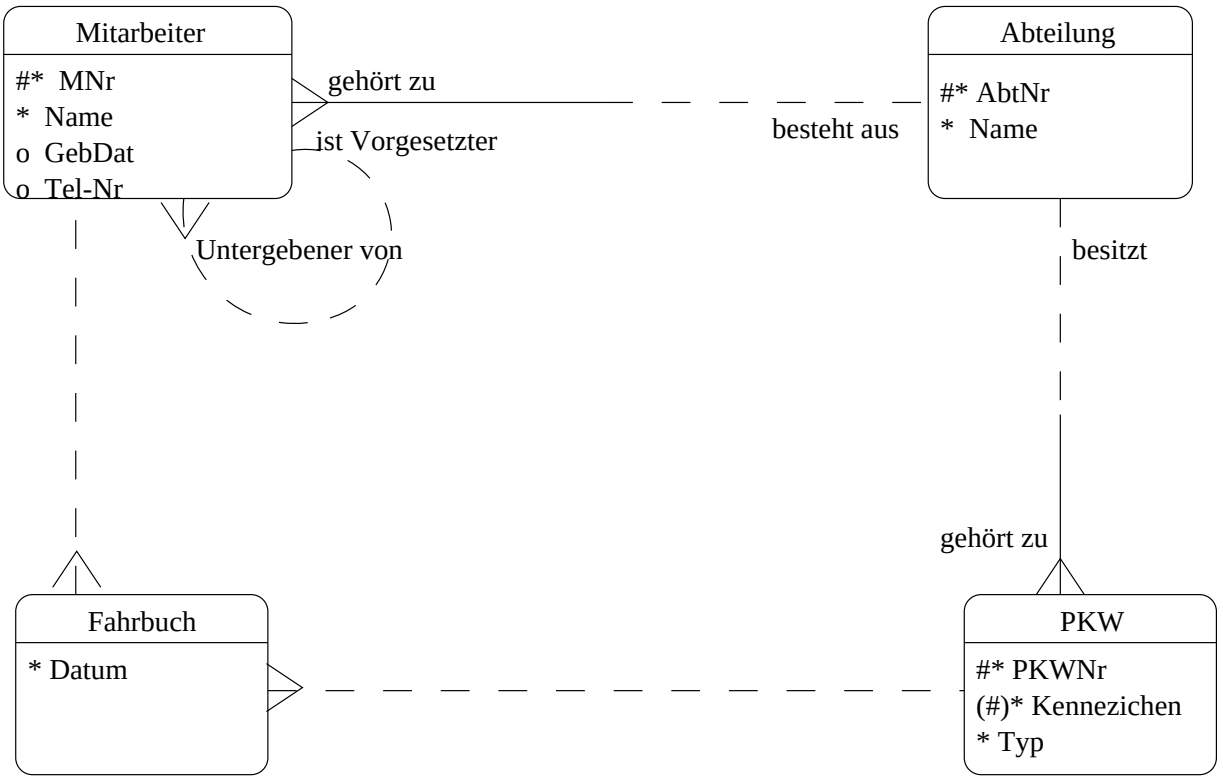

Abbildung 3.3.: ER-Beispiel 2

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 Datenintigrität sicherstellen zu können, habe ich einen 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	
Fahrbuch	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.

3.8. 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 5.1.

Bei der Überlegung, welchen Datentyp man verwendet, sollte man nicht vom Normalfall ausgehen, sondern immer alle möglichen Ausnahmen betrachten. 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

4. 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.

In diesem Kapitel werden wir dialogorientiert arbeiten. Alle Befehle, die hier direkt eingegeben werden, können 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. Weiter unten wird dann auch die Benutzung durch PHP beschrieben.

4.1. 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 eine Datenbank zu installieren. Die Aufrufsyntax lautet:

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

Für Paßwort, Benutzername und DB-Name sind natürlich die entsprechenden Werte einzutragen (ohne die geschweiften Klammern!!). Diese bekommst Du vom Provider, der Dir die Datenbank zur Verfügung stellt, bzw., im Falle einer eigenen DB, gibst Du Dir selbst die Daten. ;-)

Wir richten immer zwei Accounts für dieselbe DB ein. Einen, für den sowohl Lese- als auch Schreibzugriffe erlaubt sind und einen, der nur lesend zugreifen darf. Der nur lesende Account bekommt ein ‘_ro‘ an das Ende des Benutzernamens angehängt.

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!

4.1.1. Dateien abarbeiten

Wie oben schon erwähnt, kann man die Befehle auch mit einem ASCII[1]-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 normal angezeigt werden, werden auf den Bildschirm geschrieben.

[1] American Standard Code for Information Interchange

4.1.2. 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.

5. SQL-Befehle

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 Beispiel-Datenbank wird die Datenbank, die in Kapitel 3.6 beschrieben wird, benutzt.

5.1. 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_definition's durch Kommas getrennt hintereinander stehen können. Die 'create_definition's 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 (Feldname,...)
oder UNIQUE (Feldname,...)
oder FOREIGN KEY (Feldname,...) [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.

Tabelle 5.1.: 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<= NUM <=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 oder HHMMSS oder 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 5.2.: 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)

Für Typ ist einer der Datentypen aus Tabelle 5.1 zu verwenden.
Die Bedeutung der YMHSDs ist in der Tabelle 5.2 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 ist der Name und die Telefonnummer, sowie 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` (5.10) 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.

5.2. 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.

5.3. 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!

5.4. 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, wie 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 Beispiel aus.

5.5. SELECT

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

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

Für alle, die sich fragen, warum das der mächtigste Befehl sein soll, nur ein kurzer Verweis auf Anhang C.

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>
```

5.5.1. 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. Wenn nichts angegeben wird, 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] engl. ascending
[2] engl. descending

```
+-----+------+-------+----------------+------------+--------------+
|   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;
+-----+------+-------+----------------+------------+--------------+
| 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>
```

5.5.2. GROUP BY

Die `GROUP BY`-Anweisung wird nur in Verbindung mit den Gruppenfunktionen aus Kapitel 5.6.5 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>
```

5.5.3. 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. Wird nichts angegeben, 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 fünf Zeilen zurück
mysql> select * from table LIMIT 0,5;   # dasselbe nochmal
```

5.5.4. 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. Aliases 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 5.6 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>
```

5.5.5. Alias

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

5.5.5.1. Tabellen-Alias

Tabellen-Aliase können sowohl bei der `select_expression` als auch bei der 'where_definition' zur eindeutigen Spaltenbeschreibung anstelle des Tabellennamens verwendet werden. Aliases werden verwendet, weil sie in der Regel kürzer sind als der Spaltenname. Aliases 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>
```

5.5.5.2. 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 5.6, das soll uns aber nicht weiter stören).

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

mysql>
```

Das einzige, was noch etwas stört, 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.

5.5.6. 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 ausgeben wird.

Die 'where_definition' wird auch beim Löschen (`DELETE`) und Ändern (`UPDATE`) 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 5.3 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 5.3.: 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 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 5.6 werden diese noch genauer beschrieben.

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

5.5.6.1. 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 zu lösen, schicke mir bitte eine eMail 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  |
+-------+
```

[3] engl.: regular expressions

```
| 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

5.5.6.2. 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.

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.

5.5.6.3. 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>
```

5.6. 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.

5.6.1. Mathematische Funktionen

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

Eine Auswahl der mathematischen Funktionen ist in Tabelle 5.4 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 5.4.: Mathematische Funktionen in SQL

5.6.2. Logische Operatoren

Die logischen bzw. bool’schen Operatoren haben wir bereits im Kapitel 5.5.6 kennengelernt. In Tabelle 5.5 sind sie der Vollständigkeit halber noch einmal aufgeführt.

NOT !	logisches NOT. Gibt 1 zurück, wenn das Argument 0 ist, sonst 0. Ausnahme: NOT NULL gibt NULL zurück.
AND &&	logisches UND. Gibt 0 zurück, wenn eines der Argumente 0 oder NULL ist, sonst 1.
OR ||	logisches ODER. Gibt 1 zurück, wenn eines der Argumente nicht 0 und nicht NULL ist, sonst 1.

Tabelle 5.5.: Logische Funktionen in SQL

5.6.3. Sonstige Funktionen

In Tabelle 5.6 und 5.7 sind sonstige, teilweise praktische Funktionen aufgeführt. Die Tabellen sind nicht vollständig!

|	bitweises ODER
&	bitweises UND

Tabelle 5.6.: 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 5.7.: String-Funktionen in SQL

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 entsprechend dem übergebenen `format` String.
UNIX_TIMESTAMP(date)	Gibt den Unix-Timestamp (Sekunden seit dem 1.1.1970) des Datums `date` zurück.

Tabelle 5.8.: Datum-Funktionen in SQL

5.6.4. 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 5.9) 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 mehreren Stellen auf der Seite verschieden formatiert haben will. Wie gesagt, es kommt auf den Einzelfall und die Vorlieben des Programmierers an.

`%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 5.9.: mögl. Formatierungen für DATE_FORMAT

5.6.5. 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 5.10 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)
```

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

Tabelle 5.10.: Gruppenfunktionen in SQL

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

mysql>
```

5.7. Joins

Nachdem wir jetzt die Tabellen perfekt abfragen können, wollen wir mal ein paar Tabellen miteinander verbinden. Nach unserer Normalisierung (siehe Kapitel 3.4) befinden sich einige Informationen nämlich in verschieden 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>
```

5.7.1. 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 5.5.5).

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.

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

5.7.2. 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.

5.7.3. 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.

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 genommen 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>
```

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.

Ein neues Beispiel:
Ich will alle Abteilung aufgelistet bekommen, 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     |
```

[6] Wer vergeßlich ist, kann auf Seite 31 nachschlagen :-)

```
+-----------+-------------+---------+
| 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:

```
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>
```

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

5.8. 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`.

5.9. 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 column=expression,... [WHERE where_definition]
```

Für ‘table_name‘ ist selbstverständlich der Name der Tabelle einzusetzen, in der einzelne Attribute geändert werden sollen. ‘column‘ ist durch den Spaltennamen, in dem die Attribute geändert werden sollen zu ersetzen, und ‘expression‘ durch den Wert. Falls Attribute in mehreren Spalten geändert werden sollen, können diese einfach durch 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.

5.10. 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 old_column_name create_definition
oder ADD PRIMARY KEY (index_spalte,...)
oder ADD INDEX (index_spalte,...)
oder ADD UNIQUE (index_spalte,...)
oder DROP column_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>
```

Als letztes 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).

Teil III.

Einführung PHP

6. PHP Grundlagen

6.1. Einleitung

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. zu JavaScript. Der Programmcode wird in die HTML-Datei geschrieben und nicht in einer extra Datei abgelegt. Als Script werden Programme bezeichnet, die keine eigenständigen Programme sind, weil sie nicht kompliziert genug sind und andere Programme brauchen, um ausgeführt zu werden.

Immer wenn man sich fragt, ob etwas möglich ist, muß man 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, so daß ein JavaScript-Befehl notwendig ist, um die Seite auszudrucken (für viele andere Aktionen ist aber nicht einmal JavaScript nötig).

Im vorigen Kapitel wurde SQL beschrieben, jetzt wird PHP erklärt. Dies sind zwei voneinander unabhängige Sprachen, die erst einmal nichts miteinander zu tun haben! 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.

6.2. 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-konform, so daß später Probleme auf den Programmierer zukommen können, wenn man sie benutzt.

Die Sprache PHP ist hauptsächlich von C, aber auch von Java und Perl (die ihrerseits von C beeinflußt wurden) beeinflußt. Aber auch für Pascal-Programmierer ist die Sprache nicht schwer zu erlernen.
Eine Anweisung wird immer mit einem ‘;‘ abgeschlossen.

[1] PHP Hypertext Preprocessor
[2] Hypertext Markup Language

6.2.1. Der echo-Befehl

Den wichtigsten Befehl haben wird 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. Bei den Anführungsstrichen gibt es zwei verschiedene: einmal das einfache „‘“ und dass doppelte „”“. Es gibt auch einen Unterschied zwischen den beiden. Bei den doppelten Anführungsstrichen versucht der Server den Text zu interpretieren, bei den einfachen gibt er ihn dagegen einfach aus. Weitere Erklärungen zu Anführungszeichen finden sich in Kapitel 6.2.6.4.

```
$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";  // Gibt aus: Say "Hello World!" my friend
```

Wie man sieht, müssen doppelte Anführungsstriche anders geschrieben werden. Dieses Vorgehen nennt man Quoten oder Quoting[3]. 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 quoten. Siehe auch Kapitel 10 (Fehlersuche).

6.2.2. Der print-Befehl

Neben dem `echo`- gibt es auch den `print`-Befehl. Im Endeffekt leisten sie dasselbe: Sie geben Text aus. `echo` ist ein internes Sprachkonstrukt, während hingegen `print` eine 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);
```

6.2.3. 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.

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

6.2.4. 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 (bool'schen)- und Vergleichs-Operatoren unterscheiden.

[3] engl. to quote: zitieren

6.2.4.1. Arithmetische Operatoren

Tabelle 6.1.: Arithmetische Operatoren in PHP

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	Dividend von $a und $b
$a % $b	Modulo	Rest der Division von $a und $b

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

6.2.4.2. 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!";
```

6.2.4.3. 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 6.2.: Bit-Operatoren in PHP

6.2.4.4. Logische Operatoren

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

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

6.2.4.5. Vergleichsoperatoren

Für Bedingungen braucht man Vergleichsoperatoren.

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 sind
$a xor $b	Exklusiv-ODER	true, wenn genau einer $a oder $b true sind
!$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 sind

Tabelle 6.3.: Logische Operatoren in PHP

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

Tabelle 6.4.: Vergleichsoperatoren in PHP

6.2.5. Kommentare

Für Kommentare gibt es zwei Möglichkeiten der Schreibweise:

```
echo "Noch kein Kommentar!";

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

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

echo "Kein Kommentar mehr!";
```

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

6.2.6. Variablen

Alle Variablen werden durch ein vorangestelltes '$' gekennzeichnet. Die Variablen müssen nicht vorher definiert werden. PHP verwendet den Typ, den es für richtig hält. Der Variablentyp kann auch bei der ersten Benutzung festgelegt werden, indem er in Klammern angegeben wird. In Tabelle 6.5 sind die verfügbaren Typen aufgelistet.

Tabelle 6.5.: Typen in PHP

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

6.2.6.1. 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, wird die Variable in den Typ ‚double' umgewandelt.

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 Hexadzimalzahlen ein „`0x`“ und bei Oktalzahlen eine „`0`“ vorangestellt. Diese Zahlensysteme werden häufig wegen ihrer stärkeren Hardware-Nähe genommen. Für uns sind sie im Moment eher weniger interessant.

6.2.6.2. 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.

6.2.6.3. Boolean

Mit PHP4 ist auch der Datentyp ‚Boolean‘ gekommen. PHP3 hat den boolschen Wert „true“ als Integer mit dem Wert ‚1‘ interpretiert.

6.2.6.4. String

In Strings werden Buchstaben-/Zeichenketten gespeichert. Wenn man Strings schreibt, müssen diese in Anführungszeichen geschrieben werden (siehe auch Kapitel 6.2.1).

Betrachten wir als erstes die doppelten Anführungszeichen (“). Um z.B. innerhalb der Anführungszeichen eines `echo`- oder `print`-Befehls eines 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[4]“ bezeichnet. Eine Übersicht der escaped characters gibt es in Tabelle 6.6.

\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 6.6.: escaped characters

Auch bei den einfachen Anführungszeichen gibt es escaped characters. Es sind genau zwei Stück, nämich \‘ und \\.

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.

6.2.6.5. 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 (äquivalent zu 83 dezimal)
$a = 0xbad;         \\ Hexadezimalzahl (äquivalent zu 2989 dezimal)
echo floor((0.1+0.7)*10); \\ Ausgabe ist 7
```

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

6.2.6.6. Array

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

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

Bei der 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 Klammer hinter dem Array-Namen angegeben.

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

Zeile	Name	Tage
1	Januar	31
2	Februar	28
3	März	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:

```
$monat[1]["Name"] = "Januar";          $monat[1]["Tage"] = 31;
$monat[2]["Name"] = "Februar";         $monat[2]["Tage"] = 28;
```

```
$monat[3]["Name"] = "März";          $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 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.

6.2.7. 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 größer 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 größer als b";
  $b = $a;
}
```

6.2.8. 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 größer als b";
if ($a<=$b)
  print "a ist nicht größer als b";
```

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

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

6.2.9. ELSEIF

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

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

6.2.10. 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.

6.2.11. 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 B":"");
?>
```

Ich jedoch mag ich diese Syntax nicht besonders, weil sie mir zu unübersichtlich ist. Mein Lektor ist da anderer Ansicht. ;-)

6.2.12. 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.

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
```

```
}

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

6.2.13. 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-Kenner:
Die `DO ...WHILE`-Schleife ist vom Prinzip her identisch mit `REPEAT UNTIL`.

6.2.14. FOR

`FOR`-Schleifen 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 jeder Schleife überprüft. Wenn die Bedingung erfüllt ist, wird die Schleife ausgeführt, wenn nicht, wird abgebrochen. Das ist die Laufbedingung.
Am Ende jeder Schleife wird der dritte Ausdruck ‘expr3‘ ausgeführt.
Jeder Ausdruck kann auch leergelassen 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 Variablen) und als letztes wird $i inkrementiert[5].

Mit break wird die aktuelle Schleife verlassen.

6.2.15. SWITCH

Die Funktion der SWITCH-Anweisung ist identisch mit der der CASE OF-Anweisung in Pascal. 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.

[5] Um eins erhöht

Am Anfang wird kein Code ausgeführt. Nur dann, wenn eine CASE-Anweisung mit einem Wert gefunden wird, der gleich dem Wert der 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";
}
```

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 im folgenden Fall sieht:

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

6.3. 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 PHP-Code steht, diese Datei mit <?php anfangen und mit ?> aufhören (bzw. mit einer anderen PHP-Code-Markierung, siehe Kapitel 6.2).

6.4. Funktionen

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

Bei einigen Programmiersprachen findet eine Unterscheidung zwischen Funktionen, die einen Wert zurückgeben und solchen, die keinen Wert zurückgeben statt. Z.B. in Turbo-Pascal 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. Rückgabewerte müssen nicht abgefragt werden.

Ein kleines Beispiel:

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

6.4.1. Variablenparameter

Normalerweise werden in PHP Werteparameter[6] übergeben. Will man jedoch die Änderungen der Parameter in der Funktion auch in der aufrufenden Funktion haben, muß man mit Variablenparametern[7] bzw. Referenzparametern[8] arbeiten.

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.';
```

[6] In der Funktion wird mit einer Kopie der Variablen gearbeitet
[7] Es wird mit den Originalvariablen gearbeitet, weil nur die Adresse übergeben wird
[8] Zwei Namen für dasselbe

7. PHP & HTML

7.1. Formulare

Mit PHP können Formulardaten relativ einfach eingelesen werden. Man muß im Formular als **action** den Dateinamen der PHTML-Datei angeben und als **method** empfiehlt sich ‘post‘. Bei ‘get‘ kann man sonst die Werte der Eingabe in der URL wiederfinden.

Nun ein kleines Beispiel. Die Eingaben, die auf der Seite ‘eingabe.html‘ eingetragen wurden, werden durch OK an die Datei ‘ausgabe.phtml‘ 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.phtml" 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">
  <input type="reset" value="Abbrechen">
</form>
</div>
</body>
</html>
```

Quelltext der Datei ‘ausgabe.phtml‘:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<html>
<head>
 <title>Ausgabe</title>
</head><body>
<?php
print "Feld 1:".$feld1."<br>Feld 2:".$feld2;
?>
</body>
</html>
```

7.2. Werte übergeben

Es gibt auch Situationen, da möchte man dem PHP-Skript Werte übergeben, ohne ein Formular zu verwenden. Dies ist natürlich auch möglich. Ein normaler HTML-Link sieht folgendermaßen aus:

```
<a href="datei.phtml">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.phtml?var1=Wert1&var2=2">Linktext</a>
```

Allgemeiner formuliert: An das Verweisziel (in unserem Fall ‘datei.phtml‘) wird mit einem ‘? ‘ beginnend der Variablenname und mit einem Gleichheitszeichen der Wert herangehängt. Weitere Werte werden mit einem ‘&‘ statt ‘? ‘ angehängt. Es dürfen keine Leerzeichen dabei entstehen.

Wenn man die übergebenen Werte verwendet, darf man nicht vergessen, daß **jeder** die Werte beim Aufruf verändern kann. Deshalb sollten die Variablen vor der Weiterverarbeitung auf korrekte Werte hin überprüft werden.

8. 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!

8.1. Syntax

8.1.1. mysql_connect

Syntax:

```
int mysql_connect(string hostname , string username , string password);
```

Mit **mysql_connect()** wird eine Verbindung zum Server geöffnet.

Wenn der zurückgegebene Wert positiv ist, verlief die Verbindung erfolgreich; bei Null gab es einen Fehler. 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.

8.1.2. mysql_close

Syntax:

```
int mysql_close(int 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.

8.1.3. mysql_select_db

Syntax:

```
int mysql_select_db(string database_name , int 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 genommen.

8.1.4. mysql_query

Syntax:

```
int mysql_query(string query , int link_identifier);
```

mysql_query sendet die 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 0 hat, gibt es einen Fehler. Mit dem Zeiger an sich kann man aber nicht viel anfangen; vielmehr benötig man ihn, um die folgenden Funktionen nutzen zu können. Dort wird er als Parameter `result` übergeben.

8.1.5. mysql_fetch_array

Syntax:

```
array mysql_fetch_array(int 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`.

```
<?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"];
}
?>
```

8.1.6. mysql_fetch_row

Syntax:

```
array mysql_fetch_row(int 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];
}
?>
```

8.1.7. mysql_error

Syntax:

```
string mysql_error([int 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.

8.1.8. mysql_errno

Syntax:

```
int mysql_errno([int 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.

8.1.9. mysql_insert_id

Syntax:

```
int mysql_insert_id(int link_identifier);
```

Gibt die Nummer zurück, die beim letzten `INSERT` dem Feld mit `AUTO_INCREMENT` zugewiesen wurde.

8.1.10. mysql_num_rows

Syntax:

```
int mysql_num_rows(int result);
```

Gibt die Anzahl der Zeilen im Ergebnis zurück.

8.2. Übung

8.2.1. 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 zu Stande:

```
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";
?>
```

Tipps 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.1.

8.2.2. Ergebnis-Tabelle ausgeben I

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.2.

9. PHP & HTTP

9.1. 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 gecacht 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!** Wenn PHP als CGI installiert ist, gibt es außerdem einige Einschränkungen, z.B. kann keine Authentifizierung gemacht werden.

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.

9.1.1. Weiterleiten

Wie bereits oben erwähnt, kann man, neben JavaScript, 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.

Bei dieser Anweisung sendet Apache automatisch den Statuscode 302.

9.1.2. 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');
```

[1] Request for Comments

9.1.3. 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**[2].

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 die beiden Variablen ‘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 Zeilen müssen fast genau so übernommen werden, damit es funktioniert![3] 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 ($PHP_AUTH_USER == ""
      OR !@mysql_connect("localhost",$PHP_AUTH_USER,$PHP_AUTH_PW)) {
    Header('HTTP/1.0 401 Unauthorized');
    Header('WWW-Authenticate: Basic realm="Top Secret"');
```

[2] Übrigens: PHP gibt es auch als Modul für Windows, siehe auch PHP-FAQ[7] “Wo finde ich PHP als Modul für Windows?“

[3] Beachtet man z.B. die Groß-/Kleinschreibung nicht, kann das dazu führen, dass die Authentification nicht mehr mit dem Internet Explorer, wohl aber weiterhin mit dem Netscape Navigator funktioniert!

```
    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 '@'-Zeichen vor dem `mysql_connect` hat nichts mit der `if`-Abfrage zu tun. Es sorgt dafür, daß keine Fehlermeldung beim Aufrauf von `mysql_connect` ausgegeben wird. Die Fehlermeldung würde nicht nur stören, sondern sie würde auch die Paßwortabfrage zuverlässig 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 ubernehmen, 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 Authentification 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.

10. Fehlersuche

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 (innnerhalb 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';
```

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:

```
$text = 'Alternativtext';
echo "<img src=\"bild.gif\" width=\"10\" height=\"10\" alt=\" .$text. \">";
```

Oder warum bekommen wir hier eine Fehlermeldung:

```
$text = 'Alternativtext';
echo "<img src=\"bild.gif\" width=\"10\" height="10" alt=\"" .$text. "\">";
```

Ganz einfach: Das escaping 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:

```
$text = 'Alternativtext';
echo "<img src=\"bild.gif\" width=\"10\" height=\"10\" alt=\"" .$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. Dann sieht man nämlich schon häufig den Fehler. Wenn man ihn jedoch nicht findet, benutzt man einfach den String und gibt ihn direkt am SQL-Prompt 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.

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)!

10.1. 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.3.

10.1.1. 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";
}
?>
```

10.1.2. Fehler in einer leeren Zeile?

Die Zeilennummern sind hier nur der Übersichtlichkeit halber eingefügt worden. Sie stehen natürlich nicht in der PHP-Datei.

```
1: <?php
2: for ($i=1; $i<=5; $i++){
3: echo "$i<br>\n";
4: echo "Das waren die Zahlen 1-5";
5: ?>
6:
```

10.1.3. Wieso fehlt ein ; wo eins ist?

Wie im vorigen Beispiel sind die Zeilennummern wieder nur der Übersichtlichkeit halber eingefügt worden.

```
1: <?php
2: for ($i=1; $i<=5; $i++){
3:   echo "$i<br>\n;
4: }
5: echo "Das waren die Zahlen 1-5";
6: ?>
```

Teil IV.

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

Fahrbuch

MNr	PKWNr	Datum
1	1	13.5.2000
1	2	3.5.1998

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 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($result){
  // Tabellenanfang
  echo "<table>\n";
  // 1. Tabellenzeile Anfang
  echo "  <tr>\n";
  for ($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";
}
```

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.2. 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.3.

B.3.1. 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 Variable wurde der Wert „1“ zugewiesen. Anschließend wurde dieser Wert genommen, sodass 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, indem man statt `if ($i == 1)` ein `if (1 == $i)` schriebe. 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.3.2. 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.3.3. 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 auch 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].

[1] farbliche Hervorhebung in Abhängigkeit der Bedeutung. Z.B. Text in Anführungszeichen wird grün gefärbt, reservierte Wörter in dunkelrot.

C. Alle wichtigen SQL-Befehle

Dieser Anhang ist für alle Freaks gedacht, die immer noch nicht genug haben. Es soll nur eine Kurzübersicht über alle wichtigen SQL-Befehle in kompletter Syntax gegeben werden.

Wem diese Auflistung nicht reicht, dem empfehle ich die englische mySQL-Original-Dokumentation[1].

C.1. CREATE TABLE

```
CREATE TABLE tbl_name (create_definition,...)
[table_options] [select_statement]

create_definition:
  col_name type [NOT NULL | NULL] [DEFAULT default_value] [AUTO_INCREMENT]
            [PRIMARY KEY] [reference_definition]
  or    PRIMARY KEY (index_col_name,...)
  or    KEY [index_name] (index_col_name,...)
  or    INDEX [index_name] (index_col_name,...)
  or    UNIQUE [INDEX] [index_name] (index_col_name,...)
  or    [CONSTRAINT symbol] FOREIGN KEY index_name (index_col_name,...)
            [reference_definition]
  or    CHECK (expr)

type:
        TINYINT[(length)] [UNSIGNED] [ZEROFILL]
  or    SMALLINT[(length)] [UNSIGNED] [ZEROFILL]
  or    MEDIUMINT[(length)] [UNSIGNED] [ZEROFILL]
  or    INT[(length)] [UNSIGNED] [ZEROFILL]
  or    INTEGER[(length)] [UNSIGNED] [ZEROFILL]
  or    BIGINT[(length)] [UNSIGNED] [ZEROFILL]
  or    REAL[(length,decimals)] [UNSIGNED] [ZEROFILL]
  or    DOUBLE[(length,decimals)] [UNSIGNED] [ZEROFILL]
  or    FLOAT[(length,decimals)] [UNSIGNED] [ZEROFILL]
  or    DECIMAL(length,decimals) [UNSIGNED] [ZEROFILL]
  or    NUMERIC(length,decimals) [UNSIGNED] [ZEROFILL]
  or    CHAR(length) [BINARY]
  or    VARCHAR(length) [BINARY]
  or    DATE
  or    TIME
  or    TIMESTAMP
  or    DATETIME
  or    TINYBLOB
  or    BLOB
  or    MEDIUMBLOB
  or    LONGBLOB
  or    TINYTEXT
```

[1] http://www.mysql.com/

```
  or    TEXT
  or    MEDIUMTEXT
  or    LONGTEXT
  or    ENUM(value1,value2,value3,...)
  or    SET(value1,value2,value3,...)

index_col_name:
        col_name [(length)]

reference_definition:
        REFERENCES tbl_name [(index_col_name,...)]
                    [MATCH FULL | MATCH PARTIAL]
                    [ON DELETE reference_option]
                    [ON UPDATE reference_option]

reference_option:
        RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT

table_options:
        TYPE = {ISAM | MYISAM | HEAP}
or      AUTO_INCREMENT = #
or      AVG_ROW_LENGTH = #
or      CHECKSUM = {0 | 1}
or      COMMENT = "string"
or      MAX_ROWS = #
or      MIN_ROWS = #
or      PACK_KEYS = {0 | 1}
or      PASSWORD = "string"
or      DELAY_KEY_WRITE = {0 | 1}

select_statement:
        [IGNORE | REPLACE] SELECT ...  (Some legal select statement)
```

C.2. ALTER TABLE

```
ALTER [IGNORE] TABLE tbl_name alter_spec [, alter_spec ...]

alter_specification:
        ADD [COLUMN] create_definition [FIRST | AFTER column_name ]
  or    ADD INDEX [index_name] (index_col_name,...)
  or    ADD PRIMARY KEY (index_col_name,...)
  or    ADD UNIQUE [index_name] (index_col_name,...)
  or    ALTER [COLUMN] col_name {SET DEFAULT literal | DROP DEFAULT}
  or    CHANGE [COLUMN] old_col_name create_definition
  or    MODIFY [COLUMN] create_definition
  or    DROP [COLUMN] col_name
  or    DROP PRIMARY KEY
  or    DROP INDEX index_name
  or    RENAME [AS] new_tbl_name
  or    table_options
```

C.3. DROP TABLE

```
DROP TABLE [IF EXISTS] tbl_name [, tbl_name,...]
```

C.4. DELETE

```
DELETE [LOW_PRIORITY] FROM tbl_name
    [WHERE where_definition] [LIMIT rows]
```

C.5. SELECT

```
SELECT [STRAIGHT_JOIN] [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [HIGH_PRIORITY]
       [DISTINCT | DISTINCTROW | ALL]
    select_expression,...
    [INTO OUTFILE 'file_name' export_options]
    [FROM table_references
        [WHERE where_definition]
        [GROUP BY col_name,...]
        [HAVING where_definition]
        [ORDER BY {unsigned_integer | col_name | formula} [ASC | DESC] ,...]
        [LIMIT [offset,] rows]
        [PROCEDURE procedure_name] ]
```

C.6. JOIN Syntax

```
table_reference, table_reference
table_reference [CROSS] JOIN table_reference
table_reference INNER JOIN table_reference
table_reference STRAIGHT_JOIN table_reference
table_reference LEFT [OUTER] JOIN table_reference ON conditional_expr
table_reference LEFT [OUTER] JOIN table_reference USING (column_list)
table_reference NATURAL LEFT [OUTER] JOIN table_reference
```

C.7. INSERT

```
    INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
        [INTO] tbl_name [(col_name,...)]
        VALUES (expression,...),(...),...
or  INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
        [INTO] tbl_name [(col_name,...)]
        SELECT ...
or  INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
        [INTO] tbl_name
        SET col_name=expression, col_name=expression, ...
```

C.8. REPLACE

```
    REPLACE [LOW_PRIORITY | DELAYED]
        [INTO] tbl_name [(col_name,...)]
        VALUES (expression,...)
or  REPLACE [LOW_PRIORITY | DELAYED]
        [INTO] tbl_name [(col_name,...)]
        SELECT ...
or  REPLACE [LOW_PRIORITY | DELAYED]
        [INTO] tbl_name
        SET col_name=expression, col_name=expression,...
```

C.9. UPDATE

```
UPDATE [LOW_PRIORITY] tbl_name SET col_name1=expr1,col_name2=expr2,...
    [WHERE where_definition] [LIMIT #]
```

C.10. SHOW

```
   SHOW DATABASES [LIKE wild]
or SHOW TABLES [FROM db_name] [LIKE wild]
or SHOW COLUMNS FROM tbl_name [FROM db_name] [LIKE wild]
or SHOW INDEX FROM tbl_name [FROM db_name]
or SHOW STATUS
or SHOW VARIABLES [LIKE wild]
or SHOW PROCESSLIST
or SHOW TABLE STATUS [FROM db_name] [LIKE wild]
or SHOW GRANTS FOR user
```

C.11. DESCRIBE

```
{DESCRIBE | DESC} tbl_name {col_name | wild}
```

C.12. GRANT

```
GRANT priv_type [(column_list)] [, priv_type [(column_list)] ...]
    ON {tbl_name | * | *.* | db_name.*}
    TO user_name [IDENTIFIED BY 'password']
        [, user_name [IDENTIFIED BY 'password'] ...]
    [WITH GRANT OPTION]

For the GRANT and REVOKE statements, priv_type may be specified as
any of the following:

ALL PRIVILEGES      FILE                RELOAD
ALTER               INDEX               SELECT
CREATE              INSERT              SHUTDOWN
DELETE              PROCESS             UPDATE
DROP                REFERENCES          USAGE
```

C.13. REVOKE

```
REVOKE priv_type [(column_list)] [, priv_type [(column_list)] ...]
    ON {tbl_name | * | *.* | db_name.*}
    FROM user_name [, user_name ...]
```

C.14. Kommentare

```
mysql> select 1+1;     # This comment continues to the end of line
mysql> select 1+1;     -- This comment continues to the end of line
mysql> select 1 /* this is an in-line comment */ + 1;
mysql> select 1+
/*
this is a
multiple-line comment
```

```
*/
1;
```

D. Open Publication License

Draft v1.0, 8 June 1999

D.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 äs 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.

D.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.

E. Verzeichnisse

Literaturverzeichnis

[1] Brockhaus Enzyklopädie, Band 6

[2] Duden Fremdwörterbuch

[3] IBM-Lehrgangsunterlagen „DB2 Familie - Datenbankdesign“

[4] http://www.tcx.se/

[5] http://www.php3.de/

[6] http://www.teamone.de/selfhtml/

[7] http://www.koehntopp.de/php/

Abbildungsverzeichnis

Tabellenverzeichnis

F. 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, X, xEmacs und allen die ich vergessen 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 und Jens Hatlak, nicht vergessen, die dieses Dokument erst zu dem gemacht haben, was es jetzt hoffentlich ist. Ich hoffe, es kommen noch ein paar dazu.

Und letztendlich muß ich auch noch dem Übersetzungsdienst von AltaVista danken, mit dem ich am Ende einfach stellenweise die englische PHP-Anleitung ins Deutsche übersetzt habe (Nachbearbeitung war leider dennoch nötig).

Und wie es immer so schön heißt: Ich bedanke mich noch bei allen, die ich vergessen habe.

G. Versionen

G.1. 5.12.2000

- Übungen zum Kaptiel “Fehlersuche“ hinzugefügt
- ausführliche Erklärung der einzelnen Datentypen hinzugefügt
- kleine Detailverbesserungen

G.2. 27.11.2000

- Funktion DATE_FORMAT hinzugefügt
- kleine Fehler korrigiert

G.3. 19.10.2000

- Kapitel ‘PHP & HTTP‘ erweitert
- `print` erklärt
- Beispiel zur Erklärung der Tabellenstruktur hinzugefügt
- Erklärung zu Funktionen etwas überarbeitet

G.4. 30.9.2000

- 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

G.5. 27.7.2000

- Grundzustand

Index