10.5.1 Eine Bedingung

Zum Formulieren von Bedingungen können neben einigen Funktionen wie LIKE, BETWEEN oder IN (siehe Kapitel 10.5.6) folgende Operatoren verwendet werden.

=	gleich
<	kleiner
>	größer
<>	ungleich
<=	kleiner oder gleich
>=	größer oder gleich

Beispiele für Abfragen mit einer Bedingung

Zeige die Datensätze aller Mitarbeiter an, die den Nachnamen 'Klein' haben.

```
SELECT *
FROM   Mitarbeiter
WHERE  Nachname = 'Klein';
```

```
Ausgabe:
1, Eva,   Klein, 1995-01-13, 2000, w, 2
4, Ernst, Klein, 1988-02-02, 1000, m, 3
```

Welcher Mitarbeiternachname beginnt mit einem Buchstaben von 'A' bis 'H' ?

```
SELECT Nachname
FROM   Mitarbeiter
WHERE  Nachname < 'I';
```

```
Ausgabe:  Ax
          Blei
```

Welcher Mitarbeiter verdient mehr als 2000,- € oder genau 2000,- € ?

```
SELECT Vorname, Nachname
FROM   Mitarbeiter
WHERE  Gehalt >= 2000;
```

```
Ausgabe:  Eva, Klein
          Kai, Blei
```

Welcher Mitarbeiter hat am 28.10.1967 Geburtstag?

```
SELECT Vorname, Nachname
FROM   Mitarbeiter
WHERE  Geburtsdatum = '1967-10-28';
```

```
Ausgabe:  Kai, Blei
```

Für welchen Mitarbeiter wurde kein Geburtsdatum angegeben?

```
SELECT Vorname, Nachname
FROM   Mitarbeiter
WHERE  Geburtsdatum = NULL;
```

```
Ausgabe:  Udo, Ax
```

10.5.2 Mehrere Bedingungen

Mehrere Bedingungen können im WHERE-Teil mit den folgenden logischen Operationen kombiniert bzw. negiert werden.

AND	Und-Verknüpfung
OR	Oder-Verknüpfung
NOT	Negation

Dabei wird erst das NOT dann das AND und zum Schluss das OR ausgewertet. Wird eine andere Auswertungsreihenfolge benötigt, müssen Klammern gesetzt werden.

Beispiele für Abfragen mit mehreren Bedingungen

Zeige die Datensätze aller Mitarbeiter, die den Vornamen 'Eva' und den Nachnamen 'Klein' besitzen.

```
SELECT *
FROM   Mitarbeiter
WHERE  Nachname = 'Klein'
  AND  Vorname  = 'Eva';
```

```
Ausgabe:
1, Eva,   Klein, 1995-01-13, 2000, w, 2
```

Welche Mitarbeiter arbeiten in der Abteilung mit ID '2' und verdienen mehr als 1800,- € ?

```
SELECT Vorname, Nachname, Gehalt
FROM   Mitarbeiter
WHERE  ID_Abteilung = 2
  AND  Gehalt > 1800;
```

```
Ausgabe:  Eva, Klein, 2000
```

Welche Mitarbeiter arbeiten in der Abteilung mit ID '2' und verdienen NICHT mehr als 1800,- € ?

```
SELECT     Vorname, Nachname, Gehalt
FROM       Mitarbeiter
WHERE      ID_Abteilung = 2
  AND NOT  Gehalt > 1800;
```

```
SELECT     Vorname, Nachname, Gehalt
FROM       Mitarbeiter
WHERE      ID_Abteilung = 2
  AND      Gehalt <= 1800;
```

```
Ausgabe:  Udo, Ax, 1000
```

Welche männlichen Mitarbeiter verdienen mehr als 2000,- € oder arbeiten in der Abteilung mit ID '3'?

```
SELECT  Vorname, Nachname,
        Gehalt, Geschlecht
FROM    Mitarbeiter
WHERE   Geschlecht = 'm'
  AND   (  ID_Abteilung = 3
           OR Gehalt > 2000  );
```

```
Ausgabe:  Kai, Blei, 3500, m
          Ernst, Klein, 1000, m
```

Beispiel Mitarbeiter

Mitarbeiter

ID_Mitarbeiter	Vorname	Nachname	Geburtsdatum	Gehalt	Geschlecht	ID_Abteilung
1	Eva	Klein	1995-01-13	2000	w	2
2	Kai	Blei	1967-10-28	3500	m	1
3	Udo	Ax		1500	m	2
4	Ernst	Klein	1988-02-02	1000	m	3

10.5 Abfragen auf einer Tabelle

Wenn die Tabellen der Datenbank mit Werten gefüllt sind, können mit Hilfe der SELECT-Anweisung Abfragen an die Datenbank gestellt werden. Der SELECT- Befehl ist der wichtigste und umfangreichste SQL-Befehl überhaupt. Deshalb werden zunächst, anhand von Beispielen, die Möglichkeiten von SELECT-Anweisungen aufgezeigt, die lediglich auf der obigen gezeigten Tabelle 'Mitarbeiter' basieren.

Aufbau einer Abfrage

Eine Abfrage bzw. eine SELECT-Anweisung besteht im Allgemeinen aus folgenden drei Bereichen.

- SELECT
  Der erste Teil wird durch das Schlüsselwort SELECT eingeleitet. Hinter dem Schlüsselwort werden alle Merkmale der Tabelle durch Kommata getrennt angegeben, die als Ergebnismenge ausgegeben werden sollen. Das Sonderzeichen * sollte verwendet werden, wenn der gesamte Datensatz, also alle Merkmale der Tabelle in der Ergebnismenge dargestellt werden sollen.

  Beispiel
  Wenn bei einer Abfrage nur der Vor- und Nachname eines Mitarbeiters interessiert, alle anderen Merkmale des Mitarbeiters aber völlig uninteressant sind, würde man den SELECT-Teil der Abfrage so formulieren:

```
SELECT Vorname, Nachname
```

  Sind hingegen alle Merkmale der Datensätze von Interesse wird die SELECT Anweisung wie folgt formuliert:

```
SELECT *
```

- FROM
  Der zweite Teil wird durch das Schlüsselwort FROM eingeleitet und spezifiziert die Datenquelle, aus der die Daten für die Abfrage entnommen werden. Hier können die Namen von einer aber (wie später noch gezeigt wird) auch von mehreren Tabellen stehen.

  Beispiel
  Im betrachteten Fall ist die Datenquelle einzig und allein die Tabelle 'Mitarbeiter'. Daher lautet der FROM-Teil der Abfrage:

```
FROM Mitarbeiter
```

- WHERE
  Der dritte und vorerst letzte Teil einer Abfrage beginnt mit dem Schlüsselwort WHERE. Hier können nun eine oder auch mehrere Bedingungen formuliert werden, die von den Datensätzen der Ergebnismenge erfüllt werden müssen.

  Beispiel
  Sollen nur alle männlichen Mitarbeiter angezeigt werden, kann die SQL-Anweisung um den folgenden WHERE-Teil ergänzt werden.

```
WHERE Geschlecht = 'm'
```

Zeige mir den Vor- und Nachnamen aller männlichen Mitarbeiter an.

```
SELECT Vorname, Nachname
FROM   Mitarbeiter
WHERE  Geschlecht = 'm';
```

```
Ausgabe:  Kai,   Blei
          Udo,   Ax
          Ernst, Klein
```

Zusammenfassung

Schlüsselwort	Frage	Beschreibung
SELECT	Was?	Gibt an, was (welche Merkmale) ausgegeben werden soll.
FROM	Woher?	Bestimmt die Datenquelle, aus der die Daten stammen.
WHERE	Wobei? Womit?	Legt Bedingungen fest, die erfüllt werden müssen.

10.5.3 Aggregationsfunktionen

Mit Hilfe von Aggregationsfunktionen können alle Datensätze einer Ergebnismenge zu einem einzigen Datensatz zusammengefasst werden.
Beispielsweise liefert die Abfrage

```
SELECT  Gehalt  FROM Mitarbeiter;
```

Ausgabe: 2000 3500 1500 1000

eine Auflistung der Gehälter sämtlicher Mitarbeiter.
Unter Verwendung der Aggregationsfunktion AVG

```
SELECT  AVG(Gehalt)  FROM Mitarbeiter;
```

Ausgabe: 2000

wird hingegen nur ein einziger Datenwert, nämlich das Durchschnittsgehalt aller Mitarbeiter, ermittelt.
Abfragen können auch mehrere Merkmale betreffen. Wenn dabei Aggregationsfunktionen verwendet werden, muss für jedes Merkmal separat eine Aggregationsfunktion bestimmt werden.

```
SELECT MAX(Vorname), MAX(Nachname)
FROM   Mitarbeiter;
```

Ausgabe: Udo, Klein

Es gibt folgende Aggregationsfunktionen

MIN (Merkmal)	Minimalwert
MAX (Merkmal)	Maximalwert
COUNT (*)	Anzahl der Zeilen in der Ergebnismenge
COUNT (Merkmal)	Anzahl der Zeilen, die bei dem Merkmal nicht den Wert NULL (für undefiniert) haben
COUNT (DISTINCT Merkmal)	Anzahl der Zeilen mit unterschiedlichen Werten
SUM (Merkmal)	Summenwert
AVG (Merkmal)	Durchschnittswert

Aliasnamen (Merkmale bzw. Spalten)
Gerade bei der Verwendung von Aggregationsfunktionen und Rechenoperationen (siehe Kapitel 10.5.4) im SELECT-Teil einer SQL-Abfrage entstehen Spalten mit neuen bzw. veränderten Bedeutungen. Diesen Spalten kann mit Hilfe des Schlüsselwortes AS im SELECT-Teil einer SQL-Abfrage eine sinnvolle Spaltenüberschrift gegeben werden. Hierzu folgende zwei Beispiele

```
SELECT Gehalt *1.045 AS Neugehalt ...
```

Diese SELECT Anweisung wird nur die um 4,5% erhöhten Gehälter in der Spalte 'Neugehalt' anzeigen.

```
SELECT AVG(Gehalt) AS Durchschnitt ...
```

Diese SELECT-Anweisung wird nur das Durchschnittsgehalt in der Spalte 'Durchschnitt' anzeigen.

Beispiele für die Verwendung von Aggregationsfunktionen

Ermittle das Durchschnittsgehalt aller Mitarbeiter.

```
SELECT AVG(Gehalt)  AS Durchschnitt
FROM   Mitarbeiter;
```

Ausgabe: Durchschnitt 2000

Welcher Mitarbeiter hat den alphabetisch letzten Namen?

```
SELECT MAX(Nachname) AS Letzter
FROM   Mitarbeiter;
```

Ausgabe: Letzter Klein

Zeige zusätzlich zu dem Nachnamen auch den bzw. die passenden Vornamen an.

```
SELECT  Nachname, Vorname
FROM    Mitarbeiter
WHERE   Nachname =
        (  SELECT  MAX(Nachname)
           FROM    Mitarbeiter   );
```

Ausgabe: Klein

Ausgabe: Klein, Eva
Klein,Ernst
Hinweis: Der Maximale Nachname in der Tabelle Mitarbeiter, ist der alphabetisch letzte Nachname „Klein". Daher liefert die Unterabfrage diesen Nachnamen zurück.

Unter
siehe K

Ermittle das Geburtsdatum des **ältesten** Mitarbeiters.

```
SELECT MIN(Geburtsdatum)  AS Ältester
FROM   Mitarbeiter;
```

Ausgabe: Ältester 1967-10-28

Wie viele Mitarbeiter gibt es?

```
SELECT COUNT(*)  AS Anzahl
FROM   Mitarbeiter;
```

Ausgabe: Anzahl 4

Wie viele unterschiedliche Nachnamen haben die Mitarbeiter?

```
SELECT COUNT(DISTINCT Nachname)
FROM   Mitarbeiter;
```

Ausgabe: 3

Wie viele Mitarbeiter gibt es und wie hoch ist das Durchschnittseinkommen?

```
SELECT COUNT(*), AVG(Gehalt)
FROM   Mitarbeiter;
```

Ausgabe: 4, 2000

Zeige den alphabetisch ersten Vornamen und den alphabetisch ersten Nachnamen aller männlichen Mitarbeiter an?

```
SELECT MIN(Vorname), MIN(Nachname)
FROM   Mitarbeiter
WHERE  Geschlecht = 'm';
```

Ausgabe: Ernst, Ax

10.5.4 Einfache Rechenoperationen

Im SELECT- aber auch im WHERE-Teil einer Abfrage können einfache Rechenoperationen durchgeführt werden.

+	plus
*	mal
-	minus
/	geteilt

Dezimalzahlen werden mit einem Dezimalpunkt anstelle des Kommas geschrieben. Beispielsweise 3.14 anstelle von 3,14.

Beispiele für die Verwendung von Rechenoperationen

Wie viel ist 4,9 * 1,9?

```
SELECT 4.9 * 1.9  AS Berechnung;
```

```
Ausgabe:  Berechnung 9.31
```

Wie hoch wäre das Gehalt der Mitarbeiter nach einer 4,5%-igen Gehaltserhöhung?

```
SELECT  Vorname, Nachname,
        Gehalt * 1.045 AS Neugehalt
FROM    Mitarbeiter;
```

```
Ausgabe:  Eva,    Klein,  Neugehalt 2090
          Kai,    Blei,   Neugehalt 3657.5
          Udo,    Ax,     Neugehalt 1567.5
          Ernst,  Klein,  Neugehalt 1045
```

Welche Mitarbeiter werden nach einer 5%-igen Gehaltserhöhung mehr als 2050,- € verdienen? Zeige neben dem Mitarbeiternamen auch das veränderte Gehalt an.

```
SELECT  Vorname, Nachname, Gehalt * 1.05
FROM    Mitarbeiter
WHERE   Gehalt * 1.05 > 2050;
```

```
Ausgabe:  Eva, Klein,  2100
          Kai, Blei,   3675
```

Ermittle das Durchschnittsgehalt der Mitarbeiter (ohne die Verwendung der Aggregationsfunktion AVG).

```
SELECT SUM (Gehalt) / COUNT (*)
       AS  Durchschnitt
FROM   Mitarbeiter;
```

```
Ausgabe:  Durchschnitt 2000
```

Ermittle die Differenz zwischen dem maximalen und dem minimalen Gehalt der Mitarbeiter.

```
SELECT MAX(Gehalt) - MIN(Gehalt)
       AS  Differenz
FROM   Mitarbeiter;
```

```
Ausgabe:  Differenz 2500
```

10.5.5 Datumsfunktionen

Es werden unterschiedliche Datentypen für Datums- und Zeitangaben verwendet (vergleiche Kapitel 10.2.2 Datentypen). Mit folgenden Funktionen können aus einzelnen Integervariablen Datums- und Zeitvariablen zusammengebaut werden.

Datums- und Zeitformate aus Einzelwerten

MAKEDATE(year, day-of-year)	
year	ist die Jahresangabe als Integer.
day-of-year	ist der Tag des Jahres als Integer, generiert das zugehörige Datumsformat. (YYYY-MM-DD)
MAKETIME(hour, minute, second)	
hour	Angabe der Stunde als Integer
minute	Angabe der Minute als Integer
second	Angabe der Sekunde als Integer, generiert das zugehörige Zeitformat. (HH:MM:SS)

Die aktuellen Datums- bzw. Zeitinformationen können wie folgt ermittelt werden:

Aktuelles Datum/Zeit

NOW()	Ermittelt die aktuelle Datums- und Zeitangabe.
	YYYY-MM-DD HH:MM:SS
CURDATE()	Ermittelt das aktuelle Datum.
	YYYY-MM-DD
CURTIME()	Ermittelt die aktuelle Zeit.
	HH:MM:SS

Mit den folgenden Funktionen können aus diesen Datumsformaten Teilinformationen extrahiert werden.

Teilinformationen extrahieren

DATE(timestamp)	Ermittelt das Datum aus einer Timestamp-Variablen.
TIME(timestamp)	Bestimmt die Zeit aus einer Timestamp-Variablen.
YEAR(date)	Ermittelt das Jahr aus einem Datum.
MONTH(date)	Ermittelt den Monat aus einem Datum.
DAY(date)	Ermittelt den Tag aus einem Datum.
HOUR(time)	Ermittelt die Stunden aus einer Zeitangabe.
MINUTE(time)	Ermittelt die Minuten aus einer Zeitangabe.
SECOND(time)	Ermittelt die Sekunden aus einer Zeitangabe.

Weitere Informationen aus einer Datumsangabe kann man mit folgenden Funktionen gewinnen.

Datumszusatzinformationen

MONTHNAME(date)	Ermittelt den Namen des Monats.
DAYNAME(date)	Ermittelt den Namen des Wochentages.
DAYOFWEEK(date)	Ermittelt den Wochentag. 1=Sonntag, 2 = Montag, ...
QUARTER(date)	Ermittelt das Quartal.
WEEKOFYEAR(date)	Ermittelt die Kalenderwoche.

Mit den unterschiedlichen Zeit- und Datumsformaten kann auch gerechnet werden.

Rechenfunktionen

date + INTERVAL 1 DAY	Einen Tag auf das Datum addieren.
date + INTERVAL 2 MONTH	Zwei Monate auf das Datum addieren.
date - INTERVAL 3 YEAR	Drei Jahre vom Datum abziehen.
time - INTERVAL 1 HOUR	Eine Stunde von der akt. Zeit abziehen.
time + INTERVAL 3 MINUTES	Drei Minuten auf die aktuelle Zeit addieren.
time - INTERVAL 5 SECONDS	Fünf Sekunden von der akt. Zeit abziehen.
DATEDIFF(date1, date2)	Gibt die Anzahl der Tage zwischen den beiden Datumsangaben zurück.
TIMEDIFF(time1, time2)	Ermittelt den Zeitraum zwischen den beiden Zeitangaben und liefert den Datentyp TIME zurück.

Anmerkung

Diese Auflistung der wichtigsten Datumsfunktionen ist keinesfalls vollständig. MySQL unterstützt noch viele weitere Datumsfunktionen, gerade auch was das Formatieren der Datums- und Zeitausgabe betrifft.

Aber Achtung! Bei diesen Funktionen weichen die unterschiedlichen Hersteller sehr stark voneinander ab. Die meisten hier vorgestellten Funktionen finden sich auch bei anderen Anbietern. Diese können aber evtl. anders heißen oder eine andere Syntax verlangen.

Beispiele für die Verwendung von Datumsfunktionen

Ziehe vom Datum '1.3.2001' zwei Tage ab.

```
SELECT '2001-03-01' - INTERVAL 2 DAY;
```

```
Ausgabe:  2001-02-27
```

Wie viele Tage liegen zwischen dem 3.3.2009 und dem 25.2.2009?

```
SELECT
DATEDIFF('2009-03-03','2009-02-25');
```

```
Ausgabe:  6
```

Wie viele Mitarbeiter haben am 13. eines Monats Geburtstag?

```
SELECT  COUNT(*)
FROM    Mitarbeiter
WHERE   DAY(Geburtsdatum) = 13;
```

```
Ausgabe:  1
```

Welche Mitarbeiter haben im Oktober Geburtstag?

```
SELECT  Vorname, Nachname
FROM    Mitarbeiter
WHERE   MONTH(Geburtsdatum) = 10;
```

```
Ausgabe:  Kai, Blei
```

Welche Mitarbeiter wurden vor dem Jahr 1990 geboren?

```
SELECT  Vorname, Nachname
FROM    Mitarbeiter
WHERE   YEAR(Geburtsdatum) < 1990;
```

```
Ausgabe:  Kai,    Blei
          Ernst,  Klein
```

An welchen Wochentagen haben die Mitarbeiter mit Nachnamen 'Klein' Geburtstag?

```
SELECT  Vorname, DAYNAME (Geburtsdatum)
FROM    Mitarbeiter
WHERE   Nachname = 'Klein';
```

```
Ausgabe:  Eva,    Friday
          Ernst,  Tuesday
```

```
SELECT Vorname, DAYOFWEEK(Geburtsdatum)
FROM   Mitarbeiter
WHERE  Nachname = 'Klein';
```

```
Ausgabe:  Eva, 6
          Ernst, 3
```

Welcher Mitarbeiter ist sieben Jahre älter als 'Eva'?

```
SELECT  Vorname, Nachname
FROM    Mitarbeiter
WHERE   YEAR(Geburtsdatum) =
        ( SELECT YEAR(Geburtsdatum)-7
          FROM   Mitarbeiter
          WHERE  Vorname = 'Eva'  );
```

Ausgabe: 1988

```
Ausgabe:  Ernst, Klein
```

Unteral
siehe Ka

10.5.6 Vergleichsfunktionen

Als einfache String-Vergleichsfunktion steht die Funktion LIKE zur Verfügung. Zudem können mit den Anweisungen IN und BETWEEN Mengen- bzw. Suchbereiche definiert werden, die das Formulieren von WHERE-Anweisungen deutlich vereinfachen.

LIKE

Beim LIKE-Befehl können mit Hilfe von Jokerzeichen Vergleichsmuster erstellt werden, die dann in dem WHERE-Teil einer Abfrage verwendet werden können.

Jokerzeichen

%	Steht für beliebig viele beliebige Zeichen
_ (Unterstrich)	Steht für genau ein beliebiges Zeichen

Anmerkung

Bei MS-ACCESS werden die vom Betriebssystem her bekannten Jokerzeichen ? (ein beliebiges Zeichen) und * (beliebig viele beliebige Zeichen) anstelle der in SQL gebräuchlichen Jokerzeichen verwendet!

Nenne alle Mitarbeiter, deren Nachnamen mit einem 'K' beginnen.

```
SELECT  Nachname, Vorname
FROM    Mitarbeiter
WHERE   Nachname LIKE 'K%';
```

```
Ausgabe:  Klein, Eva
          Klein, Ernst
```

Suche alle Mitarbeiternachnamen, die zwischen 'Me' und 'er' genau ein beliebiges Zeichen haben.

```
SELECT  Nachname
FROM    Mitarbeiter
WHERE   Nachname LIKE 'Me_er';
```

```
Ausgabe:  -
Achtung:  Keine Ausgabe, da in der Bei-
          spieltabelle keine Meier oder
          Meyer als Nachnamen vorkommen.
```

Nenne alle Mitarbeiter, deren Nachnamen als drittes und viertes Zeichen ein 'ei' enthält.

```
SELECT  Nachname, Vorname
FROM    Mitarbeiter
WHERE   Nachname LIKE '__ei%';
```

```
Ausgabe:  Klein, Eva
          Blei,  Kai
          Klein, Ernst
```

BETWEEN

Durch die Anweisung BETWEEN wird ein zusammenhängender Bereich mittels eines Start- und eines Endewertes bestimmt. Dabei gehören die Start- und Endewerte mit zum betrachteten Bereich.

Welcher Mitarbeiternachname beginnt mit einem Buchstaben zwischen 'A' und 'D'?

```
SELECT  Nachname
FROM    Mitarbeiter
WHERE   Nachname BETWEEN 'A' AND 'D';
```

```
Ausgabe:  Ax
          Blei
Achtung:  Ein Nachname wie etwa 'Daume'
          würde nicht ausgegeben, da
          'Daume' hinter D und somit
          nicht zwischen A und D liegt.
```

Welche Mitarbeiter beziehen ein Gehalt zwischen 1500,- € und 2500,- € ?

```
SELECT  Vorname, Nachname, Gehalt
FROM    Mitarbeiter
WHERE   Gehalt BETWEEN 1500 AND 2500;
```

```
Ausgabe:  Eva, Klein, 2000
          Udo, Ax, 1500

Achtung:  Das Gehalt 1500 € gehört auch
          zum betrachteten Bereich.
```

IN

Die IN-Anweisung ermöglicht das Zusammenstellen beliebiger Vergleichsmengen.

Welche Mitarbeiter wurden in den Jahren 1967, 1988 und 1990 geboren?

```
SELECT  Vorname, Nachname, Geburtsdatum
FROM    Mitarbeiter
WHERE   YEAR(Geburtsdatum)
        IN (1988, 1967, 1990);
```

```
Ausgabe:  Kai,   Blei,  1967-10-28
          Ernst, Klein, 1988-02-02
```

Die Rei folge spiel Rolle

Welche Mitarbeiter arbeiten in den Abteilungen mit ID '3' oder ID '1' ?

```
SELECT  Vorname, Nachname, ID_Abteilung
FROM    Mitarbeiter
WHERE   ID_Abteilung IN (1, 3);
```

```
Ausgabe:  Kai,   Blei,  1
          Ernst, Klein, 3
```

Anmerkung

Die IN-Anweisung ersetzt mehrere OR-Anweisungen.

```
WHERE    ID_Abteilung  IN  (1, 3)
```

kann beispielsweise auch durch

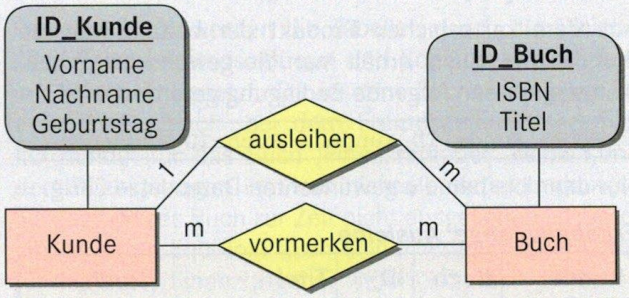
```
WHERE    ID_Abteilung = 1
 OR      ID_Abteilung = 3
```

dargestellt werden.

Beispiel Bücherei

Kunde

ID_Kunde	Vorname	Nachname	Geburtstag
1	Eva	Klein	1995-01-13
2	Kai	Blei	1967-10-28
3	Udo	Ax	
4	Ernst	Block	1988-02-02



Buch

ID_Buch	ISBN	Titel	ID_Kunde
1	1-23456-001	Leichter lachen	2
2	4-98765-002	Datenbanken	3
3	6-55444-006	Finanzratgeber	
4	7-14324-008	Haus und Garten	4

vormerken

ID_Kunde	ID_Buch
1	2
4	1
4	2

Der Fremdschlüssel 'ID_Kunde' realisiert die Beziehung 'ausleihen'.

10.6 Abfragen auf mehreren Tabellen

Bei Abfragen, die Informationen aus mehreren Tabellen kombinieren, müssen alle an der Abfrage beteiligten Tabellen im FROM-Teil der Abfrage aufgeführt werden. Um bei Namensgleichheit einzelner Merkmale in unterschiedlichen Tabellen die Eindeutigkeit zu garantieren, kann dem Merkmalsnamen der Tabellenname vorangestellt werden. Um beispielsweise alle Kunden, die sich Bücher ausgeliehen haben und die Buchtitel der von ihnen ausgeliehenen Bücher anzuzeigen, könnte der SELECT- und der FROM-Teil dieser Abfrage wie folgt aussehen:

Welcher Kunde hat welche Bücher ausgeliehen?

```
SELECT  Kunde.ID_Kunde, Nachname
        ID_Buch, Titel, Buch.ID_Kunde
FROM    Kunde, Buch;
```

Anmerkung

Da das Merkmal 'ID_Kunde' in beiden Tabellen vorkommt, muss der Tabellenname vorangestellt werden, damit das gewünschte Merkmal eindeutig identifiziert werden kann.

Diese Abfrage spiegelt die eigentliche Beziehung, nämlich die Gleichheit des Fremdschlüssels 'ID_Kunde' der Tabelle 'Buch' mit dem Schlüssel 'ID_Kunde' in der Tabelle 'Kunde' noch nicht wieder.
Würde man diese Abfrage ohne weitere Einschränkung ausführen, also ohne die Angabe eines WHERE-Teils, so würde die Ergebnismenge aus dem kartesischen Produkt[1] zwischen den beiden Tabellen 'Kunde' und 'Buch' gebildet. Das wiederum heißt, dass jeder Datensatz der einen mit jedem Datensatz der anderen Tabelle kombiniert würde. Die Ergebnismenge sähe dann wie folgt aus:

Karthesisches Produkt der Tabellen 'Buch' und 'Kunde'

Kunde. ID_Kunde	Nach-name	ID_ Buch	Titel	Buch. ID_Kunde
1	Klein	1	Leichter lachen	2
1	Klein	2	Datenbanken	3
1	Klein	3	Finanzratgeber	
2	Blei	1	Leichter lachen	2
2	Blei	2	Datenbanken	3
2	Blei	3	Finanzratgeber	
3	Ax	1	Leichter lachen	2
3	Ax	2	Datenbanken	3
3	Ax	3	Finanzratgeber	
4	Block	1	Leichter lachen	2
4	Block	2	Datenbanken	3
4	Block	3	Finanzratgeber	

Anmerkung

Neben den beiden erwünschten Ergebnisdatensätzen (hier rot hinterlegt), die tatsächlich widerspiegeln welcher Kunde welches Buch ausgeliehen hat, werden auch viele unerwünschte Ergebnisdatensätze gebildet.

1 Das kartesische Produkt zweier Mengen enthält alle möglichen Kombinationen von jedem Element der einen Menge mit jedem Element der anderen Menge.

10.6.1 Equi-Join

Aus dem kartesischen Produkt der beiden Tabellen 'Kunde' und 'Buch' erhält man die gewünschte Ergebnismenge, wenn folgende Bedingung gefordert wird.

```
WHERE Kunde.ID_Kunde = Buch.ID_Kunde
```

Nur dann bleiben die gewünschten Datensätze übrig.

Ergebnismenge: Ausleihe

Kunde. ID_Kunde	Nach- name	ID_ Buch	Titel	Buch. ID_Kunde
2	Blei	1	Leichter lachen	2
3	Ax	2	Datenbanken	3

Anmerkung

Jetzt zahlt es sich aus, für zusammengehörige Schlüssel und Fremdschlüsselmerkmale in unterschiedlichen Tabellen dieselben Namen verwendet zu haben. Dadurch wird das Formulieren der Join-Bedingung deutlich einfacher.

Somit führt die Gleichsetzung der Merkmale 'ID_Kunde' in den beiden Tabellen bzw. die Gleichsetzung von Schlüssel und Fremdschlüssel zu dem gewünschten Ergebnis. Das Kombinieren bzw. Verknüpfen von unterschiedlichen Tabellen in einer Abfrage wird allgemein als **Join** bezeichnet. Da in diesem Fall der Join über die Gleichsetzung von Schlüssel und Fremdschlüsselmerkmal erfolgt, wird dieser Join-Typ auch als **Gleichheits-Join** oder **Equi-Join** bezeichnet. Dieser Join-Typ lässt sich auch als **Inner-Join** formulieren.

Aliasnamen (Tabellen)

Damit nicht immer der ganze Tabellenname bei Namensgleichheit den Merkmalen vorangestellt werden muss, gibt es die Möglichkeit im FROM-Teil der Abfrage Aliasnamen für die Tabellen zu vereinbaren.

```
FROM  Buch B, Kunde K
```

Somit führen die folgenden beiden SQL-Anweisungen zu demselben korrekten Ergebnis.

Suche zu allen Kunden, die Bücher ausgeliehen haben, die von ihnen ausgeliehenen Buchtitel heraus.

Ohne Aliasnamen

```
SELECT  Kunde.Nachname, Buch.Titel
FROM    Buch, Kunde
WHERE   Buch.ID_Kunde = Kunde.ID_Kunde;
```

Mit Aliasnamen

```
SELECT  K.Nachname, B.Titel
FROM    Buch B, Kunde K
WHERE   B.ID_Kunde = K.ID_Kunde;
```

```
Ausgabe:  Blei, Leichter lachen
          Ax, Datenbanken
          Block, Haus und Garten
```

Beispiele für die Verwendung von Equi- bzw. Gleichheits-Joins

Wie heißt der Kunde, der das Buch mit dem Titel 'Datenbanken' ausgeliehen hat?

```
SELECT  K.Vorname, K.Nachname
FROM    Buch B, Kunde K
WHERE   B.ID_Kunde = K.ID_Kunde
 AND    B.Titel = 'Datenbanken';
```

```
Ausgabe:  Ax, Datenbanken
```

Wie viele Kunden haben Bücher ausgeliehen?

```
SELECT  COUNT (*)
FROM    Buch B, Kunde K
WHERE   B.ID_Kunde = K.ID_Kunde;
```

```
Ausgabe:  3
Hinweis:  Die Aggregationsfunktion
          COUNT(*) zählt alle Datensätze
          der Ergebnismenge.
```

Equi-Join zwischen mehr als zwei Tabellen

Dieses Prinzip der Gleichsetzung von Schlüssel- und Fremdschlüsselmerkmalen funktioniert auch, wenn mehr als zwei Tabellen an der SQL-Anweisung beteiligt sind.

Welcher Kunde hat welches Buch vorgemerkt?

```
SELECT  K.Vorname, K.Nachname, B.Titel
FROM    Kunde K, vormerken v, Buch B
WHERE   B.ID_Buch = v.ID_Buch
 AND    v.ID_Kunde = K.ID_Kunde;
```

```
Ausgabe:  Eva, Klein, Datenbanken
          Ernst, Block, Leichter lachen
          Ernst, Block, Datenbanken
```

Wie heißen die Kunden, die sich das Buch mit dem Titel 'Datenbanken' vorgemerkt haben?

```
SELECT  K.Vorname, K.Nachname
FROM    Kunde K, vormerken v, Buch B
WHERE   B.ID_Buch = v.ID_Buch
 AND    v.ID_Kunde = K.ID_Kunde
 AND    B.Titel = 'Datenbanken';
```

```
Ausgabe:  Eva,   Klein
          Ernst, Block
```

Wie viele unterschiedliche Kunden haben Bücher vorgemerkt?

```
SELECT  COUNT (DISTINCT (K.ID_Kunde))
FROM    Kunde K, vormerken v, Buch B
WHERE   B.ID_Buch = v.ID_Buch
 AND    v.ID_Kunde = K.ID_Kunde;
```

```
Ausgabe:  2
Hinweis:  Durch DISTINCT kann sicher-
          gestellt werden, dass die
          Aggregationsfunktion COUNT nur
          unterschiedliche ID-Werte zählt.
```

Inner-Join

Der Equi-Join lässt sich auch mit dem Schlüsselwort INNER JOIN formulieren. Dann allerdings wird die Join-Bedingung mit dem Schlüsselwort ON angegeben. Weitere Bedingungen können weiterhin wie bisher als WHERE-Bedingungen formuliert werden.

Suche zu allen Kunden, die Bücher ausgeliehen haben, die von ihnen ausgeliehenen Buchtitel heraus.

Mit Inner-Join

```
SELECT  K.Nachname, B.Titel
FROM    Buch B INNER JOIN Kunde K
ON      B.ID_Kunde = K.ID_Kunde;
```

Ohne Inner-Join

```
SELECT  K.Nachname, B.Titel
FROM    Buch B, Kunde K
WHERE   B.ID_Kunde = K.ID_Kunde;
```

```
Ausgabe:  Blei, Leichter lachen
          Ax, Datenbanken
          Block, Haus und Garten
```

Wie heißen die Kunden, die sich das Buch mit dem Titel 'Datenbanken' vorgemerkt haben?

Mit Inner-Join (verschachtelt)

```
SELECT  K.Vorname, K.Nachname
FROM    ( Kunde K INNER JOIN vormerken v
          ON v.ID_Kunde = K.ID_Kunde )
  INNER JOIN Buch B
        ON    B.ID_Buch = v.ID_Buch
WHERE   B.Titel = 'Datenbanken';
```

Mit Inner-Join (alternative Schreibweise)

```
SELECT  K.Vorname, K.Nachname
FROM    Kunde K
 INNER JOIN
        (vormerken v, Buch B)
  ON    (  B.ID_Buch = v.ID_Buch
    AND    v.ID_Kunde = K.ID_Kunde  )
WHERE    B.Titel = 'Datenbanken';
```

Ohne Inner-Join

```
SELECT  K.Vorname, K.Nachname
FROM    Kunde K, vormerken v, Buch B
WHERE   B.ID_Buch = v.ID_Buch
 AND    v.ID_Kunde = K.ID_Kunde
 AND    B.Titel = 'Datenbanken';
```

```
Ausgabe:  Eva,    Klein
          Ernst, Block
```

Anmerkung

MySQL unterstützt noch weitere Schreibweisen, die zur Formulierung von Equi-Join verwendet werden können.

10.6.2 Left- und Right-Join

Beim Equi-Join werden die Tabellen durch die Gleichsetzung von Fremdschlüssel und Schlüsselmerkmalen in Beziehung gesetzt. Da in dem Büchereibeispiel in der Tabelle 'Buch' bei dem Fremdschlüsselmerkmal 'ID_Kunde' für das Buch 'Finanzratgeber' keine Kunden-ID eingetragen ist, was nichts anderes bedeutet, als dass dieses Buch zur Zeit nicht ausgeliehen ist, wird der folgende Equi-Join auch keinen Ergebnisdatensatz für das Buch 'Finanzratgeber' liefern können.

Suche zu allen Kunden, die Bücher ausgeliehen haben, die von ihnen ausgeliehenen Buchtitel heraus.

```
SELECT  B.Titel, K.Nachname
FROM    Buch B, Kunde K
WHERE   B.ID_Kunde = K.ID_Kunde;
```

```
Ausgabe:  Leichter lachen, Blei
          Datenbanken,     Ax
          Haus und Garten, Block
```

Das Buch 'Finanzratgeber' taucht somit in der Ergebnismenge nicht auf. Möchte man eine komplette Bücherliste erstellen, in der die Kundennamen lediglich bei ausgeliehenen Büchern zusätzlich erscheinen, hat man ein Problem. Mit einem Equi-Join ist eine solche Abfrage nicht realisierbar.

Haupt- und Nebentabellen

Wie kann die folgende Abfrage realisiert werden?

Erstelle eine komplette Bücherliste ALLER Bücher! Diese Liste soll insbesondere auch die zurzeit nicht ausgeliehenen Bücher mit einschließen. Bei jedem ausgeliehenen Buch soll jedoch der Nachname des Kunden angezeigt werden, der das Buch ausgeliehen hat.

Bei derartigen Fragen muss zunächst zwischen Haupt- und Nebentabellen unterschieden werden.

Haupttabelle

Die Haupttabelle ist immer die Tabelle, deren Datensätze auf jeden Fall vollständig in der Ergebnismenge erscheinen sollen.

Nebentabelle

Die Nebentabelle ist die Tabelle, aus der bei übereinstimmenden Schlüssel- und Fremdschlüsselmerkmalen zusätzliche Informationen zur Ergebnismenge hinzugefügt werden sollen.

Da in dem Beispiel eine vollständige Bücherliste verlangt wird, ist die Tabelle 'Buch' in diesem Fall die Haupttabelle. Nur dann, wenn ein Buch tatsächlich ausgeliehen worden ist, wird zur Ergebnismenge der entsprechende Kundennachname aus der Tabelle 'Kunde' hinzugefügt. Die Tabelle 'Kunde' stellt somit in diesem Fall die Nebentabelle dar.

Buch

ID_Buch	ISBN	Titel	ID_Kunde
1	1-23456-001	Leichter lachen	2
2	4-98765-002	Datenbanken	3
3	6-55444-006	Finanzratgeber	
4	7-14324-008	Haus und Garten	4

Haupttabelle

Nebentabelle

Kunde

ID_Kunde	Vorname	Nachname	Geburtstag
1	Eva	Klein	1995-01-13
2	Kai	Blei	1967-10-28
3	Udo	Ax	
4	Ernst	Block	1988-02-02

Welche der beiden Tabellen die Haupttabelle ist, kann man im FROM-Teil einer Abfrage mit Hilfe der Schlüsselwörter LEFT JOIN oder RIGHT JOIN festlegen. Somit gibt es zwei äquivalente Schreibweisen, um eine Tabelle als Haupttabelle auszuzeichnen. Im obigen Beispiel sehen die entsprechenden FROM-Teile, die die Tabelle 'Buch' als Haupttabelle und die Tabelle 'Kunde' als Nebentabelle festlegen, wie folgt aus:

```
FROM    Buch LEFT JOIN Kunde
```

oder

HAUPT-Tabelle LINKS

HAUPT-Tabelle RECHTS

```
FROM    Kunde RIGHT JOIN Buch
```

Verwendet man die Schlüsselwörter LEFT JOIN, so muss sich links davon die Haupttabelle befinden. Entsprechend befindet sich die Haupttabelle bei der Verwendung der Schlüsselwörter RIGHT JOIN rechts von diesen. Bei Left- oder Right-Join-Abfragen wird das Schlüsselwort ON zur Definition der eigentlichen Verbindung (Join) verwendet. Weitere Bedingungen können mit dem Schlüsselwort WHERE hinzugefügt werden.
Unter Verwendung des Left-Joins könnte die gewünschte Liste aller Bücher wie folgt ermittelt werden.

```
SELECT  B.Titel, K.Nachname
FROM    Buch B LEFT JOIN Kunde K
ON      B.ID_Kunde = K.ID_Kunde;
```

Beispiele für die Verwendung von Left- bzw. Right-Joins (zwischen zwei Tabellen)

Erstelle eine komplette Bücherliste ALLER Bücher! Diese Liste soll insbesondere auch die zurzeit nicht ausgeliehenen Bücher mit einschließen. Bei jedem ausgeliehenen Buch soll jedoch der Nachname des Kunden angezeigt werden, der das Buch ausgeliehen hat.

```
SELECT  B.Titel, K.Nachname
FROM    Buch B LEFT JOIN Kunde K
ON      B.ID_Kunde = K.ID_Kunde;
```

```
SELECT  B.Titel, K.Nachname
FROM    Kunde K RIGHT JOIN Buch B
ON      B.ID_Kunde = K.ID_Kunde;
```

```
Ausgabe:  Leichter lachen , Blei
          Datenbanken     , Ax
          Finanzratgeber
          Haus und Garten , Block
```

Anmerkung

Im FROM-Teil können auch weiterhin, wie hier gezeigt, Aliasnamen für Tabellen vereinbart werden.

Erstelle eine komplette Kundenliste ALLER Kunden! Diese Liste soll insbesondere auch Kunden, die zurzeit kein Buch ausgeliehen haben, mit einschließen. Bei Kunden, die ein oder mehrere Bücher ausgeliehen haben, sollen die zugehörigen Buchtitel angezeigt werden.

```
SELECT  K.Vorname, K.Nachname, B.Titel
FROM    Kunde K LEFT JOIN Buch B
ON      B.ID_Kunde = K.ID_Kunde;
```

```
SELECT  K.Vorname, K.Nachname, B.Titel
FROM    Buch B RIGHT JOIN Kunde K
ON      B.ID_Kunde = K.ID_Kunde;
```

```
Ausgabe:  Eva,   Klein
          Kai,   Blei,  Leichter lachen
          Udo,   Ax,    Datenbanken
          Ernst, Block, Haus und Garten
```

Erstelle eine komplette Liste ALLER Kunden, deren Nachname mit einem Buchstaben zwischen 'B' und 'O' beginnt. Diese Liste soll auch Kunden, die zurzeit kein Buch ausgeliehen haben, mit einschließen. Bei Kunden, die ein oder mehrere Bücher ausgeliehen haben, sollen die zugehörigen Buchtitel angezeigt werden.

```
SELECT  K.Vorname, K.Nachname, B.Titel
FROM    Kunde K LEFT JOIN Buch B
ON      B.ID_Kunde = K.ID_Kunde
WHERE   K.Nachname BETWEEN 'B' AND 'P';
```

```
SELECT  K.Vorname, K.Nachname, B.Titel
FROM    Buch B RIGHT JOIN Kunde K
ON      B.ID_Kunde = K.ID_Kunde
WHERE   K.Nachname BETWEEN 'B' AND 'P';
```

```
Ausgabe:  Eva,   Klein
          Kai,   Blei,  Leichter lachen
          Ernst, Block, Haus und Garten
```

11 Benutzerschnittstelle

Daten, auf die mit Hilfe von SQL zugegriffen werden kann, müssen zur einfachen und anschaulichen Eingabe, Weiterverarbeitung und Darstellung für den Benutzer grafisch aufbereitet werden.
Für die Erzeugung und Gestaltung solcher Benutzerschnittstellen bieten die meisten Anbieter von RDBMS produktbezogene individuelle Möglichkeiten an. Auch clientseitige Datenbanken wie beispielsweise ACCESS bieten diverse Möglichkeiten der Formular- und Berichtgestaltung.
Wenn man eine Benutzerschnittstelle entwickeln möchte, die relativ unabhängig von der verwendeten Datenbanksoftware ist, kann man den Zugriff auf die Daten, beispielsweise von höheren Programmiersprachen wie C++ oder Java aus, über die Schnittstelle ODBC (Open Database Connectivity/„Offene Datenbank-Verbindungsfähigkeit") bzw. JDBC realisieren. Diese Umsetzung ist für LAN's (Lokal Area Network) gut geeignet, da dadurch dem Programmierer alle Möglichkeiten der Programmiersprache zur Verfügung stehen.
Datenbankanwendungen, auf die nicht nur innerhalb eines LAN sondern weltweit (evtl. sogar öffentlich) zugegriffen werden soll, werden in der Regel webbasierend realisiert. Zur Umsetzung der Benutzerschnittstelle wird bei webbasierenden Datenbankanwendungen häufig die serverseitige Skriptsprache PHP eingesetzt.

Dieses Kapitel setzt Erfahrungen und Grundkenntnisse im Bereich PHP-Programmierung und HTML-Formular-Erstellung voraus.

Im Kapitel 13 Nachschlagen sind Zusammenfassungen und Befehlsübersichten zu diesen Themen zu finden.

11.1 LAMP / XAMPP

Die Skriptsprache PHP ist integraler Bestandteil von LAMP[1], WAMP[2] oder XAMP[3] aber auch in XAMPP[4] Systemen und wird zusammen mit dem Datenbankmanagementsystem MySQL verwendet.
In den folgenden beiden Abschnitten werden die prinzipiellen Arbeitsweisen und die Vor- und Nachteile von LAMP- und XAMPP-Systemen vorgestellt. Das Kapitel 11.2 zeigt anhand des einfachen Gästebuchbeispiels das grundsätzliche Zusammenspiel von HTML und PHP, bevor im Kapitel 11.3 auf Sicherheitsaspekte beim Betrieb eines öffentlichen und damit angreifbaren Webservers eingegangen wird.

1 LAMP Abkürzung für Linux, Apache-Webserver, MySQL, PHP
2 WAMP Abkürzung für Windows, Apache-Webserver, MySQL, PHP
3 XAMP das X steht für die Betriebssystemunabhängigkeit sonst wie LAMP
4 XAMPP wie XAMP, aber zusätzlich mit der Scriptsprache Perl

11.1.1 LAMP

LAMP ist ein Akronym für:
- **L**inux Betriebssystem
- **A**pache Webserver
- **M**ySQL Datenbank
- **P**HP Skriptsprache

Komponenten

Linux

Linux ist das Betriebssystem, das die Basis eines LAMP-Systems bildet. Von ihm stammt das L aus der Abkürzung LAMP.

Apache

Apache ist der Webserver, der auf dem Betriebsystem läuft und der für die Bereitstellung der HTML- und PHP-Seiten zuständig ist.

MySQL

MySQL ist das relationale Datenbankmanagementsystem (RDBMS), das die Datenbanken und deren Zugriffsrechte verwaltet. Mit SQL als Datenbankschnittstelle kann auf das RDBMS, beispielsweise von PHP aus, zugegriffen werden.

PHP

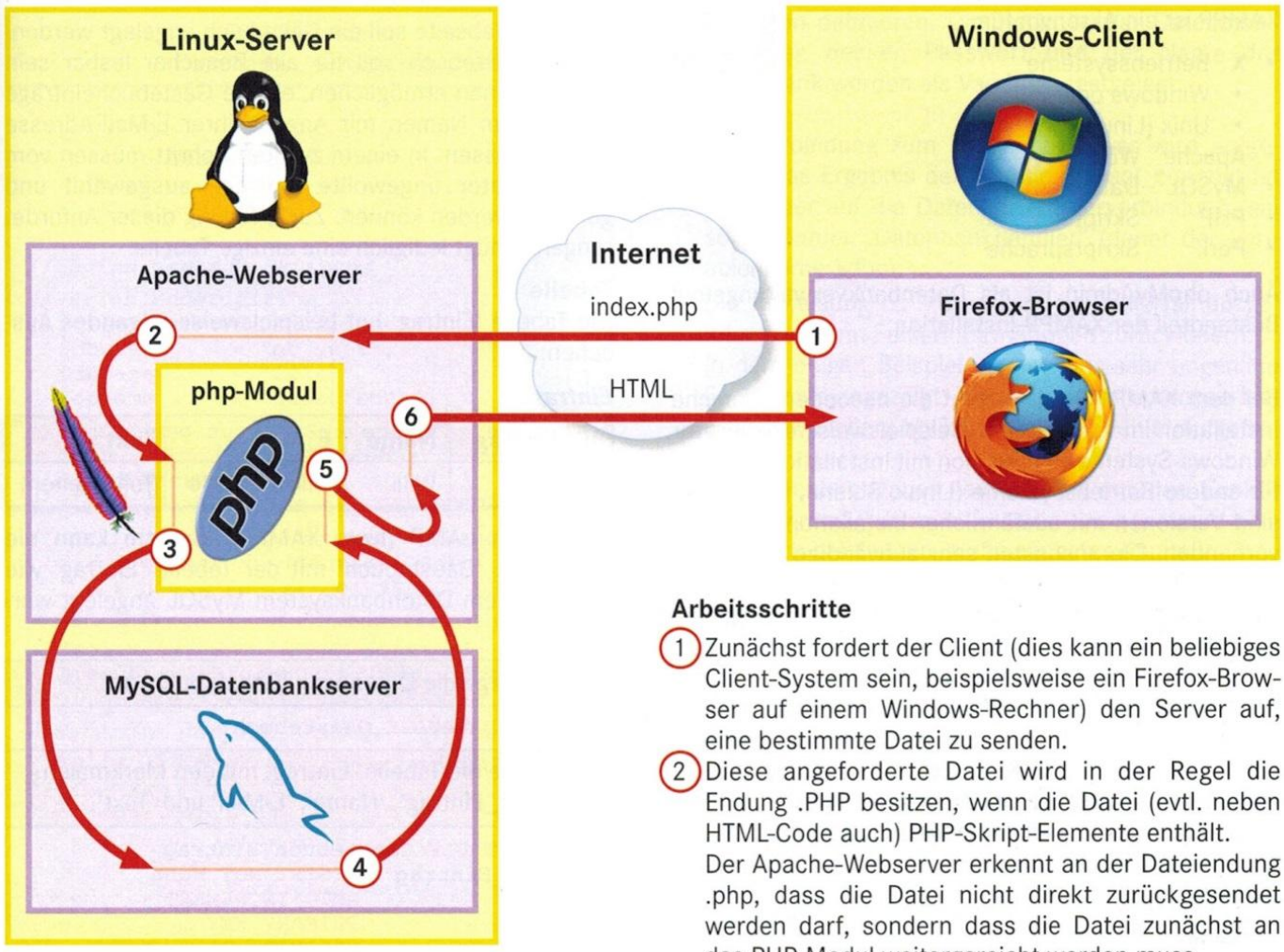
PHP ist eine, als Modul in den Apache-Webserver integrierte Skriptsprache. Sie wird serverseitig ausgeführt und wird zur Erzeugung von HTML-Code verwendet. Mit PHP-Funktionen kann der Zugriff auf eine MySQL-Datenbank realisiert werden.

---

phpMyAdmin

phpMyAdmin ist ein in PHP geschriebenes, webbasiertes Verwaltungstool für MySQL-Datenbankserver. Es ist schnell und unkompliziert zu installieren und vereinfacht die Datenbankadministration erheblich.

Arbeitsweise



Der entscheidende Unterschied zwischen der Skriptsprache PHP und anderen Skriptsprachen (wie etwa JavaScript) besteht darin, dass die PHP-Programme vom Webserver ausgeführt werden. An den aufrufenden Browser werden nur die PHP-Programmausgaben (in der Regel reiner HTML-Code und nicht das eigentliche PHP-Programm) gesendet. Der Browser selbst muss lediglich diesen HTML-Code umsetzen und anzeigen. Mit der Programmausführung hat der Browser auf dem Client nichts zu tun[1].

Im Folgenden wird die Abarbeitung einer typischen Browseranfrage an einen Apache-Webserver anhand des obigen Schaubildes vorgestellt. Dabei ist die angeforderte Datei keine reine HTML-Datei (z.B. index.html) sondern eine HTLM Datei mit integrierten PHP Skripten. Diese Dateien haben üblicherweise die Endung PHP (z.B. index.php). PHP-Dateien Anfragen werden nicht vom Apache-Webserver direkt beantwortet, sondern müssen wie folgt abgearbeitet werden:

Arbeitsschritte

1. Zunächst fordert der Client (dies kann ein beliebiges Client-System sein, beispielsweise ein Firefox-Browser auf einem Windows-Rechner) den Server auf, eine bestimmte Datei zu senden.
2. Diese angeforderte Datei wird in der Regel die Endung .PHP besitzen, wenn die Datei (evtl. neben HTML-Code auch) PHP-Skript-Elemente enthält.
   Der Apache-Webserver erkennt an der Dateiendung .php, dass die Datei nicht direkt zurückgesendet werden darf, sondern dass die Datei zunächst an das PHP-Modul weitergereicht werden muss.
3. Das PHP-Modul übersetzt alle PHP-Skript-Anteile der Datei und führt diese aus. Das PHP-Skript kann nun wiederum Datenbankbefehle in Form von SQL-Anweisungen enthalten. In einem solchen Fall muss vom PHP-Modul aus die Verbindung zum Datenbankserver hergestellt und die SQL-Abfrage an den Datenbank-Server übergeben werden.
4. Der Datenbankserver MySQL nimmt die SQL-Abfrage entgegen, führt diese auf der angegebenen Datenbank aus und liefert die Ergebnismenge an das PHP-Skript zurück.
5. Das PHP-Skript bereitet die Ergebnismenge so auf, dass daraus reiner HTML-Code entsteht.
6. Die dynamisch erstellte Datei enthält jetzt ausschließlich HTML-Code und wird von dem Apache-Webserver an den aufrufenden Browser auf dem Client zurückgeschickt.

Vorteile

- Es müssen keine Annahmen oder Voraussetzungen bezüglich des Client-Browsers gemacht werden (jeder Standardbrowser zeigt die Seite an).
- Der Anwender bekommt von dem serverseitigen Programm nichts mit (Sicherheitsaspekt).

1 Im Gegensatz zu z.B. JavaScript-Programmen. Diese Programme werden direkt an den Browser gesendet, der dann für die Ausführung des Skripts verantwortlich ist.

11.1.2 XAMPP

XAMPP ist ein Akronym für:

- **X** Betriebssysteme
  - Windows oder
  - Unix (Linux)
- **A**pache WEB-Server
- **M**ySQL Datenbank
- **P**HP Skriptsprache
- **P**erl Skriptsprache

Auch phpMyAdmin ist als Datenbankverwaltungstool Bestandteil der XAMPP-Installation.

Bei dem XAMPP-System steht die besonders einfache Installation im Vordergrund. Beispielsweise existiert für Windows-Systeme eine Version mit Installationsroutine, für andere Betriebssysteme (Linux, Solaris, Mac OS X) sind Versionen mit ausführlicher Installationsanleitung vorhanden. Die ansonsten sehr aufwändige Konfiguration der Teilkomponenten entfällt. Das XAMPP-System ist in erster Linie für Entwickler geeignet, die möglichst schnell ein kompaktes Testsystem aufsetzen möchten. Zum Einsatz als Produktivsystem (z.B. als öffentlicher Webserver) ist ein XAMPP-System wegen der Einschränkungen in Hinblick auf die Sicherheit nicht zu empfehlen. Da ein XAMPP-System alle Komponenten eines LAMP-Systems (oder auch eines WAMP-Systems und zusätzlich die Skriptsprache PERL) enthält, ist die zuvor beschriebene Arbeitsweise des LAMP-Systems auf das XAMPP-System übertragbar.

Vorteile

- Wie beim LAMP-System!
- Leichte Installierbarkeit.
- Geringer Konfigurationsaufwand.
- Für unterschiedliche Betriebssysteme erhältlich.

Nachteil

- Sollte nicht als Produktivsystem verwendet werden. (Aufgrund von Sicherheitslücken)

> **Anmerkung**
>
> Aufgrund der einfacheren Installation und der Verfügbarkeit für unterschiedliche Betriebssysteme befindet sich auf der Buch-CD ein XAMPP-System.
>
> Da die Skriptsprache Perl in diesem Buch nicht verwendet wird, könnte aber ebenso gut ein LAMP oder WAMP-System aufgesetzt werden.

11.2 Beispiel Gästebuch

Für eine Webseite soll ein Gästebuch angelegt werden. Dieses Gästebuch soll für alle Besucher lesbar sein und soll ihnen ermöglichen, eigene Gästebucheinträge unter ihrem Namen mit Angabe ihrer E-Mail-Adresse zu hinterlassen. In einem zweiten Schritt müssen vom Administrator ungewollte Einträge ausgewählt und gelöscht werden können. Zur Erfüllung dieser Anforderungen genügt lediglich eine einzige Tabelle.

Tabelle

Die Tabelle 'Eintrag' hat beispielsweise folgendes Aussehen:

Eintrag

ID_Eintrag	Name	E_Mail	Text
1	Willi	willi@mail.de	Tolle Seiten!

Auf einem LAMP (bzw. XAMPP) -System kann die Datenbank 'Gaestebuch' mit der Tabelle 'Eintrag' wie folgt auf dem Datenbanksystem MySQL angelegt werden.

Lege die Datenbank 'Gaestebuch' an.

```
CREATE DATABASE Gaestebuch;
```

Erzeuge die Tabelle 'Eintrag' mit den Merkmalen 'ID_Eintrag', 'Name', 'E-Mail' und 'Text'.

```
CREATE TABLE Gaestebuch.Eintrag
(    ID_Eintrag    BIGINT NOT NULL
                   AUTO_INCREMENT
                   PRIMARY KEY ,
     Name          VARCHAR( 25 ),
     E_Mail        VARCHAR( 25 ),
     Text          TEXT                );
```

Testweise könnten auch erste Datensätze über SQL in die Tabelle 'Eintrag' eingefügt werden.

Lege den Testdatensatz in der Tabelle 'Eintrag' an: '1, Willi, willi@mail.de, Tolle Seiten!'

```
INSERT INTO Gaestebuch.Eintrag
(Name, E_Mail, Text) VALUES
('Willi','willi@mail.de','Tolle Seiten!');
```

Benutzer

Zudem sollte ein eigener Datenbankbenutzer angelegt werden, der nur auf der Datenbank 'Gaestebuch' die notwendigen Rechte besitzt.

Lege den Benutzer 'gb_user' mit Passwort '12345' an.

```
CREATE USER gb_user@localhost
IDENTIFIED BY '12345';
```

Mit 'localhost' wird der eigene, der lokale Rechner bezeichnet.

Gib dem Benutzer 'gb_user' das Lese-, Einfüge- und Löschrecht auf der Tabelle 'Eintrag'.

```
GRANT SELECT, INSERT, DELETE ON
Gaestebuch.Eintrag * TO gb_user@localhost;
```

11.2.1 PHP Gästebuch anzeigen

PHP-Skriptdatei
Dateiname: 'anzeige.php'

```
<html>
   <body bgcolor='#cccccc'>
      <h1> Gästebuch </h1>
      <hr>
      <table border='1'>
<?php
//Beginn des php-Skriptes
//Variablendefinition
   $dbserver    = "localhost";
   $dbuser      = "gb_user";
   $dbpasswort  = "12345";                                    (1)
   $dbname      = "Gaestebuch";
//Verbindung zum DB-Server aufbauen                          (2)
   $dbh = mysql_connect($dbserver,$dbuser,
      $dbpasswort)
      or die ("Fehler bei CONNECT");                          (3)
//Verbindung zur Datenbank aufbauen
   mysql_select_db ($dbname, $dbh)
      or die ("Fehler bei SELECT_DB");                        (4)
//SQL-Abfrage an die Datenbank senden
   $sql = "SELECT * FROM Eintrag";                            (5)
   $result = mysql_query ($sql, $dbh)
      or die ("Fehler bei QUERY");                            (6)
//Ergebnis der SQL-Abfrage verarbeiten
   while ($row=mysql_fetch_row($result))                      (7)
   {
      echo "<tr>\n";
      foreach ($row as $i)
      {
          echo "<td>$i</td>\n";                              (8)
      }
      echo "</tr>\n";
   }
//Datenbankverbindung schließen
   mysql_close($dbh);                                         (9)
// Ende des PHP-Skriptes
?>
      </table>
   </body>
</html>
```

Das HTML-Dokument beinhaltet einen PHP-Skript. PHP-Skripte beginnen mit dem Tag <?php und enden mit dem Tag ?>. In einer HTML-Datei können mehrere PHP-Skripte an den unterschiedlichsten Stellen eingefügt werden. Alle Programmzeilen zwischen dem beginnenden und dem endenden PHP-Tag werden vom PHP-Parser (engl. to parse „analysieren") übersetzt und ausgewertet. Nach der Auswertung bleibt ein HTML-Dokument übrig, das kein PHP mehr enthält.
Die Skriptsprache PHP stellt unterschiedliche MySQL-Befehle zur Verfügung, mit denen Verbindungen zu MySQL-Datenbanken aufgebaut, SQL-Befehle übertragen, Abfrageergebnisse entgegengenommen und Datenbankverbindungen abgebaut werden können. Einige dieser Befehle sind bereits in dem obigen Beispielprogramm zu finden.

Programmablauf

(1) Variablen definieren. Der Rechner, der Datenbankbenutzer, dessen Passwort und der Name der Datenbank werden als Variablen festgelegt.

(2) `mysql_connect()`
Die Verbindung zum Datenbankserver wird aufgebaut. Das Ergebnis der Funktion mysql_connect ist ein Zeiger auf die Datenbankserververbindung, ein so genannter „Datenbankhandler" (daher der Variablenname $dbh).

(3) Die Anweisung `or die` wird im Fehlerfall ausgeführt und kann Fehlerinformationen zurückliefern.
In dem obigen Beispiel werden nur sehr ungenaue Fehlertexte angezeigt, die gerade mal erkennen lassen, bei welchem MySQL-Befehl ein Fehler aufgetreten ist.
Diese Fehlerangaben lassen sich mit den Funktionen `mysql_errno()` und `mysql_error()` erheblich präzisieren (siehe nächstes Kapitel).

(4) `mysql_select_db()`
Der Befehl mysql_slect_db baut mit Hilfe des Datenbankhandlers und des Datenbanknamens die eigentliche Datenbankverbindung auf. Ab jetzt kann über den Datenbankhandler direkt auf die Datenbank zugegriffen werden.

(5) SQL-Anweisung der Variablen `$sql` zuweisen.

(6) `mysql_query()`
Der Befehl mysql_query führt mit Hilfe des Datenbankhandlers die SQL-Anweisung auf der ausgewählten Datenbank aus. Als Ergebnis wird die Ergebnismenge ($result) der SQL-Anweisung zurückgeliefert.

(7) `mysql_fetch_row()`
Mit dem Befehl mysql_fetch_row können aus der Ergebnismenge ($result) einer SQL-Abfrage einzelne Datensätze ($row) separiert werden. Das Ergebnis dieses Befehls ist ein eindimensionales Array, in das der betreffende Datensatz abgelegt wird. Daher wird die `while`-Schleife für jeden Datensatz der Ergebnismenge einmal durchlaufen bis kein Datensatz mehr vorliegt.

(8) Mit der `foreach`-Schleife werden alle Datenfelder ($i) aus dem jeweiligen Datensatz ($row) ausgelesen. Für jedes Datenfeld wird die foreach-Schleife genau einmal durchlaufen.

(9) `mysql_close()`
Datenbankverbindung abbauen.

> **Anmerkung**
>
> Dieses Beispiel kann nur funktionieren, wenn die Datei ausgabe.php über den Apache-Webserver abgerufen und ausgewertet wird. Der Client kann mit den PHP-Skript-Elementen nichts anfangen. Öffnet man die Datei 'ausgabe.php' direkt im Browser führt dies nicht zum gewünschten Ergebnis.

Nur die Ausgabe des PHP-Skriptes (das sind alle Angaben hinter einem 'echo' Befehl) verbleiben nach der Auswertung in der Datei, so dass letztendlich folgende HTML-Datei entsteht und an den Client zurückgeschickt wird. In dieser HTML-Datei ist der Bereich, der durch das PHP-Skript erzeugt worden ist, rot hinterlegt.

HTML-Code, den man sich im Browser anzeigen lassen kann.

```
<html>
  <body bgcolor='#cccccc'>
    <h1> Gästebuch </h1>
    <hr>
    <table border='1'>
    <tr>
      <td>1</td>
      <td>Willi</td>
      <td>willi@mail.de</td>
      <td>Tolle Seiten!</td>
    </tr>
    </table>
  </body>
</html>
```

Im Client-Browser selbst wird in etwa folgende Seite zu sehen sein.

11.2.2 PHP Include-Dateien/Funktionen

Einige Aufgaben, wie beispielsweise der Verbindungsaufbau zur Datenbank, werden innerhalb eines Projektes häufig an unterschiedlichsten Stellen und in unterschiedlichen Dateien verwendet. Derartige Aufgaben können in PHP durch eigenständige Funktionen realisiert werden. Diese Funktionen lassen sich zu separaten Include-Dateien zusammenfassen, die sich dann bei Bedarf an den entsprechenden Stellen in das Projekt einbinden (includieren) lassen. Da unabhängig davon, ob man das Gästebuch anzeigen, neue Datensätze in das Gästebuch einfügen oder aber bestehende Datensätze aus dem Gästebuch löschen möchte, immer wieder die Verbindung zur Datenbank 'Gaestebuch' aufgebaut und SQL-Anweisungen an die Datenbank gesendet werden müssen, kann man sich sehr viel Arbeit sparen, wenn auch hier mit einer solchen Include-Datei gearbeitet wird. Im Folgenden wird eine Include-Datei vorgestellt, die wesentliche Datenbankfunktionen repräsentiert.

PHP-Include-Datei 'db_funktionen.php'

```
<?php
// Testmodus aktivieren / deaktivieren
  define ("TESTMODUS", true);                    (1)

//Funktion zum Verbindungsaufbau
  function db_connect()
  {
    $dbserver     = "localhost";
    $dbuser       = "gb_user";                   (2)
    $dbpasswort   = "12345";
    $dbname       = "Gaestebuch";
    $dbh = mysql_connect($dbserver,
      $dbuser, $dbpasswort)
      or die db_fehler("connect"));
    mysql_select_db ($dbname, $dbh)
      or die(db_fehler("select"));
    return $dbh;
  }

//Abfragefunktion mit Verbindungsaufbau
  function db_query($sql)
  {
    if (TESTMODUS) {echo $sql;}                  (3)
    $dbh=db_connect();
    $result=mysql_query ($sql, $dbh)
      or die (db_fehler("query"));
    db_close();
    return $result;
  }

// Verbindungsabbau
  function db_close()
  {                                              (4)
    mysql_close();
  }

//Fehlerbehandlung
  function db_fehler($fehler)
  {
    if(TESTMODUS)                                (5)
    {
      echo
        "Fehler beim MySQL-Befehl " .
        $fehler .
        "<li> Fehlernummer errno = " .
        mysql_errno() .
        "<li> Fehlertext error = " .
        mysql_error();
    }
  }
?>
```

Diese einmal geschriebene Includedatei enthält Funktionen, die jetzt an unterschiedlichen Stellen eines Projektes wiederverwendet werden können.

Beschreibung der Funktionen/Konstanten

(1) `TESTMODUS`
Die Konstante TESTMODUS kann während der Entwicklungsphase aktiviert werden (auf den Wert true gesetzt werden), um Kontroll- und Fehlerausgaben zu erhalten. Im Echtbetrieb sollte die Konstante auf den Wert false gesetzt sein.

(2) `db_connect()`
Die Funktion db_connect kann ohne Übergabeparameter verwendet werden, um die Verbindung zu einer fest vorgegebenen Datenbank herzustellen. Der Datenbankhandler wird als Ausgabewert zurückgegeben.

(3) `db_query()`
Der Funktion wird lediglich die SQL-Anweisung übergeben. Der Verbindungsauf- und -abbau zur Datenbank wird von der Funktion automatisch erledigt. Als Return-Wert erhält man die Ergebnismenge der SQL-Abfrage.

(4) `db_close ()`
Diese Funktion ist nur der Vollständigkeit halber hier aufgeführt. Sie macht nichts anderes als die MySQL-Funktion mysql_close() logisch auf die Include-Datei 'db_funktionen' abzubilden.

(5) `db_fehler()`
Diese Funktion erlaubt das genauere Analysieren und Anzeigen von Datenbankfehlern. Dazu wird die Fehlernummer und auch der Fehlertext mithilfe der MySQL-Funktionen mysql_errno() und mysql_error() ermittelt und angezeigt.

In der ursprünglichen Datei 'anzeige.php' kann jetzt mit Hilfe des Befehls include_once() auf die Datei 'db_funktionen.php' und die darin enthaltenen Funktionen zurückgegriffen werden. In dem veränderten Anzeigeprogramm (siehe nächste Spalte!) sind die dafür notwendigen Änderungen rot unterlegt.

11.2.3 PHP Klassen und Objekte

Mit PHP kann auch objektorientiert programmiert werden. Anstelle der vorgestellten Include-Datei könnte z.B. auch ein 'Datenbankobjekt' angelegt und verwendet werden, das die vorgestellten Funktionen als Objektmethoden erhält. Als Objektvariablen sind die Verbindungsdaten wie Servername, Datenbankname, etc. denkbar. Da aber bereits Grundkenntnisse in PHP zum Verständnis der hier vorgestellen Beispiele vorausgesetzt werden, wird auf die objektorientierte Programmierung unter PHP nicht weiter eingegangen.

PHP-Skriptdatei
Dateiname: 'anzeige.php'

```
<html>
  <body bgcolor='#cccccc'>
     <h1> Gästebuch </h1>
     <hr>
     <table border='1'>
<?php
// Beginn des php-Skriptes
// Include-Datei einbinden und nutzen.
  include_once ("db_funktionen.php");
  $sql = "SELECT * FROM Eintrag";
  $result = db_query($sql);
// Ergebnis der SQL-Abfrage verarbeiten
  while ($row=mysql_fetch_row($result))
  {
     echo "<tr>\n";
     foreach ($row as $i)
     {
        echo "<td>$i</td>\n";
     }
     echo "</tr>\n";
  }
// Ende des PHP-Skriptes
?>
     </table>
  </body>
</html>
```

11.2.4 PHP Dateneingabe

Als erste Erweiterung soll dem Benutzer die formulargestützte Eingabe neuer Gästebucheinträge ermöglicht werden. Die Dateneingabe kann über ein HTML-Formular erfolgen. Diese HTML-Formulardatei muss die eingegebenen Daten dann an ein PHP-Skript übertragen, welches die Eingabedaten in die MySQL-Datenbank speichert. Somit sind für diese zusätzlichen Anforderungen zwei weitere Dateien notwendig.

- 'eingabe1.html': Die HTML-Formulardatei
- 'eingabe2.php': Die Datei mit dem PHP-Skript, das die Daten von der Formulardatei 'eingabe1.html' erhält und in die Datenbank 'Gaestebuch' speichert.

11.2.5 HTML Formulardatei 'eingabe1.html'

Das Tag <form> leitet in HTML ein Formular ein. Durch unterschiedliche Attribute dieses Tags kann die Zieldatei und die Art der Parameterübergabe zu dieser Zieldatei festgelegt werden. Die Datei 'eingabe1.html' (auf der nächsten Seite) enthält ein HTML-Formular mit Eingabefeldern für den Namen, die E-Mailadresse und den eigentlichen Gästebucheintrag. Zudem werden zwei Schalter generiert. Ein Schalter zum Bestätigen und Absenden der Daten und ein weiterer, um die Formulareingaben zu löschen.

HTML-Eingabeformular 'eingabe1.html'

```
<html>
  <body bgcolor='#cccccc'>
    <h1> Gästebucheintrag</h1><hr>
    <form action='eingabe2.php'                    (1)
      method='post'>
      <table border='0'>
        <tr>
          <td>Name:</td>
          <td><input type='text'
            name='Name'></td>
        </tr>
        <tr>
          <td>E-Mail:</td>
          <td><input type='text'                   (2)
            name='E_Mail'></td>
        </tr>
        <tr>
          <td>Text: </td>
          <td><textarea cols='16'                  (3)
            rows='5' name='Text'>
            </textarea>
          </td>
        </tr>
        <tr>
          <td><input type='submit'
            name='OK'                              (4)
            value='speichern'>
          </td>
          <td><input type='reset'
            name='Reset'                           (5)
            value='zurücksetzen'>
          </td>
        </tr>
      </table>
    </form>
  </body>
</html>
```

Dieses HTML-Formular wird im Browser, zunächst noch ohne die gemachten Einträge, wie folgt dargestellt:

Programmablauf/Erläuterungen

(1) `<form>`

Mit dem <form>-Tag wird das HTML-Formular eingeleitet. Die dabei verwendeten Parameter haben folgende Bedeutung:

- `action='eingabe2.php'`
  Dieser Parameter legt fest, dass sobald der Absendeschalter ('submit' siehe Punkt 4) betätigt wird, die Formulardaten an die angegebene Folgedatei ('eingabe2.php') übergeben werden.
- `method='post'`
  Dieser Parameter bestimmt, wie die Formulardaten an die Folgedatei übergeben werden.

(2) `<input type='text'>`

Mit diesem HTML-Tag werden einfache Formular-Eingabefelder definiert. Die Option

- `name='E-Mail'`
  legt beispielsweise den Namen 'E-Mail' als Variablennamen fest, unter dem der Eingabetext später in der Zieldatei 'eingabe2.php' wieder abgerufen werden kann.

(3) `<textarea>`

Mit diesem HTML-Tag können Formular-Textbereiche festgelegt werden, die mehrere Zeilen (rows) und Spalten (columns) umfassen können. Dadurch wird aber nur die Größe des Textbereiches festgelegt und ist nicht im Sinne einer Tabelle zu verstehen.

(4) `<input type='submit'>`

Das HTML-Formular-Tag <input type='submit'> erzeugt einen Absendeschalter. Wird dieser Schalter aktiviert, werden die Formulardaten an die im <form>-Tag festgelegte Zieldatei übertragen. Der Parameter value erlaubt das Festlegen der Schalterbeschriftung.

(5) `<input type='reset'>`

Dieses HTML-Formular-Tag erzeugt einen Zurücksetzen-Schalter, der alle bisherigen Formulareingaben wieder löscht.

Anmerkung

- Die Tabelle, in HTML realisiert durch die Tags <table> <tr> <td> </td> </tr></table>, sind nur zur übersichtlicheren Gestaltung des Formulars eingefügt worden.
- Bei allen HTML-Formular-Tags ist der Parameter 'name' von besonderer Bedeutung, da sich dahinter der Variablenname verbirgt, über den das zugehörige Formularelement bzw. dessen Inhalt angesprochen werden kann.

11.2.6 Kontrolle der Übergabeparameter

Bevor man daran geht, die Eingabedaten, die aus dem Eingabeformular 'eingabe1.html' an die Datei 'eingabe2.php' übergeben werden, in der Datenbank zu speichern, sollte man zunächst überprüfen, ob die Datenübergabe zwischen den beiden Dateien fehlerfrei funktioniert. Dies kann mit Hilfe der PHP-Funktion phpinfo() geschehen, die sich sehr einfach in die Datei 'eingabe2.php' einbauen lässt.

Die Datei 'eingabe2.php' mit PHP-Code zur Kontrolle der Übergabeparameter.

```
<?php
 phpinfo();
?>
```

Durch den Aufruf der Funktion phpinfo() werden sämtliche Umgebungsvariablen aufgelistet, auf die von PHP aus zugegriffen werden kann. Neben sehr vielen anderen (in diesem Zusammenhang unwichtigen) Informationen, sollte man (nach langem Suchen) im Abschnitt 'PHP Variables' das Array '_POST' wiederfinden.

Da das Formular 'eingabe1.html' die Formulardaten mit Hilfe der Methode 'post' übergibt, müssen hier unter den jeweiligen Namen der Formularelemente die eingegebenen Werte wiederzufinden sein.

PHP Variables

Variable	Value
_POST["Name"]	Walter
_POST["E-Mail"]	walter@gmx.de
_POST["Text"]	Ich finde die Seiten auch toll!
_POST["OK"]	speichern

Anmerkung

Erst wenn die richtigen Variablennamen als Arrayindex des Arrays _POST und die richtigen Eingabewerte angezeigt werden, macht es Sinn die eigentliche Datei 'eingabe2.php' zu erstellen, mit der dann die übergebenen Werte in die Datenbank übertragen werden.

11.2.7 Gästebucheintrag speichern 'eingabe2.php' (Speicherdatei)

Das tatsächliche Speichern der Datensätze gestaltet sich relativ einfach, da auf die Include-Datei 'db_funktionen.php' zurückgegriffen werden kann. Beispielsweise führt die Funktion 'db_query()' dieser Include-Datei neben der eigentlichen SQL-Abfrage auch den Verbindungsauf- und -abbau zur Datenbank durch.

Speicherdatei 'eingabe2.php' speichert die Daten aus dem Eingabeformular in die Datenbank.

```
<html>
  <body bgcolor='#cccccc'>
    <h1>  Neuer Gästebucheintrag </h1>
    <hr>  Ein neuer Gästebucheintrag
          wird gespeichert.
<?php
// Übergabeparameter einlesen
  $Name    = $_POST['Name'];
  $E_Mail  = $_POST['E_Mail'];                    (1)
  $Text    = $_POST['Text'];
// Includedatei einbinden
  include_once ("db_funktionen.php");
// SQL-Anweisung zusammenstellen
  $sql="INSERT INTO Eintrag
    (Name, E_Mail, Text) Value                    (2)
    ('$Name','$E_Mail', '$Text');";
// SQL-Anweisung ausführen
  db_query($sql);                                 (3)
?>
  </body>
</html>
```

Programmablauf/Erläuterungen

(1) `$_POST` (Parameter einlesen)
Aus dem Array $_POST können die Übergabeparameter ausgelesen und in lokale PHP-Variablen übertragen werden. Als Arrayindex wird der Name des jeweiligen Formularelementes (siehe Formulardatei 'eingabe1.html') verwendet.

(2) `$sql` (Zusammenbau der SQL-Anweisung)
Bei dem Zusammenbau der SQL-Anweisung kann auf die zuvor ausgelesenen Übergabeparameter in den Variablen $Name, $E_Mail und $Text Bezug genommen werden.

(3) `db_query()`
Diese selbstgeschriebene Funktion ist Bestandteil der Includedatei 'db_funktionen.php', die zuvor über den Befehl 'include_once' eingebunden worden ist. Die Funktion 'db_query()' steuert den Verbindungsaufbau zur Datenbank, das Ausführen der SQL-Anweisung und den Abbau der Datenbankverbindung.

11.2.8 PHP Gästebucheinträge löschen 'loeschen1.php' (Formulardatei)

Zum Löschen ungewollter Gästebucheinträge werden wiederum zwei Dateien benötigt. Zum einen die Datei 'loeschen1.php', welche die Gästebucheinträge als Mehrfachauswahl in einem Formular darstellt und somit das Auswählen der zu löschenden Gästebucheinträge ermöglicht und zum anderen die Datei 'loeschen2.php', die dann die ausgewählten Datensätze aus der Datenbank 'Gaestebuch' löscht.

Formulardatei 'loeschen1.php' ermöglicht das Auswählen der zu löschenden Gästebucheinträge.

```
<html>
  <body bgcolor='#cccccc'>
    <h1> Gästebucheinträge löschen </h1>
    <hr>
    <form action='loeschen2.php'                    (1)
      method='post'>
      <table border='1'>
<?php
// Includedatei einbinden und nutzen
  include_once ("db_funktionen.php");               (2)
  $sql="SELECT * FROM Eintrag";
  $result = db_query($sql);
// Abfrageergebnis aufbereiten
  while ($row=mysql_fetch_row($result))
  {
    echo "<tr>\n";
// Zusätzliche Auswahlspalte integrieren
    echo "<td><input type='checkbox'
      name='loeschen[]'                             (3)
      value='$row[0]'></td>";
    foreach ($row as $i)
    {
      echo "<td>$i</td>\n";                         (4)
    }
    echo "</tr>\n";
  }
?>
      </table><br>
      <input type='submit'
        name='OK'
        value='löschen'>                            (5)
      <input type='reset'
        name='Reset'
        value='zurücksetzen'>
    </form>
  </body>
</html>
```

Besonderheiten dieser Aufgabenstellung:

- Das Eingabeformular muss erst mit Hilfe eines PHP-Skriptes aus den Datensätzen der Tabelle 'Eintrag' erzeugt werden.
- Es können mehrere Datensätze gleichzeitig ausgewählt und gelöscht werden. Daher müssen die ID-Werte der zu löschenden Datensätze mit Hilfe einer Arrayvariablen von der Formulardatei 'loeschen1.php' an die Löschdatei 'loeschen2.php' übergeben werden.

Programmablauf/Erläuterungen

(1) `<form>`
Die Datei 'loeschen2.php' wird als Folgedatei bestimmt, welche die Formulardaten, mittels der POST-Methode, zugestellt bekommt.

(2) `include_once() / db_query()`
Die Includedatei 'db_funktionen.php' wird eingebunden und ermöglicht so das Nutzen der Funktion 'db_query()' zum vereinfachten Ausführen von SQL-Anweisungen.

(3) `<input type='checkbox'>`
In einer zusätzlichen Spalte wird dem eigentlichen Datensatz das Formularelement 'checkbox' vorangestellt, um so eine Mehrfachauswahl zu ermöglichen. Wichtig sind hierbei die folgenden Attribute:

- `name='loeschen[]'`
  Der Name 'loeschen' wird als Variablenname für dieses Formularelement festgelegt. Da mehrere Werte ausgewählt werden können, muss diese Variable (mit den eckigen Klammern) als Array definiert werden.
- `value='row[0]'`
  Der Inhalt der Variablen wird auf den ersten Wert der jeweiligen Datensatzzeile festgesetzt. Diese entspricht dem ID-Wert des jeweiligen Gästebucheintrages.

(4) `foreach ($row as $i)`
Die Gästebucheinträge werden in einer HTML-Tabelle aufbereitet.

(5) `<input type='submit'>`
`<input type='reset'>`
Absende- und Zurücksetzschalter werden dem Formular hinzugefügt.

11.2.9 PHP Gästebucheinträge löschen 'loeschen2.php' (Löschdatei)

Mit der Kontrollfunktion phpinfo() kann die Parameterübergabe überprüft werden.

Die Datei 'loeschen2.php' mit PHP-Code zur Kontrolle der Übergabeparameter.

```
<?php
 phpinfo();
?>
```

Im Bereich 'PHP Variables' sollten jetzt die ID_Werte der ausgewählten Gästebucheinträge in der Arrayvariablen '$_POST['loeschen']' wiederzufinden sein.

PHP Variables

Variable	Value
_POST["loeschen"]	Array ([0] => 1 [1] => 2)
_POST["OK"]	löschen

Werden die Parameter richtig übergeben, so kann aus der Variablen '$_POST['loeschen']' abgelesen werden, dass im obigen Beispiel zwei Gästebucheinträge gelöscht werden sollen. Die beiden Gästebucheinträge lassen sich über die ID-Werte eindeutig bestimmen, so dass lediglich der folgende SQL-Befehl zum Löschen der Datensätze ausgeführt werden muss:

Lösche die Gästebucheinträge aus der Tabelle 'Eintrag', deren ID-Wert einem ID-Wert der angegebenen Menge {1, 2} entspricht.

```
DELETE FROM Eintrag
WHERE ID_Eintrag IN (1,2);
```

Die Schwierigkeit für das nachfolgende Löschprogramm 'loeschen2.php' besteht also darin, aus den Übergabeparametern die obige SQL-Anweisung zusammenzustellen. Insbesondere die rot unterlegte Löschmenge muss aus den Übergabeparametern dynamisch ermittelt werden. Ansonsten unterscheidet sich dieses Programm nur unwesentlich von der Datei 'einfügen2.php'.

Löschdatei 'loeschen2.php' löscht die ausgewählten Datensätze aus der Datenbank 'Gaestebuch'.

```
<html>
  <body bgcolor='#cccccc'>
    <h1> Löschen von Einträgen </h1>
    <hr> Ein oder mehrere Gästebuch
         einträge werden gelöscht.
<?php
// Übergabeparameter einlesen
  $loeschen=$_POST["loeschen"];                    (1)
//Löschmenge zusammenstellen
  $menge="(";
  foreach ($loeschen as $i)
  {
    $menge=$menge ."$i,";                          (2)
  }
  $menge=substr($menge, 0, -1);
  $menge=$menge . ")";
// Includedatei einbinden
  include_once ("db_funktionen.php");              (3)
// SQL-Anweisung zusammenstellen
  $sql="DELETE FROM Eintrag
    WHERE ID_Eintrag IN $menge;";                  (4)
// SQL-Anweisung ausführen
  db_query($sql);                                  (5)
?>
  </body>
</html>
```

Programmablauf/Erläuterungen

(1) `$_POST` (Parameter einlesen)
Das Array 'loeschen' wird als Übergabeparameter ausgelesen.

(2) `$menge`
Die Menge der ID-Werte wird zusammengebaut, im Beispiel der String „(1,2)". Dabei werden folgende Funktionen verwendet.

- `$menge = $menge . "$i,"`
  Mit dem Operator '.' (Punkt) können zwei Teilstrings zusammengefügt werden.
- `$menge = substring ($menge, 0, -1)`
  Mit dieser Anweisung wird das letzte überflüssige Zeichen (hier das Zeichen ',' (Komma)) aus dem String $menge entfernt.

(3) `include_once ("db_funktionen.php")`
Die Includedatei wird eingebunden, um die Funktion 'db_query()' nutzen zu können.

(4) `$sql` (Zusammenbau der SQL-Anweisung)
In die SQL-Anweisung wird der zuvor ermittelte String $menge eingebunden.

(5) `db_query()`
Diese selbstgeschriebene Funktion aus der Includedatei regelt den Vebindungsauf- und -abbau zur Datenbank und führt die SQL-Anweisung aus.

11.3 Sicherheit

Das Thema Sicherheit und insbesondere die Websicherheit ist sehr komplex und wird gerne vernachlässigt. Auch das Beispiel 'Gästebuch' des vorangegangenen Kapitels ist NICHT angriffsicher programmiert und sollte so auf keinen Fall auf einem öffentlichen Webserver verwendet werden. Gerade „Einsteiger" sind froh, wenn eine Webanwendung überhaupt funktioniert. Diese Einstellung ändert sich meistens erst, wenn eigene Webserver durch Angreifer zerstört, für deren Zwecke missbraucht oder Daten manipuliert worden sind. Im schlimmsten Fall bekommt man von derartigen kriminellen Aktivitäten erst gar nichts mit. Das Kapitel 11.3.1 'SQL-Inclusions' (Einschließungen) sensibilisiert am Beispiel von böswilligen SQL-Manipulationen für das Thema Websicherheit, bevor im Kapitel 11.3.2 Sicherheitshinweise und Tipps zum Schutz eigener Anwendungen gegeben werden. Allerdings wird in diesem Buch das Thema Sicherheit nur angerissen werden.

11.3.1 SQL-Inclusions

Angenommen zur Autorisierung auf dem Web-Server soll eine Benutzerverwaltung angelegt werden. Jeder Benutzer soll sich in Zukunft mit seiner Kennung und seinem Passwort anmelden, bevor er Zugang zu den Web-Seiten erhält. In der Tabelle 'User' sind die dafür notwendigen Benutzerdaten auf einem MySQL-System hinterlegt.

User

ID_User	Kennung	Passwort
1	admin	hsk8iwz
2	mustermann	emma123

Anmeldeformular 'anmelden1.html'

Die Anmeldeinformationen werden mit der Methode POST aus dem Anmeldeformular 'anmelden1.html' an die folgende Anmeldekontrolle 'anmelden2.php' übertragen.

Anmeldekontrolle 'anmelden2.php'

```
<html>
  <body bgcolor='#cccccc'>
    <h1> Anmeldekontrolle</h1><hr>
<?php
// Übergabeparameter auslesen
  $Kennung      =$_POST["Kennung"];
  $Passwort     =$_POST["Passwort"];                    (1)
// Includedatei einbinden
  include_once ("db_funktionen.php");
//Kontrollabfrage zusammenstellen                       (2)
  $sql=" SELECT  Kennung
         FROM    User
         WHERE   Kennung='$Kennung'
           AND   Passwort='$Passwort'";
// SQL-Abfrage ausführen                                 (3)
  $result=db_query($sql);
// Wenn Datensatz gefunden,
// dann erfolgreiche Anmeldung!
  if ( mysql_num_rows($result) > 0 )                    (4)
  {
    $kennung=mysql_fetch_row($result);
    echo "Willkommen! ".$kennung[0].
         " Sie sind jetzt angemeldet.";
  }
  else
  {
    echo  "Anmeldung fehlgeschlagen!"
  }
?>
  </body>
</html>
```

Programmablauf/Erläuterungen

(1) `$_POST` (Parameter einlesen)
Die Übergabeparameter 'Kennung' und 'Passwort' werden ausgelesen.

(2) `$sql` (Zusammenbau der SQL-Anweisung)
Aus den Übergabeparametern wird die SQL-Abfrage zur Überprüfung der Anmeldedaten zusammengestellt.

(3) `db_query()`
Der Befehl regelt den Datenbankverbindungsauf- und -abbau und das Ausführen der SQL-Anweisung.

(4) `mysql_num_rows($result) > 0`
Die Funktion mysql_num_rows gibt die Anzahl der, in der Ergebnismenge $result enthaltenen, Datensätze zurück. Wurde eine entsprechende Kennung mit zugehörigem Passwort in der Tabelle 'User' gefunden, wird dieser Wert zwangsläufig größer als null sein. Auf diese Weise kann überprüft werden, ob eine Anmeldung erfolgreich war oder nicht.

Wo liegt das Problem?
Gibt man in dem Anmeldeformular die Daten

```
Kennung: admin
Passwort: hsk8iwz
```

ein, dann werden in dem Programm 'anmelden2.php' diese Werte nach der Parameterübergabe an die Variablen '$Kennung' und '$Passwort' gebunden und somit zu folgender SQL-Anweisung zusammengesetzt:

```
$sql=" SELECT  Kennung
       FROM    User
       WHERE   Kennung='admin'
         AND   Passwort='hsk8iwz'";
```

Da sich in der Tabelle 'User' ein Benutzerdatensatz mit Kennung 'admin' und Passwort 'hsk8iwz' befindet, kann die Anmeldung wie erwartet korrekt durchgeführt werden.

Der Angreifer kennt das Passwort einer ihm bekannten Kennung nicht
Problematisch wird es erst, wenn der Anwender, wie bereits im Anmeldeformular angedeutet, die folgenden Daten eingibt:

```
Kennung: admin'#
Passwort:
```

In dem Fall entsteht durch die Parameterübergabe und das Zusammenfügen der Variablen $Kennung und $Passwort zur SQL-Abfrage folgende Anweisung:

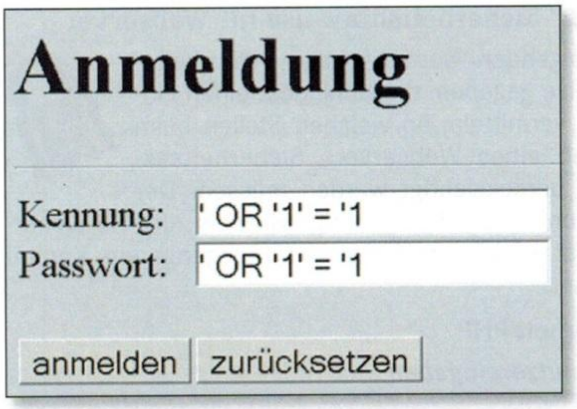
```
$sql=" SELECT  Kennung
       FROM    User
       WHERE   Kennung='admin'#'
         AND   Passwort=''";
```

Leider wird beim Ausführen der SQL-Anweisung das Zeichen '#' als der Beginn eines Kommentars interpretiert, sodass alles nach dem '#' Zeichen ignoriert wird. Letztendlich entsteht die folgende SQL-Abfrage OHNE den rot hervorgehobenen Teil.

```
$sql=" SELECT  Kennung
       FROM    User
       WHERE   Kennung='admin'#'
         AND   Passwort=''";
```

Wenn ein Angreifer weiß, dass die Kennung 'admin' auf einem System existiert, dann kann er sich auf diese Weise, auch ohne Kenntnis des zugehörigen Passwortes, an dem System anmelden. Durch das Einschleusen des Kommentarzeichens in die SQL-Anweisung wird hier der Kontrollmechanismus ausgehebelt.

Dem Angreifer sind nicht nur die Passwörter, sondern auch die Kennungen unbekannt.
Selbst dann, wenn der Angreifer weder Kennungen noch Passwörter kennt, besteht durchaus die Möglichkeit sich bei einem ungeschützten System mit Hilfe von SQL-Inclusions unerlaubten Zutritt zu verschaffen. Sehr vielversprechend ist beispielsweise folgender Versuch:



Die Formulardaten werden wieder an die Datei 'anmelden2.php' übertragen und dort zu folgender SQL-Abfrage zusammengesetzt:

```
$sql=" SELECT  Kennung
       FROM    User
       WHERE   Kennung=''OR '1'='1'
         AND   Passwort=''OR '1'='1'";
```

Da '1'='1' immer richtig ist und diese Bedingung mit der logischen Operation OR mit der eigentlichen Bedingung (Kennung =' ') verknüpft wird, entsteht ein Ausdruck der immer zu wahr ausgewertet wird. Diese SQL-Anweisung wird daher die Menge aller Kennungen als Ergebnismenge zurückliefern. Unglücklicherweise ist das Anmeldeskript 'anmelden2.php' so programmiert, dass in einem solchen Fall eine Anmeldung mit der ersten gefundenen Kennung erfolgt. In diesem Beispiel ist dies zufällig wieder die 'admin'-Kennung.

Fazit
SQL-Inclusions aber auch PHP-Inclusions bieten unzählige Anfriffsmöglichkeiten. Im Extremfall[1] können Eingaben wie

```
';DROP DATABASE;#
```

ganze Datenbanken zerstören. Deshalb ist eine der wichtigsten Grundregeln im Bezug auf sichere Webprogrammierung:
ALLE Benutzereingaben MÜSSEN richtig validiert (auf Zulässigkeit überprüft) werden, BEVOR sie weiterverarbeitet werden. Dabei spielt es keine Rolle, ob sie über GET[2] oder POST an den Server geschickt werden oder ob sich die Werte in Cookies[3] oder in 'hidden fields'[4] befinden.

1 Glücklicherweise nicht beim mysql_query-Befehl, da hierbei keine Mehrfach-SQL-Befehle zulässig sind.
2 Die GET-Methode bietet eine weitere Möglichkeit Eingabeparameter aus einem HTMLFormular zu übertragen. Dabei werden die Werte für den Benutzer sichtbar als Parameter übertragen. Im Allgemeinen ist die Methode POST dem GET vorzuziehen.
3 Cookies erlauben das clientseitige Speichern von beliebigen Informationen.
4 Versteckte Felder werden vom Browser ignoriert und sind für den Benutzer unsichtbar. In HTML-Formularen können so Zusatzinformationen versteckt und übertragen werden.

1.3.2 Sicherheitshinweise für Webserver

n Folgenden werden einige Sicherheits-inweise gegeben, die zumindest einen Ein-ruck vermitteln, an welchen Stellen beim etrieb eines Webservers Sicherheitsas-ekte berücksichtigt werden müssen. Der rste und besonders wichtige Hinweis zur PHP-Sicher-eit ist bereits aus dem vorangegangenen Kapitel ekannt.

icherheit PHP

Benutzereingaben

Alle Benutzereingaben vor der Weiterverarbeitung validieren!

Fehleranzeige deaktivieren
Fehler in Protokolldatei schreiben. Die Ausgabe von Fehlermeldungen sollte nur während der Programmentwicklung erfolgen.

Variablen initialisieren
Variablen sollten vor ihrer ersten Benutzung initialisiert werden!

Verwaltungstools entfernen
Verwaltungstools wie phpMyAdmin zum Anlegen und Verwalten von Datenbanken werden auf einem Produktivserver nicht mehr zwingend benötigt und sollten gelöscht (bzw. deaktiviert) werden.

Include-Dateien
Include-Dateien können systemkritische Informationen, wie beispielweise die Verbindungsdaten zur Datenbank, enthalten. Daher sollten diese Dateien nicht im öffentlichen Bereich des Webservers (unterhalb des Verzeichnisses htdocs) sondern in einem nicht öffentlichen Bereich abgelegt werden. Die PHP-Skripte können Include-Dateien auch aus nichtöffentlichen Bereichen einbinden.

Systemsicherungen
Auch Kopien oder Systemsicherungen gehören nicht in den öffentlichen Bereich des Webservers (unterhalb des htdocs -Verzeichnisses).

icherheit HTML

HTML-Kommentare
Keine Kommentare in HTML-Dateien zurücklassen, die Angreifern ungewollt Informationen liefern.

icherheit Apache

Server-Banner verstecken
Es kann und sollte verhindert werden, dass der Webserver seine Identität und seine Version auf Anfragen oder bei Fehlermeldungen preisgibt.

- ***Zugriff nur auf bestimmte Dateien erlauben***
  Der Web-Server kann mit Hilfe von .htaccess Dateien oder über die Konfigurationsdatei httpd.conf so eingestellt werden, dass er nur noch bestimmte Dateien (z.B nur noch Dateien mit der Dateiendung .php) anzeigt. Alle anderen Dateien werden vor dem direkten Zugriff geschützt und könnten ggf. über ein entsprechendes php-Skript dargestellt werden.
- ***Das mod_spelling deaktivieren***
  Durch dieses Modul werden einfache Tippfehler bei der direkten Angabe von URL's korrigiert.
- ***Das mod_autoindex deaktivieren***
  Durch dieses Modul wird ein Verzeichnisbaum angezeigt, wenn keine index.html oder index.php Datei gefunden wird.

Sicherheit Betriebssystem

- ***HTTP-Port***
  Nur die Ports öffnen, die unbedingt für den Betrieb des Web-Servers notwendig sind.
  - HTTP (Port 80/8080)
- ***HTTPS-Port***
  Anmeldungen und Passwörter nur über sichere Verbindungen (HTTPS) durchführen :
  - HTTPS (Port 443)
- ***SSH***
  Wenn der Webserver von externen Clients aus gewartet und mit Daten versorgt werden muss, kann der SSH-Zugang verwendet werden. Ein FTP Zugang ist nicht notwendig! Ein Datentransfer kann z.B. über SSH mit dem Programm 'WinSCP' erfolgen
  - SSH (Port 22)
  - Evtl. SSH vom Standardport 22 auf einen unbekannten Port verlegen.
  - Anzahl möglicher Anmeldeversuche für SSH-Zugang heruntersetzen.
  - Der Root- oder Admin-Kennung keine direkte Anmeldung über SSH erlauben.
    - Besser eine separate sehr eingeschränkte SSH-Kennung anlegen. Von dieser Kennung aus kann dann zur Administratorkennung gewechselt werden.
  - Den SSH-Zugang nicht über Passwörter sondern über Schlüssel absichern.

Da gerade durch die Validierung der Benutzereingaben die Sicherheit eines Webservers entscheidend verbessert werden kann, folgt auf der nächsten Seite eine Funktion 'eingabeKontrolle()' mit der beliebige Sonderzeichen und Schlüsselwörter kontrolliert aus einer Benutzereingabe entfernt werden können. Auf diese Weise können SQL- und PHP-Inclusions verhindert werden.

Programm zur Eingabevalidierung

```
function eingabeKontrolle(&$eingabe)
//Die Funktion überprüft und verändert den Eingabeparameter direkt! (Seiteneffekt!!!)
//Sollte ein Eingabefehler vorliegen, wird:
// 1. Die Eingabe durch den Text "- Eingabefehler -" ersetzt!
// 2. Der boole'sche Wert true zurückgegeben!
{
//Liste der unzulässigen Steuerzeichen
   $tauschen  = array("&&", "||", ";", "\\", "\"", "'");                    (1)
//Liste der unzulässigen Schlüsselwörter (Achtung Großschreiben!)
   $finden    = array("AND", "OR", "SELECT", "DELETE", "INSERT", "UNION");  (2)
   $alarm = "-"; //Variable wird beim Fund von Schüsselwörtern gesetzt.
   $eingabe = trim($eingabe); //entfernt führende und nachfolgende Leer-/Sonderzeichen
   $ausgabe = $eingabe;
   $ausgabe = strip_tags($ausgabe); //entfernt HTML-Tags
   $ausgabe = str_replace($tauschen, "_Sonderzeichen_", $ausgabe); //entfernt Sonderz.
   $eingabe_gross = strtoupper($ausgabe); //in Großbuchstaben umwandeln
//Schlüsselwörter suchen und $alarm bei Fund setzen
   foreach ($finden as $schluesselwort)
   {
      $regausdruck = "/$schluesselwort/";
      $ausgabe_teile = preg_split($regausdruck, $eingabe_gross); //Stringteilung
      $anfang = "A"; $ende= "E"; //A für Anfang; E für Ende
      foreach ($ausgabe_teile as $wert)
      {
         $anfang = substr($wert, 0,1); //Teilstringbildung
         if (   !preg_match("/^[A-Z0-9]+$/",$anfang) //Mustersuche
             && !preg_match("/^[A-Z0-9]+$/",$ende) {$alarm = $schluesselwort;}
         $ende = substr($wert, -1);
      }
   }
//Ursprüngliche Eingabe mit dem veränderten Ausgabewert vergleichen
   if ($ausgabe == $eingabe && $alarm == "-")
   { //Eingabe korrekt
      return false;
   }
   else
   { //Eingabe fehlerhaft
      $eingabe = "- Eingabefehler -";
      return true;
   }
}
```

Es gibt in PHP zahlreiche Funktionen, die das Validieren eines Strings ermöglichen und vereinfachen. Die hier vorgestellte Funktion hat den Vorteil, dass die Arrays

(1) `$tauschen` um beliebige Sonderzeichen und

(2) `$finden` um beliebige Schlüsselwörter erweitert werden können, die dann aus der Benutzereingabe herausgefiltert werden.

Wichtig ist, dass diese Funktion direkt mit dem Originalwert (auf der Referenz) arbeitet und diesen verändert. Die Funktion gibt den boole'schen Wert true zurück, wenn in der Benutzereingabe eines der Schlüsselwörter oder Sonderzeichen gefunden wurde.

11.3.3 Angriffe erkennen

Durch die im vorigen Abschnitt beschriebenen Maßnahmen lässt sich die Sicherheit eines Webservers maßgeblich erhöhen. Um Angriffe aber überhaupt rechtzeitig erkennen und angemessen darauf reagieren zu können, ist es sinnvoll ein **Intrusion Detection System** (**IDS** Einbruchserkennungssystem) auf dem Webserver zu installieren.

SNORT

Snort ist ein solches open source **network intrusion prevention and detection system (IDS/IPS)**, das sehr häufig zur Überwachung bei LAMP oder XAMPP-Systemen zum Einsatz kommt.