Kurzreferenz

PHP

Autor: Michael Puff

Erstellt: 2011-02-25

http://www.michael-puff.de
mail@michael-puff.de

Inhaltsverzeichnis

Tabellenverzeichnis

1 Grundlagen

1.1 Syntax

- Ein PHP-Skript wird in
  `<?php ... ?>`
  eingeschlossen.
- Jede Zeile endet mit einem Semikolon.
- PHP unterscheidet zwischen Groß- und Kleinschreibung.

1.2 Kommentare

- Einzeilige Kommentare werden mit
  `//`
  oder
  `#`
  eingeleitet.
- Blockkommentare bestehen aus
  `/*... */`

1.3 Funktionen

```
function name($param1, $param2, $optparam = 5)
{
  anweisungen
  return rückgabewert;
}
```

- Optionale Parameter werden mit einem Wert vorbelegt.
- Sie sind beim Funktionsaufruf nicht erforderlich.
- Rückgabewerte werden mit `return` zurückgegeben.
- Code nach *return* wird nicht mehr ausgeführt. Die Funktion wird beendet und verlassen.

2 PHP und HTML

- PHP Code wird mit `<?php ... ?>` in HTML-Dateien eingebunden.
- HTML-Dateien mit PHP Code müssen die Endung php tragen.
- Die Textausgabe erfolgt mit dem Befehl *echo*:
  ```
  echo "Dies ist ein Text.";.
  ```
- Zeichenketten können sowohl in Anführungszeichen als auch in einfache Hochkommas gesetzt werden.
- Wird mit *echo* eine Zeichenkette ausgegeben, die wiederum Anführungszeichen enthält, müssen diese mit
  maskiert werden:
  ```
  echo "Dies ist ein Zitat: \"Hello world.\"";
  ```
- Einfache Hochkommas können ohne sie zu maskieren ausgegeben werden, wenn die Zeichenkette von Anführungszeichen umschlossen ist:
  ```
  echo "Dies ist ein Zitat: 'Hello world.'\n";.
  ```

3 Variablen und Datentypen

3.1 Variablen

- Deklaration durch vorangestelltes Dollarzeichen ($)
- Zulässige Zeichen: Zeichen, Ziffer und Unterstrich
- Variablen sind *case sensitive*
- Variablen werden nicht deklariert
- Erstellung erfolgt durch Wertzuweisung:

```
$text = "Dies ist ein Text";
```

3.2 Datentypen

Datentyp	Beschreibung
Integer	Ganzzahlen. `$zahl = 5;`
Oktalzahl	Oktalzahl. Bildschirmausgabe erfolgt dezimal. Oktalzahlen beginnen mit einem O. `$ok_zahl = O123;`
Hexadezimalzahl	Hexadezimalzahl. Bildschirmausgabe erfolgt dezimal. Hexadezimalzahlen beginnen mit 0x. `$hex_zahl = 0x123;`
Double	Fließkommazahl. `$zahl = 4.5;`
String	Zeichenkette. `$text = "Dies ist ein Text";`
Boolean	Wahrheitswert. *true/false*. `$wert = true;`

Tab. 3.1: Datentypen

3.3 Arrays

3.3.1 Einfache Arrays

```
$arrayname[index] = "Wert1";
```

```
<?php
  $vorname[0] = "Peter";
  $vorname[1] = "Hans";
  $vorname[2] = "Robert";
  // Ausgabe
  echo($vorname[0]);
  echo($vorname[1]);
  echo($vorname[2]);
?>
```

Mit der Funktion *print_r* werden Informationen über eine Variable in leichter lesbarer Form ausgegeben. Einzelne Werte werden lediglich ausgegeben, bei Arrays und Objekten werden Schlüssel/Wert-Paare ausgegeben.

3.3.2 Assoziative Arrays

```
$arrayname["Schlüssel"] = "Wert1";
```

```
<?php
  $kfz["GS"] = "Goslar";
  $kfz["KS"] = "Kassel";
  $kfz["HH"] = "Hamburg";
  // Ausgabe Element Hamburg
  echo $kfz["HH"];
?>
```

Syntax der Kurzschreibweise:

```
$arrayname = array("Schlüssel1" => "Wert1",
                   "Schlüssel2" => "Wert2");
```

3.4 Konstanten

```
define("NAME", Wert);
```

```
<?pp
  define("ZAHL", 2);
  // Ausgabe
  echo ZAHL;
?>
```

Es gibt von PHP reservierte Konstanten. So gibt die Konstante __*FILE*__ den Pfad der aktuellen Datei zurück.

4 Operatoren

Zuweisungsoperator ist das Gleichheitszeichen: =.

Mit dem Punktoperator . werden Zeichenketten miteinander verbunden.

Operator	Bedeutung	Syntax	Kurzform
+	Addition	`$x = $x + $y;`	`$x += $y;`
-	Subtraktion	`$x = $x - $y;`	`$x -= $y;`
*	Multiplikation	`$x = $x * $y;`	`$x *= $y;`
/	Division	`$x = $x / $y;`	`$x /= $y;`
+1	Inkrement	`$x = $x + 1;`	`$x++;`
-1	Dekrement	`$x = $x - 1;`	`$x--;`
%	Modulo	`$x = $x % $y;`	`$x %= $y;`

Tab. 4.1: Rechenoperatoren

Operator	Bedeutung
&& bzw. and	und
|| bzw. or	oder
Xor	entweder-oder
!	Negation

Tab. 4.2: Logische Operatoren

Operator	Bedeutung
==	gleich
===	gleicher Wert und gleicher Typ
!= bzw. <>	ungleich
!==	Wert und Typ nicht gleich
>	größer als
<	kleiner als
>=	größer gleich
<=	kleiner gleich

Tab. 4.3: Vergleichsoperatoren

5 Kontrollstrukturen

5.1 Verzweigungen

5.1.1 if-else

```
if (Bedingung)
{
  Anweisung1;
}
else
{
  Anweisung2;
}
```

```
$variable = (Bedingung)? $variable1 : $variable2;
```

Ist die Bedingung wahr, wird der linken Seite der Wert der Variablen links vom Doppelpunkt zugewiesen, ansonsten der Wert rechts vom Doppelpunkt.

5.1.2 elseif

```
if (Bedingung1)
{
  Anweisung1;
}
elseif (Bedingung2)
{
  Anweisung2;
}
// weiter elsif-Blöcke
else
{
  AnweisungN;
}
```

5.1.3 switch

```
switch($variable)
{
  case Wert1:
    Anweisung1;
    break;
   case Wert2:
     Anweisung2;
     break;
   default:
     Anweisung3;
}
```

- Im Unterschied zu den meisten anderen Programmiersprachen können auch Zeichenketten zur Fallunterscheidung genommen werden.
- Wichtig: Wird das *break* am Ende eines Anweisungsblocks weggelassen, wird auch der nachfolgende Anweisungsblock abgearbeitet!

5.2 Schleifen

5.2.1 for-Schleife

```
for(Start; Bedingung; In- bzw. Dekrement)
{
  Anweisung;
}
```

An Stelle des In- bzw. Dekrements kann auch eine beliebige Anweisung stehen, die nach jedem Schleifendurchlauf ausgeführt wird.

```
for($i = 0; $i < 5; $i++)
{
  echo $i;
}
```

5.2.2 while-Schleife

```
while(Bedingung)
{
  Anweisung;
}
```

5.2.3 do-while-Schleife

```
do
{
  Anweisung;
}
while(Bedingung);
```

5.2.4 foreach-Schleife

```
foreach($array as $value)
{
  echo $value;
}
```

6 Wichtige Funktionen

Funktion	Beschreibung
sort()	Sortiert ein Array aufsteigend.
rsort()	Sortiert ein Array absteigend.
asort()	Sortiert ein assoziatives Array aufsteigen. Die Beziehung zwischen Element und Index bleibt erhalten.
arsort()	Wie *asort()* nur absteigend.
ksort()	Sortiert ein assoziatives Array nach dem Index.

Tab. 6.1: Sortierfunktionen

Funktion	Beschreibung
date(format)	Aktuelle Zeit und Datum ausgeben. Wobei *format* aus Formatbeschreibern als Zeichenkette aus Tabelle 6.3 (Seite: 12) zusammengesetzt sein kann.
checkdate(monat, tag, jahr)	Überprüft Datum auf Gültigkeit. Rückgabe *true* oder *false*.
strtotime(string datum)	Umrechnung eines Datums in englischer Schreibweise in einen UNIX Zeitstempel. Bei ungültigen Angaben ist der Rückgabewert -1.

Tab. 6.2: Datumsfunktionen

Format	Beschreibung
a	“am“ oer „pm“
A	„AM“ oder „PM“
d	Zweistelliger Tag des Monats
D	Wochentag, kurz , englisch
F	Monatsname, englisch
g	Stunde zwischen 1 und 12
G	Stunde zwischen 0 und 23
h	Stunde zwischen 1 und 12, zweistellig
H	Stunde zwischen 0 und 23, zweistellig
i	Minuten von 0 - 59
I	Sommer- und Winterzeit. 1 = Sommerzeit, 0 = Winterzeit
j	Tag von 1 bis 31
l	Tag der Woche, englisch
L	Schaltjahr. 1 = Schaltjahr, 0 = kein Scahltjahr
m	Monat von 01 bis 12
M	Monat, kurz, englisch
n	Monat von 1 bis 12
s	Sekunde von 00 bis 59
t	Anzahl der Tage des aktuellen Monats
T	Zeitzone des Rechners
U	Sekunden seit 1970-01-01 (UNIX Zeitstempel)
w	Wochentag von 0 bis 6, 0 = Sonntag
y	Jahr, zweistellig
Y	Jahr, vierstellig
z	Tag des Jahres von 0 bis 365

Tab. 6.3: Datumsformatbeschreiber

Funktion	Beschreibung
strchr(zeichenkette, zeichen)	Sucht nach *zeichen* in *zeichenkette*. Rückgabewert ist der Rest der Zeichenkette.
strrchr(zeichenkette, zeichen)	Wie *strchr()* nur von hinten.
strstr(zeichenkette, teilzeichenkette)	Sucht nach *teilzeichenkette* in *zeichenkette*. Rückgabewert ist der Rest der Zeichenkette inklusive der Teilzeichenkette.
strtr(zeichenkette, alt, neu)	Ersetzt *alte* durch *neu* in *zeichenkette*.
strpos(zeichenkette, zeichen[, offset])	Gibt die numerische Position des ersten Vorkommens von *zeichen* innerhalb von *zeichenkette* zurück. *offset* bezeichnet die Startposition der Suche. Als *zeichen* werden auch Zeichenketten akzeptiert.
str_replace(teilzeichenkette, ersatz, zeichenkette)	Ersetzt *teilzeichenkette* durch *ersatz* in *zeichenkette*.
substr_count(zeichenkette, substr)	Zählt *substr* in *zeichenkette*.
strlen(zeichenkette)	Ermittelt die Anzahl der in *zeichenkette* enthaltenen Zeichen.
explode(trennzeichen, zeichenkette)	zerlegt eine Zeichenketten an hand des Trennzeichens in ein Array.
strcomp(zeichenkette1, zeichenkett2)	Vergleicht zwei Zeichenketten. 0 = Zeichenketten sind gleich, 1 = Zeichenkette 1 ist größer, -1 = Zeichenkette 2 ist größer. Groß- und Kleinschreibung wird berücksichtigt.
strtoupper(zeichenkette)	Wandelt alle Zeichen in *zeichenkette* in Großbuchstaben um.
strtolower(zeichenkette)	Wandelt alle Zeichen in *zeichenkette* in Kleinbuchstaben um.
ucfirst(zeichenkette)	Wandelt das erste Zeichen von *zeichenkette* in Großbuchstaben um.
ucwords(zeichenkette)	Wandelt das erste Zeichen aller Wörter in Großbuchstaben um.
stripslashes(zeichenkette)	Entfernt aus *zeichenkette* alle Quotes.
strip_tags(zeichenkette)	Entfernt HTML- und PHP-Tags aus *zeichenkette*.

Tab. 6.4: Zeichenkettenfunktionen

Funktion	Beschreibung
array_unshift(array, element1, ...)	Fügt Elemente am Anfang eines Arrays hinzu.
array_shift(array)	Entfernen des ersten Elementes aus dem Array.
array_push(array, element1, ...)	Fügt Elemente am Ende eines Arrays hinzu.
array_pop(array)	Entfernt das letzte Element aus dem Array.
array_pad(array, anzahl, element)	*element* wird an das so lange *array* angefügt bis *anzahl* Elemente erreicht ist. *anzahl* > 0 = Anfügen an Ende, *anzahl* < 0 = Anfügen an Anfang.
array_slice(array, position, anzahl)	Schneidet ab *position* die *anzahl* Elemente aus. Rückgabewert ist ein neues Array. Ist *anzahl* negativ wird von hinten ausgeschnitten.
array_splice(array, position, anzahl, element)	Ersetzt alle *anzahl* Elemente ab *position* durch *element*. Wird der letzte Parameter wegeglassen werden die *anzahl* Elemente ab *position* gelöscht.
array_value(array)	Liest alle Elemente eine assoziativen Arrays aus. Rückgabewert ist ein neues Array.
array_keys(array)	Liest alle Schlüssel eine assoziativen Arrays aus. Rückgabewert ist ein neues Array.
array_flip(array)	Vertauscht Werte und Schlüssel bzw. Indizes eines (assoziativen(Arrays. Rückgabewert ist ein neues Array.
array_reverse(array)	Dreht die Reihenfolge der Elemente eines Arrays um. Rückgabewert ist ein neues Array.
array_merge(array1, array2, n)	Fasst die Arrays 1 bis n in einem neuen Array zusammen.
array_diff(array1, array2[, array])	Ermittelt die Unterschiede zwischen Arrays. Vergleicht array1 mit array2 und gibt die Unterschiede zurück. Gibt ein Array mit allen Werten von array1 zurück, die in keinem der anderen Arrays vorhanden sind.
in_array(needle, haystack[, bool strict])	Prüft, ob ein Wert in einem Array existiert. Wenn der dritte Parameter auf TRUE gesetzt wird vergleicht *in_array()* nicht nur den Wert sondern auch den Typ des gesuchten Wertes *needle* mit den Elementen des Arrays.
unset(array[i])	Löscht Element *i* aus *array*.

Tab. 6.5: Arrayfunktionen

Funktion	Beschreibung
die(meldung)	Gibt eine Meldung aus und beendet das aktuelle Skript.
list(...)	list() wird verwendet, um eine Liste von Variablen innerhalb einer Operation zuzuweisen.
error_log(meldung, typ[, ziel])	Sendet eine Fehlermeldung an das Log-File des Webservers, einen TCP Port oder eine Datei. Der erste Parameter, message, ist die Fehlermeldung, die mitgeschrieben werden soll. *Type* gibt an, wo die Meldung abgelegt werden soll. 0: Log-Systems Betriebssystem, 1: *meldung* wird via Email an *ziel* gesendet, 2: *meldung* wird über den PHP-Debugger gesendet, 3: *meldung* wird an die Datei angefügt, die mit *ziel* angegeben wurde.
print_r(array)	Gibt Variablen-Informationen in lesbarer Form aus.

Tab. 6.6: Sonderfunktionen

```php
<?php

$info = array('Kaffee', 'braun', 'Koffein');

// Auflisten aller Variablen
list($drink, $color, $power) = $info;
echo "$drink ist $color und $power macht es zu etwas besonderem.\n";

// Ein paar davon auflisten
list($drink, , $power) = $info;
echo "$drink hat $power.\n";

// Oder nur die dritte ausgeben
list( , , $power) = $info;
echo "Ich brauche $power!\n";

?>
```

7 Klassen

7.1 Grundstruktur

```
class klassenname
{
  [variablendeklaration}
  [variablendeklaration]

  function methode1()
  {

  }

  function methode2()
  {

  }
}
```

- Der Konstruktor hat die gleiche Bezeichnung wie die Klasse.
- Ein Objekt wird mit dem Schlüsselwort *new* erzeugt:
  `$obj = new classe1().`
- Der Zugriff und Aufruf von Attributen und Methoden erfolgt über den Selbstzeiger *$this*:
  `echo $this->variable1;.`
- Mit dem Schlüsselwort *extends* wird die Basisklasse angegeben:

```
class AbgeleiteteKlasse extends Basisklasse
{

}
```

7.2 Instantiieren einer Klasse

```
myObj = new myClass();
```

7.3 Sichtbarkeiten

Private
Eine Methode oder ein Attribut was als *private* deklariert ist, darf nur innerhalb der Klasse genutzt werden. Eine Veränderung eines Attributes oder Aufrufen einer Methode von außen ist nicht möglich. Auch Erben einer Klasse haben keinen Zugriff darauf. Es ist allgemein üblich Attribute als *private* zu deklarieren und zum Setzen und Abfragen eine *public* Methode (Getter und Setter Methoden) zu schreiben.

Public
Methoden und Attribute die als *public* deklariert werden sind überall und für jeden sichtbar.

Protected
protected ist ähnlich wie private. Methoden und Attribute sind nach außen nicht sichtbar oder erreichbar. Der Unterschied ist, das sie jedoch in abgeleiteten Klassen verfügbar sind.

statische Variablen oder Methoden
Mit dem Schlüsselwort *static* werden Methoden oder Eigenschaften einer Klasse als statisch deklariert. Statisch heißt, dass diese Methoden oder Eigenschaften verwendet werden können, ohne dass ein Objekt aus der Klasse erzeugt wurde. Drr Zugriff erfolgt über den Klassennamen:

```
echo MeineKlasse::$statische_eigenschaft;
```

8 Datenbankanbindung (MySQL)

8.1 Wichtige Funktionen

Funktion	Beschreibung
mysql_connect(host, benutzer, passwort)	Stellt eine Verbindung zu dem DMS her. Rückgabewert ist ein Verweis auf den DBMS oder *false*.
mysql_close(verbindung)	Schließt die Verbindung zu dem DBMS auf die *verbindung* verweist.
mysql_select_db(dbname, verbindung)	Wählt die Datenbank *dbname* auf dem mit *verbindung* verbundenen DBMS aus.
mysq_db_query(dbname, sqlBefehl, verbindung)	Führt die in *sqlBefehl* angegebene Abfrage auf die in *dbname* angegebene Datenbank mit der *verbindung* aus. Rückgabewert ist eine Ergebnismenge.
mysql_query(sqlBefehl, verbindung)	Führt die in *sqlBefehl* angegebene Abfrage aus. Zuvor muss eine Datenbank ausgewählt worden sein. Rückgabewert ist eine Ergebnismenge.
mysql_error()	Liefert den Fehlertext der zuvor ausgeführten MySQL Operation.
mysql_fetch_array(ergebnis)	Liefert einen Datensatz als assoziatives Array, als numerisches Array oder beides. Liefert ein Array das dem aktuellen Datensatz entspricht oder FALSE, wenn keine weiteren Datensätze vorliegen. Als Schlüssel für die assoziativen Indizes werden die Feldnamen benutzt.
mysql_fetch_row(ergebnis)	Liefert einen Datensatz aus dem Anfrageergebnis mit der übergebenen Kennung. Der Datensatz wird als Array geliefert. Wiederholtes Aufrufen von mysql_fetch_row() liefert den nächsten Datensatz des Anfrageergebnisses oder FALSE, wenn keine weiteren Datensätze verfügbar sind.
mysql_num_fields(ergebnis)	Liefert die Anzahl der Felder einer Ergebnismenge.
mysql_num_rows(ergebnis)	Liefert die Anzahl der Zeilen einer Ergebnismenge.
mysql_field_name(ergebnis, index)	Gibt den Namen des mit *index* bezeichneten Feldes aus.

Tab. 8.1: MySQL mit PHP

mysql_fetch_array:

```
<?php
mysql_connect("localhost", "mysql_user", "mysql_password")
    or die("Keine Verbindung möglich: " . mysql_error());
mysql_select_db("mydb");

$result = mysql_query("SELECT id, name FROM mytable");

while ($row = mysql_fetch_array($result, MYSQL_NUM)) {
    printf ("ID: %s  Name: %s", $row[0], $row[1]);
}

mysql_free_result($result);
?>
```

mysql_fetch_row:

```
<?php
$result = mysql_query("SELECT id,email FROM people WHERE id = '42'");
if (!$result) {
    echo 'Abfrage konnte nicht ausgeführt werden: ' . mysql_error();
    exit;
}
$row = mysql_fetch_row($result);

echo $row[0]; // 42
echo $row[1]; // Der Wert von email
?>
```

8.2 Kurzeinführung in SQL

8.2.1 Anlegen von Datenbanken und Tabellen

Eine Datenbank wird über ein SQL-Query angelegt. Die SQL-Syntax zum Anlegen einer Datenbank lautet:

```
CREATE DATABASE <name>
```

Anlegen einer Tabelle:

```
CREATE TABLE <tabellenname>(<feldname> <datentyp> <weitere Attribute>,
     <feldname> <datentyp> <weitere Attribute>, ...)
```

Indiziert man eine Spalte, wird diese separat, sortiert abgelegt und verwaltet. Das hat zur Folge das Abfragen schneller und effizienter bearbeitet werden können. Man sollte die Felder einer Tabelle indiziere, die am häufigsten sortiert bzw. abgefragt werden. Hinzukommt, dass man über die ID eines Datensatzes auf den selbigen zugreift. Also sollte auch dieses Feld indiziert werden. Ein großer Nachteil von Indizes ist die Tatsache, dass sämtliche Datenänderungen langsamer werden (Die Sortierung muss ja neu aufgebaut werden). Man muss also immer abwägen, ob ein Index auf Feld Sinn macht. Indizes werden mit dem SQL-Schlüsselwort `KEY` oder `INDEX` definiert. Dann folgt der Name des Indexes und Klammern das zu indizierende Feld:

```
PRIMARY KEY(id), KEY idx_name (name), KEY idx_vorname (vorname)!.
```

Ein vollständiger Query für das Anlegen einer Adressdatenbank könnte jetzt so aussehen:

```
CREATE TABLE Kontakte(id INT NOT NULL AUTO_INCREMENT, name varchar(20)
    , vorname varchar(20), strasse varchar(55), plz int, ort varchar
    (50),
telefon1 varchar(17), telefon2 varchar(17), fax varchar(17), email1
    varchar(50), email2 varchar(50), url varchar(50), gebdat date,
firma varchar(25), ts timestamp, PRIMARY KEY(id), KEY idx_name (name),
     KEY idx_vorname (vorname))
```

`NOT NULL`: Das Feld muss einen Wert enthalten.
`AUTO_INCREMENT`: Der Inhalt des Feldes wird bei jedem neuen Datensatz inkrementiert.

Gelöscht wird eine Tabelle mit

```
DROP TABLE <tabellenname>
```

8.2.2 Datensätze einfügen, editieren und löschen

Datensatz einfügen:

```
INSERT INTO <tabellenname>(<feldname>, <feldname>,...) VALUES('<wert>'
    , '<wert>', ...)
```

Datensatz editieren:

```
UPDATE <tabellenname> SET <feldname>='<wert>' WHERE <feldname>='<wert>
    '
```

Wichtig hier bei ist die `WHERE`-Klausel mit der wir den SQL-Ausdruck nur auf einem bestimmten Datensatz anwenden. Würde dies weglassen, wären alle Datensätze betroffen. Günstiger weise nimmt man hier ein Feld, welches einen Datensatz eindeutig identifiziert. Man kann auch Felder der Tabelle kombinieren, um einen Datensatz eindeutig zu identifizieren.

Datensatz löschen:

```
DELETE FROM <tabellenname> WHERE <feldname>='<wert>'
```

8.2.3 Filter-Operationen

Die SELECT-Anweisung startet eine Abfrage. Aufgrund der Syntax kann eine SELECT-Anweisung auch als „SFW-Block“ (SELECT, FROM, WHERE) bezeichnet werden. Jeder Abfrage gibt eine neue Tabelle mit den Datensätzen zurück, die den Abfragekriterien entsprechen.

Ein SELECT-Statement hat folgende allgemeine Syntax:

```
SELECT <feldname>, <feldname>, <...> FROM <tabellenname>
    WEITERE_SQL_ANWEISUNGEN
```

Will man sich nur alle Nachname und die Vornamen ausgeben lassen, so sieht das passende SQL-Statement so aus:

```
SELECT name, vorname FROM kontakte
```

Will man sich alle Spalten einer Tabelle ausgeben lassen, kann man anstatt alle Spaltennamen hinzuschreiben auch ein „*“ als allgemeinen Platzhalter angeben:

```
SELECT * FROM kontakte
```

ORDER BY:
Mit ORDER BY wird festgelegt, nach welcher Spalte bzw. welchen Spalten sortiert werden soll. Mit ASC werden die Zeilen aufsteigend, mit DESC absteigend sortiert. Ist nichts angegeben, wird aufsteigend sortiert. Hier ein einfaches Beispiel, Datensätze nach dem Nachnamen sortieren:

```
SELECT name, vorname FROM kontakte ORDER BY name
```

Will man nach mehreren Spalten gleichzeitig sortieren, gibt man die weiteren Spalten einfach durch ein Komma getrennt mit an:

```
SELECT name, vorname FROM kontakte ORDER BY name, ort
```

WHERE:
Mit WHERE kann man gezielt Datensätze filtern.

```
SELECT * FROM <tabellenname> WHERE <feldname>='<wert>'
```

Alle Meiers aus einer Adress-Datenbank:

```
SELECT * FROM kontakte WHERE name='meier'
```

Die Ausdrücke können auch mit AND, OR und NOT miteinander verknüpft werden. Desweiteren ist es möglich Platzhalter zu verwenden: «_» steht für ein beliebiges Zeichen und «%» für eine beliebige Zeichenkette. Auch kann man natürlich WHERE noch mit ORDER BY und weiteren SQL-Ausdrücken kombinieren.

LIKE:
Immer dann, wenn man in Textfeldern im Suchmuster Platzhalter oder Jokerzeichen verwenden will, können die Vergleichsoperatoren nicht verwendet werden. Statt dessen muss man in diesen Fällen auf den Operator LIKE zurückgreifen. Sollen zum Beispiel alle Personen mit der Vorwahl „0561“ gefunden werden, sähe dies so aus:

```
SELECT name, vorname, telefon1 FROM kontakte WHERE telefon1 LIKE '
    %0561%'
```

BETWEEN:
BETWEEN wählt alle Spalten aus die zwischen den oberen und unteren Wert liegen.

```
SELECT name, vorname, gebdat FROM kontakte WHERE gebdat BETWEEN '
    1980-01-01' and '2005-01-01'
```

Diese Abfrage liefert uns alle Personen, die zwischen 1. Januar 1980 und 1. Januar 2005 geboren wurden. Man beachte die Angabe des Datums: yyyy-mm-dd. So und nicht anders muss es angegeben werden, damit es der mySQL Server versteht.

IN:
Der letzte Operator aus dieser Gruppe ist der IN-Operator. Er wird benutzt, wenn man nicht mit einem einzelnen Wert, sondern mit einer Wertemenge vergleichen will. Beispiel: Wir wollen alle Personen die entweder „Schmidt“ oder „Meier“ heißen. Mit dem Vergleichsoperator „=“ und einer Oder-Verknüpfung wird das bei vielen Werten, die zu vergleichen sind, schnell recht unübersichtlich. Einfacher geht es mit dem IN-Operator:

```
SELECT name, vorname FROM kontakte WHERE name IN ('schmidt', 'meier')
```

9 Ausnahmebehandlung

- Exceptions werden mit *throw* geworfen und
- mit `catch(Exception $e)` abgefangen.
- Da PHP eine Garbage Collection besitzt, gibt keine Ressourcenschutzblöcke mit *finally*.

```php
<?php
function inverse($x) {
    if (!$x) {
       throw new Exception('Division durch Null.');
    }
    else return 1/$x;
}

try {
    echo inverse(5) . "\n";
    echo inverse(0) . "\n";
} catch (Exception $e) {
    echo 'Exception abgefangen: ',  $e->getMessage(), "\n";
}

// Ausführung fortsetzen
echo 'Hallo Welt';
?>
```

10 Formulare

Abb. 10.1: Beispiel Formular

HTML Code:

```
<html>
  <head>
    <title>Beispiel Formular</title>
  </head>
  <body>
    <form action='formulardaten.php' method='POST'>
      <!-- Texteingabefelder -->
      <p>Vorname: <input name='vorname' type='text'></input></p>
      <p>Nachname: <input name='name' type='text'></input></p>
      <p>
        <!-- Radionbuttons -->
```

```
        <input checked type='radio' name='farbe' value='rot'></input>
            Rot<br>
        <input type='radio' name='farbe' value='blau'></input>Blau
      </p>
      <p>
        <!-- Checkboxen -->
        <input type='checkbox' name='preis1' value='Auto'></input>Auto
            <br>
        <input type='checkbox' name='preis2' value='Reise'></input>
            Reise
       </p>
       <p>
         <!-- Listenauswahlfeld  -->
         <select name='farben' size='4'>
           <option>Rot</option>
           <option selected>Blau</option>
           <option>Grün</option>
           <option>Gelb</option>
         </select>
       </p>
       <!-- Mehrzeiliges Texteingabefeld -->
       <p><textarea name='eingabefeld' cols='40' rows='5'>Vorgegebener
            Text</textarea></p>
       <p><input type='Submit' name='senden' value='OK'></input></p>
    </form>
  </body>
</html>
```

- Mit der Methode *POST* werden die Daten im HTML-Header übertragen. Sie sind somit nicht direkt einsehbar. Es können eine große Menge an Daten versendet werden.
- Wird die Methode *GET* verwendet, werden die Daten über die URL übertragen. Die Daten sind einsehbar und die Datenmenge ist auf die maximal mögliche Länge der URL begrenzt.
- Im Attribut *action* wird das aufzurufende PHP-Skript angegeben, welches die Daten entgegen nimmt.
- Die Bezeichner der Eingabefelder dienen PHP als Variablennamen beim Empfang.
- Der Input-Typ *Submit* erzeugt eine Schaltfläche, die das Formular abschickt. Wird der Input-Typ *Reset* gewählt, wird eine Schaltfläche erzeugt, welche die Formulardaten wieder löscht und das Formular zurücksetzt.
- *value* bestimmt die Beschriftung der Schaltfläche.
- Das Attribut *name* spielt bei Schaltflächen keine Rolle.
- Wird das *size*-Attribut gesetzt, bestimmt es die Höhe bzw. Anzahl an sichtbaren Zeilen in einem Auswahlfeld.

- Wird das Attribut *value* belegt, wird der Wert von Value in das PHP-Skript übertragen und nicht die Beschriftung des Eintrages.
- Die Option *selected* bzw. *checked* wählt einen Eintrag bzw. Option aus.

Zugehöriges PHP-Skript:

```
<html>
  <head>
    <title>Übertragene Formulardaten</title>
  </head>
  <body>
    <?php
      $vorname = $_POST['vorname'];
      $name = $_POST['name'];
      $farbe = $_POST['farbe'];
      $preis_1 = $_POST['preis1'];
      $preis_2 = $_POST['preis2'];
      $gewinn = $preis_1." ". $preis_2;
      $farben = $_POST[farben];
      echo "<p>Vorname: ".$vorname."</p>\n";
      echo "<p>Nachname: ".$name."</p>\n";
      echo "<p>Farbe: ".$farbe."</p>\n";
      echo "<p>Gewinn: ".$gewinn."</p>\n";
      echo "<p>Farbauswahl: ".$farben."</p>\n";
    ?>
  </body>
</html>
```

- Mit `$_POST['variablenname']` werden die Daten „abgeholt“. Bzw. entsprechend mit *$_GET*.

Formulardaten sollten immer auf ihre Gültigkeit bzw. Korrektheit überprüft werden. Besonders wichtig ist dies beim Versand von E-Mails oder wenn Daten in eine Datenbank übernommen werden sollen. Dabei muss darauf geachtet erden, dass der Benutzer keine PHP-Befehle, SQL-Abfragen oder unerwünschte HTML-Tags eingibt. Siehe dazu die Funktionen *stripslashes und strip_tags*.

11 Arbeiten mit Dateien

Textdateien zeilenweise auslesen:

```
if (filesize($filename) > 0) {
        $fp = @fopen($filename, "r") or die ("Kann Datei nicht lesen."
            );
        while($line = fgets($fp, 1024)){
                echo ($line)."<br>";
        }
        fclose($fp);
}
```

Funktion	Beschreibung
basename(pfad)	Extrahiert Dateiname aus Pfad *pfad*.
chdir(verzeichnis)	Wechselt in das Verzeichnis *verzeichnis*.
closedir(verzeichnis_handle)	Schließt das Handle auf ein Verzeichnis.
copy(quelle, ziel)	Datei kopieren.
dirname(pfad)	Extrahiert den Verzeichnisnamen aus *pfad*.
disk_free_space(verzeichnis)	Liefert den freien Speicherplatz von Verzeichnis *verzeichnis*.
disk_total_space(verzeichnis)	Liefert die Gesamtgröße von Verzeichnisses *verzeichnis*.
fclose(datei_handle)	Schließt die Datei *datei_handle*.
feof(datei_handle)	Prüft, ob der Dateizeiger am Ende der Datei *datei_handle* steht.
fgets(datei_handle, laenge)	Liest Zeile aus Datei *datei_handle* ab den Dateizeiger bis zum Zeilenumbruch oder die mit *laenge* angegebene Anzahl Zeichen.
file(dateinamen)	Einlesen von Datei *dateiname* in ein Array.
file_exists(dateiname)	prüft, ob Datei *dateiname* oder Verzeichnis existiert.
file_get_contents(dateiname)	Liest Datei *dateiname* in eine Zeichenkette.
file_put_contents(dateiname)	Schreibt Zeichenkette in Datei *dateiname*.
fileatime(dateiname)	letzte Zugriffsdatum von *dateiname*.
filectime(dateiname)	letzte Änderungsdatum von *dateiname*.
filemtime(dateiname)	letzte dateiänderung von *dateiname*.
filesize(dateiname)	Dateigröße von *dateiname*.
filetype(dateiname)	Dateityp von *dateiname*.
fopen(dateiname, modus)	Öffnet Datei *dateiname*. „w“- schreiben, „r“ - lesen, „a“ - anhängen.
fputs(datei_handle, text[, laenge])	Schreibt *text* ab den Dateizeiger in Datei *datei_handle*. Entweder den ganzen Text oder die mit *laenge* angegeben Anzahl Zeichen.
fseek(datei_handle, offset[, whence])	Positioniert den Dateizeiger um die mit *offset* angegebene Anzahl Bytes. Wenn *whence* nicht angegeben wird vom Dateianfang. *whence*: SEEK_SET - Setzt Position gleich offset bytes, SEEK_CUR - Setzt Position auf die aktuelle Stelle plus offset, SEEK_END - Setzt die Position ans Ende der Datei plus offset (negativ).
ftell(datei_handle)	Liefert Position des Dateizeigers von *datei_handle*.
ftruncate(datei_handle)	Kürzt Datei *datei_handle*.
fwrite(datei_handle, zeichenkette[, laenge])	Schreibt *zeichenkette* in Datei *datei_handle*. *laenge* gibt die zu schreibende Anzahl Zeichen an.
mkdir(verzeichnis)	Erstellt Verzeichnis *verzeichnis*.
opendir(verzeichnis)	Öffnet Verzeichnis *verzeichnis*.
readdir(verzeichnis_handle)	Liest die nächste Datei in Verzeichnis *verzeichnis_handle*.
rename(alt, neu)	Datei umbenennen.
rmdir(verzeichnis)	Löscht Verzeichnis *verzeichnis*.
touch(dateiname)	Legt leere Datei *dateiname* an.
unlink(dateiname)	Löscht Datei *dateiname*.
write(handle, text)	Schreibt (*text)* in die mit *handle* bezeichnete Datei.

Tab. 11.1: Dateifunktionen

Literaturverzeichnis

[1] Jasmin Schmidt, Oliver Leiss: *PHP 5 - Das Einsteigerseminar*. cbhv, 1. Auflage, 2007, 3-8266-7283-6

[2] Internet: *PHP Manual*. http://www.php.net/manual, Stand: Dezember 2010