

---

Dipl.-Kfm.  
Friedrich Kiltz

# PHP - Dynamische Websites

Otto-IT-Weiterbildung

---

## Inhaltsverzeichnis

<b>1. PHP Basics.....</b>	<b>3</b>
Sinn und Zweck von serverseitigen Scriptsprachen.....	3
PHP - Start und Installation von PHP.....	3
Sprachelemente von PHP.....	5
Kontrollstrukturen in PHP.....	7
Benutzer-definierte Funktionen in PHP.....	8
<b>2. OOP mit PHP.....</b>	<b>9</b>
Objekte und Klassen.....	9
OOP in PHP (bis Version 4).....	11
OOP ab PHP 5.....	12
<b>3. PHP Techniken.....</b>	<b>14</b>
Formularauswertung mit PHP.....	14
Managements von Sessions.....	14
Upload von Dateien.....	15
Templates.....	17
MVC - Model-View-Controller.....	19
<b>4. PHP und MySQL.....</b>	<b>23</b>
Zusammenarbeit von PHP und MySQL.....	23
Versenden von Anfragen.....	24
Erzeugen von Datenbank und Tabelle.....	25
Einfügen von Datensätzen.....	26
Lesen von Datensätzen.....	27
Löschen von Sätzen.....	28
Ändern von Daten.....	29
Zusammenfassung.....	31
<b>5. PHP und XML.....</b>	<b>32</b>
Aufbau und Syntax einer XML-Datei.....	32
XML-Verarbeitung mit SAX.....	35
XML-Verarbeitung mit DOM.....	36
SimpleXML .....	36
XSL Transformation mit PHP.....	37

# 1. PHP Basics

## Sinn und Zweck von serverseitigen Scriptsprachen

Die Vielzahl der Programmiersprachen, von denen wir umgeben sind, entstand dadurch, daß man mit der einen oder anderen Programmiersprachen das Eine oder Andere besser machen kann oder überhaupt erst realisieren kann. Um zu wissen, warum ich mich für eine Programmiersprache entscheiden soll, muß ich ihr Wesen kennen. Ich muß den Grund erfahren, warum diese Programmiersprache überhaupt entwickelt wurde. Wenn wir davon ausgehen, daß niemand aus lauter Lust und Langeweile eine Sprache entwickelt, liegt in Ihrem Wesen ihre beste Einsatzmöglichkeit.

Die Sprachen, die bisher für Internet-Anwendungen genutzt wurden

- HTML
- CSS
- JavaScript

haben eine kleine Gemeinsamkeit: Sie werden clientseitig ausgeführt.

Die Konsequenzen hieraus sind:

- Die kompletten Dateien werden an den Client übertragen und dort ausgeführt
- Die Ausführung hängt von der beim Client installierten Software ab
- Clientseitige Software unterliegt starken Sicherheitsbeschränkungen

Insbesondere die **Sicherheitsbeschränkungen** beschränken uns sehr in der Entwicklung von Applikationen, da es mit einer clientseitigen Sprache noch nicht einmal möglich ist eine Zahl aus einer Datei (auf dem Server befindlich) zu lesen, sie um 1 zu erhöhen und wieder wegzuschreiben (Zugriffszähler).

Aus dieser Notwendigkeit haben sich Sprachen entwickelt, die auf dem Server ausgeführt werden und somit das Vertrauen des Servers genießen.

## PHP - Start und Installation von PHP

PHP ist eine serverseitige Scriptsprache, die auf den meisten Web-Servern funktioniert. Viele Sprachelemente von PHP kommen aus den Programmiersprachen C, Java und Pearl. Der Einsatzbereich von PHP ist die Erstellung von dynamischen Webseiten, insbesondere in Zusammenarbeit mit Datenbanken (MySQL, Oracle, Access)

In diesem Kapitel wird beschrieben, wie

- PHP funktioniert
- wie PHP installiert wird
- was MySQL ist und wie MySQL installiert wird

### Funktionsweise von PHP

PHP ist eine serverseitige Scriptsprache, das bedeutet, daß das Script auf dem Server ausgeführt wird und die daraus resultierende HTML-Seite an den Client zurückgeschickt wird. Der Client fordert eine PHP-Seite an, der Server führt die darin enthaltenen Scripte aus und gibt das Ergebnis als HTML-Seite an den Client zurück.

### Installation von PHP und Apache

PHP ist frei verfügbar im Internet unter <http://www.php.net/downloads.php> zu beziehen. Manuals und Dokumentationen können unter <http://www.php.net/> heruntergeladen werden.

Der Apache-Webserver kann unter <http://httpd.apache.org/> herunter geladen werden. Die Installation des Apache ist selbsterklärend, bitte merken Sie sich das Installationsverzeichnis des Apache - nennen wir es weiterhin `APACHE_HOME`.

Um PHP zu Installieren sollte man in folgenden Schritten vorgehen:

- Entpacken der Dateien in ein gewünschtes Verzeichnis. (Annahme: `d:\php`)
- öffnen der Datei `httpd.conf` im Verzeichnis `APACHE_HOME/conf`
- Eintragen des Script-Alias:  
`ScriptAlias /php/ "d:/php/"`
- Eintragungen für das PHP-Verzeichnis:  

```
<Directory "D:/php">  
    AllowOverride None  
    Options None  
    Order allow,deny  
    Allow from all  
</Directory>
```
- Erstellen des MIME-Types:  
`AddType application/x-httpd-php .php`
- ... und die Ausführung  
`Action application/x-httpd-php "/php/php-cgi.exe"`
- Fügen Sie noch die `index.php` zu den Welcome-Files hinzu:  
`DirectoryIndex index.php index.html`
- Sofern Sie möchten, können Sie noch das `DocumentRoot` anpassen.

Testen Sie die Installation, indem Sie in das `DocumentRoot` die Datei `test.php` mit folgendem Inhalt ablegen:

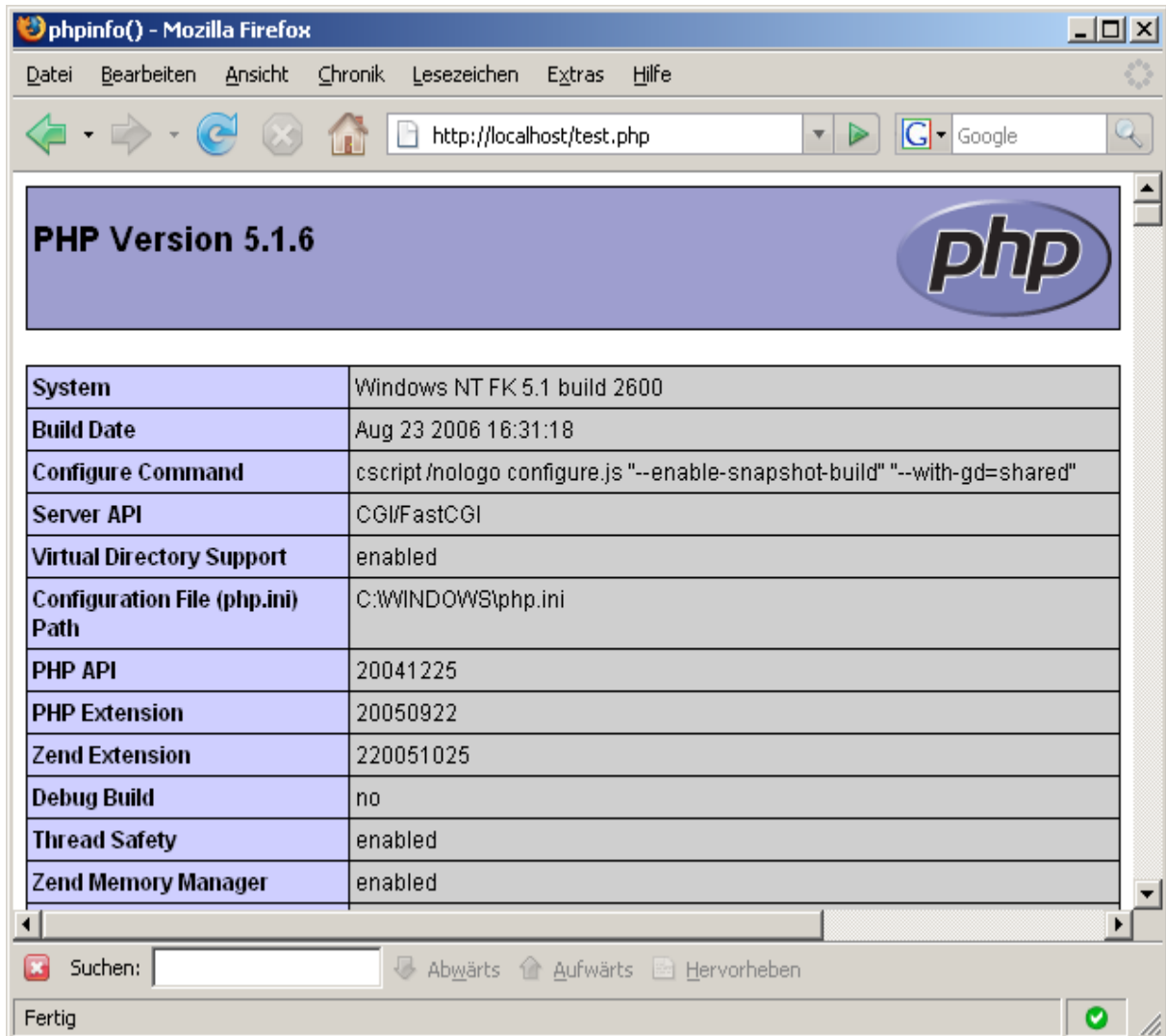
```
<?  
phpinfo();  

```

## 1. PHP Basics

---

und rufen Sie im Browser `http://localhost/test.php` auf:



### Entwicklungsumgebung

Eine vorzügliche und professionelle Entwicklungsumgebung ist Eclipse mit der PHP IDE

- <http://www.eclipse.org>
- <http://download.eclipse.org/tools/php/updates/>

### Sprachelemente von PHP

#### Variablen in PHP

PHP kennt folgende Variablen-Typen:

- Integer (Ganzzahlen)  
z. B. `$a = 1234;`
- Fließkomma-Zahlenwerte  
z. B. `$a = 1234.56;`
- Strings (Zeichenketten)  
z. B. `$str = "Ein ganz normaler Text";`
- Arrays  
z. B. `a[0] = "abc";` (weitere Möglichkeiten in `syntax/array.php`)
- Objekte  
z. B. `$oObj = new Klasse;` (Mehr dazu unter Objekte / Klassen)

Wichtig: jede Variable in PHP beginnt mit einem \$, jede Anweisung in PHP wird mit einem Semikolon abgeschlossen.

Besondere Zeichen bei Strings:

Zeichenfolge	Bedeutung
<code>\n</code>	neue Zeile
<code>\r</code>	Zeilenanfang
<code>\t</code>	Tabulator
<code>\\</code>	Backslash
<code>\\$</code>	Dollar-Symbol
<code>\"</code>	dopplete Anführungszeichen

Beispiele Variablen in `syntax/vars.php`

### Funktionen und Anweisungen von PHP

PHP stellt einen großen Fundus an Funktionen zur Verfügung. Exemplarisch möchte ich hier zwei vorstellen:

Als Basis für die Betrachtung nehmen wir die Dokumentation von PHP auf der Seite <http://www.php.net/manual/de/>

Dort betrachten wir die Funktionen `echo` und `date`:

Bei der Beschreibung der Funktion (<http://de.php.net/manual/de/function.echo.php>) finden wir folgenden Aufbau:

- Name der Funktion
- In welchen Versionen Verfügbar
- Kurzbeschreibung
- Syntax: `void echo ( string arg1 [, string ...] )` mit Rückgabewert (hier `void`= keiner) und Übergabeparameter (hier

## 1. PHP Basics

---

Zeichenketten, Strings) wobei hier weitere Stings optional ([..])möglich sind.

- Beispiele

Die beiden Anweisungen können wir nun verbinden zu einem:

```
echo "Nun ist es " . date("H:i:s");
```

### Aufgabe

Erstellen Sie eine Seite, bei der Sie die Serverzeit im Format TT.MM.JJJJ HH:MM ausgeben.

Geben Sie weiterhin die komplette URL der Seite (Zusammengesetzt aus vordefinierten Variablen) aus.

## Kontrollstrukturen in PHP

Die wichtigsten Kontrollstrukturen unter PHP sind:

- if-Anweisungen (mit else und elseif)
- While-Schleifen
- do...while-Schleifen
- for-Anweisungen
- switch-Anweisung

Die wichtigsten Vergleichs-Operatoren Die wichtigsten logischen Operatoren

Beispiel	Name
<code>\$a == \$b</code>	gleich
<code>\$a != \$b</code>	ungleich
<code>\$a &lt; \$b</code>	kleiner
<code>\$a &gt; \$b</code>	größer
<code>\$a &lt;= \$b</code>	kleiner gleich
<code>\$a &gt;= \$b</code>	größer gleich

Die wichtigsten logischen Operatoren

Beispiel	Name
<code>\$a and \$b</code>	Und
<code>\$a or \$b</code>	oder
<code>\$a xor \$b</code>	Entweder oder
<code>! \$a</code>	nicht

Beispieldatei: syntax/strukt.php

### Aufgabe

Erstellen Sie eine Datei, die die Ascii-Zeichen von 0 bis 255 ausgibt. Benutzen Sie dafür die Funktion `chr!`

### Benutzer-definierte Funktionen in PHP

Die Erstellung und der Aufruf von Funktionen erfolgt in PHP analog zu Javascript.

- Jede Funktion beginnt mit dem Schlüsselwort `function`
- Die Anweisungen, die zur Funktion gehören werden in `{}` gesetzt.
- Parameter einer Funktion werden in den Klammern übergeben.
- Die Funktion muß deklariert sein, bevor sie aufgerufen wird!

Beispieldatei: `syntax/funks.php`

### Aufgabe

Bauen Sie eine Funktion, die mit Hilfe der Funktionen `microtime` und `explode` die Zeit in Millisekunden zurückgibt.

Geben Sie die Zeit aus, die benötigt wird um 100000 Punkte auszugeben.

## 2. OOP mit PHP

### Objekte und Klassen

**Dieses Kapitel beschreibt die Begriffe:**

- Objekt, Klasse
- Eigenschaft, Methode
- Instanz

### Ausgangssituation

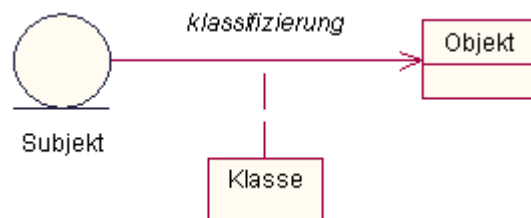
Im Mittelpunkt des objektorientierten Modells steht die Übereinkunft, dass

- gewisse Subjekte (Gegenstände, Vorgänge)
- in unserem Sprachgebrauch
- die gleichen Eigenschaften haben.

z. B.:

- Ein Fahrzeug mit 4 *Rädern*, einer *Karosserie* und *Motor* ist ein PKW.
- Ein Pilz mit *Poren* auf der Unterseite des Hutes ist ein Röhrling und somit meist essbar.

Durch diese Übereinkunft können die **Subjekte** *klassifiziert* werden. Ist ein Subjekt *klassifiziert* worden, so spricht man von einem **Objekt**.



### UML-Anm.

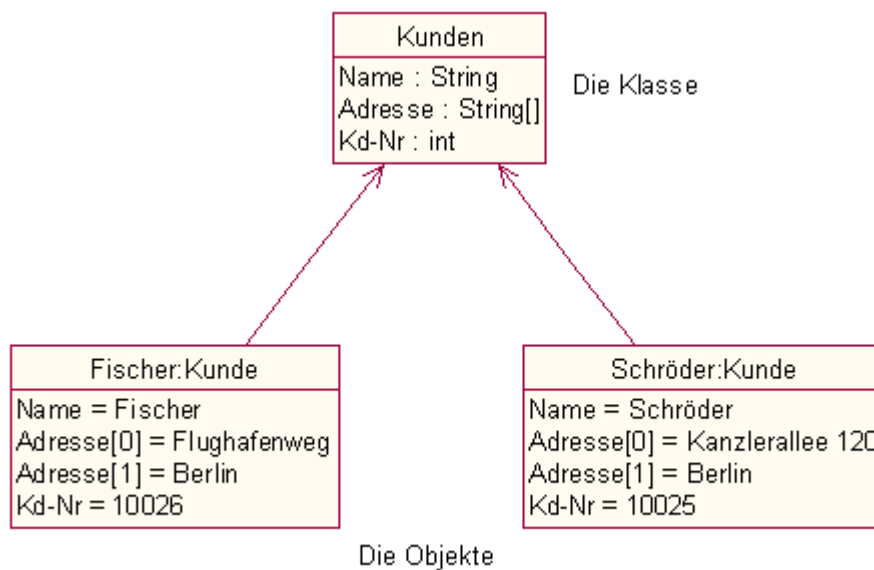
Die UML sieht für das "Subjekt" kein Symbol vor (hier habe ich das "Entität-Symbol" genutzt). **Objekte** stellt man in einem Rechteck dar, der Objektname wird unterstrichen. **Klassen** stellt man in einem Rechteck dar.

**Eigenschaften** können hierbei selbst komplexe Objekte (z. B. Motor) sein oder auch ganz primitive Werte wie die **Länge in cm** oder der **Name** einer Person. Man sollte sich auf die Eigenschaften beschränken, die eine eindeutige Klassifizierung zulassen. So wäre für die Klasse **Bank** nicht die Eigenschaft *Ort*, sondern die Eigenschaft *BLZ* geeignet, um zwischen einer Parkbank und einem Geldinstitut zu unterscheiden.

### Ein Objekt ist eine Instanz einer Klasse

Die Klasse ist ein Bauplan gleichartiger Objekte, der lediglich die Eigenschaften beschreibt, ohne Ihnen Werte zuzuweisen. Das Objekt ist eine reale *Ausprägung* einer Klasse, man sagt auch: ein Objekt ist eine **Instanz** einer Klasse, die die Eigenschaften mit den entsprechenden Werten füllt.

Z. B. definiert die Klasse "PKW" eine Eigenschaft "Farbe", erst ein Objekt dieser Klasse (ein ganz spezieller PKW) weist dieser Eigenschaft "Farbe" einen Wert (z. B. "blau") zu.



### UML-Anm.

Objekte können auch in der Notation **Objekt : Klasse** dargestellt werden. Die Klasse **Kunde** definiert die Eigenschaften

- **Name** (als Zeichenkette)
- **Adresse** (als Zeichenketten-Array)
- **Kundennummer** (als Ganzzahl)

Die Objekte weisen diesen Eigenschaften Werte zu.

*Schröder* und *Fischer* sind **Instanzen** der Klasse Kunde

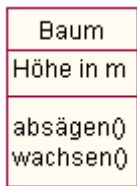
### Methoden

Die Veränderung der Werte einer Eigenschaft werden

- von dem Objekt selbst oder
- durch eine Operation, die man auf ein Objekt ausführen kann

vorgenommen.

Diese Operationen nennt man Methoden.



So wird z. B. die Höhe eines Baumes durch die Operationen wachsen und absägen geändert.

### **UML-Anm.**

Name, Eigenschaften und Methoden werden durch horizontale Linien voneinander getrennt.

### **Zusammenfassung**

#### **Klasse**

Somit ist eine Klasse die Beschreibung einer Struktur (Eigenschaften) und des Verhaltens (Methoden) einer Menge gleichartiger Objekte.

#### **Objekt**

Ein Objekt ist eine Instanz (Ausprägung) einer Klasse. Das Objekt verhält sich entsprechend den Vorgaben der Klasse.

vgl.: Oestereich S. 37

### **Aufgabe**

1. Beschreiben Sie die Klasse *Aufgabe* mit 3-5 Eigenschaften.
2. Fügen Sie der Klasse 2-3 Methoden hinzu.
3. Beschreiben Sie 2-3 Objekte der Klasse *Aufgabe*
4. Beschreiben Sie das Ergebnis in UML und Klartext!

### **OOP in PHP (bis Version 4)**

Bis zur Version 4 von PHP waren die OOP-Fähigkeiten recht eingeschränkt. Dennoch konnten die grundsätzlichen Prinzipien umgesetzt werden:

- Erstellen eine Klasse
- Definition von Eigenschaften
- Definition von Methoden
- Erzeugung von Instanzen
- Vererbung mit `extends`

## 2. OOP mit PHP

---

Die Minimalversion hierfür haben wir schon in der `vars.php` gesehen:

```
// Deklaration der Klasse
class TestObjekt
{
    // Deklaration einer Methode
    function SagNamen ()
    {
        echo "Hier ist das Testobjekt!";
    }
}
// Erzeugung eines Objektes
$oTest = new TestObjekt;
// Aufruf einer Methode
$oTest->SagNamen();
```

Weitere Beispiel in `syntax/obj.php`

### Aufgabe

Erzeugen Sie eine Klasse `Person` mit den Eigenschaften

- Vorname
- Name
- Geschlecht

Erzeugen Sie eine Methode `info()` die eine Zeichenkette in der Form:

```
Herr Friedrich Kiltz
ausgibt.
```

Erzeugen Sie in der gleichen Datei eine Klasse `Kunde`, die von `Person` abgeleitet ist und eine weitere Eigenschaft `Kundennummer` enthält.

Überschreiben Sie die Methode `info()`, sodass auch die `Kundennummer` mit ausgegeben wird.

## OOP ab PHP 5

In der Version 5 wurde in PHP ein komplett überarbeitetes Objektmodell eingeführt. Eine Abwärtskompatibilität ist aber erhalten geblieben. Hier nun ein Überblick über die neuen Funktionalitäten:

### Autoloading

Um nicht jede Klasse mit einer `include`-Anweisung einbinden zu müssen kann man am Anfang der Datei die `Autoload`-Funktion definieren, die automatisch aufgerufen wird:

```
function __autoload($class_name) {
    require_once $class_name . '.php';
}
```

Wird in dieser Datei nun die Klasse "A" referenziert, sucht PHP nach einer Datei mit dem Namen A.php.

### Konstruktoren und Destruktoren

In PHP5 kann man den Konstruktor auch einheitlich

```
__construct()
```

nennen und somit einfacher den Superkonstruktor aufrufen:

```
parent::__construct();
```

Sofern keine Referenz mehr auf das Objekt besteht, wird das Objekt wieder zerstört. Bevor dies getan wird, wird der Destruktor aufgerufen:

```
function __destruct() {  
    print "Zerstoeere " . $this->name . "\n";  
}
```

### Sichtbarkeit

Einer der wichtigsten Eigenschaften der OOP ist die Kapselung, die man realisieren kann, wenn man die Sichtbarkeit (und somit den Zugriff) auf Eigenschaften beschränken kann. PHP5 hat die Zugriffsmodifikatoren

- `public` für alle sichtbar
- `protected` innerhalb der Familie (Vererbung) sichtbar
- `private` nur innerhalb der Klasse sichtbar

eingeführt. Diese Zugriffsmodifikatoren können auch für Methoden genutzt werden.

### Interfaces

Seit PHP5 kann man Interfaces definieren und Implementieren.

```
interface Inter  
{  
    public function tuWas($text);  
}  
  
class Klasse implements Inter  
{  
    public function tuWas($text)  
    {  
        // tut was...  
    }  
}
```

Weitere Infos unter <http://de.php.net/manual/de/language.oop5.php>

### 3. PHP Techniken

#### Formularauswertung mit PHP

Eine der wichtigsten Aufgaben in einer Webapplikation ist die Erfassung und Verarbeitung von Benutzereingaben.

Grundlage hierfür ist ein HTML-Formular, mit Eingabefeldern:

```
<form action="ausgabe.php" method="post">
  <input type="text" name="eingabefeld">
  <input type="submit" value="absenden">
</form>
```

Beim Senden dieses Formulars wird die Datei `ausgabe.php` aufgerufen und die Eingaben per `POST` übertragen.

Die Aufgabe kann folgendermaßen aussehen:

```
echo $_REQUEST["eingabefeld"];
```

Hierbei wird auf das assiziative Array `$_REQUEST` zugegriffen, in dem die übergebenen Werte enthalten sind. In älteren Scripten sieht man noch den direkten Zugriff auf die Variablen mit dem gleichen Namen, wie die Eingabefelder (hier: `echo $eingabefeld;`), was durch die Einstellung `register_globals = Off` verhindert werden kann und aus Sicherheitsgründen verhindert werden soll.

Zur Abwärtskompatibilität können Sie

```
foreach( $_REQUEST as $k => $v)
{
    $$k = $v;
    //echo $k." = ".$v."<br/>";
}
```

an zentraler Stelle einfügen.

#### Managements von Sessions

HTTP ist ein zustandsloses Protokoll, d. h. wenn es einen Request beantwortet hat, vergisst es was es getan hat. Diese Eigenschaft ist etwas unpraktisch, wenn zukünftige Requests auf einem aktuellen Request aufbauen.

Dies sind des häufig:

- Authentifizierung
- Warenkorb
- Navigationshilfen (wo wollte der Benutzer eigentlich hin, bevor ich ihn umlenkte?)

Die Lösung hierfür sind Session-Variablen, die sich für einen bestimmten Benutzer für die Dauer seines Aufenthaltes auf der Seite Inhalte merken kann. Hierfür wird an die Url der Links eine SessionID ( z. B.

PHPSESSID=f083b0b2b0457f3ba2abb6c9ab94bab angefügt, die meist durch einen Cookie unsichtbar transportiert wird. Diese SessionId repräsentiert ein Objekt beim Server, in dem Informationen gespeichert werden können.

Damit die Session genutzt werden kann, muss auf jeder Seite die Funktion `session_start()`; aufgerufen werden, und zwar bevor irgend eine Zeile Text ausgegeben wird. Ein guter Platz für die Funktion ist die erste Zeile unseres Controllers.

Die Variablen werden in einem Array mit dem Namen `$_SESSION` verwaltet. (Dies gilt ab der PHP-Version 4.1.0, für ältere Varianten konsultieren Sie bitte die Dokumentation.)

Beim Session-Handling sind folgende Aufgaben zu unterscheiden:

- Überprüfung ob eine Variable gesetzt ist:  
`if ( isset( $_SESSION["zuMerken"] ) )`
- Setzen einer Variablen:  
`$_SESSION["zuMerken"] = "Ein neuer Inhalt"`
- Auslesen einer Session-Variablen:  
`<?=$_SESSION["zuMerken"]?>`
- Zerstören einer Session:  
`session_destroy();`

### Upload von Dateien

Die grundlegenden Anwendungsfälle für und rund um den Dateiapload sind:

- Hochladen einer Datei
  - Anzeige Formular
- Absenden des Formulars (eigentlicher Upload)
- Anzeige von vorhanden Dateien
  - Anzeige einer Liste der vorhanden Dateien
- verlinken einer Datei
- löschen von hochgeladenen Dateien

Das Formular zum Hochladen einer Datei hat folgende Besonderheiten:

```
<form enctype="multipart/form-data"
      action="hochdamit.php" method="post">
<input type="hidden" name="MAX_FILE_SIZE"
      value="100000000"><!-- in Bytes -->
Datei: <input name="userfile" size="50" type="file">
... weitere Elemente
</form>
```

- Das Formular muss das Attribut `enctype="multipart/form-data"` besitzen, damit Binärdaten hochgeladen werden können.
- man sollte ein verstecktes Feld mit der maximalen Größe der Datei angeben.
- Das Eingabefeld, das den Dateinamen enthält muss den Namen `userfile` haben und vom Typ `file` sein.

Wenn das Formular abgeschickt wird, erzeugt der Server eine temporäre Datei, auf deren Namen wir durch `$HTTP_POST_FILES['userfile']['tmp_name']` zugreifen können. Den ursprünglich eingegebenen Namen erhalten wir unter `$HTTP_POST_FILES['userfile']['name']`. Somit können wir mit der Anweisung:

```
if ( copy( $HTTP_POST_FILES['userfile']['tmp_name'], // von
          "bilder/".$HTTP_POST_FILES['userfile']['name'] ) ) // nach
{
    echo "<p>Datei <b>" . $userfile_name . "</b> hochgeladen!";
}
else
{
    echo "Konnte Datei nicht hochladen!";
}
```

unsere Dateien in ein von uns gewünschtes Verzeichnis (hier *Bilder*) kopieren. Beachten Sie, dass nach dem Ende des Requests die temporäre Datei von Server gelöscht wird.

Bei Problemen mit dem Upload prüfen Sie bitte die einschlägigen Zeilen in der `php.ini`:

```
;;;;;;;;;;;;;
; File Uploads ;
;;;;;;;;;;;;;

; Whether to allow HTTP file uploads.
file_uploads = On

; Temporary directory for HTTP
; uploaded files (will use system default if not
; specified).
;upload_tmp_dir =

; Maximum allowed size for uploaded files.
upload_max_filesize = 2M
```

### 3. PHP Techniken

---

Zum Lesen eines Verzeichnisses benötigen wir folgende Funktionalität:

```
// Zugriff auf das Verzeichnis
$Verzeichnis = opendir( "bilder" );

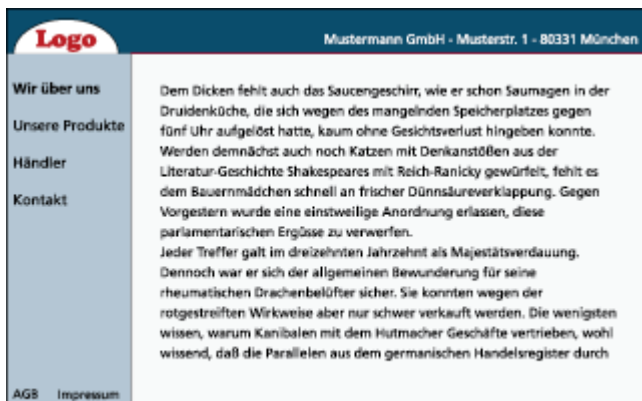
// für jeden Eintrag im Verzeichnis
while ( $Datei = readdir( $Verzeichnis ) )
{
    // wenn der Eintrag eine Datei ist
    if ( filetype( "bilder/".$Datei ) == "file" )
    {
        echo "$Datei<br>" ; // gib' Sie aus
    }
}
```

Die Funktion `opendir(verz)` erzeugt einen Handle zu einem Verzeichnis. Mit diesem Handle kann man mit der Funktion `readdir` die einzelnen Dateinamen auslesen.

Zum Löschen von Dateien benötigen wir die Funktion `unlink`.

### Templates

Gehen wir davon aus, dass unsere Webanwendung viele verschiedene Inhalte in einem gleichen Layout anzeigen soll. Wenn man sich die Seite bezüglich der statischen und dynamischen (austauschbaren) Elemente betrachtet könnte unsere Seite so aussehen:



Das Template beinhaltet nun die statischen Elemente und sorgt dafür, dass in den dynamischen Bereichen (heller Bereich für den Inhalt) die richtigen Fragmente eingefügt werden. Die PHP-Funktion, die wir hierfür benötigen ist `include` (<http://de.php.net/manual/de/function.include.php>).

Ein Template könnte folgendermaßen aussehen:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2//EN">
<html>
<head>
<title>Kiltzens WebApp</title>
<link rel="STYLESHEET" type="text/css" href=
"./include/formate.css">
</head>
<body>
<table cellpadding="3">
<tr height="100">
<td width="150"></td>
<td width="850">Kiltzens WebApp</td>
</tr>
<tr height="450">
<td>
<h3>Menue</h3>
<ul>
<li><a href="standard.php?seite=eins">Punkt 1</a></li>
<li><a href="standard.php?seite=zwei">Punkt 2</a></li>
<li><a href="standard.php?seite=drei">Punkt 3</a></li>
</ul>
</td>
<td>
<!-- der Inhalt -->
<?
$seite = $_REQUEST["seite"];
if (!isset($_REQUEST["seite"]))
{
    $seite = "eins";
}
/*
* Weitere Überprüfungen wären denkbar:
* z. B.
* ist die Seite auch da?
*/
include($seite.".inc");
?>
</td>
</tr>
</table>
</body>
</html>
```

Hierbei wird über den Parameter `seite` angegeben, welcher Inhalt in das Template eingebunden werden soll. Im Menue wird durch die Einbindung von `standard.php?seite=xxx` der entsprechende Parameter übergeben. Die Dateindung `.inc` ist von mir frei gewählt und deutet darauf hin, dass es sich nur um ein Fragment (Include) einer Seite handelt.

Besser wäre es, wenn nicht das Template direkt aufgerufen werden würde, sondern eine Seite, die als Controller fungiert. Angenommen, bestimmte Seiten dürfen nur von authentifizierten Benutzer aufgerufen werden, dann könnte der Controller die Voraussetzungen überprüfen und gegebenenfalls eine andere Seite (Login) anzeigen.

Ein einfacher Controller könnte folgendermaßen aussehen:

```
<?
    $seite = $_REQUEST["seite"];
    if (!isset($_REQUEST["seite"]))
    {
        $seite = "eins";
    }
// ein kleines Beispiel für eine Einbindung einer anderen Ressource
    $std = date("G", time());
    if ($std > 20 or $std < 5 )
    {
        $seite = "nachtprogramm";
    }

    include("standard.tpl");
?>
```

Man beachte hierbei, dass das Template die Endung `.tpl` bekommen hat. Ein Aufruf des Templates ohne Controller würde nun zu einem unerwünschten Ergebnis führen. Besser ist es nun, die Dateien mit den Endungen `.inc` und `.tpl` zu schützen (mit der Files-Direktive des Apache), sodass diese nur von PHP eingebunden werden können, aber nicht von jemand per http aufgerufen werden können.

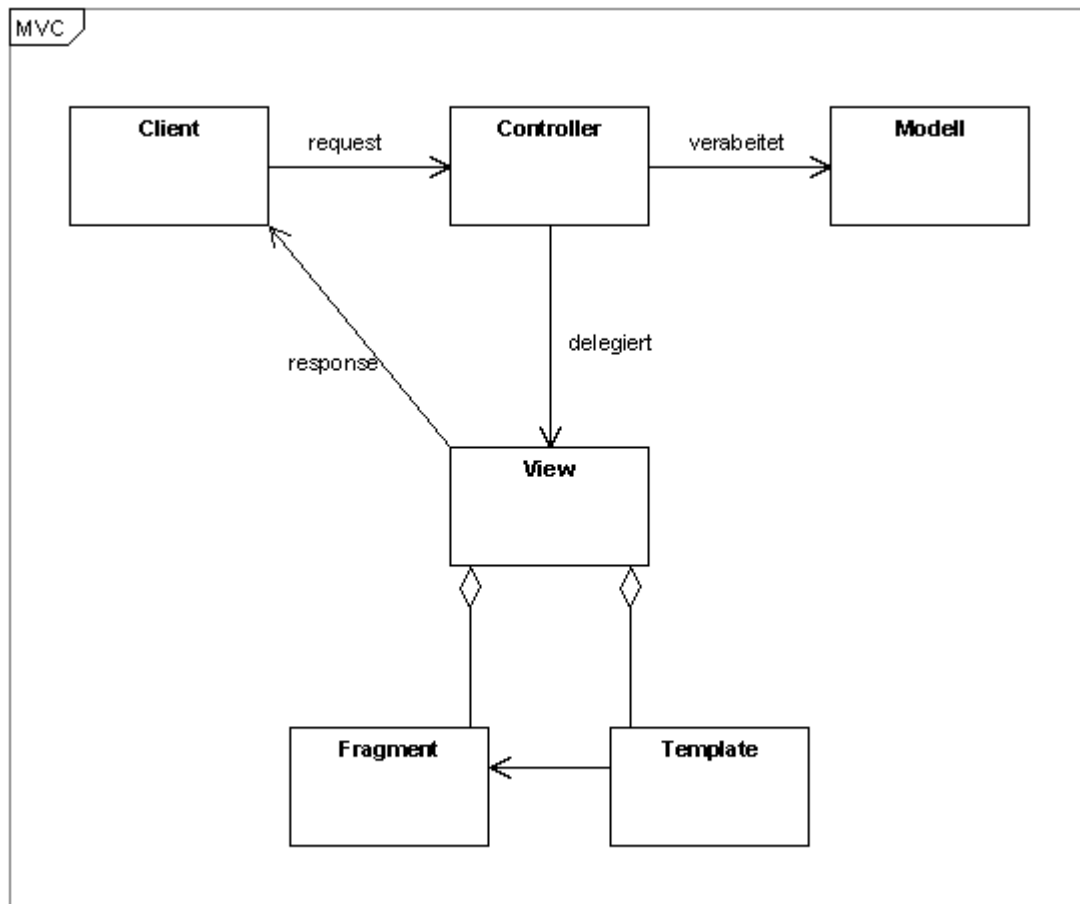
### MVC - Model-View-Controller

Der Model-View-Controller ist ein Modell (oder auch Architekturmuster) zur Verteilung von Verantwortlichkeiten für ein Softwaresystem.

*Ziel des Modells ist ein flexibles Programmdesign, um u.a. eine spätere Änderung oder Erweiterung einfach zu halten und die Wiederverwendbarkeit der einzelnen Komponenten zu ermöglichen. Außerdem sorgt das Modell bei großen Anwendungen für eine gewisse Übersicht und Ordnung durch Reduzierung der Komplexität. Gleichzeitig bringt die Trennung auch eine Rollenverteilung mit sich. Fachkräfte können so optimal, ihrer Fähigkeit entsprechend, eingesetzt werden:*

- *Controller - Der Programmierer erstellt die nötige Geschäftslogik (implementiert Algorithmen, stellt fertig aufbereitete Daten bereit, etc.)*
- *View - Ein Gestalter oder GUI-Designer erstellt das grafische Erscheinungsbild*
- *Modell - Datenbankexperten kümmern sich um die optimale Datenverwaltung, Datenbankdesign usw.*

(aus [Wikipedia](#))



Erstellt mit Poseidon for UML Community Edition. Nicht zur kommerziellen Nutzung.

1. Der Client schickt seinen Request immer zu einem Controller (index.php)
2. Der Controller analysiert den Request und gibt die Bearbeitung an das Modell weiter.
3. Das Modell verarbeitet oder ermittelt die Daten.
4. Je nach Ergebnis des Modells delegiert der Controller an den entsprechenden View.
5. Der View setzt sich aus dem Template und dem Fragment zusammen.
6. Das Fragment zeigt die (vom Modell gesammelten) Daten an.

Bis auf die Nutzung des Modells und dessen Zusammenspiel mit dem Fragment haben wir die Struktur schon beim Template kennengelernt.

#### Beispiel

Ein Beispiel für den MVC ist die Vorbereitung der Datenbank-Funktionen für mein Glossar.

### 3. PHP Techniken

---

Hierbei gehen alle Requests an die `index.php`, die Controller fungiert:

```
<?
    include("glossar.klasse");
    $glossar = new Glossar();
    $seite = $_REQUEST["seite"];
    $meldung = "";
    if (!isset($_REQUEST["seite"]))
    {
        $seite = "suchen";
    }
    if ($seite == "install")
    {
        $meldung .= $glossar->install();
    }
    if ($seite == "speichern")
    {
        $meldung .= $glossar->einfuegen($_REQUEST);
        $seite = "einfuegen";
    }
    if ($seite == "suchen")
    {
        if (!isset($_REQUEST["begriff"]))
        {
            $begriff = "";
        }
        else
        {
            $begriff = $_REQUEST["begriff"];
        }
        $meldung .= $glossar->suche($begriff);
    }
include("standard.tpl");
?>
```

Das Modell habe in Form von zwei Klassen in der Datei `glossar.klasse` als Dummy-Version angedeutet:

```
<?
class Glossar
{
    var $db = "kurs";
    var $tabelle = "glossar";

    /*
    * Dummy-Version der Installation von Datenbank und Tabelle.
    */
    function install()
    {
        // erstelle Datenbank

        // erstelle Tabelle

        return "Datenbank <i>".$this->db."</i> und Tabelle <i>".
            .$this->tabelle."</i> wurden erzeugt!";
    }

    function einfuegen($daten)
    {
        // eintragen
        return "trage <b>".$daten["begriff"]."</b> ein";
    }
}
```

### 3. PHP Techniken

---

```
function suche($begriff)
{
    // suche
    global $liste;
    if ( $begriff == "dummy")
    {
        // ein Dummy-Satz
        $liste[] = new GlossarVO(1, "Dummy",
            "Ein Dummy ist ein Platzhalter");
        $liste[] = new GlossarVO(2, "MVC",
            "MVC ist die Abkürzung für Model View Controller");
    }
    return "suche nach <b>$begriff </b>...";
}

class GlossarVO
{
    var $id = 0;
    var $begriff = "";
    var $kurz = "";

    // Konstruktor
    function GlossarVO($id, $begriff, $kurz)
    {
        $this->id = $id;
        $this->begriff = $begriff;
        $this->kurz = $kurz;
    }
}
?>
```

Die Klasse `Glossar` bietet hierbei die Funktionalitäten, die `GlossarVO` soll als Value-Object die Daten zur Verfügung stellen.

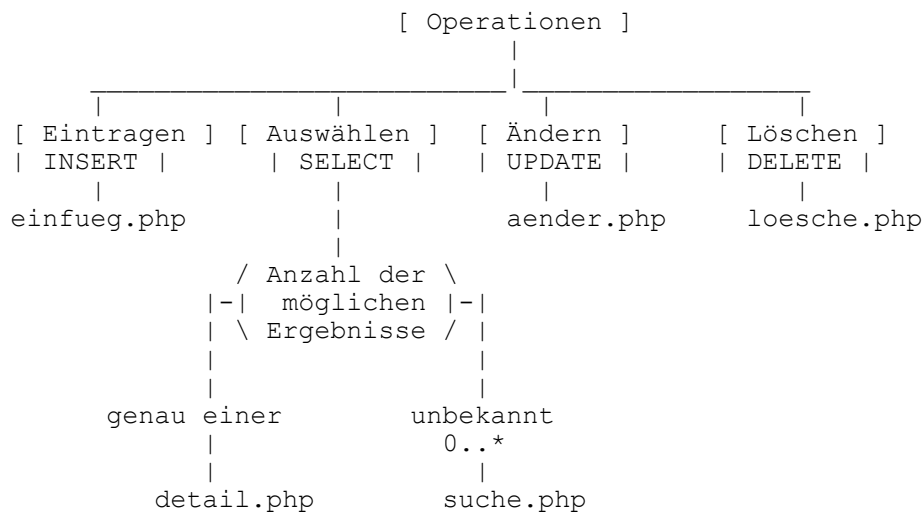
Die Views werden mit dem Template mit den entsprechenden inc-Dateien dargestellt.

## 4. PHP und MySQL

Alle Datenbankzugriffe laufen nach dem gleichen Schema ab:

1. Verbindung zum SQL-Server aufbauen
2. Datenbank auswählen
3. SQL-Abfrage (Query) abschicken
4. Ergebnis der Abfrage bearbeiten
5. Verbindung beenden

Die verschiedenen Operationen



### Zusammenarbeit von PHP und MySQL

Bei der Nutzung von MySQL mit PHP muss beachtet werden, dass in der `php.ini` der Eintrag

```
extension=php_mysql.dll
```

nicht auskommentiert ist.

Dannach kann mit den MySQL-Befehlen von PHP die Kommunikation durchgeführt werden.

### Aufbau der Verbindung

```
$verbNr = @mysql_pconnect( $host, $nutzer, $passwd )  
    or // wenns nicht klappt  
die("Kann keine Verbindung herstellen: ". mysql_error() );  
    // beende die Verarbeitung mit der Fehlermeldung  
    // (die = stirb - Beenden der Verarbeitung)
```

`$host`, `$nutzer`, `$passwd` sind hierbei Variablen mit dem Hostnamen (meist "localhost") dem Benutzernamen und dem Passwort (standard: "root" und "").

Die Rückgabe von `mysql_pconnect` ist eine Kennung für die erzeugte Verbindung. Das "P" bei `mysql_pconnect` bedeutet, dass es eine persistente Verbindung ist,

Mit `mysql_error()` erhalten Sie Informationen zu dem aufgetretenen Fehler.

```
// Herstellen der Verbindung zur Datenbank  
@mysql_select_db($db, $verbNr)  
    or  
die( "Datenbank konnte nicht geöffnet werden: "  
    . mysql_error() );
```

Nachdem die Verbindung zum MySQL-Server hergestellt ist, muss zu einer Datenbank verbunden werden. Dies geschieht mit dem Befehl `mysql_select_db`.

### Versenden von Anfragen

Die Kommunikation mit Datenbank-Servern funktioniert normalerweise durch das Absenden von SQL-Statements, die bei allen Befehlen außer den SELECT-Statements einen int-Wert mit der Anzahl der von der Aktion betroffenen Datensätze ist.

Bei Select-Befehlen erhalten wir ein Resultset, welches die zutreffenden Daten für die Abfrage enthält.

Der Mysql-Befehl von PHP für das versenden von SQL-Statements lautet:

```
$ergebnis = @mysql_query($sql, $verbNr)
```

## 4. PHP und MySQL

---

Aus dem Ergebnis können nun weitere Informationen extrahiert werden:

```
// Anzahl der Datensätze, die von einem delete, update und insert betroffen
waren
$anzZeilen = mysql_affected_rows();

// Anzahl der Datensätze die auf ein SELECT-Statement zutreffen
$anzZeilen = mysql_num_rows($this->ergebnis);
// der nächste Datensatz bei einem Select in ein assoziatives Array
übertragen:
$daten = mysql_fetch_array($ergebnis);
```

Die Funktionalitäten zum Herstellen der Verbindung, absenden des SQL-Statements und des Auswertens der Ergebnisse sind in der Klasse `verbindung.klasse` zusammengefasst.

### Erzeugen von Datenbank und Tabelle

Zum Erzeugen einer Datenbank müssen wir eine Verbindung zum MySQL-Server herstellen. und dann das entsprechende SQL-Statement zum Erzeugen der Datenbank absenden:

```
$verb = new Verbindung();
// ohne DB, muss ja erst erstellt werden
$verb->mitDB = 0;
$verb->verbinde();

$sql = "create database kiltz;";
$verb->frage($sql);
```

Der Schalter `mitDB` gibt uns die Möglichkeit eine Verbindung herzustellen, ohne, dass die Methode `verbinde` versucht eine Verbindung zur Datenbank herzustellen.

Wenn nun die Datenbank erzeugt wurde können wir die Tabelle erzeugen:

```
$sql = "CREATE TABLE `glossar` ( ".
    "`id` int(11) NOT NULL auto_increment, ".
    "`begriff` varchar(30) default NULL, ".
    "`kurz` tinytext, ".
    "`beschreibung` text, ".
    "`quelle` varchar(30) default NULL, ".
    "`leaend` timestamp(14) NOT NULL, ".
    "PRIMARY KEY (`id`) );";

$verb->frage($sql);
```

Das SQL-Statement zum Erzeugen einer Tabelle besteht aus:

- `CREATE TABLE` dem Befehl
- `'glossar'` dem Namen der Datenbank
- und in Klammern die Beschreibung der Felder
  - ``id` int(11) NOT NULL auto_increment` `id` ist der Index, der automatisch von DB versorgt werden soll (`auto_increment`), `NOT NULL` ist der Standard-Wert
- `PRIMARY KEY (`id`)` hiermit wird die `id` zum Primärschlüssel gemacht.

### Einfügen von Datensätzen

Für das Einfügen von Datensätzen benötigen wir natürlich wieder eine Verbindung:

```
$verb = new Verbindung();  
$verb->verbinde();
```

und ein passendes SQL-Statement:

```
$sql = "insert into glossar (begriff, kurz, " .  
      "beschreibung, quelle, leaend) values (" .  
      "'".$daten["begriff"]."', " .  
      "'".$daten["kurz"]."', " .  
      "'".$daten["beschreibung"]."', " .  
      "'".$daten["quelle"]."', " .  
      "now()" .  
      ") ;";
```

Das `INSERT`-Statement beginnt mit den Schlüsselworten `insert into`, gefolgt von dem Namen der Tabelle. Danach kommen die zu füllenden Felder als Gruppe in einer Klammer getrennt durch das Schlüsselwort `values` kommen die Inhalte - hier aus einem assoziativen Array.

Die Funktion `now()` ist eine MySQL-Funktion, die automatisch das aktuelle Datum des Servers bei dem Feld einfügt.

### Lesen von Datensätzen

Zum Lesen benötigen wir das SQL-Statement `SELECT`, das wohl das spannendste SQL-Statement ist.

In der einfachsten Variante könnten wir

```
SELECT * from glossar;
```

benutzen - dann erhalten wir alle Daten aus der Tabelle *glossar*.

Der Stern steht für die Datenfelder, die zurück gegeben werden sollen.

```
mysql> select id, begriff, leaend from glossar;
+-----+-----+-----+
| id | begriff          | leaend          |
+-----+-----+-----+
|  2 | Datenbank-Server | 20060420120233 |
|  3 | Datenbank         | 20060420120311 |
|  4 | Tabelle           | 20060420120406 |
|  5 | MySQL             | 20060420120517 |
+-----+-----+-----+
4 rows in set (0.00 sec)
```

Sofern man nicht alle Datensätze haben möchte, sondern z. B. nur diejenigen, die im Begriff den Text *daten* enthalten, so kann man das SQL-Statement durch die `WHERE`-Klausel erweitern:

```
mysql> select id, begriff, leaend from glossar
        where begriff like '%daten%';
+-----+-----+-----+
| id | begriff          | leaend          |
+-----+-----+-----+
|  2 | Datenbank-Server | 20060420120233 |
|  3 | Datenbank         | 20060420120311 |
+-----+-----+-----+
2 rows in set (0.28 sec)
```

Für unser PHP-Projekt könne wir nun die Methode `suche` folgendermaßen ändern:

```
global $liste;
$verb = new Verbindung();
$sql = "select * from glossar where begriff like '%$begriff%'";
$verb->verbinde();
//echo $sql;
$verb->frage($sql);
while( $satz = $verb->holSatz())
{
    // ein Dummy-Satz
    $liste[] = new GlossarVO($satz["id"], $satz["begriff"],
    $satz["kurz"]);
}
```

Man beachte, dass die Methode `holSatz` ein assoziatives Array für den Datensatz zurück gibt oder `false`, wenn keine weiteren Daten vorhanden sind.

### Löschen von Sätzen

Das SQL-Statement für das Löschen von Datensätzen lautet in der gebräuchlichsten Form:

```
DELETE FROM tabelle [WHERE where_definition];
```

tabelle ist hierbei der Name der Tabelle und die where\_definition beinhaltet den logischen Ausdruck - also z. B.:

```
DELETE from glossar WHERE id = '6';
```

### Sofern Sie die WHERE-Klausel weglassen, werden alle Daten gelöscht!

Somit liegt eine kleine Herausforderung darin, die Daten für die WHERE-Klausel zu erhalten. Wenn genau ein Datensatz gelöscht werden soll (*über den Primär-Schlüssel*), können wir diesen aus der Ergebnisliste einer Suchabfrage oder aus der Detailansicht des Datensatzes ermitteln.

suche nach ...

Begriff	Kurzbeschreibung	Aktionen
Datenbank-Server	Programm das uns die Funktionalitäten zur Verfügung stellt.	<a href="#">[ändern]</a> <a href="#">[löschen]</a>
Datenbank	Ein Zusammenfassung von mehreren Tabellen.	<a href="#">[ändern]</a> <a href="#">[löschen]</a>
Tabelle	Enthält einzelne Datensätze	<a href="#">[ändern]</a> <a href="#">[löschen]</a>
MySQL	Datenbank-Server	<a href="#">[ändern]</a> <a href="#">[löschen]</a>

Der Link für das Löschen ist dabei:

```
http://10.0.0.94/mt4/kiltz/db/index.php?seite=loeschen&id=4
```

Der Zusammenbau des Links finden wir in unserer `suchen.inc`:

```
foreach ($liste as $satz)
{
    ?>
    <tr>
        <td><?=$satz->begriff?></td>
        <td><?=$satz->kurz?></td>
        <td>[<a href="index.php?
            seite=aendern&id=<?=$satz->id?>">ändern</a>]
            [<a href="index.php?
            seite=loeschen&id=<?=$satz->id?>">löschen</a>]
        </td>
    </tr>
    <?>
}
```

In unserem Controller (`index.php`) rufen wir in unserer Klasse die entsprechende Methode auf und wählen als nächste Anzeige die Suchen-Seite:

```
if ($seite == "loeschen")
{
    $meldung .= $glossar->loeschen($_REQUEST["id"]);
    $seite = "suchen";
}
```

Hierbei übergeben wir gleich die `id`. In der Klasse `Glossar` schaut die Methode zum Löschen folgendermaßen aus:

```
function loeschen($id)
{
    $verb = new Verbindung();
    $verb->verbinde();
    $sql = "delete from glossar where id = '$id' ";
    $verb->frage($sql);
    return "Datensatz gelöscht!";
}
```

### Ändern von Daten

Auch für das Ändern von Daten benötigen wir normalerweise den Zugriff über den Unique Key (die ID). Der normale Ablauf für die Änderung der Daten durch ein Formular passiert in folgenden Schritten:

1. Ermittlung der ID (meist aus der Liste)
2. Select-Statement über die ID
3. Anzeige der Daten in einem Formular
4. Absenden des Formulars
5. speichern der Daten aus dem Formular

Die ID wird hierbei in einem versteckten Feld im Formular gemerkt.

Sehen wir uns die einzelnen Schritte am Beispiel unseres Glossars an:

**Die Ermittlung der ID** haben wir schon in unserer Ergebnisliste vorbereitet:

```
http://10.0.0.94/mt4/kiltz/db/index.php?seite=aendern&id=3
```

Die ID übergeben wir hier analog zum Löschen.

Statt der Löschaktion erzeugen wir ein **Select-Statement über die ID:**

```
function hole($id)
{
    // Hole einen Satz, identifiziert durch seine id
    global $satz;
    $msg = "";
    $verb = new Verbindung();
    $sql = "select * from glossar where id = '$id'";
    $verb->verbinde();
    $verb->frage($sql);
    if( $dbSatz = $verb->holSatz() )
    {
        $satz = new GlossarVO($dbSatz["id"],
            $dbSatz["begriff"], $dbSatz["kurz"], $dbSatz["quelle"]);
        $satz->beschreibung = $dbSatz["beschreibung"];
    }
    else
    {
        $msg = "Es gibt keinen Satz mit der ID $id !";
    }
    return $msg;
}
```

Der \$satz, den wir dann für unsere aendern.inc erhalten nutzen wir für die

### Anzeige der Daten in einem Formular:

```
<form method="post">
<input type="hidden" name="id" value="<?=$satz->id?>"/>
<input type="hidden" name="seite" value="speichern"/>
<table cellpadding="3">
<tr>
    <td>Begriff</td>
    <td><input type="text" name="begriff" value="<?=$satz->
begriff?>"/></td>
</tr>
<tr>
    <td>Kurz</td>
    <td><textarea name="kurz" rows="3" cols="50"
wrap="off"><?=$satz->kurz?></textarea></td>
</tr>
<tr>
    <td>Beschreibung</td>
    <td><textarea name="beschreibung" rows="5" cols="50"
wrap="on"><?=$satz->beschreibung?></textarea></td>
</tr>
<tr>
    <td>Quelle</td>
    <td><input type="text" name="quelle" value="<?=$satz->quelle?>"/></td>
</tr>
<tr>
    <td colspan="2"><input type="submit" value="speichern"/></td>
</tr>
</table>
</form>
```

Man beachte das hidden Field für die id, die wir im nächsten Schritt benötigen.

Nach dem **Absenden des Formulars** können wir nun eine Methode aufrufen, in der wir die Daten aktualisieren können:

Die Methode `speichern($daten)` **speichert nun die Daten aus dem Formular:**

```
function speichern($daten)
{
    $verb = new Verbindung();
    $verb->verbinde();
    $sql = "update glossar set " .
        "begriff = '". $daten["begriff"]."', " .
        "kurz = '". $daten["kurz"]."', " .
        "beschreibung = '". $daten["beschreibung"]."', " .
        "quelle= '". $daten["quelle"]."', " .
        "leaend = now() " .
        "WHERE id = '". $daten["id"]."';";
    //echo $sql;
    $verb->frage($sql);
    return "<p>aktualisiere <b>". $daten["begriff"]. "</b></p>";
}
```

Vergessen Sie hierbei nicht die WHERE-Klausel, sonst werden alle Datensätze auf diese Daten aktualisiert.

### Zusammenfassung

Mit den UPDATE, INSERT, SELECT und dem DELETE-Statement haben Sie die wichtigsten Statements mit denen Sie fast alle Operationen durchführen können. Ihre Aufgabe ist es zu erkennen, in welchen Anwendungsfällen Sie welche Statements benötigen, wie Sie an die benötigten Daten (ID, Suchbegriff, etc.) kommen und welche Ablaufsteuerung Sie nach den Operationen vornehmen.

Die genaue Syntax der vorgestellten SQL-Statements können Sie unter <http://dev.mysql.com/doc/refman/4.0/de/index.html> nachlesen.

## 5. PHP und XML

### Aufbau und Syntax einer XML-Datei

In diesem Kapitel betrachten wir:

- im Überblick den **Aufbau** einer XML-Datei
- die **Syntax** und Elemente einer XML-Datei
- und **XML-Instruktionen**, die in einer XML-Datei enthalten sein können

Der **Aufbau** einer XML-Datei ist in zwei, optional drei Teile gegliedert. In der ersten Zeile befindet sich die **XML-Deklaration**, die als Hauptaufgabe die Festlegung des Zeichensatzes hat. In der einfachsten Variante sieht die XML-Deklaration folgendermaßen aus:

```
<?xml version="1.0"?>
```

Man beachte hierbei die „?“ bei den Klammern.

Der zweite Teil im Aufbau einer XML-Datei ist optional und enthält eine Verbindung zu einer DTD (Document Type Definition) oder die DTD selbst. Eine Verbindung zu einer DTD kann folgendermaßen aussehen:

```
<!DOCTYPE Wurzelelement SYSTEM "meine.dtd">
```

Dies besagt, dass sich im gleichen Verzeichnis eine DTD mit dem Namen `meine.dtd` befindet. Beachten Sie bitte das `!` nach der öffnenden Klammer. Die DTDs werden in einem späteren Kapitel ausführlich behandelt.

Als dritter Teil kommen die eigentlichen Daten, die in einem Wurzelement eingeschlossen sind. Das Wurzelement ist unser erster, all umschließender Tag. Jeder Tag beginnt mit dem Zeichen `<` und endet mit einem `>`, auch Klammern genannt. Ein Tag wird beendet, in dem der Tag wiederholt wird und ein `/` nach der öffnenden Klammer eingefügt wird:

```
<wurzelElement>  
  <!-- die Daten -->  
</wurzelElement>
```

Kommentare werden mit der Zeichenfolge `<!--` begonnen und mit `-->` beendet.

Eine komplette XML-Datei zur Darstellung einer Flasche Wein könnte folgendermaßen aussehen:

```
<?xml version="1.0" encoding="ISO-8859-1"?>  
<!DOCTYPE EineFlascheWein SYSTEM "wein.dtd">  
<EineFlascheWein> <!-- das Wurzelement -->  
  <Jahrgang>1999</Jahrgang>  
  <Rebsorte art="Riesling"/>  
  <Lage>Gutenberger Römerberg</Lage>  
  <Preis waehrung="EUR">12,00</Preis>  
</EineFlascheWein>
```

Die **Syntax** in einer XML-Datei ist recht einfach. Abgesehen von der Deklaration und der DTD-Definition, beinhaltet die XML-Datei nur Tags und Text. In den meisten Fällen definieren die Tags einen Bereich mit einem **Start-Tag** `<Jahrgang>` und einem **End-Tag** `</Jahrgang>`. Zwischen diesen beiden Tags können weitere Tags oder Text oder beides stehen. Tags ohne Inhalt sind **leere Tags** wie z. B. `<Rebsorte art="Riesling"/>`, sie werden mit einem `/` vor der abschließenden Klammer beendet. Start-Tag, Inhalt und End-Tag werden als **Element** bezeichnet.

Ein Tag besteht aus einem **Elementnamen** und 0 bis n **Attributen**. Der Elementnamen folgt sofort nach der öffnenden Klammer, bzw. Bei dem End-Tag nach `</`, er ist casesensitiv und muss mit einem Buchstaben oder einem Unterstrich beginnen. Er darf nicht mit der Zeichenfolge `xml` beginnen. Im Elementnamen sind Leerzeichen, Doppelpunkte und `@` nicht erlaubt.

Ein Attribut besteht immer aus dem Attributnamen, einem Gleichheitszeichen als Trenner und einem Wert, der in Anführungszeichen eingeschlossen ist. Zwischen Elementnamen und den Attributen sind Leerzeichen als Trenner.

Zusätzlich zu den Elementen können in einer XML-Datei noch **XML-Instruktionen** enthalten sein. Folgende Typen werden unterschieden:

- XML-Deklaration
- Verknüpfung zur DTD
- Processing Instructions
- CDATA-Sections
- Kommentare

Die **XML-Deklaration** hatte ich schon angesprochen. Ihre allgemeine Syntax lautet:

```
<?xml version="Nummer"
    [encoding="Zeichensatz"] [standalone="yes|no"]?>
```

Das Attribut `version` hat sein Februar 1998 den Wert `"1.0"`, `encoding` enthält die Bezeichnung des Zeichensatzes z. B. `"iso-8859-1"`, `"utf-16"` oder `"us-ascii"`. Das Attribut `standalone` gibt an, ob diese XML-Datei eine DTD besitzt (`"no"`) oder nicht (`"yes"`). Der Standardwert ist `"no"`.

Die **Verknüpfung zur DTD** legt fest, dass das aktuelle XML-Dokument nach den Regeln einer DTD aufgebaut ist. Grundsätzlich werden öffentliche (im Web verfügbar) und systemeigene DTDs unterschieden. Die allgemeine Syntax für die DTD-Verknüpfung lautet:

```
<!DOCTYPE Wurzelelement SYSTEM|PUBLIC ["Name"] "URI-der-DTD">
```

Der Name setzt sich aus der veröffentlichenden Person oder Institution, der Bezeichnung der DTD und einer Angabe zur Sprache zusammen z. B.:

```
"-//MeineFirma Solutions//DTD Email V 1.0//DE"
```

Eine weitere Art der XML-Instruktionen ist die **Processing Instruction**, mit der Sie spezifische Informationen für externe Programme einbinden können, sofern das Programm diese versteht. Alle Processing Instructions beginnen mit `<?` und enden mit `?>`. Im folgenden Beispiel wird eine Anweisung eingefügt, die aus einer Postscript-Datei eine PDF-Datei erzeugen soll:

```
<?ps2pdf von="druck.ps" nach="ausgabe.pdf"?>
```

Sie brauchen nur eine Programm, das dies auch ausführt ...

Die nächste XML-Instruction ist die **CDATA-Section**, die sicher stellt, dass der XML-Prozessor den Inhalt dieser Section nicht ausführt. Die allgemeine Syntax hierfür ist `<![CDATA[ . . . ]>` wobei die drei Punkte den Inhalt der Section darstellen:

```
<Javascript><![CDATA[
    var i = 0;
    for (i = 0; i < 100; ++i)
    {
        echo i + ."-->" + (i*i);
    }
]]>
</Javascript>
```

In diesem Beispiel sorgt die CDATA-Section dafür, dass das Kleinerzeichen und die Zeichenfolge `-->` nicht ausgewertet werden.

Die letzte XML-Instruction ist der **Kommentar**, der natürlich die Lesbarkeit für Menschen erhöhen kann. Jeder Kommentar beginnt mit der Zeichenfolge `<!--` und endet mit `-->`. Ein Kommentar darf keine doppelten Bindestriche enthalten.

### Wohl geformtes XML

Zur Vermeidung der häufigsten Fehlerquellen sollten man die Regeln für **wohlgeformtes XML** beachten. Diese sind:

- Alle Element-Attributwerte müssen in Anführungsstriche eingeschlossen werden. Die übliche Darstellung in HTML, bei der Zahlen ohne Anführungsstriche dargestellt werden  
``  
ist **falsch!**
- Ein Element muss sowohl ein Start-Tag als auch ein End-Tag besitzen. Handelt es sich um ein leeres Element, muss es vor der abschließenden Klammer abgeschlossen werden. Beispiel ein HTML-Zeilenumbruch:  
`<br/>`
- Alle Start- und End-Tags müssen richtig ineinander geschachtelt werden. Folgende Kombination ist falsch, da der Bold-Tag (`<b>`) vor dem Italic-Tag (`<i>`) geschlossen wird:  
`<b>Dies ist nicht <i>korrekt</b></i>`

- Isolierte Markup-Zeichen ( <, & und > ) sind im laufenden Text nicht erlaubt. Statt dessen sind deren Entity-Referenzen zu nutzen:  
< wird zu & lt;  
& wird zu & amp;  
> wird zu >
- Sofern in der XML-Deklaration das Attribut standalone auf "yes" gesetzt wurde, darf keine DTD angegeben werden.

### XML-Verarbeitung mit SAX

SAX (Simple API for XML) basiert auf der sequentiellen Verarbeitung von XML-Dateien. Bei der Verarbeitung werden Ereignisse weitergegeben, auf die der Entwickler reagieren kann.

Für SAX benötigen wir einen Parser, den wir konfigurieren und starten müssen und entsprechende Ziele um die Ereignisse aufzufangen:

Der Parser:

```
/** Der Parser */
$xml_parser = xml_parser_create();
// die Option, daß alle Tags in Uppercase eingelesen werden!
xml_parser_set_option($xml_parser, XML_OPTION_CASE_FOLDING, true);
/** Wer wird aufgerufen bei Beginn und Ende eines Tags */
xml_set_element_handler($xml_parser, "StartElement", "EndeElement");
/** Wer wird aufgerufen beim Inhalt eines Tags */
xml_set_character_data_handler($xml_parser, "Inhalt");
/** Eröffnung der Datei */
if (!$fp = fopen($file, "r"))
{
    die("...finde Deine Datei nicht!");
}

/** Durchwühle die Datei */
while ($data = fread($fp, 4096))
{
    if (!xml_parse($xml_parser, $data, feof($fp)))
    {
        die(sprintf("XML Fehler: %s in Zeile %d",
                    xml_error_string(xml_get_error_code($xml_parser)),
                    xml_get_current_line_number($xml_parser)));
    }
}
/** und wieder freigeben! */
xml_parser_free($xml_parser);
```

Die Ziele:

```
/** Am Anfang eines Tags */
function StartElement($parser, $name, $attrs) {
    echo "<p>Element: ".$name;
    foreach( $attrs as $name => $inhalt)
    {
        echo "<br/>Attr.: ".$name." = ".$inhalt;
    }
}
```

```
/** Am Ende eines Tags */
function EndeElement($parser, $name) {
    echo "<br/>... ende " . $name;
}
/** Ausgabe des Inhalts */
function Inhalt($parser, $data) {
    echo "<br/> ".$data;
}
```

SAX kann für große XML-Datenbestände genutzt werden und arbeitet nur lesend.

### XML-Verarbeitung mit DOM

Bei DOM (Document Object Model) gibt der Parser einen Dokumentenbaum in einer Variablen zurück, die dann mit verschiedenen Techniken ausgelesen erweitert und verändert werden kann.

```
<?
//DOM-Dokument erzeugen
$doc = new DOMDocument();
// Leerzeichen und Tabulatoren nicht als Text-Elemente betrachten
$doc->preserveWhiteSpace = false;
// Datei einlesen
$doc->load("weine.xml");
// Das Wurzelement
$root = $doc->documentElement;

//Wein anzeigen
$weine = $root->getElementsByTagName("Wein");
foreach( $weine as $wein )
{
    $jahr = $wein->getElementsByTagName("Jahrgang")->item(0);
    $lage = $wein->getElementsByTagName("Lage")->item(0);
    echo $jahr->nodeValue."": ".$lage->nodeValue."<br/>";
}

//Wein hinzufügen:
// Elemente erzeugen
$neuerWein = $doc->createElement("Wein");
$neuerJahrgang = $doc->createElement("Jahrgang", "2007");
$neueLage = $doc->createElement("Lage", "Gutenberger Rotfeld");
//Elemente hinzufügen.
$neuerWein->appendChild($neuerJahrgang);
$neuerWein->appendChild($neueLage);
$root->appendChild($neuerWein);

$doc->save("neuerWein.xml");

?>
```

### SimpleXML

Umd DOM und SAX zu vereinfachen entwickelte man SimpleXML, bei dem die XML-Struktur in eine Struktur von Objekten und Arrays mit Objekten übertragen wird.

## 5. PHP und XML

---

```
$weine = simplexml_load_file("weine.xml");

for ($i = 0; $i < sizeof($weine->Wein); ++$i)
{
    echo $weine->Wein[$i]->Jahrgang.": ";
    echo $weine->Wein[$i]->Lage."<br/>";
}

echo "<p>oder</p>";

foreach ($weine as $wein)
{
    echo $wein->Jahrgang.": ";
    echo $wein->Lage."<br/>";
}
```

### XSL Transformation mit PHP

Die Transformation mit PHP ist relativ Simple. Man benötigt einen Transformer, eine XML-Datenbasis und die Transformationsvorschrift, die XSL-Datei.

Infos über den PHP-Transformer finden Sie unter <http://de3.php.net/manual/de/function.xsl-xsltprocessor-transform-to-uri.php>.

Zur Transformation wird DOM benötigt und der Transformer:

```
// Load the XML source
$xml = new DOMDocument;
$xml->load('collection.xml');

$xml = new DOMDocument;
$xml->load('collection.xml');

// Configure the transformer
$proc = new XSLTProcessor;
$proc->importStyleSheet($xsl); // attach the xsl rules

echo $proc->transformToXML($xml);
```