Die Programmiersprache PHP trägt ihren Namen als eine auf sich selbst verweisende Abkürzung aus ***P**ersonal **H**ome **P**age Tools* und ***H**ypertext **P**reprocessor*.

Obwohl sie in der Syntax sehr an JAVA und C++ erinnert, handelt es sich hier im Gegensatz zu allen anderen modernen Programmiersprachen um eine INTERPRETER-Sprache und nicht um eine COMPILER-Sprache. D.h. Es wird nicht der komplette Quelltext in eine maschinennahe Sprache übersetzt, wo die Übersetzung bei erfolgreicher Compilierung unabhängig vom Quelltext ein lauffähiges Programm darstellt. Vielmehr wird in PHP Zeile für Zeile übersetzt und jede Zeile nach ihrer Übersetzung sogleich als einzelne Zeile, ohne Kenntnis des Programmrestes, ausgeführt.

Eine weitere Besonderheit von PHP ist, dass nur diejenigen Teile des Quelltextes, die durch die besondere Markierung **<?php** ... **?>** gekennzeichnet sind, übersetzt und ausgeführt werden. Nicht auf diese Weise markierter Text wird einfach Buchstabe für Buchstabe, bzw. Zeichen für Zeichen in das erzeugte HTML-Dokument übernommen.

PHP wird (... neben Java-Script) zur Erstellung dynamischer Websites eingesetzt, wobei ihr Vorteil darin liegt, dass die PHP-Befehle ausschließlich auf einem unzugänglichen PHP-Server im Netz ausgeführt werden und der USER nur ein reines HTML-Script zur Anzeige bekommt, was den Umgang mit Datenbanken sicherer macht. Gleichzeitig ist dies im Vergleich zu Java-Script, das auf dem Rechner des USERs ausgeführt wird, auch der größte Nachteil, da bei PHP immer vollständige HTML-Seiten geladen werden müssen und Änderungen nicht lokal und damit schnell am USER-PC stattfinden.

Das Ergebnis der Verarbeitung in PHP kann nur als HTML-Seite über einen Browser ausgegeben werden.

Inhaltsverzeichnis

PHP-Verarbeitung

<?php ... **?>**

zwischen dem PHP-**Anfangs-TAG** und dem PHP-**End-TAG** werden dort alle PHP-Befehle vom PHP-Interpreter zeilenweise ausgeführt.

Jeder beliebige Text, der außerhalb dieser TAGs steht, wird als Text dem erzeugten HTML-Dokument mitgegeben. Dies gilt auch für alle HTML-Tags, wie
, <table>, </table>, <b>, ...

Variablen

$a

$ich _bin_eine_Variable

Buchstaben oder Buchstabenketten ohne Leerzeichen, Umlaute oder andere Sonderzeichen die auf **$** folgen, werden von PHP automatisch als Variablen erkannt. Dabei spielt es keine Rolle, um welchen Typ von Variable es sich handelt. Character, String, Boolean, Integer, Float erkennt PHP automatisch im verwendeten Kontext. Die Behandlung der Variablennamen erfolgt kontextsensitiv, also unter Berücksichtigung von Groß und Kleinschreibung. In einer einzelnen Variable kann auch ein ganzes Array gespeichert sein.

$b[0], **$b[1]**, **$b[2]**, ... , **$b[i]**

Stellen die zweite mögliche Variablenart, das Array, ein ganzes Feld von Variablen mit gleichem Variablennamen, einzig unterschieden durch die Zahl in eckigen Klammern dar. Auch hier wird der Variablentyp automatisch erkannt. Eine Definition der Anzahl der benötigten Variablen ist nicht notwendig.

$b[0][1], **$b[0][2]**, **$b[0][3]**, ...
$b[1][1], **$b[1][2]**, **$b[1][3]**, ... , **$b[i][k]**
$b[i][k]...[m]

Selbstverständlich sind auch mehrdimensionale Arrays möglich. Oder auch solche:

$b['*Name1***']**, **$b['***Name2***']**, ...

Tragen Spalten, z.B. In einer Datenbank *Namen* zur Kennzeichnung ist es auch möglich diese Arrays mit Hilfe ihrer Namen eindeutig aufzurufen. Die Namen müssen dabei in Hochkommas stehen.

'3'.$b.$c.'Text1'.$k."Text2"

Variablen können können immer miteinander und immer mit Text verknüpft werden. Dies ist unabhängig davon, von welchem Variablentyp eine Variable ursprünglich war. Als Verknüpfungszeichen dient der **Punkt**.

Eine Besonderheit in PHP ist, dass man sich mit einer erfolgten Belegung bei einer Variablen nicht auf den entsprechenden Typ festlegt. Ein einfaches Beispiel, mit

$a = 128;
$a = $a / 2;
$a = $a / 3;
$a = $a . "Halleluja";
$a = $a . "10101";
$a = $a * 6;

ist $a eine Variable vom Typ Integer. Durch Division durch 2 bleibt der Variablentyp Integer. Wird jetzt durch 3 dividiert ändert sich der Variablentyp automtisch auf Float und trägt jetzt den Wert 21,3333333333.
Wird nun an die Variable $a mittels des **Punktes** das Wort Halleluja angehängt, handelt es sich bei $a nun um eine Variable vom Typ String, an den auch Zahlen angehängt werden
können, was den Variablentyp nicht ändert.
Mit der Variable $a vom Typ String können tatsäch-lich numerische Operationen durchgeführt werden, z.B. Die Multiplikation mit
der Zahl 6. Dabei wird die numerische Operation so weit ausgeführt, bis nicht numerische Zeichen auftauchen und dann automatisch beendet. Das Ergebnis ist also wieder 128.

siehe *Beispiel 01*

$b = ''Kind";
$b[0] = ''K";
$b[1] = ''i";
$b[2] = ''n";
$b[3] = ''d";

Dabei sind alle Variablen im Zweifelsfall eben vom Typ STRING. Ein STRING ist aber nichts anderes, als ein ARRAY vom Typ CHARACTER. Das heißt, obwohl man es einer String-Variablen nicht ansehen kann, verbirgt sich dahinter ein geordnetes Variablenfeld, welches jeweils einzelne Zeichen enthält. Dieses Array beginnt immer mit dem Index 0.
Deshalb kann man immer auf die einzelnen Zeichen in einer Zeichenkette, also einem String, oder in PHP jeder Variablen zugreifen.

siehe *Beispiel 03*

Bildschirmausgabe

echo ‘Text ausgeben’**;** Der Text zwischen den Hochkommas wird ausgegeben.

echo **“**Text ausgeben**”;** Der Text kann auch zwischen Gänsefüßchen stehen, PHP verarbeitet beides in gleicher Weise und auch gemischt aber jeweils nur mit gleichem Anfangs- und Endzeichen.
Soll am Ende ein Zeilenumbruch erfolgen muss dies mittels HTML-TAG

echo ‘Text ausgeben
’**;**

<?php echo ‘Text ausgeben’ **?>**
 mitgegeben werden. Möglich wäre auch eine Kombination von PHP und HTML, die zwar unübersichtlich ist, aber trotzdem häufig verwendet wird.

siehe *Beispiel 01*

echo $a; gibt den Inhalt der Variablen $a auf dem Bildschirm aus.

echo $a[0].$a[6]; gibt entweder das 1. und das 7. Zeichen der Variablen $a oder den 1. und den 7. Vertreter des Arrays $a[i] das auf dem Bildschirm aus.

Tastatureingaben über HTML (**kein** Bestandteil von PHP)

siehe *Beispiel 01*

Üblicher Weise wird für Tastatureingaben die HTML-Variante **<form action=”...” method=”...”> ... </form>**, also **nicht** innerhalb der PHP-TAGs, sondern als reiner HTML-Text, verwendet. Dabei werden die eingegebenen Daten an ein Programm, auf irgend einem Server im Internet, geschickt, welches die Daten dann verarbeitet.
Bei der Verschickung von Daten im Internet können diese offen oder heimlich verschickt werden. Die Daten werden aber in beiden Fällen unverschlüsselt übertragen.

Methode: **get** Bei offener Versendung werden die versendeten Daten im Adressfeld angezeigt.
Methode: **post** Bei heimlicher Versendung werden die versendeten Daten nicht angezeigt.

In HTML gibt es für ein Form-Blatt folgende “Befehle”:

<form action="test.php"> Mindestens damit beginnt jedes Formblatt. Der Inhalt von **action** bezeichnet das Programm, an welches die eingegebenen Daten weitergegeben werden.

Ist keine besondere Weitergabemethode angegeben, erfolgt die Weitergabe der Daten automatisch mit der Methode **get**.

</form> Damit wird das Formblatt geschlossen. Dies ermöglicht nach einander mehrere Formblätter auf einer Seite, die an verschiedene Auswertungsprogramme oder an ein und das selbe Programm verschickt werden, wobei sie dann Parameter enthalten die verschiedene Verarbeitungen mit ein und dem selben Programm ermöglichen.

<form action="test.php” method="post"> Erzeugt ein Formblatt welches die Daten heimlich, also mit der Methode **post** weiterleitet.

<form action="test.php?bbb=Text" method="post"> Erzeugt ein Formblatt welches die Daten innerhalb des Formblatts heimlich mit der Methode **post** weiterleitet und die mit der Variablen **bbb** mittels **?bbb=Text** direkt an die Adresse angehängten Daten (... hier deren Inhalt **Text**) offen mit der Mathode **get** weitergibt, was dann in der Adresszeile sichtbar ist.

Innerhalb des Formblatts stehen folgende Eingabemöglichkeiten zur Varfügung:

<input type="*text*" name="*z*"> Ist als **type** *text* angeben, kann hier beliebiger und als solcher sichtbarer Text eingegeben werden. Der Text wird als Inhalt der frei wählbaren Variablen *z* weitergegeben und ist im Empfängerprogramm unter diesem Variablennamen zur weiteren Verarbeitung verfügbar. Die Anzahl der Zeichen ist beschränkt.

<input type="*password*" name="*n*"> Ist als **type** *password* angeben, kann hier beliebiger Text eingegeben werden. Allerdings wird der Text im HTML-Formular nicht angezeigt. Der Text wird als Inhalt der frei wählbaren Variablen *n* weitergegeben und ist im Empfängerprogramm unter diesem Variablennamen zur weiteren Verarbeitung verfügbar. Die Anzahl der Zeichen ist beschränkt.

<input type="*...*" name="*z*" value="*Vorgabe*"> Unabhängig vom jeweils gewählten Typ **type** kann im in HTML angezeigten Eingabefeld durch **value** irgend ein Text (... hier *Vorgabe*) gezeigt werden.

<input type="..." name="*z*" value="**<?php $ccc ?>**"> Wird das Formular auf einem PHP-Server erzeugt, kann als angezeigter Vorgabewert auch der Inhalt irgend einer PHP-Variable (... hier **$ccc**) angezeigt werden.

<textarea name="s" cols="50" rows="6"> </textarea> Erlaubt längere Texte einzugeben. Dabei gibt **cols** die Anzahl der Zeichen pro Zeile und **rows** die Anzahl der im HTML-Text angezeigten Zeilen an. Es können jedoch auch mehr Zeichen, als die im HTML-Text dargestellten eingegeben werden.

<textarea name="s">**das war drin**</textarea> Im in HTML angezeigten Eingabefeld kann auch eine Vorgabe (... hier: *das war drin*) angezeigt werden.

<textarea name="s">**<?php $ccc ?>**</textarea> Wird das Formular auf einem PHP-Server erzeugt, kann als angezeigter Vorgabewert auch der Inhalt irgend einer PHP-Variable (... hier **$ccc**) angezeigt werden.

<input type="submit" value="*abschicken*"> Um das Formular (... Formblatt) an das under **action** in <form ... > angegebene Programm abzuschicken bedarf es eines anklickbaren Auslösers.
Dabei kann unter **value** ein angezeigter Text eingegeben werden.

<input type="submit"> Erfüllt den Zweck auch, es muss kein Anzeigetext angegeben werden.

Auswertung von Tastatureingaben aus HTML in PHP

Wie immer in PHP wird jede denkbare, mehr oder weniger benutzerfreundliche oder strikt sicherheitsorientierte Variante angeboten. Letztich ist für die Verarbeitung der übertragenen Daten entscheidend, wie der verwendete PHP-Server individuell konfiguriert wird.

Ob zwischen GET- und POST-Übergaben unterschieden wird ist eine Frage des persönlichen Sicherheitsbedarfs, den jeder Administrator lokal auf seinem PHP-Server selbst wählen darf. Wer sich gegen Attacken bestmöglich wappnen möchte unterscheidet strikt zwischen den beiden Variantenarten (... so ist auch die Einstellung des internen PHP-Servers am Bach-Gymnasium).

siehe *Beispiel 01*

Wer (... und das ist die Standardeinstellung im Internet) problemlos, unisono beides nutzen möchte, setzt seine Einstellungen so, dass nicht zwischen den beiden Variantenarten Get und Post bei der Variablenannahme unterschieden wird. Das macht eben auch das Handling einfacher.

siehe *Beispiel 01*a

Verzweigung

if(*Bedingung* **) { ... }** Ist die *Bedingung* erfüllt (z.B. $i <= 5 oder $b !="Käse", ...), wird der Programmtext in den geschweiften Klammern ausgeführt. Ist sie nicht erfüllt, wird der Programmtext in den geschweiften Klammern ignoriert.

if(*Bedingung* **) { ... } else { ... }** Ist die *Bedingung* erfüllt (z.B. $i <= 5 oder $b !="Käse", ...), wird der Programmtext in den 1. geschweiften Klammern ausgeführt. Ist sie nicht erfüllt, wird der Programmtext in den 2. geschweiften Klammern ausgeführt.

Logische Operationen für die Bedingung sind:

== vergleicht zwei Werte, bei Gleichheit ist die Bedingung erfüllt
!= vergleicht zwei Werte, bei Ungleichheit ist die Bedingung erfüllt
<= ist der linke Wert kleiner oder gleich dem rechten, ist die Bedingung erfüllt
>= ist der linke Wert größer oder gleich dem rechten, ist die Bedingung erfüllt
< ist der linke Wert kleiner als der rechte, ist die Bedingung erfüllt
> ist der linke Wert größer als der rechte, ist die Bedingung erfüllt

&& verknüpft zwei Bedingungen mit UND, es müssen also beide erfüllt sein, damit die Bedingung insgesamt erfüllt ist

|| verknüpft zwei Bedingungen mit ODER, es muss also nur eine der beiden erfüllt sein, damit die Bedingung insgesamt erfüllt ist

siehe *Beispiel 02*

So ist es auch möglich, mit Hilfe von Klammern logische Operationen aneinander zu binden, also die in der Klammer jeweils zuerst zu testen und das Ergebnis mit einer dritten zu verknüpfen.

($a<=5 && $b>2) || $a == 7 Diese Bedingung ist wahr wenn sowohl $a kleiner gleich fünf und gleichzeitig $b größer, als zwei ist. Sie ist auch wahr, wenn $a sieben ist, egal was $b ist.

Schleifen

siehe *Beispiel 03*

for($i=1; $i<=7; $i++) {...}	Diese Schleife läuft von 1, 2, ... ,7 Bei jedem Durchlauf wird $i um 1 erhöht. Dabei werden also 7 Mal die Befehle in den geschweiften Klammern ausgeführt. Dabei kann $i in den geschweiften Klammern benutzt werden, soll aber dort nicht verändert werden.
for($i=1; $i<7; $i++) {...}	Diese Schleife läuft von 1, 2, ... ,6
for($i=1; $i!=7; $i++) {...}	Diese Schleife läuft von 1, 2, ... ,6
for($i=1; $i==7; $i++) {...}	Diese Schleife läuft nicht, sie erzeugt nicht einmal einen Durchlauf

while (*Bedingung* **) {**...**}**	Die Befehle in den geschweiften Klammern werden so lange ausgefüht, wie die *Bedingung* erfüllt, also wahr, ist. Als *Bedingung* kommen alle bei Verzweigungen möglichen, also auch Kombinationen mit allen logischen Operatoren in Betracht.

Datenbanken

Egal, ob sie nun von Microsoft, SAP, Oracle oder frei programmiert wurden, wie mySQL, stehen in einer Datenbank mit einem *Datenbanknamen* mehrere Tabellen mit eigenen *Tabellennamen*. Jede Tabelle wiederum besteht aus mehreren Spalten mit *Spaltennamen* und vielen Zeilen, die aufeinander folgen.

Um eine Datenbank überhaupt ansprechen zu können benötigt man einige grundlegende Informationen über dieselbe. Man muss wissen, wo sie sich befindet, etwa auf dem Rechner, auf dem auch der PHP-Server läuft (... mysql.localhost). Man benötigt einen zugelassenen Benutzernamen (... Schueler) und das zugehörige Passwort (... Schnurzelputz) und den Namen der Datenbank (... BachGymn) und der Tabelle (... Informatik), mit der man kommunizieren möchte.

All diese Angaben, bis auf den Tabellennamen, möchte man in der Regel nicht öffentlich preisgeben. Deshalb versteckt man sie irgendwo an einer unzugänglichen Stelle auf dem PHP-Server und gibt für das ausführende PHP lediglich die entsprechenden Variablen weiter.

$db_host = "mysql.localhost";
$db_user = "Schueler";
$db_pass = "Schnurzelputz";
$db_name = "BachGymn";

Wir werden genau diese Variablen mittels **include** einer kleinen Datei mit Namen **DBdat.php** aus dem Verzeichnis **Hofsaess** laden, welches nicht unser eigenes Verzeichnis ist, aber auf der selben Ebene liegt. Den Kontakt zur Datenbank stellen wir dann ausschließlich durch Nutzung der Variablen her.

Selbstverständlich ist es auch von Vorteil, wenn man die Spaltennamen der ausgewählten Tabelle namens **Informatik** kennt. Dies ist allerdings nicht zwingend notwendig. Unsere Tabelle hat insgesamt 5 Spalten mit den Spaltennamen **id**, **datum**, **Name**, **Vorname**, **Text**..

Kontakt zur Datenbank herstellen

include("../Hofsaess/DBdat.php"); Die Zugangsdaten übernehmen wir direkt aus einem kleinen Textfile in unser Programm. Dieser Textfile wird somit Teil unseres PHP-Programms und vom PHP-Server auch als solcher ausgewertet. Dabei werden unsichtbar die Variablen:

$db_host = "mysql.localhost";
$db_user = "Schueler";
$db_pass = "Schnurzelputz";
$db_name = "BachGymn";

gesetzt. Damit verfügen wir jetzt über die Inhalte der Variablen $db_host, $db_user, $db_pass, $db_name.

../ Aus dem aktuellen Verzeichnis geht es eine Ebene zurück in das Verzeichnis, in dem unser Verzeichnis liegt.

Hofsaess/ Betreten des Verzeichnisses *Hofsaess*, das im selben Verzeichnis liegt, wie unseres.

Dbdat.php Aufruf der kleinen Textdatei *Dbdat.php* zur Implementierung im PHP-Programm.

$DB = **mysql_connect**($db_host,$db_user,$db_pass); Stellt den Kontakt zum Ort, an dem sich die Datenbank befindet, her. Dabei gibt $DB entweder den Wert 0 oder 1 zurück, je nachdem, ob die Anmeldung erfolgreich war oder nicht. Dies könnte dann im Programm ausgewertet werden, um dem User anzuzeigen, ob dieser Ort mit seinen Zugangsdaten momentan überhaupt erreichbar ist.

$DN = **mysql_select_db**($db_name); Stellt den Kontakt zur Datenbank her. Dabei gibt $DN entweder den Wert 0 oder 1 zurück, je nachdem, ob die Anmeldung erfolgreich war oder nicht. Dies könnte dann im Programm ausgewertet werden, um dem User anzuzeigen, ob diese Datenbank überhaupt erreichbar ist.

siehe *Beispiel 11*

Daten aus einer Datenbank-Tabelle auslesen

$ergebnis = **mysql_query** ("**SELECT** * **FROM** Informatik"); Prinzipiell genügt dies, um die Datenbank-Tabelle *Informatik* komplett in der Reihenfolge, in der die Daten in der Datenbank gespeichert sind, auszulesen und als komplettes Array in *$ergebnis* zu speichern.

SELECT * liest alle Spalten der Datenbank-Tabelle aus

SELECT `id` , `Name`, `Text` liest nur die Spalten id, Name, Text aus der Datenbank-Tabelle aus

FROM Informatik sorgt dafür, dass in der gewählten Datenbank nur die Tabelle *Informatik* ausgelesen wird.

ORDER BY Name sorgt dafür, dass die Auslese der Datenbank in alphabetisch **auf**steigender Form nach den Einträgen in der Spalte *Name* vorgenommen wird.

ORDER BY Vorname **DESC** sorgt dafür, dass die Auslese der Datenbank in alphabetisch **ab**steigender Form nach den Einträgen in der Spalte *Vorname* vorgenommen wird.

Da der Auslesetext in **mysql_query**(...) sehr lang sein kann, wird normalerweise nicht die direkte Form benutzt. Üblich ist, den Auslesetext in einer separaten Variablen z.B. $query bereitzustellen und dann mit **mysql_query**(...) zu verknüpfen.

$query = "**SELECT** * **FROM** Informatik **ORDER BY** Name"; ist eine geeignete Form dieser separaten Variable.

$ergebnis = **mysql_query(**$query**)**; Liest die geamte Datenbank-Tabelle *Informatik* mit allen Spalten aufsteigend geordnet nach den Einträgen in der Spalte *Name* aus und übergibt das Array an *$ergebnis*.

$daten = **Mysql_fetch_array(** $ergebnis**)** übergibt das Riesen-Array der ganzen Datenbankauslese in *$ergebnis* zeilenweise, als kleines Zeilen-Array, an $daten

while($daten = **mysql_fetch_array(**$ergebnis**)**) { ... } Mit einer WHILE-Schleife werden die Daten-Arrays zeilenweise an *$daten* übertragen, solange bis keine weiteren Zeilen mehr vorhanden sind.

Innerhalb der geschweiften Klammern können die einzelnen Daten $daten[...] des Arrays $daten dann weiterverarbeitet werden.

$daten[**0**], $daten[**1**], $daten[**2**], ... Ruft die Daten so, wie sie von links nach rechts in der Datenbank-Tabelle stehen ab

siehe *Beispiel 11*

$daten['**id**'], $daten['**Nachname**'], $daten['**Text**'] kennt man die Spaltennamen der Datebkank-Tabelle, können die Daten aus dem Zeilen-Array $daten auch mit Hilfe der Spaltennamen abgerufen werden.
Diese Methode hat den Vorteil, dass eine bestehende Datenbank jederzeit um weitere Spalten erweitert werden kann, ohne dass bereits bestehende Software zur Auslese neu überarbeitet werden muss.

siehe *Beispiel 11*a

Üblicher Weise werden die Daten in einer HTML-Tabelle präsentiert, wenn sie in großer Menge dargeboten werden. Das hat auch den Vorteil, dass man Sprungadressen zu Programmen, welche die Daten in einer Zeile verändern oder ganze Zeilen löschen, im Anschluss an die Datenausgabe zeilensensitiv angeboten werden können.

Datensatz aus einer Datenbank-Tabelle löschen

Um einen Datensatz löschen zu können, muss er eindeutig identifizierbar sein. Aus diesem Grund ist in der Datenbank die Spalte **id** vorhanden. Bei der Anlage eines neuen Datensatzes wird die **id** automatisch von der Datenbank selbst eindeutig vergeben. D.h. Um einen Datensatz zu löschen muss zuerst die Datenbank ausgelesen und alle Daten aufgelistet werden. Dabei kann z.B. wie in ***Beispiel 11a*** eine Tabellenzelle erzeugt werden, die ein Programm zum Löschen aufruft und die eindeutige **id** des zu löschenden Datensatzes weiter gibt:

<td><a href="**loeschen.php?id=*<?php echo $daten["id"]; ?>*">löschen</td>

$wegdamit = $_GET["id"]; Im Programm **loeschen.php** wird die mit der Methode GET übergebene Variable **id** in $wegdamit gespeichert um dann mit der Zeile:

mysql_query("**DELETE FROM** *Informatik* **WHERE id='$wegdamit'**");

den Datensatz endgültig zu löschen. Dieser Vorgang ist endgültig, also irreversibel.

siehe *Beispiel 12*

Daten in einer Datenbank-Tabelle ändern

siehe *Beispiel 13*

Um einen Datensatz ändern zu können, muss er eindeutig identifizierbar sein. Aus diesem Grund ist in der Datenbank die Spalte **id** vorhanden. Bei der Anlage eines neuen Datensatzes wird die **id** automatisch von der Datenbank selbst eindeutig vergeben. D.h. um einen Datensatz zu ändern muss zuerst die Datenbank ausgelesen und alle Daten aufgelistet werden. Dabei kann z.B. wie in ***Beispiel 11a*** eine Tabellenzelle erzeugt werden, die ein Programm zum Ändern aufruft und die eindeutige **id** des zu ändernden Datensatzes weiter gibt:

```
<td><a href="aendern.php?id=<?php echo $daten["id"]; ?>">a&auml;ndern</a></td>
```

Im Quelltext sieht dies für die id = 15 dann so aus:

```
<td><a href="aendern.php?id=15">a&auml;ndern</a></td>
```

Um Änderungen vornehmen zu können, müssen dem User die zu veränderbaren Daten in einer Form präsentiert werden, in der sie sich per Tastatureingabe ändern lassen.
Dazu muss zuerst die Verbindung zur Datenbank aufgebaut und der entsprechende Datensatz gelesen werden.

```
$id = $_GET["id"];
```

Im Programm **aendern.php** wird die mit der Methode GET übergebene Variable **id** in $id gespeichert um dann mit der Zeile:

```
$ergebnis=mysql_query("SELECT * FROM Informatik WHERE id='$id'");
$daten = mysql_fetch_array($ergebnis);
```

ausgelesen und im Array $daten bereit gestellt zu werden

Die Tastatureingaben werden in HTML über **<form> ... </form>** vorgenommen:

```
<form action="aendern.php?todo=anders&id=<?php echo $id; ?>" method="post">
Name: <input type="text" name="z" value="<?php echo $daten['Name']; ?>">
Vorname: <input type="text" name="n" value="<?php echo $daten['Vorname']; ?>">
Text: <textarea name="s" cols="50" rows="6"><?php echo $daten['Text']; ?></textarea>
Eingaben mit Programm <i>aendern.php</i>: <input type="submit" value="ändern">
</form>
```

Hierbei werden die Daten aus der Datenbank mit z.B. **value="<?php echo $daten['Name']; ?>"** als Anzeigetext in die INPUT-Tags übernommen. Nach der Veränderung werden die Daten durch die Vorgabe **action="aendern.php?todo=anders&id=<?php echo $id; ?>"** via HTML wieder an das ursprüngliche Programm geschickt. Dabei werden mit der Methode GET todo=anders und id=id mitgeschickt. Todo=anders ist notwendig, damit das Einpflegen in die Datenbank nur dann erfolgt, wenn **<form> ... </form>** auch wirklich benutzt wurde.

Die Variablen werden nach GET- und POST-Methode getrennt bereitgestellt:

```
$id = $_GET["id"];   $todo = $_GET["todo"];
$z = $_POST["z"];  $n = $_POST["n"];  $s = $_POST["s"];
```

Und falls $todo == ‘anders’ gesetzt ist in die Datenbank mittels **UPDATE** eingepflegt:

```
if($todo=='anders') {$query="UPDATE Informatik SET Name='$z', Vorname='$n', Text='$s' WHERE id='$id'";
                     $ergebnis=mysql_query($query);
                     $todo=”nix”;
                    }
```

Wichtig ist dabei $todo einen anderen Wert als aendern zuzuweisen, um fehlerhafte, unabsichtliche Änderungen zu vermeiden.

Datensatz neu in eine Datenbank-Tabelle einfügen

siehe *Beispiel 14*

Um einen neuen Datensatz einzugeben muss die Verbindung zur Datenbank aufgebaut werden.

Die Tastatureingaben werden in HTML über **<form> ... </form>** vorgenommen:

```
<form action="neu.php?todo=neu" method="post">
neuer Name: <input type="text" name="z"><br>
neuer Vorname: <input type="text" name="n"> <br>
neuer Text: <textarea name="s" cols="50" rows="6"></textarea><br><br>
Neueingaben mit Programm <i>neu.php</i>: <input type="submit" value="eintragen">
</form>
```

Durch die Vorgabe **action="neu.php?todo=neu"** via HTML werden die eingegebenen Daten wieder an das ursprüngliche Programm geschickt. Dabei wird mit der Methode GET todo=neu mitgeschickt. Todo=neu ist notwendig, damit das Einpflegen in die Datenbank nur dann erfolgt, wenn **<form> ... </form>** auch wirklich benutzt wurde.

Die Variablen werden nach GET- und POST-Methode getrennt bereitgestellt:

```
$todo = $_GET["todo"];
$z = $_POST["z"];  $n = $_POST["n"];  $s = $_POST["s"];
```

Und falls $todo == ‘neu’ gesetzt ist in die Datenbank mittels **INSERT INTO** eingepflegt:

```
if($todo=='neu') {   $query="INSERT INTO Informatik SET Name='$z', Vorname='$n', Text='$s' ";
                     $ergebnis=mysql_query($query);
                     $todo=”nix”;
                 }
```

Wichtig ist dabei $todo einen anderen Wert als aendern zuzuweisen, um fehlerhafte, unabsichtliche Eintragungen zu vermeiden.