

Datenbanken und SQL

Kapitel 6

Datenbankprogrammierung mit PHP

Datenbankprogrammierung mit PHP

- ▶ **Zusammenspiel: Browser, Webserver und Datenbank**
- ▶ **Kurzeinführung in HTML und PHP**
- ▶ **Einfache Datenbankzugriffe**
- ▶ **Fehlerbehandlung**
- ▶ **Verwendung von Sessionvariablen**
- ▶ **Umfangreiche Lese- und Schreibzugriffe auf Datenbanken**
- ▶ **Transaktionsbetrieb**
- ▶ **Arbeiten mit binären Daten (Bilder) in Datenbanken**

Browser und Webserver (1)

▶ Browser

- ▶ Beispiele: Internet Explorer, Firefox, Chrome, Safari, usw.
- ▶ Browser sind lokal installiert
- ▶ Browser lesen HTML-Seiten und geben diese formatiert aus
- ▶ Browser kennen Javascript und Flash als Programmiersprachen

▶ Webserver

- ▶ Beispiele: IIS unter Windows, Apache unter Unix und Windows
- ▶ Webserver bieten Dienste an
- ▶ Webdienste können statisch sein (nur HTML)
- ▶ Webdienste können dynamisch sein (JSP, PHP, Perl, ASP, ...)

Browser und Webserver (2)

Ergebnis vom
Browser dargestellt



über Telefonleitung, Glasfaser
oder drahtlos (LAN, WLAN)

Anfrage



Ergebnis

statisch (nur HTML)
dynamisch (PHP, ...)



Im Browser:
<http://www.google.de>

vom Name-Server umgesetzt
in z.B.: 194.205.10.255

Webserver:
verarbeitet Anfrage

Webserver und Datenbanken (1)

▶ Webserver

- ▶ Dynamische Dienste benötigen meist viele Daten
- ▶ Daten werden in Dateien gespeichert oder in Datenbanken
- ▶ Webserver greift über Datenbankschnittstellen zu

▶ Datenbank

- ▶ Datenbank steht im Netz (Internet, Cloud)
- ▶ Datenbank ist geschützt
- ▶ Datenbank kann vom Webserver aufgerufen werden
- ▶ Datenbank speichert neue Daten und gibt Daten zurück

Webserver und Datenbanken (2)

Ergebnis vom
Browser dargestellt



Anfrage



Ergebnis



Abruf
von
Daten



Im Browser:
<http://www.google.de>

Webserver:
verarbeitet Anfrage

Datenbank:
liefert Daten

verarbeitet
Datenbankdaten

lokales schnelles LAN
oder Internet

Aufbau einer HTML-Seite

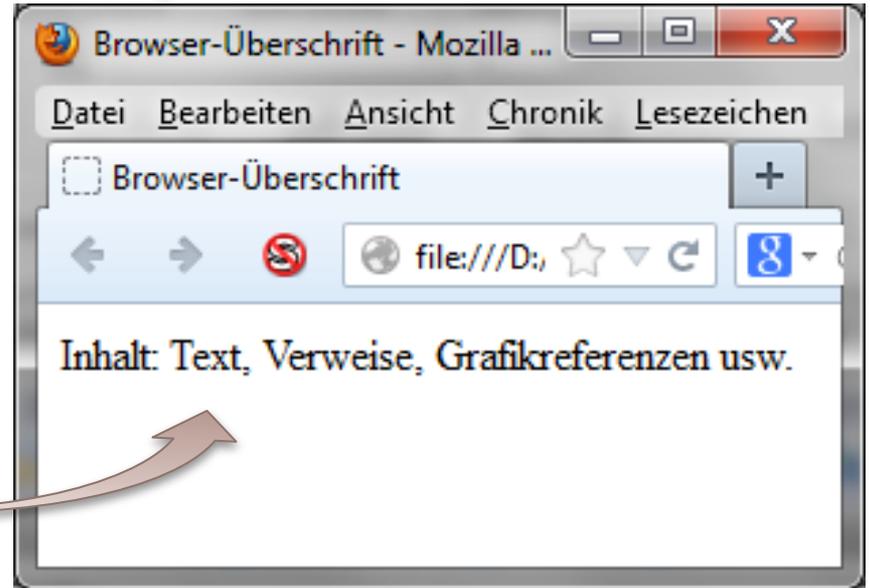
► **Empfehlung:** <http://de.selfhtml.org>

► **Aufbau:**

```
<html>  
  <head>  
    <title>Browser-Überschrift</title>  
  </head>  
  <body>  
    Inhalt: Text, Verweise, Grafikreferenzen usw.  
  </body>  
</html>
```

Tag

Ende-Tag



Einfache Textformatierung

- ▶ Der Browser bricht den Text automatisch um
- ▶ Wichtig sind also nur Formatangaben:
 - ▶ Absatzformatierungen, Überschriften, Fettdruck, ...

▶ Beispiel:

h: Hierarchy

p: Paragraph

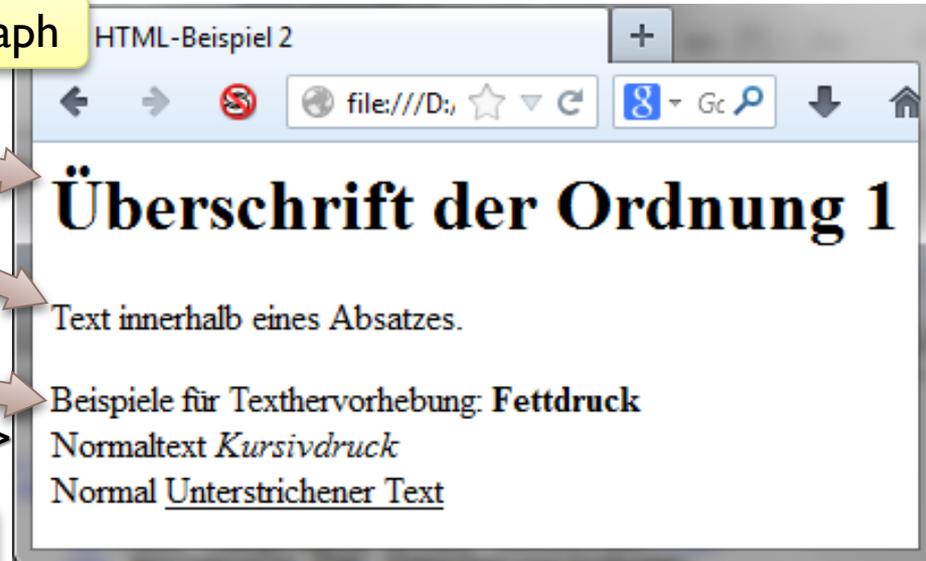
```
<h1>Überschrift der Ordnung 1</h1>
<p>Text innerhalb eines Absatzes.</p>
<p> Beispiele für Text hervorhebung:
  <b>Fettdruck</b><br/>
  Normaltext <i>Kursivdruck</i><br/>
  Normal <u>Unterstrichener Text</u></p>
```

b: Bold

u: Underline

i: Italic

br: Break



Tabellen in HTML

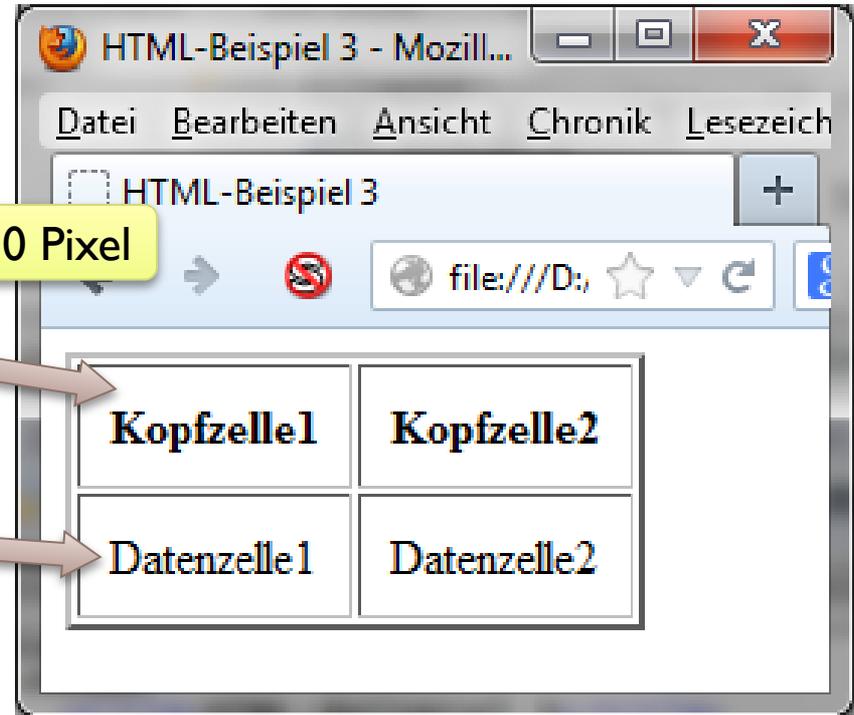
- ▶ Tabellen sind in HTML 4.0 ein Formatierungswerkzeug!

- ▶ Beispiel:

Randstärke: 2 Pixel

```
<table border="2" cellpadding="10">
<tr>
<th>Kopfzelle 1 </th>
<th>Kopfzelle 2</th>
</tr>
<tr>
<td>Datenzelle 1 </td>
<td>Datenzelle 2</td>
</tr>
</table>
```

Textabstand: 10 Pixel



Weitere Möglichkeiten

▶ Links:

▶ Link auf andere Webseite:

`Google-Startseite`

Linkadresse

Hinweis zu Adresse

▶ Link auf interne Seite *beispiel.html* im Unterverzeichnis *test*

`Beispielseite`

▶ Bild anzeigen:

▶ Bild *img.jpg* aus Verzeichnis *image* anzeigen:

``

Link auf Bild

Max. Höhe
und Breite

„/“ !

▶ Manche Tags besitzen keine Endetags, z.B. `
`, ``

„/“ beachten!

Formulare in HTML

- ▶ Zu Formularen gibt es ein Form-Tag: `<form ... >`
- ▶ Es gibt beispielsweise folgende Formularfelder:
 - ▶ Textfeld: `<input type="text" ... />`
 - ▶ Radiobutton: `<input type="radio" ... />`
 - ▶ Checkbox: `<input type="checkbox" ... />` falls `>|`
 - ▶ Auswahlfeld: `<select size="20" ... > <option ... > ...`
 - ▶ Combobox: `<select size="1" ... > <option ... > ...` falls `|`
 - ▶ Button: `<input type="submit" ... />`
 - ▶ Reset-Button: `<input type="reset" ... />`

Funktionsweise von Formularen

```
<form action="start.php" method="post">
```

Methode **post**: Interne Übergabe der Formularinhalte
Methode **get**: Übergabe der Formularinhalte in Browserzeile

```
<input ... />
```

... hier die Datei „start.php“ aufgerufen.

Beim Klick auf den Submit-Button wird ...

Formular abschicken

```
<input type="submit" value="Formular abschicken"/>  
</form>
```

Beispiel zu Formularen (1)

Linkadresse

Methode post

```
<form action="formulare.php" method="post">
  <p>Server: <input type="text" size="20"
    name="Server"/></p>
  <p>Datenbank: <input type="text" size="20"
    name="Datenbank"/></p>
  <p>Kennung: <input type="text" size="20"
    name="Kennung" value="bike"/></p>
  <p>Passwort: <input type="password"
    size="20" name="Passwort"/></p>
  <p><input type="submit" value="Start"/></p>
</form>
```

The screenshot shows a web browser window displaying a form. The form contains four text input fields and one submit button. The fields are labeled 'Server:', 'Datenbank:', 'Kennung:', and 'Passwort:'. The 'Kennung:' field contains the text 'bike'. The 'Passwort:' field is masked with black dots. The submit button is labeled 'Start'. Arrows from the code on the left point to each of these elements in the screenshot.

Die Formatierung muss selbst gesetzt werden!

Beispiele zu Formularen (2)

<p>Geschlecht:

männlich <input type="radio" name="Anrede" value="Herr" checked />

weiblich <input type="radio" name="Anrede" value="Frau"/></p>

Vorab gesetzt

<p>Interesse:

Lesen <input type="checkbox" name="Interesse1" value="Buch" checked />

Filme ansehen <input type="checkbox" name="Interesse2" value="Video"/>

Musik hören <input type="checkbox" name="Interesse3" value="Audio"/> </p>

Vorab gesetzt

<p>Familienstand:

<select name="Familienstand" size="1">

<option> ledig </option>

<option> verheiratet </option>

<option> geschieden </option>

<option> verwitwet </option>

</select></p>

Combobox

Inhalt der
Combobox

Geschlecht: männlich weiblich

Interesse: Lesen Filme ansehen Musik hören

Familienstand:

- ledig
- verheiratet
- geschieden
- verwitwet

PHP

- ▶ PHP = PHP Hypertext Preprocessor
- ▶ 1995: von Rasmus Lerdorf vorgestellt
- ▶ PHP wird in HTML eingebettet
- ▶ PHP-Dateien besitzen die Endung „.php“
- ▶ PHP wird auf einem Webserver ausgeführt
- ▶ ~70% aller Serverprogramme sind in PHP geschrieben
- ▶ PHP ist stark an Perl, C und Java angelehnt

Gemeinsamkeiten: PHP und Java/C

- ▶ **Blockstruktur, Anweisungen, Funktionen:**

- ▶ { ... } Strichpunkt am Ende einer Anweisung

- ▶ **Kontrollstrukturen:**

- ▶ if, switch

- ▶ for, while, do ... while

Identische Syntax!

- ▶ **Operatoren:**

- ▶ nur minimale Unterschiede, zusätzliche Operatoren in PHP
- ▶ Konkatenierungsoperator bei Strings: Punkt (.,')

- ▶ **Groß- und Kleinschreibung wird unterschieden**

Unterschied: PHP zu Java/C

	in PHP
Datentypen	Sechs Datentypen: int, bool, double, string, array, object
Variablen	Erstes Zeichen ist immer das Dollarzeichen (,\$')
Operatoren	Zusätzlich: ===, !==; Operator -> für Objektmethoden
Deklaration von Variablen	Deklaration nicht erforderlich; Variablen ändern den Datentyp dynamisch
Zeichenketten	Eigene Funktionalität, ähnlich zu Java
Funktionen	Funktionsnamen sind case-insensitiv

Wichtige Funktionen in PHP

<code>trim(str)</code>	entfernt Leerzeichen am Anfang und Ende von <code>str</code>
<code>strlen(str)</code>	gibt die Länge der Zeichenkette <code>str</code> zurück
<code>strpos(str1, str2)</code>	gibt das erste Vorkommen von <code>str2</code> in der Zeichenkette <code>str1</code> zurück; bzw. <code>false</code> , falls nicht enthalten
<code>strcmp(str1, str2)</code>	vergleicht die Zeichenketten <code>str1</code> und <code>str2</code>
<code>strcasecmp(str1, str2)</code>	vergleicht die Zeichenketten <code>str1</code> und <code>str2</code> unabhängig von Groß- und Kleinschreibung
<code>substring(str, pos, len)</code>	gibt einen Teilstring von <code>str</code> der Länge <code>len</code> ab Position <code>pos</code> zurück
<code>implode(str, feld)</code>	liefert Zeichenkette mit allen Feldelementen zurück, die mittels der Zeichenkette <code>str</code> verknüpft werden
<code>stripslashes(str)</code>	entfernt Entwertungszeichen <code>,\'</code> in der Zeichenkette <code>str</code>
<code>echo param1, ...</code>	gibt die Parameterliste auf HTML aus
<code>isset(var)</code>	liefert <code>true</code> , wenn Variable <code>var</code> existiert, sonst <code>false</code>
<code>unset(var)</code>	setzt die Variable <code>var</code> zurück, <code>var</code> existiert nicht mehr

Zeichenketten in PHP

- ▶ Zeichenketten entweder: "Dies ist ein String."
- ▶ Zeichenketten oder: 'Dies ist ein String.'
- ▶ Unterschied:

- ▶ In ""-Strings werden Variablen substituiert

- ▶ Beispiel:

```
$zahl = 100 ;
```

```
echo "<p>Die Zahl $zahl ist größer als Null</p>";
```

```
echo '<p>Die Zahl $zahl ist größer als Null</p>';
```

Hier wird \$zahl durch Inhalt ersetzt

Gänsefüßchen

Hochkomma

hier nicht

- ▶ Ausgabe:

Die Zahl 100 ist größer als Null

Die Zahl \$zahl ist größer als Null

Formulare und PHP

- ▶ Alle Formularfelder besitzen den Parameter **name**
- ▶ Die jeweiligen Formularinhalte werden mit diesem Parameter identifiziert.
- ▶ Beispiel:

```
<form action="formulare.php" method="post">  
<p>Server: <input type="text" size="20" name="Server"/></p>
```
- ▶ In der Datei `formulare.php` ist der Inhalt verfügbar unter:
`$_POST['Server']`
 - ▶ Analoges gilt für Methode `get`: **`$_GET['Server']`**

Formulare mit PHP auslesen

POST-Variablen

formulare.html

formulare.php

Datenbanken und SQL

Einfaches Formular

Server:

Datenbank:

Kennung:

Passwort:

```
<h3>Ausgabe von Formulardaten </h3>
HTML: Wir geben die Variablen aus:
<?php
  echo "<p>PHP: Start von PHP</p>";
  $server = $_POST['Server'];
  $datenbank = $_POST['Datenbank'];
  $benutzer = $_POST['Kennung'];
  $passwort = $_POST['Passwort'];
  echo "<p>PHP: Server: $server;
  Datenbank: $datenbank;
  Kennung: $benutzer,
  Passwort: wird nicht verraten.</p>";
?>
<p>HTML: Ende.</p>
```

Datenbanken und SQL

Ausgabe von Formulardaten

HTML: Wir geben die Variablen aus:

PHP: Start von PHP

PHP:
Server: localhost
Datenbank: ora21c
Kennung: bike
Passwort: wird nicht verraten

HTML: Ende

PHP-Start

PHP-Ende

per POST übergeben

Variablen ausgeben

Felder in PHP (1)

▶ Felder in PHP flexibel: Felder, Listen, Aufzählungen!

▶ Beispiele:

```
$feld = array( 0, 2, 4, 6, 8 );
```

Feld mit 5 Elementen:
\$feld[0]=0, \$feld[1]=2 usw.

```
$feld[ ] = 10;
```

Nächstes Feld: \$feld[5]=10

```
$feld[10] = 20;
```

\$feld[6] existiert nicht, ebenso nicht 7, 8 und 9!

```
$feld["test"] = 100;
```

Assoziative Felder:
Indexe sind Strings

```
$feld["wert"] = "Ende";
```

```
echo "<p>Anzahl der Feldelemente: " . count($feld) . "</p>
```

```
<p>Inhalt:</p>";
```

```
foreach ($feld as $inhalt)
```

Konkatenierung

Anzahl der
Feldelemente: 9

```
echo "<p>$inhalt</p>";
```

Ausgabe aller
Werte mit foreach

Felder in PHP (2)

- ▶ **Felder besitzen einen Index (Schlüssel, key) und einen Inhalt (context). Es gibt den Zuordnungsoperator „=>“**
- ▶ **Also: key => context (z.B. 4=>8)**

- ▶ **Beispiele:**

Beginnend bei 0,
dann 1 usw.

feld[10]=20

feld["test"]=100

```
$feld = array( 0=>0, 2, 4, 6, 8, 10, 10=>20, "test"=>100, "wert"=>"Ende" );  
echo "<p>Anzahl der Feldelemente: " . count($feld) . "</p>
```

```
<p>Inhalt:</p>";
```

```
foreach ($feld as $key => $context)  
    echo "<p>feld[$key] = $context</p>";
```

Ausgabe in Schleife:

feld[0]=0

feld[1]=2

feld[2]=4

usw.

PHP auf Datenbankzugriff vorbereiten

- ▶ **Einstellungen in php.ini (je nach Hersteller):**
 - ▶ `extension=php_pdo_oci.dll`
 - ▶ `extension=php_pdo_mysql.dll`
 - ▶ `extension=php_pdo_sqlsrv_83_nts_x64.dll` 
- ▶ **Microsoft liefert eigene Treiber (z.B. für PHP 8)**
- ▶ **Im Unterverzeichnis ext von PHP müssen diese Treiber installiert sein!**
- ▶ **Die SQL Server Treiber werden von PHP nicht mitgeliefert. Download von Microsoft!**

Schnittstelle: PHP ↔ Datenbank (1)

▶ Datenbankunabhängige Programmierung mit PDO

▶ PDO = PHP Data Objects

▶ Konstruktor der Klasse PDO stellt Verbindung her:

```
$conn = new PDO("oci:dbname=$datenbank",  
                2. Parameter: Kennung, $kennung, $passwort);
```

3. Parameter:
Passwort

```
$conn = new PDO("sqlsrv:server=$server; dbname=$datenbank",  
                $kennung, $passwort);
```

```
$conn = new PDO("mysql:host=$server;dbname=$datenbank",  
                $kennung, $passwort);
```

1. Parameter:
Verbindung

Variable SERVER/HOST

Variable DBNAME

Schnittstelle: PHP \leftrightarrow Datenbank (2)

- ▶ Variable SERVER/HOST: Internetadresse der Datenbank
- ▶ Variable DBNAME: Name der Datenbank
- ▶ Variable KENNUNG und PASSWORT: Identifikation

	Variable Server/Host	Variable Dbname
Oracle lokal	(nicht verwendet)	Datenbank
Oracle entfernt	(nicht verwendet)	//Serveradresse/Datenbank
SQL Server lokal	localhost\sqlexpress	Datenbank
SQL Server entfernt	Serveradresse\sqlexpress	Datenbank
MySQL lokal	localhost	Datenbank
MySQL entfernt	Serveradresse	Datenbank

tatsächlichen
Installationsnamen verwenden

Erster Zugriff auf Datenbank

Kein Hochkomma!

► Einloggen und Transaktionsstart:

Einloggen am
Beispiel Oracle

```
$conn = new PDO("oci:dbname=$_POST[ Datenbank ]",  
                $_POST['Kennung'], $_POST['Passwort']);  
echo "<p>Die Verbindung zur Datenbank wurde hergestellt.</p>";
```

```
$conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

```
$conn->setAttribute(PDO::ATTR_CASE, PDO::CASE_UPPER);
```

Bei Fehler:
Ausnahme werfen

```
$conn->beginTransaction();
```

Objekt -> Methode

1. Parameter: Attribut
2. Parameter: Wert

Spaltennamen in
Großbuchstaben ausgeben

Aus Kompatibilitätsgründen
(Oracle)

Wichtig!
Startet Transaktion in PDO

Ausführen eines Select Befehls

- ▶ **Voraussetzung:**

- ▶ Verbindung mit Datenbank ist hergestellt (siehe letzte Folie)

- ▶ **Beispiel:**

- ▶ Lesen des Namens und Wohnorts von Mitarbeiter 2

```
$sql = " Select Name, Ort From Personal Where Persnr = 2 " ;
```

```
$stmt = $conn->query($sql);
```

Abspeichern in
Objekt vom Typ
PDOStatement

SQL-Befehl
ausführen

Oracle mag keinen
Strichpunkt!!

Auslesen des Ergebnisses

▶ Voraussetzung:

- ▶ Ergebnis steht in Objekt (`$stmt`) vom Typ `PDOStatement`

▶ Beispiel:

- ▶ Ausgabe des Namens und Wohnorts von Mitarbeiter 2

```
if ($row = $stmt->fetch())  
{  
    echo "<p>Der Mitarbeiter mit der Persnr 2 heißt  
        $row[ NAME ] und wohnt in $row[ ORT ]. </p>";  
}  
else  
{  
    echo "<p>Der Mitarbeiter mit dieser Nummer 2 existiert nicht! </p>";  
}
```

Lesen der 1. Zeile aus `$stmt`
Abspeichern im assoziativen Feld `$row`

kein
Hochkomma

Strings dürfen mehrzeilig sein!

falls keine 1. Zeile existiert,
liefert `fetch` `false` zurück

Assoziativer Wert `NAME`
(Großbuchstaben!)

Analog:
`ORT`

Hinweise zum Auslesen

- ▶ **Assoziative Namen beginnen mit einem Buchstaben**
- ▶ **Assoziative Namen enthalten keine Sonderzeichen**
 - ▶ Also: Im Select-Befehl gegebenenfalls Aliasnamen verwenden!
- ▶ **Die Methode fetch liefert assoziative und indizierte Werte**
- ▶ **In unserem Beispiel erhalten wir:**

```
$row[ 'NAME' ]      // da Select Name, ...  
$row[ 'ORT' ]      // da Select ..., Ort, ...  
$row[ 0 ]          // erste Spalte, entspricht $row[ 'NAME' ]  
$row[ 1 ]          // zweite Spalte, entspricht $row[ 'ORT' ]
```

anfällig gegen spätere Änderungen

Beenden einer Transaktion / Sitzung

- ▶ Eine Transaktion wird beendet mit den Methoden:
 - ▶ `commit` // übernimmt alle Daten dauerhaft
 - ▶ `rollback` // setzt alle Änderungen zurück
- ▶ Eine Sitzung wird beendet
 - ▶ durch Rücksetzen der Sitzungsvariable (z.B. `$conn = null;`)
- ▶ Am Ende einer PHP-Datei erfolgen folgende Aktionen:
 - ▶ Alle noch offenen Transaktionen werden zurückgesetzt (Rollback)
 - ▶ Alle noch offenen Verbindungen werden beendet
- ▶ Unser Beispiel:
 - Führt Commit durch
 - Beendet die Verbindung/Sitzung

```
$conn->commit();  
$conn = null;  
echo "<p>Die Verbindung zur Datenbank wurde geschlossen. </p>";
```

Einführung in die Fehlerbehandlung

- ▶ In PDO: Fehlerklasse PDOException
- ▶ Methoden der Fehlerklassen Exception und PDOException:
 - ▶ `getMessage` gibt einen Fehlertext aus
 - ▶ `getCode` gibt den Fehlercode aus
 - ▶ `getLine` gibt Fehlerzeile aus
- ▶ PDO wirft Fehler vom Typ PDOException, falls:
 - ▶ `PDO::ATTR_ERRMODE` ist `PDO::ERRMODE_EXCEPTION`
 - ▶ Wird nach Verbindungsaufbau mit Methode `setAttribute` gesetzt
- ▶ Trennung von Code und Fehlercode:
 - ▶ Try-Block enthält den Code
 - ▶ Catch-Blöcke fangen gegebenenfalls geworfene Fehler ab

Fehlerbehandlung am Beispiel

```
try {  
    $conn = new PDO("oci:dbname=$_POST[Datenbank]", $_POST['Kennung'],  
                   $_POST['Passwort']);  
  
    // ... Zugriffe  
    $conn->commit();  
    echo "<p>Die Verbindung zur Datenbank wird geschlossen. </p>";  
}  
catch (PDOException $e) {  
    echo "<p>PDO-Fehler in Zeile ", $e->getLine(), "mit Code ", $e->getCode(),  
        "</p><p>Fehlertext: ", $e->getMessage(), "</p>";  
}  
catch (Exception $e) {  
    echo "<p>Fehler in Zeile ", $e->getLine(), "mit Code ", $e->getCode(),  
        "</p><p>Fehlertext: ", $e->getMessage(), "</p>";  
}
```

Normaler Programmcode, nicht belastet durch Fehlerbehandlungen

Fängt zunächst alle Datenbankfehler ab

Von Exception abgeleitete Klasse

Fängt noch alle anderen PHP-Fehler ab

Standardfehlerklasse in PHP

Hinweise zur Fehlerbehandlung

- ▶ Im Fehlerfall wird in den ersten zutreffenden Catch-Block gesprungen
- ▶ Alle im dazugehörigen Try-Block definierten Variablen sind dann nicht mehr gültig. Im Beispiel:
 - ▶ \$conn existiert im Catch-Block nicht mehr
 - ▶ Die Verbindung wurde also aufgelöst!
 - ▶ Die Transaktion wurde also zurückgesetzt!
 - ▶ Ein explizites „\$conn->rollback()“ führt im Catch-Block zu einem Fehler und damit zum Absturz des Programms
- ▶ Wir verwenden im Catch-Block ausschließlich die Methoden der Klassen Exception und PDOException

Auslesen mehrerer Datenzeilen

- ▶ **Programmidee:**
 - ▶ Eingabe eines Suchstrings
 - ▶ Ausgabe aller Mitarbeiter, die Suchstring im Namen enthalten
- ▶ **Realisierung:**
 - ▶ Komfort: Unterstützung der wiederholten Eingabe eines Suchstrings ohne erneute Eingabe der Anmeldedaten
 - ▶ Ein- und Ausgabe im gleichen PHP-Programm
- ▶ **Problem:**
 - ▶ Beim ersten Aufruf des PHP-Programms ist ein Einloggen noch nicht möglich, da die Anmeldedaten erst einzugeben sind!
- ▶ **Lösung:**
 - ▶ Funktion **isset**
 - ▶ Funktion überprüft, ob Variable schon existiert

Anwendung der Funktion isset

sich selbst aufrufen:

```
<form action="mitarbeiter.php" method="post">
  <p>Bitte Daten zum Einloggen eingeben</p>
  ...      <!-- Server- und Datenbankdaten -->
  <p>Kennung eingeben:</p>
  <input type="Text" name="Kennung" size="20" value=
    <?php echo isset($_POST['Kennung']) ? $_POST['Kennung'] : "bike" ;?> />
  <p>Passwort eingeben:</p>
  <input type="Password" name="Passwort" size="20" value=
    <?php echo isset($_POST['Passwort']) ? $_POST['Passwort'] : "" ;?> />
  <p>Suchzeichen zur Mitarbeitersuche:</p>
  <input type="Text" name="Suchstring" size="30" value=
    <?php echo isset($_POST['Suchstring']) ? $_POST['Suchstring'] : "" ;?> />
  <input type="Submit" value="Weiter" />
</form>
```

Beim ersten Aufruf gibt es diese POST-Variable noch nicht

1. Aufruf: Vorgabe „bike“

Ab 2. Aufruf: Letzter Wert

Problem: Passwort ist sichtbar!

Mit Suchstring: Suche von Mitarbeitern

```
if (isset($_POST['Kennung'])) {  
    $suche = trim($_POST['Suchstring']); // ev. Leerzeichen entfernen  
    if (strlen($suche) == 0) {  
        echo "<p>Es wurde kein Suchstring eingegeben. </p>";  
    } else {  
        try {  
            echo "Suche nach Mitarbeitern, die im Namen den String $suche enthalten.";
```

Code wird erst ab
Zweitaufruf durchlaufen

eventuelle Leerzeichen entfernen

Bei leerer Eingabe
reagieren

hier: Einloggen, setAttribute, beginTransaction

```
$sql = "Select Persnr, Name, Ort, GebDatum, Gehalt, Vorgesetzt  
    From Personal  
    Where Upper( Name ) Like Upper( '%$suche%' ) ";  
$stmt = $conn->query( $sql );
```

Upper: Unabhängig
von Groß- und
Kleinschreibung

Suche mit Like
und Wildcart

Ausgabe in eine Tabelle (1)

```
if ( !($row = $stmt->fetch()) ) {  
    echo "Mitarbeiter mit dem Teilstring im Namen existiert nicht!";  
}  
else {  
?>         <!-- Tabelle aufbauen -->  
<table border cellpadding=10>  
  <tr> <!-- Erste Tabellenzeile -->  
    <th>Persnr </th>  
    <th>Name </th>  
    <th>Ort </th>  
    <th>GebDatum </th>  
    <th>Gehalt </th>  
    <th>Vorgesetzter? </th>  
  </tr>
```

Überprüfen, ob
Ergebnisse vorliegen

wenn nein:
darauf hinweisen

wenn ja:
Tabelle aufbauen

Erste Zeile
(Überschriftenzeile)
ausgeben

Ausgeben in eine Tabelle (2)

```
<?php
  do {
?> <tr>
  <td> <?php echo $row["PERSNR"] ?> </td>
  <td> <?php echo $row["NAME"] ?> </td>
  <td> <?php echo $row["ORT"] ?> </td>
  <td> <?php echo $row["GEBDATUM"] ?> </td>
  <td> <?php echo $row["GEHALT"] ?> </td>
  <td> <?php echo ($row["VORGESETZT"] == null) ? "Ja": "Nein"; ?></td>
</tr>
<?php
  } while ($row = $stmt->fetch());
?></table>
```

Mit do-while-Schleife
Daten auslesen

Neue Zeile

Alle Spalten
ausgeben

Überprüfen auf NULL

auslesen, solange
Daten vorhanden sind

Beenden mit
Commit

Typisches Auslesen von Daten (1)

▶ Variante I:

- ▶ Mittels Fetch in einer While-Schleife
- ▶ Keine Überprüfung der Daten vor der Schleife

```
while ( $row = $stmt->fetch() )
```

```
{
```

```
    /* Ausgabe der Attribute der aktuellen Zeile  
       mit Hilfe des assoziativen Feldes $row */
```

```
}
```

Typisches Auslesen von Daten (2)

▶ Variante 2:

- ▶ Mittels Fetch, If-Anweisung und Do-While-Schleife
- ▶ Überprüfen der Daten vor der Schleife

```
if ( !($row = $stmt->fetch()) )
```

Vorab wird erste
Zeile überprüft

```
    echo "<p>Keine Daten</p>";
```

```
else
```

```
do {
```

Jetzt muss erst die erste Zeile
ausgegeben werden, bevor
weitere Daten gelesen werden

```
    // Ausgabe der Attribute ab der ersten Zeile mit $row
```

```
} while ( $row = $stmt->fetch() );
```

also: Do-While

Auslesen mit Fetch: CURSOR

- ▶ Vor dem ersten Fetch-Aufruf zeigt Cursor vor die 1. Zeile
- ▶ Bei jedem Aufruf wird betreffende Zeile gelesen, und der Cursor geht zur nächsten Zeile
- ▶ Zeigt Cursor hinter die letzte Zeile, liefert der Folgebefehl false

Beim Start



Cursor *\$stmt*: 

Im nächsten Schritt



Am Ende

1	Maria Forster	Regensburg	05.07.79	JA
2	Anna Kraus	Regensburg	09.07.75	NEIN
3	Ursula Rank	Frankfurt	04.09.67	NEIN
4	Heinz Rolle	Nürnberg	12.10.57	NEIN
5	Johanna Köster	Nürnberg	07.02.84	NEIN
6	Marianne Lambert	Landshut	22.05.74	JA
7	Thomas Noster	Regensburg	17.09.72	NEIN
8	Renate Wolters	Augsburg	14.07.79	NEIN
9	Ernst Pach	Stuttgart	29.03.92	NEIN

Sessionvariable

- ▶ Sessionvariable gelten über PHP-Seiten hinweg
- ▶ Damit können Daten eingelesen werden und über viele Seiten verwendet werden
- ▶ Feld mit dem Namen `$_SESSION`:
 - ▶ `$_SESSION['Name der Variable']`
 - ▶ Analoges Handling wie `$_POST`, `$_GET`
- ▶ Voraussetzung:
 - ▶ PHP-Seite muss in Zeile 1 als Sessionseite definiert werden mit:

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

Beispiel mit Sessionvariablen

```
$_SESSION[ 'Kennung' ] = "abc12345";
```

Zwei
Sessionvariablen
erzeugt

```
$_SESSION[ 'Nr' ] = 17;
```

```
if (isset($_SESSION[ 'Kennung' ]))
```

Überprüfen, ob Sessionvariable
Kennung existiert

```
{
```

```
    echo "Variable Kennung existiert, wird nun wieder entfernt.";
```

```
    unset($_SESSION[ 'Kennung' ]);
```

Sessionvariable Kennung
existiert nun nicht mehr

```
}
```

```
echo "<p>Die Sessionvariable Nr hat den Wert:
```

Inhalt einer
Sessionvariable ausgeben

```
$_SESSION[ Nr ].</p>";
```

Arbeiten mit Datenbanken und Session

▶ Startseite:

- ▶ Benutzerdaten anfordern (z.B. Kennung und Passwort)

▶ Folgeseite:

- ▶ Benutzerdaten aus POST-Variablen auslesen und in SESSION-Variablen speichern

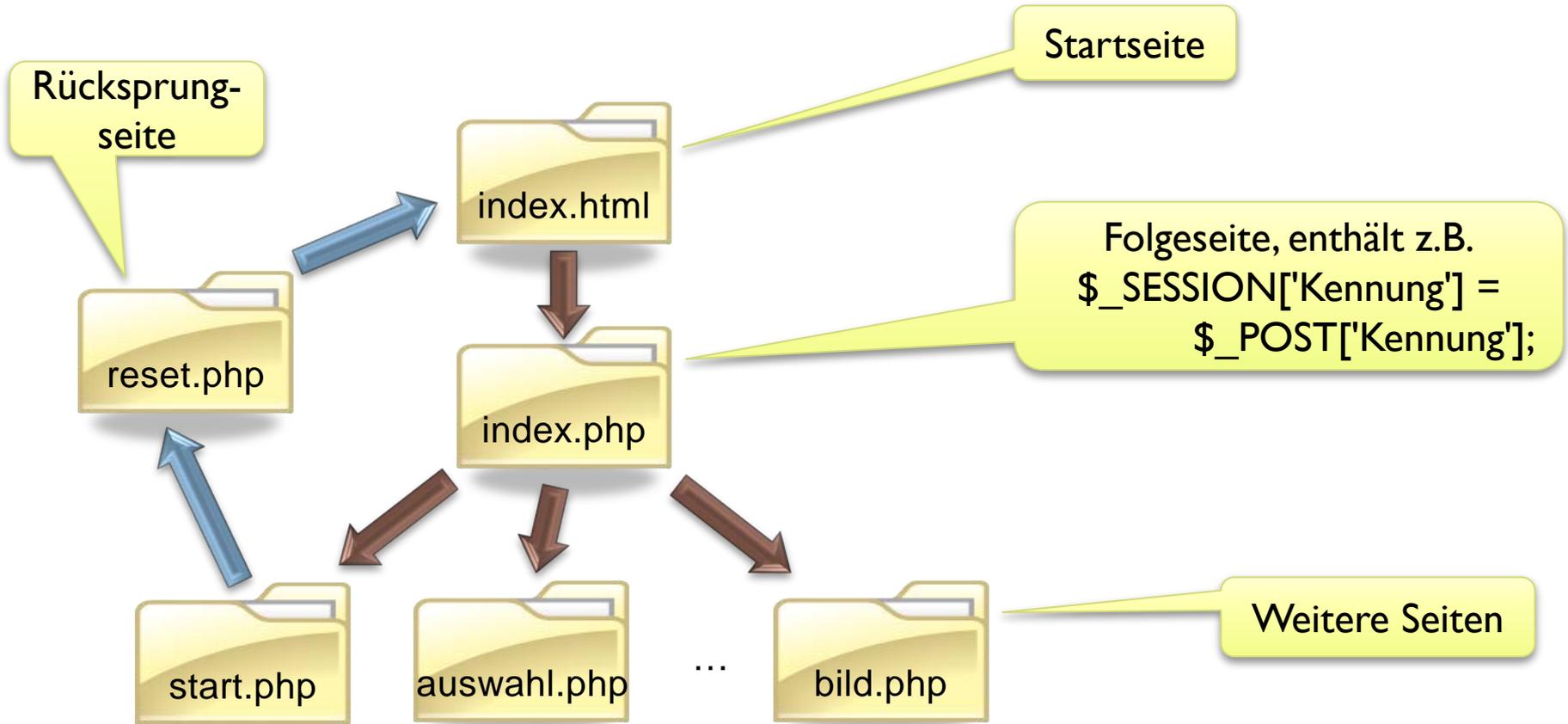
▶ Weitere Seiten:

- ▶ SESSION-Variablen verwenden (z.B. zum Einloggen in Datenbank)
- ▶ Um Quereinstieg zu unterbinden: Existenz der SESSION-Variablen überprüfen

▶ Rücksprungseite

- ▶ Zurücksetzen der SESSION-Variablen

Beispiel zu Session



Handling der weiteren Seiten

- ▶ Verhindern des Quereinstiegs mit Vermeidung von Folgefehlern:

```
if ( !isset($_SESSION['Kennung']) ||
    !isset($_SESSION['Passwort']) ||
    !isset($_SESSION['Server']) ||
    !isset($_SESSION['Datenbank']) )
    echo "Gehen Sie zur <a href='start.html'>Startseite</a>";
else
{
    $conn = new PDO( ...
```

Überprüfung, ob
SESSION-Variablen
gesetzt

Wenn nein,
dann zurück
zur Startseite

Wenn ja, dann
zugreifen

Auswahl mittels Select-Boxen

Datenbanken und SQL

Edwin Schicker

Dies ist die Datei *auswahl.php* im Sessionteil, die Datei ruft sich selbst wieder auf.

Die Verbindung zur Datenbank OracleUTF8 wurde hergestellt.

Bitte wählen Sie einen Artikel aus und dazu einen oder mehrere Kunden aus. Durch Klick auf den Weiter-Button wird eine Liste generiert, die angibt, wann und wie oft diese genau diesen Artikel in Auftrag gegeben haben.

Sofortiger Zugriff dank
SESSION möglich

Bitte wählen Sie einen Artikel aus:

Herren-City-Rad

Combobox

Bitte wählen Sie einen oder mehrere Kunden aus:

Fahrrad Shop
Zweirad-Center Staller
Maier Ingrid
Rafa - Seger KG
Biker Ecke

Select-Box mit
Mehrfachauswahl

Weiter

Ausgabe der Artikel in Combo-Box

```
$sqlI = "Select Anr, Bezeichnung From Artikel";  
$stmtI = $conn->query($sqlI);  
  
if ( $rowI = $stmtI->fetch() )  
{ echo "<select name=\"Artikelnr\" size=\"I\">" ;  
  do  
  { echo "<option value=\""$rowI[ANR]\"";  
    if (isset($_POST['Artikelnr']) and $_POST['Artikelnr'] == $rowI['ANR'])  
      echo " selected";  
    echo "> $rowI[BEZEICHNUNG]</option>";  
  } while ($rowI = $stmtI->fetch());  
  echo "</select>";  
} else ...
```

Alle Artikel lesen

Existieren Artikel?

Combobox: size=I

... aber Artikelnummern merken

Vorherige Auswahl merken

Artikelnamen anzeigen ...

In Schleife alles ausgeben

Ausgabe der Kunden in Select-Box (1)

```
$sql2 = "Select Nr, Name From Kunde";
```

Alle Kunden lesen

```
$stmt2 = $conn->query($sql2);
```

```
if ( $row2 = $stmt2->fetch() )
```

Select-Box mit 5 Einträgen

```
{ echo "<select multiple name=\"Kundnr[]\" size=\"5\">" ;
```

Mehrfachauswahl

Feldübergabe

▶ Problem: Mehrfachauswahl

▶ Lösung: Übergabe eines Feldes im Parameter Name

▶ Im Feld werden nur alle angeklickten Elemente aufgenommen

▶ Problem: Wie merken wir uns die angeklickten Kunden?

Ausgabe der Kunden in Select-Box (2)

```
do {  
    echo "<option value=\""$row2[NR]\"";  
    if (isset($_POST['Kundnr']) and  
        in_array($row2['NR'], $_POST['Kundnr'] )  
        echo " selected";  
    echo "> $row2[NAME]</option>";  
} while ($row2 = $stmt2->fetch());  
  
echo "</select>";
```

... Kundennummer merken

auswählen, falls NR in der Auswahlliste
\$_POST['Kundnr'] enthalten

Kundenname anzeigen ...

Ausgabe der dazugehörigen Aufträge

- ▶ Die markierten Angaben sind als Tabelle auszugeben
- ▶ Die Ausgabe in eine Tabelle wurde bereits behandelt

Alle Auftragsdaten ...

```
$sql = "Select A.Auftrnr, Kundnr, Datum, Persnr, Anzahl, Gesamtpreis  
From Auftrag A Inner Join Auftragsposten AP
```

... für die ausgewählten Kunden ...

```
On A.Auftrnr=AP.Auftrnr
```

```
Where Kundnr In (" . implode( ",", $_POST['Kundnr'] ) . ")
```

```
And Artnr = $_POST[Artikelnr]";
```

... und den Artikel

```
$stmt = $conn->query($sql);
```

Implode: Fasst Feldelemente zu einer durch Komma getrennten Aufzählung zusammen

Verwendung einer Textarea

- ▶ Textarea:
 - ▶ Mehrzeiliges Eingabefeld
- ▶ Beispiel:

```
<form action="select.php" method="post">
  <textarea name="Eingabe" rows="10" cols="60" wrap="virtual" >
  <?php
    if (isset($_POST['Eingabe']))
      echo trim(htmlspecialchars($_POST['Eingabe']));
  ?>
  </textarea><br/>
  <input type="Submit" value="Ausführen"/>
</form>
```

10 Zeilen à 60 Spalten

Automatischer Umbruch

Ab 2. Aufruf letzte Eingabe anzeigen

Verhindert nicht erwünschten HTML-Code

Auswerten beliebiger SQL-Befehle

- ▶ **Befehl hängt vom ersten Wort ab**
 - ▶ Also: Erstes Wort der Eingabe isolieren

- ▶ **Realisierung:**

- ▶ `$Eingabe = trim($_POST['Eingabe']);`

Eingabe
bereinigen

- ▶ `$stmt = $conn->query($Eingabe);`

SQL-Befehl abschicken

- ▶ `$erstesWort = substr($Eingabe, 0, 6);`

Enthält die ersten 6 Zeichen
der Eingabe

Falls erstes Wort: SELECT

```
if (strCaseCmp($erstesWort,"SELECT")==0) {  
    if ( $row = $stmt->fetch(PDO::FETCH_ASSOC) ) {  
        echo "<table border=\"2\" cellpadding=\"2\"><tr>";  
        foreach ( $row as $colname => $data )  
            echo "<th> $colname </th>";  
        echo "</tr>";  
        do {  
            echo "<tr>";  
            foreach ( $row as $data )  
                echo "<td>" . ($data == null ? "(null)" : $data) . "</td>";  
            echo "</tr>";  
        } while ( $row = $stmt->fetch(PDO::FETCH_ASSOC) );  
        echo "</table>"; } }
```

Select-Befehl

nur assoziative
Werte, keine
Index-Werte!

Key-Werte der ersten
Zeile lesen, um
Spaltennamen auszugeben

Schleife über alle Zeilen

Schleife über alle Spalten:
Daten ausgeben

falls NULL: (null) ausgeben

Falls anderer Befehl

Vergleich unabhängig von Groß- und Kleinschreibung

- ▶ Kein Select, also: keine Ergebnisausgabe
- ▶ DML-Befehl: Ausgabe der Anzahl der manipulierten Zeilen

```
elseif (strCaseCmp($erstesWort,"INSERT")==0)
    echo $stmt->rowCount(), " Zeile(n) wurde(n) eingefügt.<br>";
elseif (strCaseCmp($erstesWort,"UPDATE")==0)
    echo $stmt->rowCount(), " Zeile(n) wurde(n) geändert.<br>";
elseif (strCaseCmp($erstesWort,"DELETE")==0)
    echo $stmt->rowCount(), " Zeile(n) wurde(n) gelöscht.<br>";
else
    echo "Ein DDL-Befehl wurde ausgeführt.<br>";
```

Anzahl der manipulierten Zeilen

SQL Injection

▶ Ausgangslage:

- ▶ PHP Programme werden im Internet ausgeführt
- ▶ PHP Programme enthalten SQL-Befehle
- ▶ In Internetmasken werden Daten eingegeben!!

▶ Problem:

- ▶ Unberechtigte Eingaben können zu Sicherheitslücken führen

▶ Zwei Probleme werden vorgestellt

- ▶ Unerlaubter PHP-Code wird eingeschleust
- ▶ Unerlaubte SQL-Befehle werden ausgeführt

Beispiel 1: Schadcode ausführen

Dies ist die Datei *injection1.php* im Sessionteil; die Datei ruft sich selbst wieder auf.

Geben Sie einen SQL-Befehl ein.

Achtung: Jeder Einzelbefehl wird **committed!**

Im ungeschützten Modus kann Schadsoftware eingeschleust werden. Bitte geben Sie als Beispiel ein: `</textarea></td></tr></table>Hier könnte Schadsoftware stehen`

Im geschützten Modus stört diese Eingabe nicht!

SQL-Befehl
(ohne Semikolon!):

```
</textarea></td></tr></table>Hier könnte Schadsoftware  
stehen
```

Ungeschützt:



Geschützt:



Ausführen



Beispiel 1: Ergebnis

Dies ist die Datei *injection1.php* im Sessionteil; die Datei ruft sich selbst wieder auf.

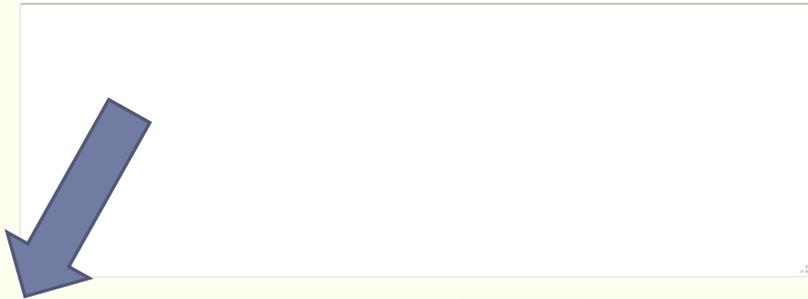
Geben Sie einen SQL-Befehl ein.

Achtung: Jeder Einzelbefehl wird **committed!**

Im ungeschützten Modus kann Schadsoftware eingeschleust werden. Bitte geben Sie als Beispiel ein: `</textarea></td></tr></table>` *Hier könnte Schadsoftware stehen*

Im geschützten Modus stört diese Eingabe nicht!

SQL-Befehl
(ohne Semikolon!):



Hier könnte Schadsoftware stehen Ungeschützt: Geschützt:

Die Verbindung zur Datenbank Oracle wurde hergestellt.

Datenbankfehler in Zeile 88 mit Fehlercode HY000

Fehlertext: SQLSTATE[HY000]: General error: 900 OCIStmtExecute: ORA-00900: Ungültige SQL-Anweisung (ext\pdo_oci\oci_statement.c:148)

Beispiel 1: Der Code

► Ungeschützt:

Textarea

```
<td><textarea name="Eingabe" rows="10" cols="60" wrap="virtual">
<?php
    echo isset($_POST['Eingabe']) ? trim($_POST[Eingabe]) : ""
?>
</textarea></td>
```

Gefahr: Kopie der letzten
Eingabe wieder ausgeben

► Geschützt:

```
trim(htmlspecialchars($_POST[Eingabe]))
```

Wirkung von htmlspecialchars

▶ Eingabe:

`</textarea></td></tr></table>`Hier könnte Schadsoftware stehen

▶ Ausgabe ohne htmlspecialchars:

... `<td><textarea name="Eingabe" rows="10" cols="60" wrap="virtual">
</textarea></td></tr></table>`Hier könnte Schadsoftware stehen
`</textarea></td>` ...

▶ Ausgabe mit htmlspecialchars:

... `<td><textarea name="Eingabe" rows="10" cols="60" wrap="virtual">
</textarea></td></tr></table>`Hier könnte
Schadsoftware stehen `</textarea></td>` ...

▶ Ergebnis:

- Im zweiten Fall gibt es keine Tags! Angezeigt wird aber ‚<‘ und ‚>‘ !

Beispiel 2: Informationen beschaffen

▶ Idee:

- ▶ Beispiel *mitarbeiter.php* wird erweitert:
- ▶ Die Daten der Vorgesetzten werden nicht angezeigt!

▶ Code:

```
$sql = "Select Persnr, Name, Ort, Gebdatum, Gehalt, Vorgesetzt  
      From Personal  
      Where Vorgesetzt Is Not Null  
      And Upper(Name) Like Upper('%$suche%' );
```



▶ Folgerung:

- ▶ Die Daten der Vorgesetzten sind geschützt!
- ▶ **Schön wär's !!!!!**

Beispiel 2: Es sieht korrekt aus!

Bitte geben Sie die Suchzeichen ein:

Ungeschützt:

Geschützt:

Weiter

Die Verbindung zur Datenbank OracleUTF8 wurde hergestellt.

Die Datenbank BIKE wird jetzt nach allen Mitarbeitern durchsucht, die im Namen den Teilstring >er< enthalten.

Ungeschützt: `Select Persnr, Name, Ort, Gebdatum, Gehalt, Vorgesetzt From Personal Where Vorgesetzt IS NOT NULL And Upper(Name) Like Upper('%er%')`

Ergebnis:

Persnr	Name	Ort	GebDatum	Gehalt	Vorgesetzter?
5	Johanna Köster	Nürnberg	07.02.84	2100	Nein
7	Thomas Noster	Regensburg	17.09.72	2500	Nein
8	Renate Wolters	Augsburg	14.07.79	3300	Nein
9	Ernst Pach	Stuttgart	29.03.92	800	Nein

Ganz normal: Die Vorgesetzten-Daten fehlen

Beispiel 2: Wir tricksen SQL aus!

Die Datenbank BIKE wird jetzt nach allen Mitarbeitern durchsucht, die im Namen den Teilstring >) Or ('%'=< enthalten.

Ungeschützt: `Select Persnr, Name, Ort, Gebdatum, Gehalt, Vorgesetzt From Personal Where Vorgesetzt IS NOT NULL And Upper(Name) Like Upper('%') Or ('%'='%)`

Ergebnis:

Persnr	Name	Ort	GebDatum	Gehalt	Vorgesetzter?
1	Maria Forster	Regensburg	05.07.79	1800	Ja
2	Anna Kraus	Regensburg	09.07.75	2300	
4	Heinz Rolle	Nürnberg	12.10.57	3300	Nein
5	Johanna Köster	Nürnberg	07.02.84	2100	Nein
6	Marianne Lambert	Landshut	22.05.74	4100	Ja
3	Ursula Rank	Frankfurt	04.09.67	2700	Nein
7	Thomas Noster	Regensburg	17.09.72	2500	Nein
8	Renate Wolters	Augsburg	14.07.79	3300	Nein
9	Ernst Pach	Stuttgart	29.03.92	800	Nein

Eingabe: `) Or ('%' = '`

ergibt:
... `Where Vorgesetzt Is Not Null
And Upper(Name) Like Upper('%')
Or ('%' = '%')`

Beispiel 2: Wir schlagen zurück

- ▶ Wir verwenden statt der Methode query:
 - ▶ prepare → bindParam → execute
 - ▶ Und übergeben die Eingaben an eine Variable
 - ▶ Damit ist die SQL-Struktur vorgegeben und kann nicht per Eingabe manipuliert werden
 - ▶ Eine SQL Injection ist nicht mehr möglich
- ▶ Folgende Befehle sind völlig gleichwertig:

```
$stmt = $conn->query( $sql );
```

Parsen und Ausführen



gleichwertig

```
$stmt = $conn->prepare( $sql );  
$stmt->execute( );
```

Parsen des Befehls

Ausführen

Beispiel 2: Der neue Code

```
$sql = "Select Persnr, Name, Ort, Gebdatum, Gehalt, Vorgesetzt  
      From Personal  
      Where Vorgesetzt Is Not Null  
      And Upper(Name) Like Upper(:name) ";  
$stmt = $conn->prepare($sql);  
$stmt->bindParam(':name', $str, PDO::PARAM_STR);  
$str = "%$suche%";  
$stmt->execute();
```

Platzhalter

Vorbereiten des Befehls

Platzhalter mit
Variable \$str vom Typ
PDO::PARAM_STR
verknüpfen

Übertragen und ausführen

Suchzeichen der Variable \$str
zuweisen (inkl. Prozentzeichen)

▶ PHP-Problem am Rande:

- ▶ Bei bindParam muss eine Variable übergeben werden!
- ▶ Die Prozentzeichen müssen daher explizit hinzugefügt werden.

Beispiel 3: Zugriff ohne Passwort

▶ Idee:

- ▶ Zum Zugriff muss der Mitarbeiter mit Passwort eingegeben werden.
- ▶ Das Passwort ist einfachheitshalber vorgegeben. Es lautet: *Freitag 13*

▶ Code:

```
$sql1 = "Select *  
      From Personal  
      Where Trim(Name) = '$kenn' And '$passwort' = '$pass' ";  
$stmt = $conn->query($sql1);
```

Hier ist intern
„Freitag 13“ hinterlegt

▶ Unsicher: Das Einloggen gelingt mit:

- ▶ Maria Forster' --
- ▶ ' Or 1=1 –

\$kenn, \$pass sind die
Eingabevariablen

▶ Das Passwort wird nicht benötigt!

Beispiel 3: Der neue Code mit bindParam

```
$sql2 = "Select *  
    From Personal  
    Where Trim(Name) = :kenn And '$password' = :pass ";  
$stmt = $conn->prepare($sql2);  
$stmt->bindParam(':kenn', $kenn, PDO::PARAM_STR);  
$stmt->bindParam(':pass', $pass, PDO::PARAM_STR);  
$stmt->execute();
```

Platzhalter

Vorbereiten des Befehls

Übertragen und ausführen

Platzhalter mit Variablen \$kenn und \$pass vom Typ PDO::PARAM_STR verknüpfen

- ▶ **Jetzt müssen Name und Passwort eingegeben werden!**

Klasse PDOException

Eigenschaft/Methode	Beschreibung
getMessage()	gibt eine ausführliche Fehlermeldung zurück
getCode()	gibt die Fehlernummer zurück (SQLSTATE-Code)
getLine()	gibt Fehlerzeile zurück
\$errorInfo	Feld mit 3 Indizes: 0: Fehlercode (SQLSTATE-Code) 1: Fehlercode des Herstellers 2: Ausführliche Fehlermeldung

bekannt

Normierter Fehlercode

entspricht getCode

entspricht getMessage

In Oracle Zugriff auf interne Codes erforderlich, da Fehlercode meist "HY000"

SQLSTATE

Code	Ursache
'00'	Erfolgreiche Beendigung
'01'	Warnung
'02'	Daten nicht gefunden
'08'	Verbindungsaufbau-Fehler
'0A'	Merkmal wird nicht unterstützt
'22'	Datenfehler (z.B. Division durch Null)
'23'	(Tabellen/Spalten-)Bedingung ist verletzt
'24'	Ungültiger Cursor-Status
'25'	Ungültiger Transaktions-Status
'2A'	SQL-Syntax- oder Zugriffsfehler
'2B'	Abhängiges Privileg existiert
'2D'	Nichterlaubte Transaktionsbeendigung
'34'	Ungültiger Cursorname
'3D'	Ungültiger Katalogname
'3F'	Ungültiger Schemaname
'40'	Rollback
'42'	Syntax- oder Zugriffsfehler
'44'	Check-Bedingung ist verletzt

▶ SQLSTATE:

▶ 5 stelliger String

▶ Erste 2 Zeichen

▶ Hauptcode

▶ Weitere 3 Zeichen

▶ Subcode

▶ z.B. Erfolg:

▶ "00000"

▶ z.B. Deadlock:

▶ "40001"

Beispiel: Transaktionsbetrieb

- ▶ **Provozieren von Deadlocks (SQLSTATE: 40001)**
- ▶ **Deadlock:**
 - ▶ Eine Verklemmung, bei der zwei oder mehr Transaktionen gegenseitig auf die Freigabe eines oder mehrerer Locks warten
- ▶ **Deadlockbehandlung im Catch-Teil**
- ▶ **Realisierung:**
 - ▶ 2 Internetanwendungen ändern Daten
 - ▶ Dabei holen die Transaktionen Locks
 - ▶ Bei ungeschickter Anforderung der Daten: Deadlock

Beispielanwendung

► Attribut Sperre in Relation Kunde manipulieren

Kundnr	Name	Sperre	Kunde1	Kunde2
5	Biker Ecke	0	<input checked="" type="radio"/>	<input type="radio"/>
1	Fahrrad Shop	0	<input type="radio"/>	<input checked="" type="radio"/>
6	Fahrräder Hammerl	0	<input type="radio"/>	<input type="radio"/>
3	Maier Ingrid	1	<input type="radio"/>	<input type="radio"/>
4	Rafa - Seger KG	0	<input type="radio"/>	<input type="radio"/>
2	Zweirad-Center Staller	0	<input type="radio"/>	<input type="radio"/>

Zuerst Kunde 5 sperren

Dann Kunde 1 sperren

In zweiter
Anwendung
genau umgekehrt!

Kann zu Deadlock führen!

Angaben übernehmen

Realisierung

```
$sql = "Update Kunde
```

```
    Set Sperre = Sperre + 1
```

```
    Where Nr = $_POST[Kunde1]";
```

Daten des ersten Kunden ändern

```
$stmt1 = $conn->query($sql);
```

```
ob_flush(); flush();
```

Änderungen sofort auf Browser anzeigen

```
sleep(10);
```

10 Sekunden warten, um parallele Anwendung zu starten

```
$sql = "Update Kunde
```

```
    Set Sperre = Sperre + 1
```

```
    Where Nr = $_POST[Kunde2]";
```

Daten des zweiten Kunden ändern

```
$stmt2 = $conn->query($sql);
```

Auftreten eines Deadlocks

▶ Anwendung 1:

- ▶ Kundel: Kundnr = 5
- ▶ Kunde2: Kundnr = 1

Kundnr	Name	Sperre	Kundel	Kunde2
5	Biker Ecke	0	<input checked="" type="radio"/>	<input type="radio"/>
1	Fahrrad Shop	0	<input type="radio"/>	<input checked="" type="radio"/>

- ▶ Sperre für Kundnr=5, Warten von 10 sec., Sperre für Nr 1 besetzt

▶ Anwendung 2:

- ▶ Kundel: Kundnr = 1
- ▶ Kunde2: Kundnr = 5

Kundnr	Name	Sperre	Kundel	Kunde2
5	Biker Ecke	0	<input type="radio"/>	<input checked="" type="radio"/>
1	Fahrrad Shop	0	<input checked="" type="radio"/>	<input type="radio"/>

- ▶ Sperre für Kundnr=1, Warten von 10 sec., Sperre für Nr 5 besetzt

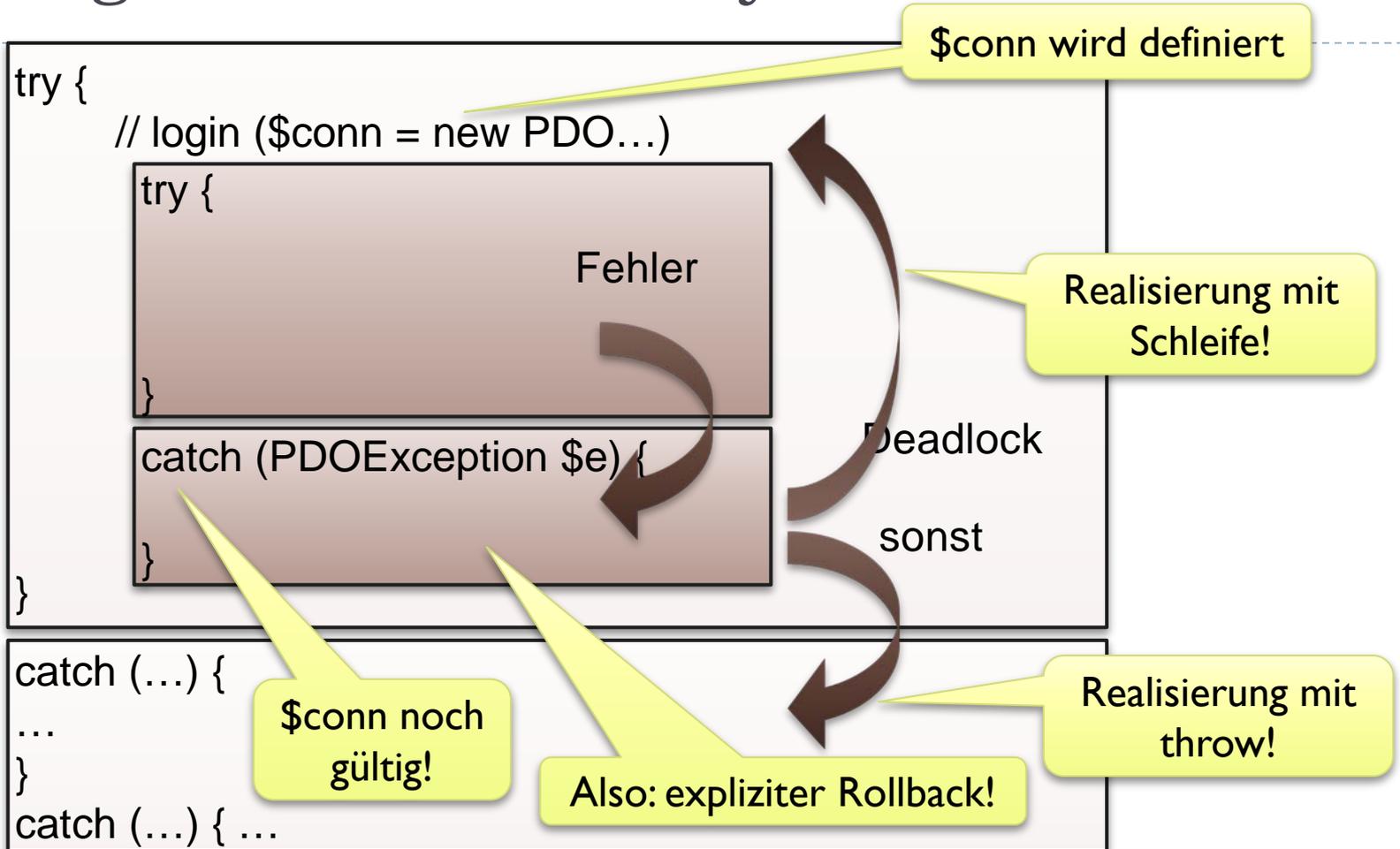
▶ Beide warten auf Freigabe einer Sperre → DEADLOCK

▶ Voraussetzung: Anwendungen starten innerhalb von 10 sec.

Behandlung des Deadlocks

- ▶ **Falls Deadlock eintritt:**
 - ▶ Eine der beiden Transaktionen erhält Fehlermeldung „40001“
 - ▶ Diese Transaktion führt einen Rollback durch und startet neu
 - ▶ Damit werden Sperren freigegeben
 - ▶ Dadurch kann andere Transaktion weiter arbeiten
- ▶ **In unserer PDO Anwendung:**
 - ▶ Fehler führen zum Sprung in Catch-Block
 - ▶ Damit wird automatisch ein Rollback durchgeführt
 - ▶ Aber: Wie können wir die Transaktion wiederholen?
 - ▶ Aber: Unsere Connection-Variable gibt es nicht mehr!

Lösung: Geschachtelte Try-Catch-Blöcke



Realisierung im Überblick

```
try {  
    $conn = new PDO( ... );    // Einloggen  
    $versuch = 0;            // Noch keine Ausführung der Transaktion
```

```
while ($versuch < 2) {  
    $versuch++;            // Erster Versuch!  
    try {  
        $conn->beginTransaction();    // Transaktionsmodus  
        // Zugriff 1  
        ob_flush(); flush(); sleep(10);    // 10 Sekunden warten  
        // Zugriff 2  
        $conn->commit();  
        $versuch = 2;            // fertig, kein Wiederholen!  
    } //end try  
    catch { ... }    // → siehe naechste Folie  
} // end while
```

Erster Update

Warten

Zweiter Update

```
} // end try
```

Lösung in PHP

```
catch (PDOException $e) {
```

```
    $code = $e->getCode();
```

```
    if ($code == "HY000")
```

```
        $code = $e->errorInfo[1];
```

```
    if ( $versuch < 2 and ($code == "40001" or $code == 60) )
```

```
    {
```

```
        echo "Synchronisationsprobleme! Es wird nochmals gestartet." ;
```

```
        $conn->rollback( );
```

```
    } else
```

```
        throw $e;
```

```
    }
```

Code abfragen

Falls kein
korrekter Code ...

... herstellerspezifischen Code auslesen

Erstversuch

Deadlock-Code

Oracle Deadlock-Code

Enorm wichtig!

Ausnahme weiterreichen
zu äußerem Catch-Block

Weitere Anmerkungen

- ▶ **Innerer Try-Catch-Block ist umhüllt von Schleife**
 - ▶ Bei fehlerfreiem Ablauf wird diese nur einmal durchlaufen
 - ▶ Bei Deadlock-Fehler kann Zähler (z.B. \$versuch) mitgeführt werden, um Endlosschleifen generell zu verhindern
- ▶ **Problem:**
 - ▶ Auch Browser synchronisieren Sessions!
 - ▶ Überlagerung von Lockproblemen
 - ▶ Empfehlung:
 - ▶ `session_write_close();` // gleich nach `session_start();`
 - ▶ → keine Protokollierung der Session → keine Serialisierung der Browser

Arbeiten mit Large Objects (LOB)

▶ Large Objects:

- ▶ Binary Large Objects (BLOB): Binär gespeicherte Objekte
- ▶ Character Large Objects (CLOB): Sehr lange Zeichenketten

	Max. Größe eines BLOB	Max. Größe eines CLOB
Oracle	8 – 128 TByte	8 – 128 TByte
SQL Server	2 GByte (Image)	1 – 2 GByte
MySQL	4 GByte (LONGBLOB)	4 GByte (LONGTEXT)

Zusätzlich:
FILESTREAM

Large Objects

- ▶ **Large Objects (LOBs) können direkt in die Datenbank aufgenommen werden:**
 - ▶ Gleiche Verwaltung wie die Daten
 - ▶ Gleiche Zugriffsrechte wie die Daten
 - ▶ Keine Zugriffe von außerhalb möglich
- ▶ **LOBs können gespeichert und wieder ausgelesen werden**
- ▶ **CLOBs zusätzlich:**
 - ▶ Direkte Bearbeitung und Durchsuchen in Datenbank möglich
- ▶ **Speichern von Büchern, Bildern, Musik, Videos, ...**

CLOB

BLOB

BLOB

BLOB

Beispiel: Ablegen von Bildern

▶ Idee:

- ▶ In Personaltabelle werden die Fotos der Mitarbeiter mit aufgenommen
- ▶ Technische Gründe: Bildtyp wird benötigt (JPEG usw.)

▶ Implementierung:

ALTER TABLE Personal

ADD Bild BLOB Default EMPTY_BLOB() ;

ALTER TABLE Personal

ADD Bildtyp CHAR(40) ;

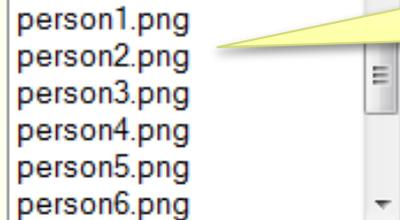
Oracle

SQL Server: Add Bild Varbinary(max);

MySQL: Add Bild LongBlob;

Bilder und PHP: blob_ein1.php

Bildauswahl auf Server:



person1.png
person2.png
person3.png
person4.png
person5.png
person6.png

Alle Dateien aus
Unterverzeichnis /image
in Select-Box anzeigen ...

Datei vom PC hochladen:



... oder Bild auf
lokalem Rechner
suchen und hochladen

Neue Funktionen:

- scandir
- is_dir

Neues Formularelement:

```
<input type="file" name=...
```

Implementierung: blob_ein1.php (1)

```
<form action="blob_ein2.php" method="post"
Ruft blob_ein2.php auf          enctype="multipart/form-data">
```

```
<p>Bildauswahl auf Server:
```

```
<input type="radio" name="Auswahl" value="Server" checked/>
```

```
<select name="Bild" size="6">
```

```
<?php
```

```
    $dir = "./image";
```

Existiert Verzeichnis ./image?

```
    if (is_dir($dir))
```

```
        if ($filearray = scandir($dir))
```

Liest alle Dateien des Verzeichnisses in Feld \$filearray

```
            foreach ($filearray as $name)
```

```
                if (!is_dir($name))
```

```
                    echo "<option> $name </option>";
```

\$filearray komplett auslesen. Dateien (keine Verzeichnisse) in Select-Box anzeigen

```
?> </select>
```

Verzeichnis nicht leer?

Implementierung: blob_ein1.php (2)

Datei vom PC hochladen:



Durchsuchen_

Ausgewähltes Bild anzeigen

Hier zusätzlich
Tabellenformatierung!

<p>Datei vom PC hochladen:

<input type="radio" name="Auswahl" value="Lokal"/>

Angabe der akzeptierten
Dateitypen, hier: Bilder

<input type="file" name="Bilddatei" size="60" accept="image/*"/> </p>

<p> <input type="Submit" value="Ausgewähltes Bild anzeigen"/></p>

</form>

Eigenschaften des Typs \$_FILES

- ▶ Dateien werden mit \$_FILES übertragen, nicht \$_POST
- ▶ \$_FILES besitzt vier assoziative Werte:
 - ▶ \$_FILES['Bilddatei']['name']
 - Name des Input-Files
 - Name der Datei, die auf dem PC ausgewählt wurde
 - ▶ \$_FILES['Bilddatei']['type']
 - Dazugehöriger Dateityp
 - ▶ \$_FILES['Bilddatei']['tmp_name']
 - Name der Datei, in den die ausgewählte Datei temporär hochgeladen wurde (inkl. Pfad)
 - ▶ \$_FILES['Bilddatei']['size']
 - Größe der Datei

Zwischenschritte (blob_ein2.php) (1)

- ▶ Kopieren der hochgeladenen Datei ins Verzeichnis upload:

```
copy( $_FILES['Bilddatei']['tmp_name'] ,  
      './upload/'.$_FILES['Bilddatei']['name'] );
```

- ▶ Merken der Bilddaten in Session-Variable:

```
$_SESSION['Bild'] = "./upload/" . $_FILES['Bilddatei']['name'];  
$_SESSION['Bildtyp'] = $_FILES['Bilddatei']['type'];
```

bzw.

```
$_SESSION['Bild'] = "./image/" . $_POST['Bild'];  
$_SESSION['Bildtyp'] = "image/png";
```

Falls Bild vom PC
hochgeladen

Falls PNG-Bild auf
Server ausgewählt

Zwischenschritte (blob_ein2.php) (2)

▶ Formular:

- ▶ Anzeige des ausgewählten Bildes
- ▶ Aufforderung zur Angabe der Personalnummer
 - ▶ um das Bild einem Mitarbeiter zuzuordnen
 - ▶ Besser: Alle Mitarbeiter auflisten (→ Übung)

Dies ist die Datei *blob_ein2.php* im Sessienteil, die die Datei *blob_ein3.php* aufruft.

Folgendes Bild wurde ausgewählt:

Ausgewähltes Bild:



Bitte geben Sie die Personalnummer an:

Bild in DB einfügen

Einfügen des Bildes in Datenbank

▶ Problem:

- ▶ Die Datenbank muss wissen, dass ein LOB vorliegt!
- ▶ Bei der Übergabe eines Bildes müssen wir also mitteilen, dass dies ein Element vom Typ `PDO::PARAM_LOB` ist!
- ▶ Wir verwenden wieder die Methoden `prepare` und `execute`!
- ▶ Wir übergeben mit der Methode `bindParam` diese LOB-Variable

Parsen und Vorbereiten des Update

```
$sql = "Update Personal
      Set   Bildtyp = ?,
      Bild = EMPTY_BLOB( )
      Where Persnr = $_POST[Persnr]
      Returning Bild Into ?";
```

Oracle

Platzhalter 1

Bild einfügen: Zunächst leeren Blob vorhalten (nur in Oracle)

Platzhalter 2

Vorbereiten des Befehls

```
$stmt = $conn->prepare($sql);
```

Bild binär zum Lesen öffnen

```
$fp = fopen( $_SESSION['Bild'], 'rb' );
```

```
$stmt->bindParam( 1, $_SESSION['Bildtyp'] );
```

1. Fragezeichen mit Variable verknüpfen

```
$stmt->bindParam( 2, $fp, PDO::PARAM_LOB );
```

```
$stmt->execute( );
```

2. Fragezeichen mit Variable \$fp als LOB verknüpfen

```
fclose( $fp );
```

Übertragen und ausführen

Ist Bild wirklich in Datenbank?

▶ Zum Überprüfen:

▶ Wir testen die Länge des LOB

- ▶ in Oracle: `Dbms_Lob.Getlength(Binärdatei)`
- ▶ in SQL Server: `DataLength(Binärdatei)`
- ▶ in MySQL: `Length(Binärdatei)`

▶ z.B.:

```
$sql = "Select Persnr, Name, Ort, Gebdatum, Gehalt,  
        Vorgesetzt, Dbms_Lob.Getlength(Bild) As Bildlaenge  
From Personal  
Where Upper(Name) Like Upper( '%$Name%' ) ";
```

Gibt Bildgröße
in Byte zurück

Bild auslesen (mitarbeiterblob.php)

```
do {  
?><tr>  
    <td> <?php echo $row["PERSNR"];?> </td>  
    // analog: NAME, ORT, GEBURTSDATUM, GEHALT  
    <td> <?php echo ($row["VORGESETZT"] == null)? "Ja" : "Nein";?> </td>  
<?php  
    if ($row["BILDLAENGE"] > 0)  
        echo "<img src=\"bild.php?id=$row[PERSNR]\" height=250/> ";  
    else  
        echo "<td> <p>-- kein Bild --</p> </td>";  
    echo "</tr>";  
} while ( $row = $stmt->fetch() );  
?>
```

Alle Daten ausgeben
(wie in mitarbeiter.php)

Liegt Bild vor? (falls
Bildlänge größer 0!)

... dann Bild anzeigen!
(siehe nächste Folie)

Schleife über
alle Mitarbeiter

Funktionsweise des Auslesens

- ▶ Auszulesen ist ein Bild → IMG-Tag
- ▶ Bild steht in Datenbank und liegt nicht als Bilddatei vor
- ▶ Also: Datenbank öffnen und mit Select-Befehl auslesen
- ▶ Dies geschieht in einer eigenen PHP-Datei
- ▶ Aufruf dieser PHP-Datei mit GET-Parameter
 - ▶ um das Bild des gewünschten Mitarbeiters zu erhalten

```
echo "<img src=\"bild.php?id=$row[PERSNR]\" height=250/> "
```

z.B. Bild von Mitarbeiter 2:
src="bild.php?id=2"

Get-Parameter

Datei bild.php

Nur Bildausgabe, daher:
Keine Tags, auch kein <html>

```
<?php
session_start();
$conn = new PDO($_SESSION['Parameter1'],
                $_SESSION['Kennung'], $_SESSION['Passwort']);
$sql = " Select Bild, Bildtyp From Personal
        Where Persnr = $_GET[id] ";
$stmt = $conn->query($sql);
$stmt->bindColumn(1, $blob, PDO::PARAM_LOB);
$stmt->bindColumn(2, $bildtyp);
$stmt->fetch(PDO::FETCH_BOUND);
header("Content-Type: $bildtyp");
fpassthru($blob);
?>
```

Get-Parameter!

1. Parameter Bild
→ \$blob als LOB !

2. Parameter Bildtyp → \$bildtyp

Keine Ausgabe in Feld, sondern
in Variable mittels bindColumn!

Ausgabe nicht mit echo, da
\$blob nur Zeiger (Oracle)

Datenbankunabhängiges Programmieren

- ▶ PDO ermöglicht weitgehend datenbankunabhängige Zugriffe (Problem: LOBs)
- ▶ Herstellerspezifisches SQL sollte beachtet werden, z.B.:

zu beachten	wegen
PDO::ATTR_CASE auf PDO::CASE_UPPER setzen	Oracle
Kein Semikolon am Befehlsende	Oracle
Im Join nur den Operator ON verwenden	SQL Server
In der From-Klausel auf den Bezeichner AS verzichten	Oracle
Kein Leerzeichen nach einer Aggregatfunktion	MySQL
Kein Full Outer Join	MySQL

Zusammenfassung zu PDO

- ▶ **Klasse PDO:**
 - ▶ zur Verwaltung der Datenbankverbindung
- ▶ **Klasse PDOStatement:**
 - ▶ zur Bearbeitung eines SQL-Befehls
- ▶ **Klasse PDOException:**
 - ▶ zur Fehlerbehandlung

Methoden der PDO-Klassen

Methode	aus Klasse	Kurzbeschreibung
Konstruktor PDO()	PDO	baut Verbindung zur Datenbank auf
setAttribute()	PDO	setzt Attribute einer Verbindung
beginTransaction()	PDO	startet eine Transaktion
commit()	PDO	beendet eine Transaktion
rollback()	PDO	setzt eine Transaktion zurück
query()	PDO	führt einen SQL-Befehl aus
prepare()	PDO	bereitet eine Abfrage vor
execute()	PDOStatement	führt eine vorbereitete Abfrage aus
fetch()	PDOStatement	liest die nächste Zeile
rowCount()	PDOStatement	gibt Anzahl manipulierter Zeilen aus
bindParam()	PDOStatement	bindet Parameter an eine Abfrage
bindColumn()	PDOStatement	verbindet Spalten mit Variablen

PHP-Funktionen (1)

trim(str)	entfernt Leerzeichen am Anfang und Ende von <i>str</i>
strlen(str)	gibt die Länge der Zeichenkette <i>str</i> zurück
strpos(str1, str2)	gibt das erste Vorkommen von <i>str2</i> in der Zeichenkette <i>str1</i> zurück; bzw. false, falls nicht enthalten
strcmp(str1, str2)	vergleicht die Zeichenketten <i>str1</i> und <i>str2</i>
strcasecmp(str1, str2)	vergleicht <i>str1</i> und <i>str2</i> unabhängig von Groß- und Kleinschreibung
substring(str, pos, len)	gibt Teilstring von <i>str</i> der Länge <i>len</i> ab Position <i>pos</i> zurück
htmlspecialchars(str)	wandelt HTML-Zeichen (z.B. <, >) in Ersatzzeichen um
in_array(such, feld)	liefert true, falls <i>such</i> im Feld vorkommt
stripslashes(str)	entfernt Entwertungszeichen, \ in der Zeichenkette <i>str</i>
implode(str, feld)	liefert Zeichenkette mit allen Feldelementen zurück, die mittels der Zeichenkette <i>str</i> verknüpft werden
echo param1, ...	gibt die Parameterliste auf HTML aus
isset(var)	liefert true, wenn Variable <i>var</i> existiert, sonst false
unset(var)	setzt die Variable <i>var</i> zurück, <i>var</i> existiert nicht mehr

PHP-Funktionen (2)

<code>copy(file1, file2)</code>	kopiert Datei file1 in die Datei file2
<code>fopen(file, str)</code>	öffnet Datei file mit der im String str angegebenen Methode und liefert einen Dateizeiger zurück
<code>fclose(fp)</code>	schließt Datei, auf die der Dateizeiger fp verweist
<code>header(str)</code>	gibt den im String str angegebenen Header aus
<code>fpasssthru(blob)</code>	gibt das Binärobjekt aus, auf den blob verweist
<code>is_dir(str)</code>	gibt true zurück, falls str ein Verzeichnis ist, sonst false
<code>scandir(str)</code>	liefert alle Dateinamen des Verzeichnisses str in einem Feld zurück
<code>session_start()</code>	erster Befehl einer Session-Seite
<code>session_write_close()</code>	schreibt kein Session-Protokoll und vermeidet dadurch Synchronisierung der Browser
<code>ob_flush()</code>	gibt Inhalt auf dem Webserver sofort aus
<code>flush()</code>	gibt Inhalt im lokalen Browser sofort aus
<code>sleep(sec)</code>	wartet für sec Sekunden