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)

statisch (nur HTML) dynamisch (PHP, …)

Anfrage

Ergebnis

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

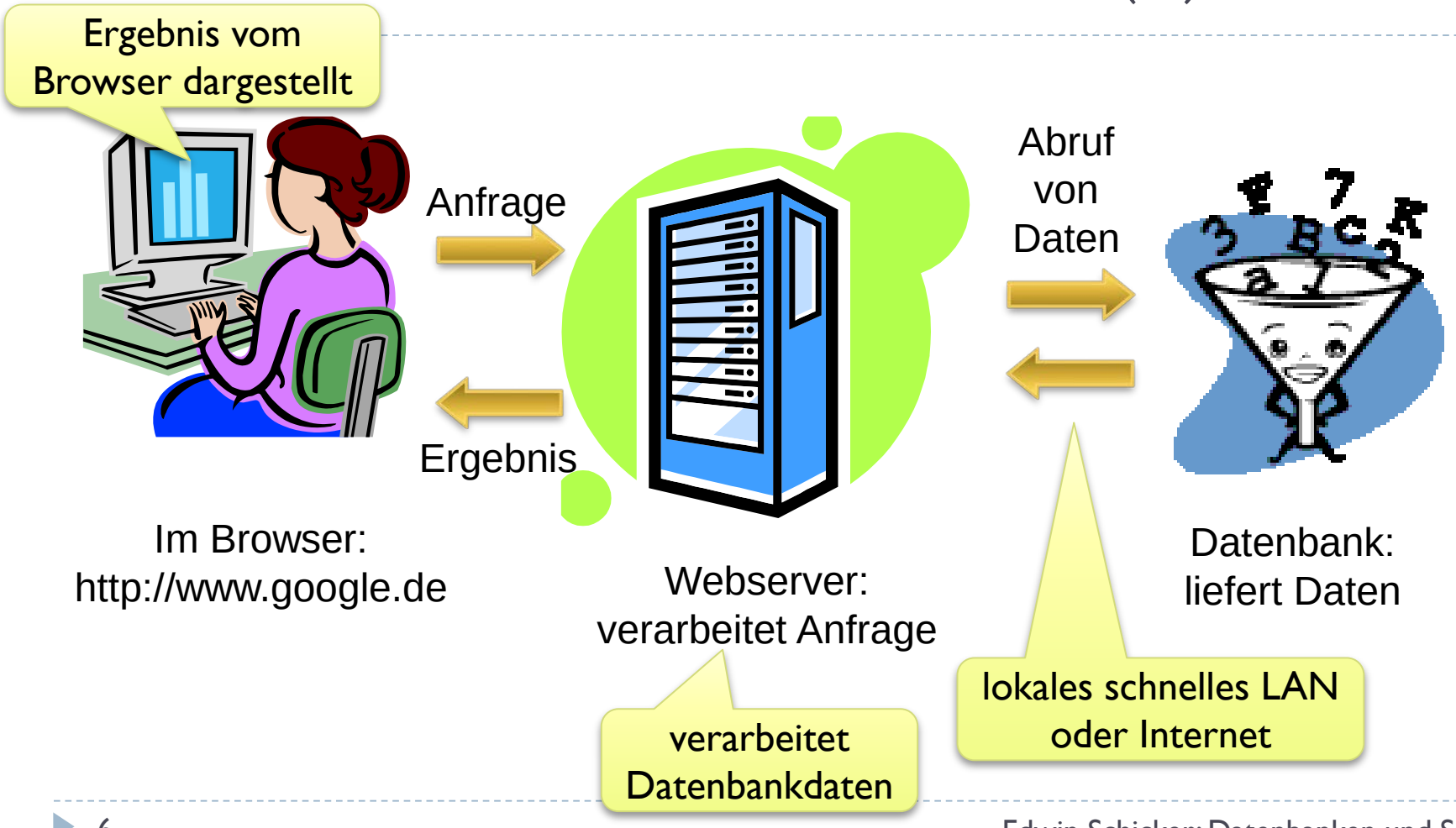
vom Name-Server umgesetzt in z.B.: 194.205.10.255

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)



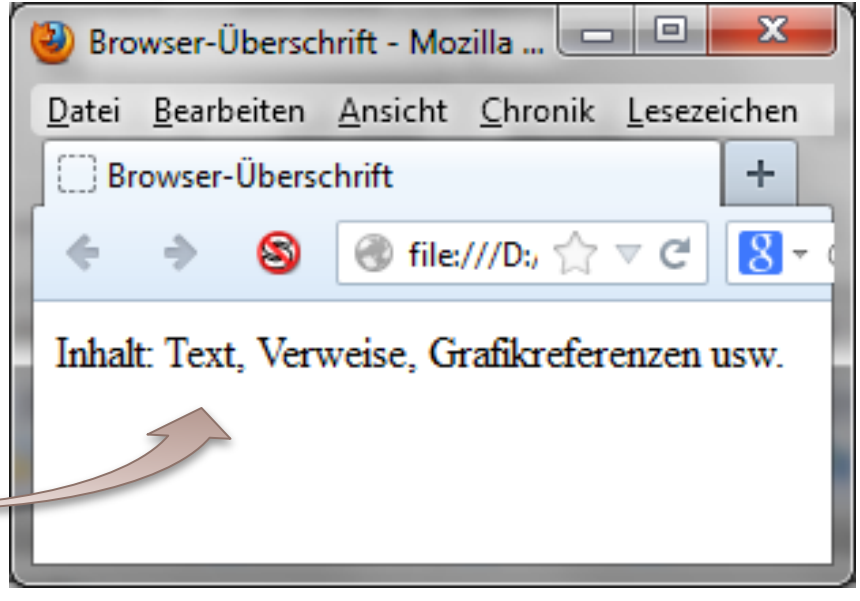
Aufbau einer HTML-Seite

- **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:

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

h: Hierarchy

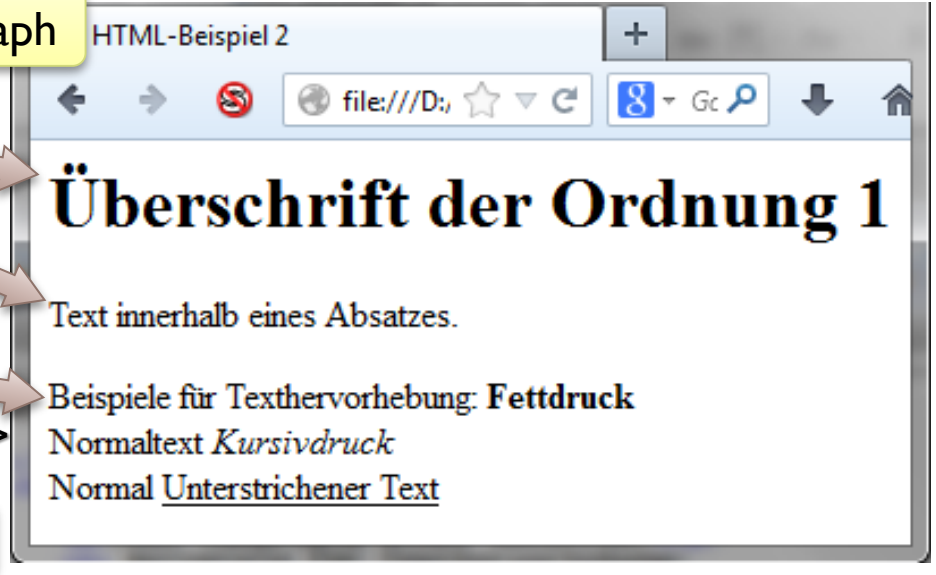
p: Paragraph

b: Bold

u: Underline

i: Italic

br: Break



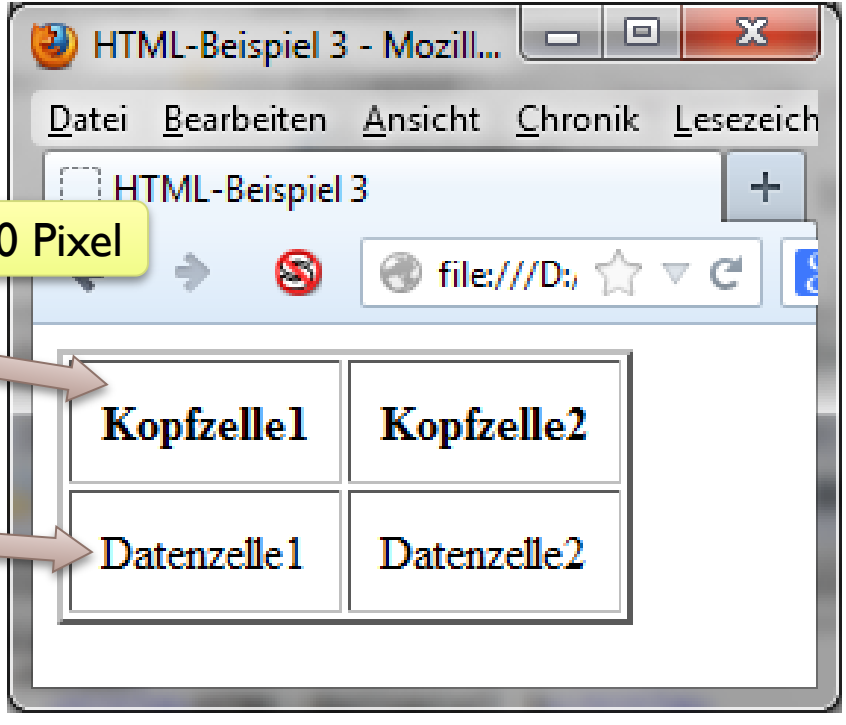
Tabellen in HTML

- Tabellen sind in HTML 4.0 ein Formatierungswerkzeug!
- Beispiel:

Randstärke: 2 Pixel

Textabstand: 10 Pixel

```
<table border="2" cellpadding="10">
 <tr>
  <th>Kopfzelle1</th>
  <th>Kopfzelle2</th>
 </tr>
 <tr>
  <td>Datenzelle1</td>
  <td>Datenzelle2</td>
 </tr>
</table>
```



Weitere Möglichkeiten

- **Links:**
  - Link auf andere Webseite:

    ```
    <a href="http://www.google.de">Google-Startseite</a>
    ```

    Linkadresse

    Hinweis zu Adresse
  - Link auf interne Seite *beispiel.html* im Unterverzeichnis *test*

    ```
    <a href="test/beispiel.html">Beispielseite</a>
    ```
- **Bild anzeigen:**
  - Bild *img.jpg* aus Verzeichnis *image* anzeigen:

    ```
    <img src="image/img.jpg" height="200" width="300"/>
    ```

    Link auf Bild

    Max. Höhe und Breite

    „/“ !
- **Manche Tags besitzen keine Endetags, z.B.
, <img/>**

  „/“ 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" … />
  - Auswahlfeld: <select size="20" … > <option … > …
  - Combobox: <select size="1" … > <option … > …
  - Button: <input type="submit" … />
  - Reset-Button: <input type="reset" … />

falls >1

falls 1

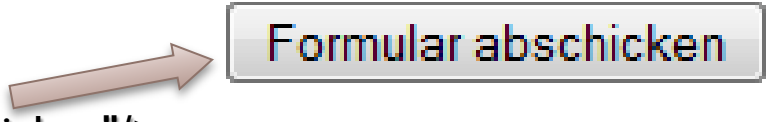
Funktionsweise von Formularen

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

  <input …                              />

  …

  …

  …

  …

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

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

… hier die Datei „start.php“ aufgerufen.

Beim Klick auf den Submit-Button wird …

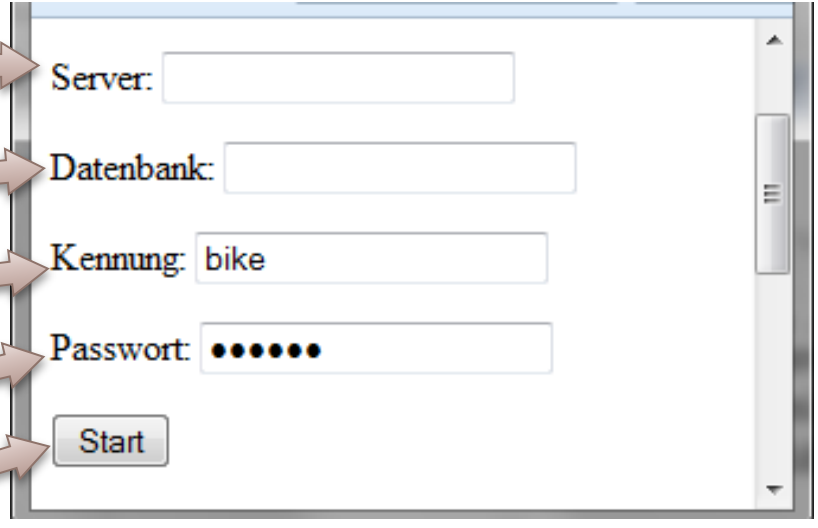


Beispiel zu Formularen (1)

Linkadresse

Methode post

```
<form action="start.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>
```



Die Formatierung muss selbst gesetzt werden!

Beispiele zu Formularen (2)

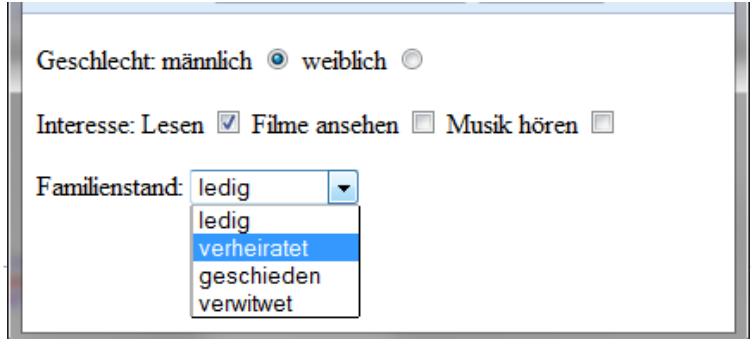
```
<p>Geschlecht:
   männlich <input type="radio" name="Anrede" value="Herr" checked />
   weiblich <input type="radio" name="Anrede" value="Frau"/></p>
<p>Interesse:
   Lesen <input type="checkbox" name="Interesse1" value="Buch" checked />
   Filme ansehen <input type="checkbox" name="Interesse2" value="Video"/>
   Musik hören <input type="checkbox" name="Interesse3" value="Audio"/> </p>
<p>Familienstand:
   <select name="Familienstand" size="1">
     <option> ledig </option>          <option> verheiratet </option>
     <option> geschieden </option>     <option> verwitwet </option>
   </select></p>
```

Vorab gesetzt

Vorab gesetzt

Combobox

Inhalt der Combobox



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

trim(str)	entfernt Leerzeichen am Anfang und Ende von str
strlen(str)	gibt die Länge der Zeichenkette str zurück
strpos(str1,str2)	gibt das erste Vorkommen von str2 in der Zeichenkette str1 zurück; bzw. false, falls nicht enthalten
strcmp(str1,str2)	vergleicht die Zeichenketten str1 und str2
strcasecmp(str1,str2)	vergleicht die Zeichenketten str1 und str2 unabhängig von Groß- und Kleinschreibung
substring(str,pos,len)	gibt einen Teilstring von str der Länge len ab Position pos zurück
implode(str,feld)	liefert Zeichenkette mit allen Feldelementen zurück, die mittels der Zeichenkette str verknüpft werden
stripslashes(str)	entfernt Entwertungszeichen ‚\' in der Zeichenkette str
echo param1,…	gibt die Parameterliste auf HTML aus
isset(var)	liefert true, wenn Variable var existiert, sonst false
unset(var)	setzt die Variable var zurück, var 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="start.php" method="post">
<p>Server: <input type="text" size="20" name="Server"/></p>
```

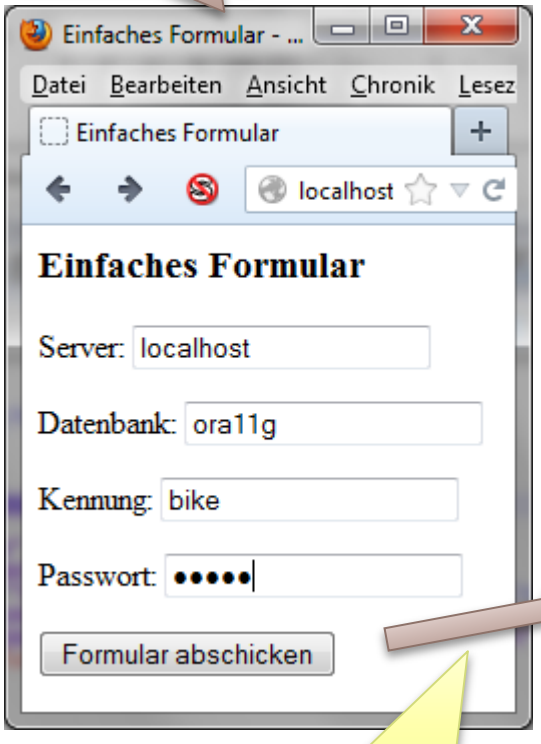
- In der Datei start.php ist der Inhalt verfügbar unter:

```
$_POST[ 'Server' ]
```

  - Analoges gilt für Methode get: $_GET['Server']

Formulare mit PHP auslesen

start.html



per POST übergeben

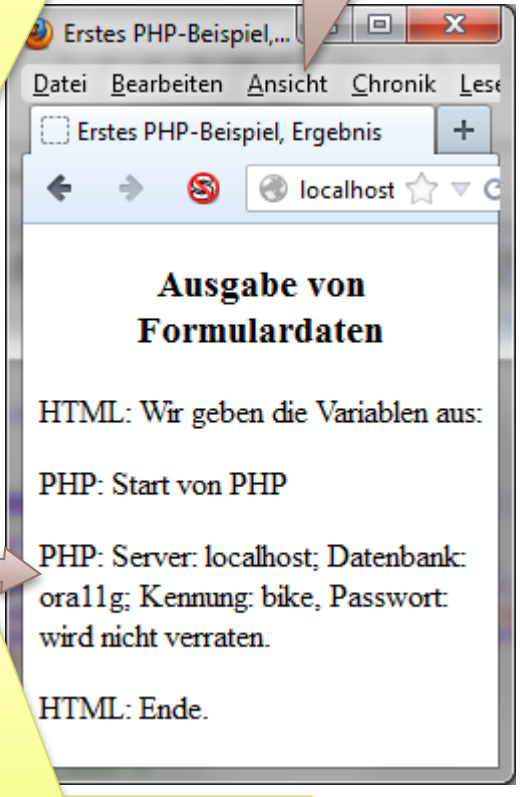
```
<h3>Ausgabe von Formulardaten </h3>
HTML: Wir geben die Variablen aus:
<?php
  echo "<p>PHP: Start von PHP</p>";
  $server = $_POST['Server'];
  $datenbank = $_POST['Datenbank'];
  $benutzer = $_POST['Kennung'];
  $passwort = $_POST['Passwort'];
  echo "<p>PHP: Server: $server;
     Datenbank: $datenbank;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

PHP-Start

PHP-Ende

POST-Variablen

start.php



Variablen ausgeben

Felder in PHP (1)

- **Felder in PHP flexibel: Felder, Listen, Aufzählungen!**
- **Beispiele:**

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

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

Nächstes Feld: $feld[5]=10

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

Assoziative Felder: Indexe sind Strings

Konkatenierung

Anzahl der Feldelemente: 9

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 => $content)
  echo "<p>feld[$key] = $content</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_54_nts.dll
- Microsoft liefert eigene Treiber (z.B. für PHP 5.4)
- Im Unterverzeichnis ext von PHP müssen diese Treiber installiert sein!
- Die SQL Server Treiber werden von PHP nicht mit geliefert. 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",
                $kennung, $passwort);
$conn = new PDO("sqlsrv:server=$server; dbname=$datenbank",
                $kennung, $passwort);
$conn = new PDO("mysql:host=$server;dbname=$datenbank",
                $kennung, $passwort);
```

3. Parameter: Passwort

2. Parameter: Kennung

1. Parameter: Verbindung

Variable SERVER/HOST

Variable DBNAME

Schnittstelle: PHP ←→ 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

- **Einloggen und Transaktionsstart:**

```
$conn = new PDO("oci:dbname=$_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);

$conn->beginTransaction();
```

Kein Hochkomma!

Einloggen am Beispiel Oracle

Bei Fehler: Ausnahme werfen

Spaltennamen in Großbuchstaben ausgeben

Aus Kompaktibilitätsgründen (Oracle)

Objekt -> Methode

1. Parameter: Attribut
2. Parameter: Wert

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);
```

Oracle mag keinen Strichpunkt!!

Abspeichern in Objekt vom Typ PDOStatement

SQL-Befehl ausführen

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 = false;)
- 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 = false;
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.";


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

Code wird erst ab Zweitaufruf durchlaufen

eventuelle Leerzeichen entfernen

Bei leerer Eingabe reagieren

hier: Einloggen, setAttribute, beginTransaction

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

solange auslesen, solange Daten vorhanden sind

Beenden mit Commit

Typisches Auslesen von Daten (1)

- **Variante 1:**
  - 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()) )
    echo "<p>Keine Daten</p>";
else
do {
   // Ausgabe der Attribute ab der ersten Zeile mit $row
} while ( $row = $stmt->fetch() ) ;
```

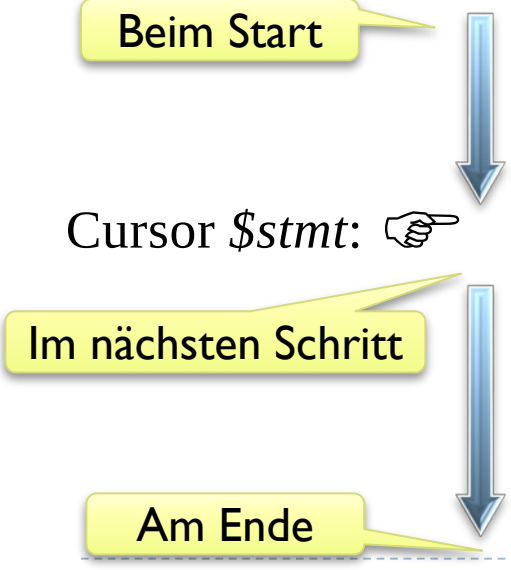
Vorab wird erste Zeile überprüft

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

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



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

if (isset($_SESSION[ 'Kennung' ]))
{
   echo "Variable Kennung existiert, wird nun wieder entfernt.";
   unset($_SESSION[ 'Kennung' ]);
}
echo "<p>Die Sessionvariable Nr hat den Wert:
                                    $_SESSION[ Nr ].</p>";
```

Zwei Sessionvariablen erzeugt

Überprüfen, ob Sessionvariable Kennung existiert

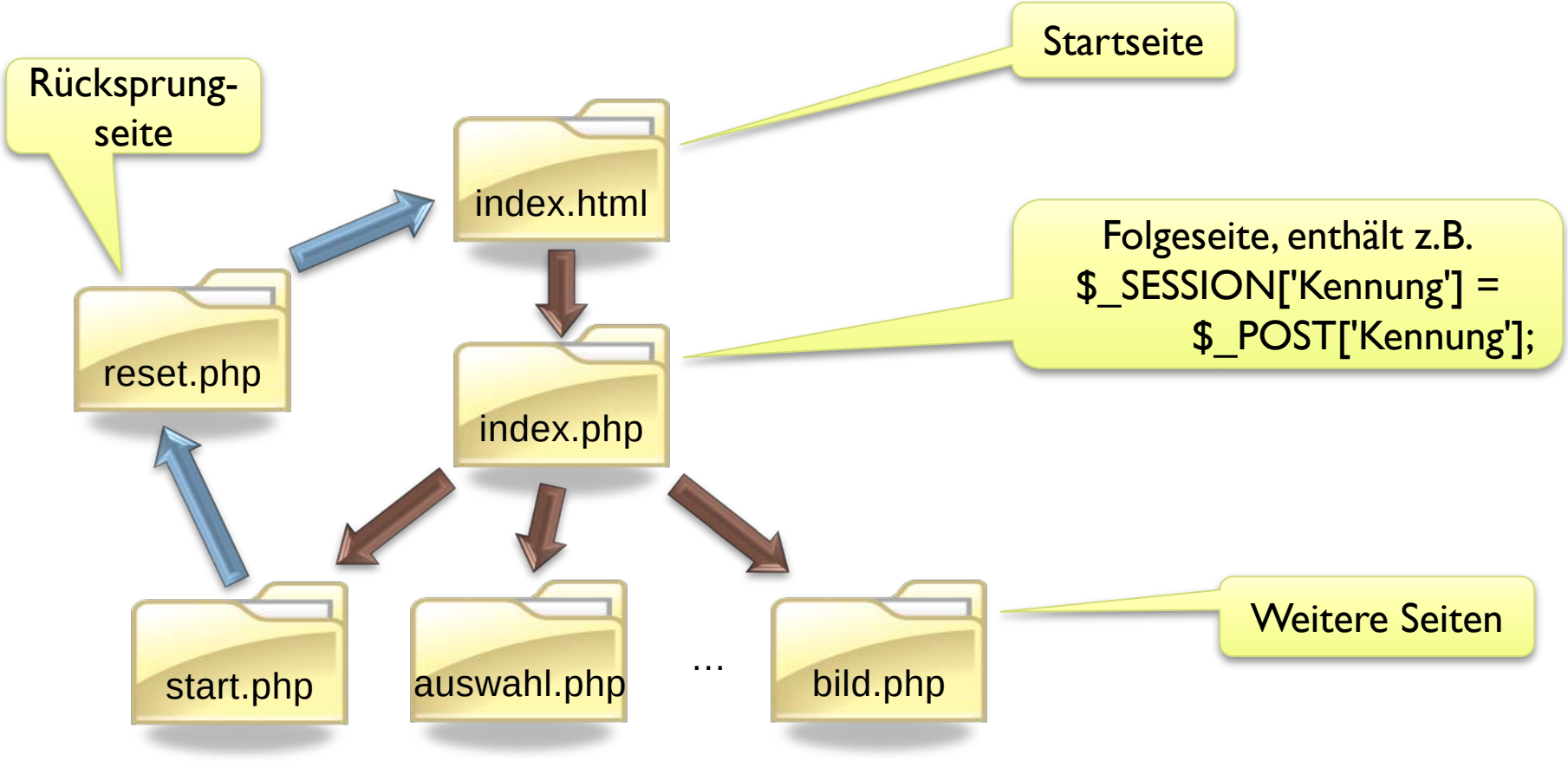
Sessionvariable Kennung existiert nun nicht mehr

Inhalt einer Sessionvariable ausgeben

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:**

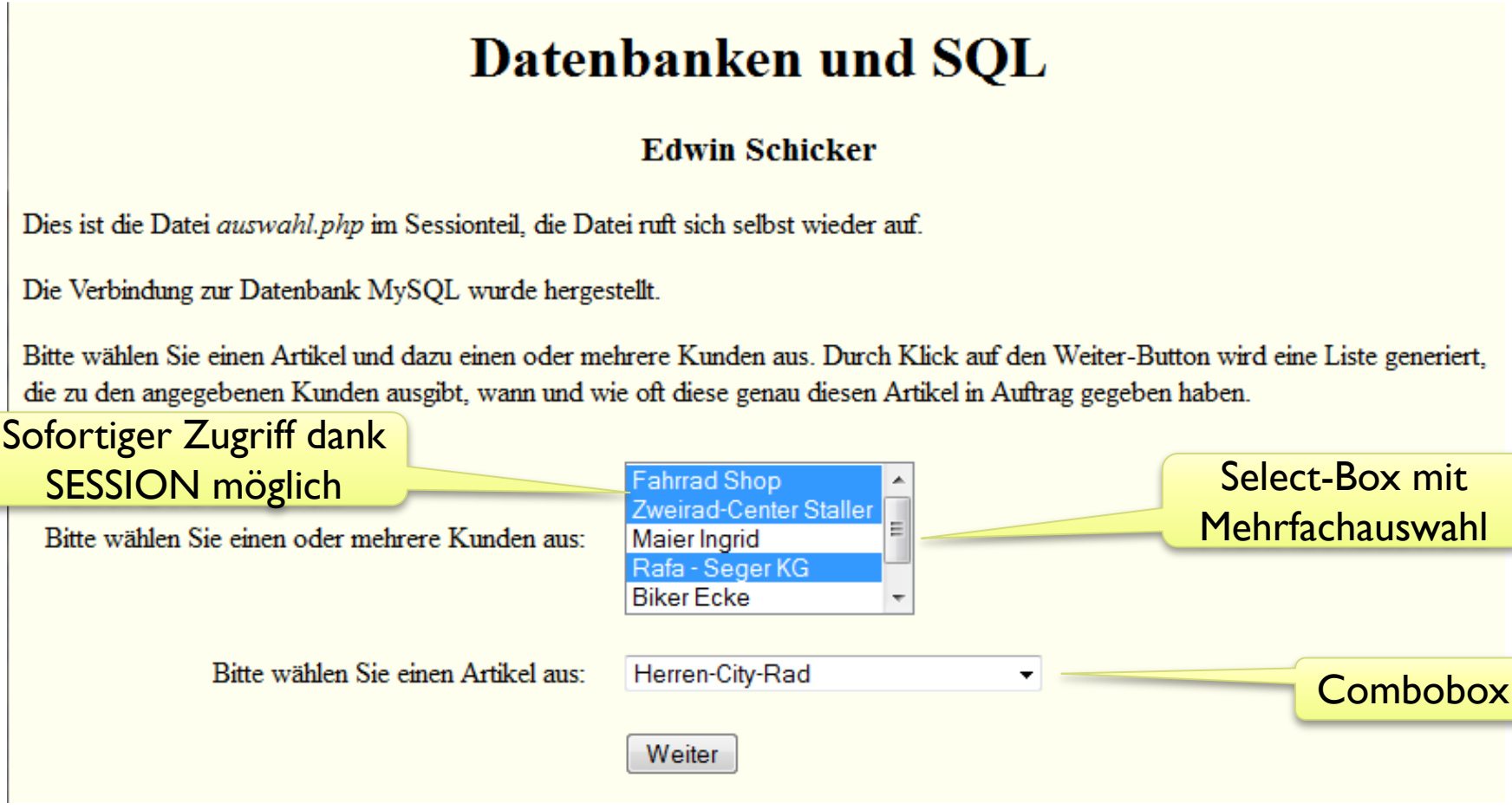
```
if ( !isset($_SESSION['Kennung']) ||
     !isset($_SESSION['Passwort']) ||
     !isset($_SESSION['Server']) ||
     !isset($_SESSION['Datenbank'])  )
   echo "Gehen Sie zur <a href="start.html">Startseite</a>");
else
{       $conn = new PDO( …
```

Überprüfung, ob SESSION-Variablen gesetzt

Wenn nein, dann zurück zur Startseite

Wenn ja, dann zugreifen

Auswahl mittels Select-Boxen



Ausgabe der Artikel in Combo-Box

```
$sql2 = "Select Anr, Bezeichnung From Artikel";
$stmt2 = $conn->query($sql2);

if ( $row2 = $stmt2->fetch() )
{  echo "<select name=\"Artikelnr\" size=\"1\">" ;

   do
   {  echo "<option value=\"$row2[ANR]\"";
      if (isset($_POST['Artikelnr']) and $_POST['Artikelnr'] == $row2['ANR'])
         echo " selected";
      echo "> $row2[BEZEICHNUNG]</option>";
   } while ($row2 = $stmt2->fetch());

   echo "</select>";
} else …
```

Alle Artikel lesen

Existieren Artikel?

Combobox: size=1

… aber Artikelnummern merken

Vorherige Auswahl merken

Artikelnamen anzeigen …

In Schleife alles ausgeben

Ausgabe der Kunden in Select-Box (1)

```
$sql1 = "Select Nr, Name From Kunde";
$stmt1 = $conn->query($sql1);

if ( $row1 = $stmt1->fetch() )
{  echo "<select multiple name=\"Kundnr[]\" size=\"5\">" ;
```

Alle Kunden lesen

Select-Box mit 5 Einträgen

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)

```
if (isset($_POST['Kundnr']))
   $alleKunden = implode( "  ", $_POST['Kundnr'] );
else
   $alleKunden ="";

do {
   echo "<option value=\"$row1[NR]\"" ;
   if (strpos($alleKunden, $row1['NR']) !== false) echo " selected";
   echo "> $row1[NAME]</option>";
} while ($row1 = $stmt1->fetch());

echo "</select>";
```

Trennzeichen zwischen Elementen

Implode: Fasst alle Feldelemente zu einem String zusammen

… aber Kundennummer merken

!==

Auswählen, falls Kundnr im String 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
                                        On A.Auftrnr=AP.Auftrnr
        Where  Kundnr  In (" . implode( "," ,  $_POST['Kundnr']) . ")
          And  Artnr = $_POST[Artikelnr]";

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

… für die ausgewählten Kunden …

… und den Artikel

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($_POST['Eingabe']))
        echo trim(htmlspecialchars(stripslashes($_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

Entfernt Entwertungszeichen

Auswerten beliebiger SQL-Befehle

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

```
$Eingabe = trim(stripslashes($_POST['Eingabe']));
$stmt = $conn->query($Eingabe);
$pos = strpos($Eingabe, " ");
$erstesWort = substr($Eingabe, 0, $pos);
```

Eingabe bereinigen

SQL-Befehl abschicken

Erstes Leerzeichen suchen, eventuell zusätzlich: \n, \r, \t

Erstes Wort geht bis zum ersten Leerzeichen (bzw. \n, \t, \r)

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

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

Vergleich unabhängig von Groß- und Kleinschreibung

```
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

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

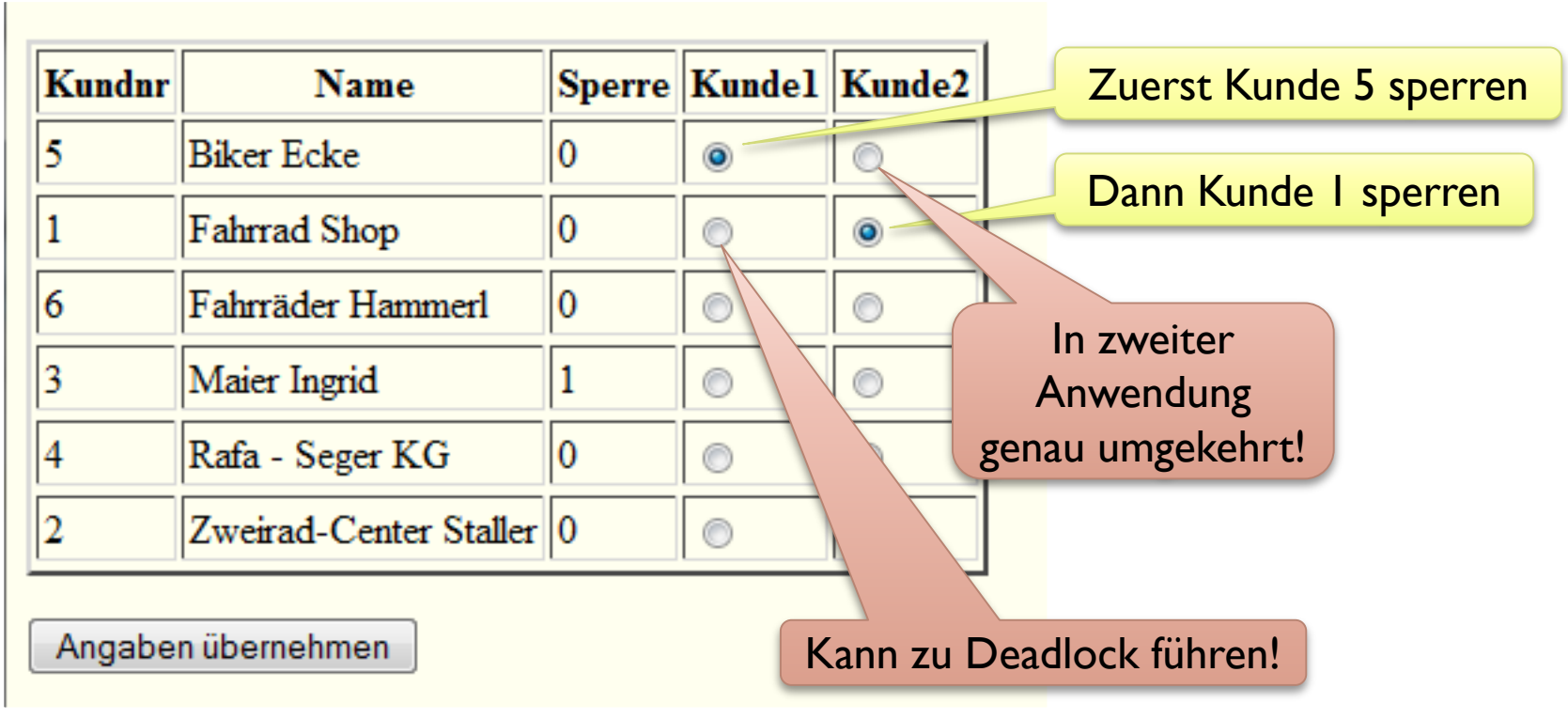
- SQLSTATE:
  - 5 stelliger String
  - Erste 2 Zeichen
    - Hauptcode
  - Weitere 3 Zeichen
    - Subcode
  - z.B. Erfolg:
    - "00000"
  - z.B. Deadlock:
    - "40001"

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



Realisierung

```
$sql = "Update Kunde
            Set Sperre = Sperre + 1
            Where Nr = $_POST[Kunde1]";
$stmt1 = $conn->query($sql);

ob_flush();  flush();
sleep(10);

$sql = "Update Kunde
            Set Sperre = Sperre + 1
            Where Nr = $_POST[Kunde2]";
$stmt2 = $conn->query($sql);
```

Daten des ersten Kunden ändern

Änderungen sofort auf Browser anzeigen

10 Sekunden warten, um parallele Anwendung zu starten

Daten des zweiten Kunden ändern

Auftreten eines Deadlocks

- Anwendung 1:
  - Kunde1: Kundnr = 5
  - Kunde2: Kundnr = 1
  - Sperre für Kundnr=5, Warten von 10 sec., Sperre für Nr 1 besetzt
- Anwendung 2:
  - Kunde1: Kundnr = 1
  - Kunde2: Kundnr = 5
  - 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.

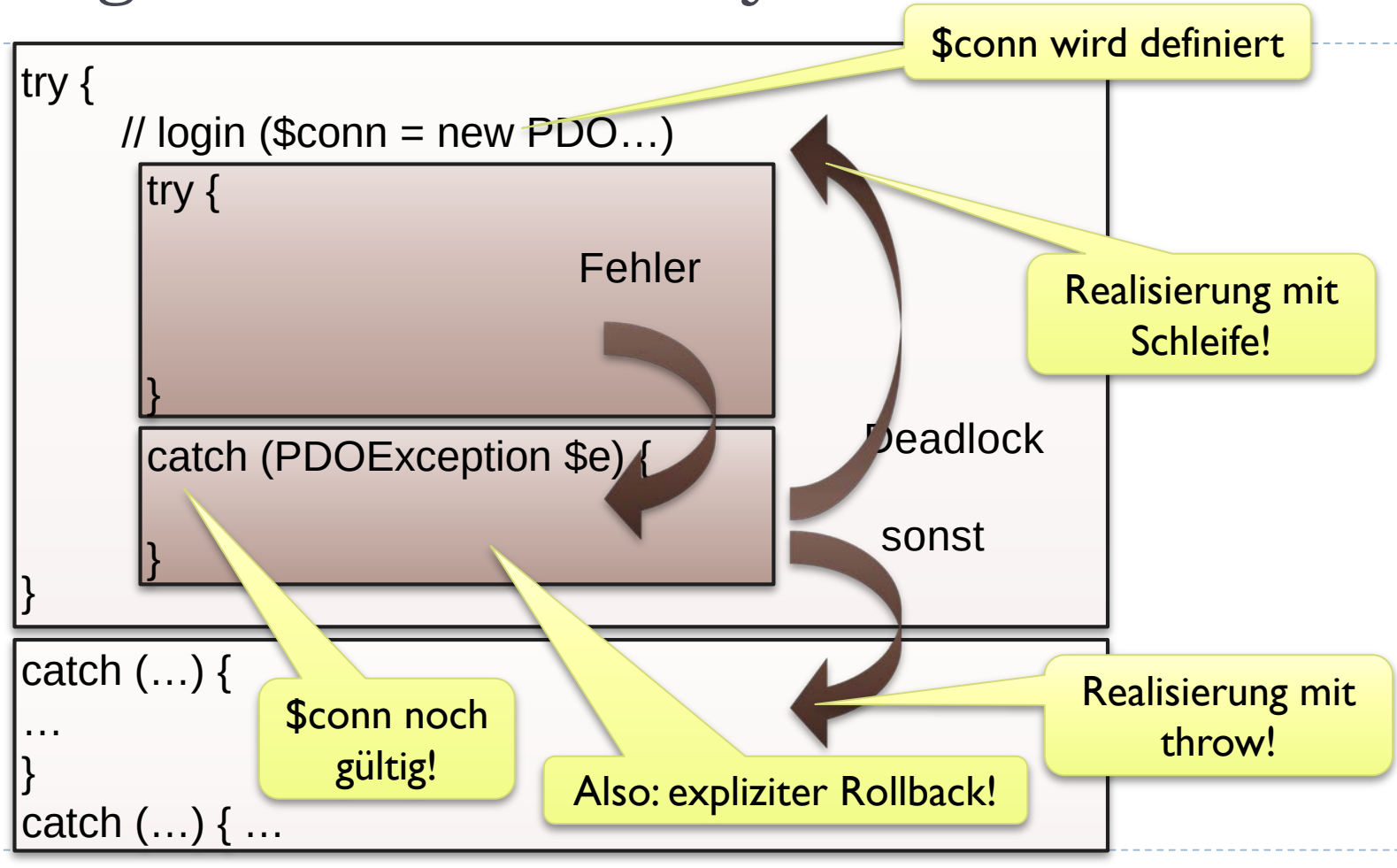
Kundnr	Name	Sperre	Kunde1	Kunde2
5	Biker Ecke	0	◉	○
1	Fahrrad Shop	0	○	◉

Kundnr	N…	Sperre	Kunde1	Kunde2
5	…r Ecke	0	○	◉
	Fahrrad…	0	◉	○

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
              // Zugriff1
            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
}  // end try
```

Erster Update

Warten

Zweiter Update

Lösung in PHP

```
catch (PDOException $e) {
   $code = $e->getCode();
   if ($code == "HY000")
      $code = $e->errorInfo[1] ;

   if ( $versuch <= 1 and ($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:

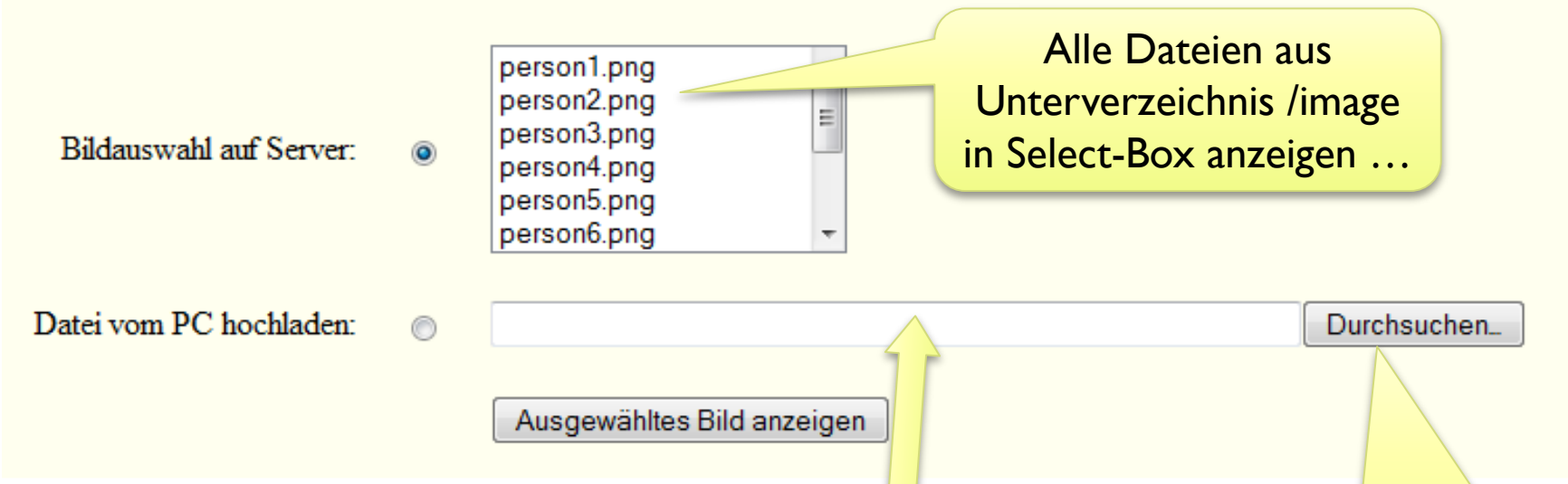
```
ALTER TABLE Personal
  ADD  Bild BLOB Default EMPTY_BLOB() ;
ALTER TABLE Personal
  ADD  Bildtyp CHAR(40) ;
```

Oracle

SQL Server: Add Bild Varbinary(max);

MySQL: Add Bild LongBlob;

Bilder und PHP: blob_ein1.php



Neue Funktionen:
- scandir
- is_dir

Neues Formularelement:
<input type="file" name=…

Implementierung: blob_ein1.php (1)

```
<form action="blob_ein2.php" method="post"
                                    enctype="multipart/form-data">
  <p>Bildauswahl auf Server:
  <input type="radio" name="Auswahl" value="Server" checked/>
  <select name="Bild" size="6">
<?php
   $dir = "./image";
   if (is_dir($dir))
    if ($filearray = scandir($dir))
     foreach ($filearray as $name)
      if (!is_dir($name))
        echo "<option> $name </option>";
?> </select>
```

Ruft blob_ein2.php auf

Notwendig, wenn file-Typ verwendet wird

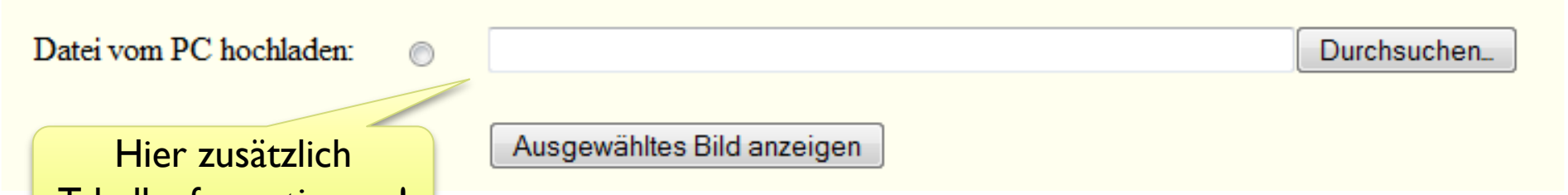
Existiert Verzeichnis ./image?

Liest alle Dateien des Verzeichnisses in Feld $filearray

Ver-zeich-nis nicht leer?

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

Implementierung: blob_ein1.php (2)



Hier zusätzlich Tabellenformatierung!

```
<p>Datei vom PC hochladen:
  <input type="radio" name="Auswahl" value="Lokal"/>

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

  <p> <input type="Submit" value="Ausgewählte Bild anzeigen"/></p>
</form>
```

Angabe der akzeptierten Dateitypen, hier: Bilder

Eigenschaften des Typs $_FILES

- Dateien werden mit $_FILES übertragen, nicht $_POST
- $_FILES besitzt vier assoziative Werte:
  - $_FILES['Bilddatei']['name']
    - Name der Datei, die auf dem PC ausgewählt wurde
    - Name des Input-Files
  - $_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'];
```

Falls Bild vom PC hochgeladen

bzw.

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

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)

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!
  - Mit der Methode query ist dies nicht möglich
  - Wir verwenden die Methoden prepare und execute!
- **Gleichwertig:**

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

Parsen und Ausführen

Parsen des Befehls

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

Ausführen

gleichwertig

Parsen und Vorbereiten des Update

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

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

$stmt->bindParam( 1, $_SESSION['Bildtyp'] ) ;
$stmt->bindParam( 2, $fp, PDO::PARAM_LOB ) ;
$stmt->execute( ) ;

fclose( $fp ) ;
```

Platzhalter

Oracle

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

Platzhalter

Vorbereiten des Befehls

Bild binär zum Lesen öffnen

1. Fragezeichen mit Variable verknüpfen

2. Fragezeichen mit Variable $fp als LOB verknüpfen

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.:

Gibt Bildgröße in Byte zurück

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

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

```
<?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);
?>
```

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

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

PHP-Funktionen (2)

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