S5: HTML-php-SQL-ORM

Lehrinhalt

Programmiersprachen sind wie alle Sprachen eine Kommunikationsform für Inhalt/Konzepte.

Das bedeutet es ist davon auszugehen, dass die Sprache das ausdrücken kann was ich für meine Aufgabe brauche, ansonst ist es das falsche Werkzeug.

Das bedeutet auch, dass man nicht das ganze Handbuch auswendig lernt bevor man anfangen kann (ich habe keine Ahnung von der Grammatik die ich täglich beim Sprechen benutze und ich habe sprechen gelernt bevor ich das Wort Gramatik oder dessen Bedeutung kannte).

- Programmiersprachen: HTML, php, SQL. Problem: drei Sprachen. Vorteil: HTML zeigt Struktur, SQL und php zwei Programmiersprachenvarianten.
- Erstellen einer Programmbibliothek.

Projekterfassung/start: Inputs, Outputs

Als Beispielprojekt: eine Leihbücherei für die Schule. (Jeder versteht das Problem, wir brauchen keinen Domainexperten, bzw. können uns in der Schulbibliothek Antworten und Daten holen und eine Implementierung anschauen.

Um schnell zu einem Anfang zu kommen suche ich mir irgendeine Ausgabe (Output), einen Bericht. In eine Leihbibliothek braucht der Bibliothekar Bücherlisten, Leserlisten und eine Liste der ausgeliehenen Bücher (Korrektur: Bücher sind Medien, es Bücher, Zeitungen, CD-ROM, DVD, eBooks, ...).

Wir nehmen einfach Leserlisten, also eigentlich die Liste der Ausleiher (Namen sind wichtig und müssen so gut wie möglich beschreiben was sie enthalten) ... Benutzer/User.
(Heißt es Bibliothek oder Bücherei ?)

Ein Benutzerbericht wird ungefähr so ausschauen:

```
Benutzerliste xyz-Bibliothek                              Seite #

Benutzer Nr   Name
....
...
..

(Druckdatum und Uhrzeit)
```

Der Bibliotheksname steht in der Systeminformation/Konfiguration, die Seitennummer ergibt sich (hoffentlich) beim Ausdruck, Druckdatum und Uhrzeit kommen vom Betriebssystem, das bedeutet aus der Datenbank kommen Benutzer Nummer und Name.

Einen Output nehmen und das Datenmodell erstellen, mit SQL direkt ausprobieren.

Datenbank (Einschub)

Warum werden Datenbanken verwendet

- weil die Datenmenge nicht mehr in den Hauptspeicher passt. Das ist bei uns nur bedingt der Fall, weil in 8GB Hauptspeicher leicht die Daten der HTL-Schulbibliothek platz haben.

- Zugriff von mehreren Clients auf eine Datenbank.
- SQL Abfragesprache. Seit x-Jahren ist NoSQL eine alternative (ob Multivaluesystem die Vorgänger waren ? weiß ich nicht)

Keiner der drei Punkte ist technisch relevant, für mich ist eine Datenbank etwas dem ich die Daten zur Ablage gebe das mir eine Methode bietet sie wieder zu bekommen (auch wenn der Computer ausgeschalten wird) also ein Persistenz-irgendwas.

Relationale Datenbanken muss man kennen, weil sie viel verwendet werden (den Unterschied zu NoSQL (komischer Name weil SQL die Abfrage/Manipulationssprache ist) anschauen ?). Relational heißen sie weil sie auf einem mathematischen Relationsmodell beruhen (denke ich).
Einfacher:

- eine relationale Datenbank ist eine dauerhafte Ablage von Daten
- in Form von Tabellen. Tabellen sind (hier in der DB) Listen von
- Einträgen definierter Struktur.

```
+----+------+---------+------------+--------+-----+
| Nr | Name | Vorname | GebDatum   | Email  | Tel |
+----+------+---------+------------+--------+-----+
|  5 | Abc  | Defg    | 2000-01-02 | a@b.at | ..  |
| 15 | Abc  | Defg    | 2001-01-02 | b@b.at | ..  |
+----+------+---------+------------+--------+-----+
```

Wichtig ist eine Spalte hat einen definerten Typ, es können also in "GebDatum" nur Datumswerte abgelegt werden, keine Texte, Zahlen, Formeln und es kann nur "ein" Wert abgelegt werden (erste Normalform)

SQL Kommandozeile

Obige Tabelle ist schon recht Nahe an dem was wir für die Benutzerverwaltung brauchen (ein guter Anfang). Die "Nr" ist die Benutzernummer, zur eindeutigen Identifikation der Einträge einer Tabelle gibt es in relationalen Datenbanken einen "PRIMARY KEY", dazu nehme ich immer eine eigene Spalte "ID".

SQL, structured query language, ist ein Versuch in einfachem Englisch mit dem Softwaresystem zu kommunizieren.

Um die Tabelle zu erzeugen brauchen wir

- ein SQL Datenbanksystem: mysql (oder sqlite)
- die Kommandozeile: "CREATE DATABASE bib;" erzeugt eine Datenbank "bib".
- "use bib;" setzt "bib" als unsere default Datenbank.
- ```
  CREATE TABLE user (
    ID   int PRIMARY KEY auto_increment,
    Nr       VARCHAR(16),
    Name     VARCHAR(64),
    Vorname  VARCHAR(64),
    GebDatum DATE,
    Email    VARCHAR(128),
    Tel      VARCHAR(32)
  );
  ```

  Erzeugt eine Tabelle. "auto_increment" ist mysql-spezifischund führt dazu dass die Datenbank den Wert für "ID" selbst vergibt, wenn keiner angegeben wird.
  "PRIMARY KEY" bedeutet der "ID" muß für jede Zeile (Eintrag) in der Tabelle eindeutig sein und darf nicht "NULL" sein.

"NULL" ist ein Nichtwert, wenn man das Geburtsdatum nicht kennt, wird der Wert "NULL" abgespeichert und nicht irgendein Wert der nicht sein kann (welcher wäre das ?).

Wenn ein Eintrag für "Name" verpflichten sein soll, schreibt man "Name VARCHAR(64) NOT NULL" in der Tabellendefinition (ein leerer String '' oder ein Leerzeichen ' ' reichen aber um diese Prüfung zu umgehen).

Wenn "Nr" eindeutig sein soll "Nr VARCHAR(16) UNIQUE" schreiben.
  ```

```
CREATE TABLE user (
  ID   int PRIMARY KEY auto_increment,
  Nr       VARCHAR(16) UNIQUE,
  Name     VARCHAR(64) NOT NULL,
  Vorname  VARCHAR(64) NOT NULL,
  GebDatum DATE,
  Email    VARCHAR(128),
  Tel      VARCHAR(32)
);
```

Werte können nur Zeilenweise eingefügt werden:

```
INSERT INTO user (Nr, Name, Vorname) VALUES('L1', 'abc', 'defg');
```

wenn die selbe Zeile nocheinmal eingegeben wird, bekommt man die Fehlermeldung: "ERROR 1062 (23000): Duplicate entry 'L1' for key 'Nr

```
INSERT INTO user (Nr, Name, Vorname) VALUES('L11', 'abc', 'defg');
```

Abfrage des Tabelleninhalts.

```
select * from user;
+----+------+------+---------+----------+-------+------+
| ID | Nr   | Name | Vorname | GebDatum | Email | Tel  |
+----+------+------+---------+----------+-------+------+
|  1 | L1   | abc  | defg    | NULL     | NULL  | NULL |
|  3 | L11  | abc  | defg    | NULL     | NULL  | NULL |
+----+------+------+---------+----------+-------+------+
2 rows in set (0,02 sec)
```

Details zu SQL sind im Handbuch auf mysql.com nachschaubar (EBNF erklären ?)

Den Bericht ausgeben

HTML, HTTP (Einschub)

HyperTextMarkupLanguage und HyperTextTransportProtocol

Markup bedeutet Auszeichnung, das heißt es gibt die Möglichkeit Teile des Textes auszuzeichnen, in bestimmten Aspekten hervorzuheben, z.B. Fettdruck.

HTML benutzt als Escapezeichen "<" und ">", der Text zwischen diesen beiden Klammern ist ein HTML-Kommandowort. "fettgedruckt" ... die Zeichen zwischen "" und "" werden dickgedruckt (bold) dargestellt.

Die zentrale Idee von HyperText ist (für mich) der Hyperlink

```
<a href="http://wikipedia.de/index.html">hier</a> nachschauen.
```

In einem HTML-Anzeigeprogramm (Browser) steht dann "hier nachschauen" und wenn auf "hier" geklickt wird, verbindet sich der Browser zum Port 80 des Servers wikipedia.de und fordert dort die Datei "index.html" an.

```
telnet wikipedia.de 80
GET /index.html HTTP/1.0
```

Der Server schickt dann den Inhalt der Datei oder einen HTTP-Header mit Fehlernummer.

PHP

Um den Inhalt der vom server verteilten Datei bei jeder Anfrage anders gestalten zu können, z.B. das aktuelle Datum ausgeben, gibt es viele Möglichkeiten, CGI, perl, php, usw.

PHP ist eine eingebettete Sprache bzw. verwendet das HTML Escapeelement.

```
ein normaler <b>html</b> Text mit aktueller Serverzeit:
<?php echo strftime("%H:%M:%S"); ?>
```

Steht dieser text in "zeit.php" und der http-Server ist korrekt konfiguriert, wird die Dateian den php-Interpreter übergeben und der http-Server liefert die Ausgabe des php-Interpreters an den Browser.
Der php-Interpreter gibt den ganzen Text der nicht zwischen "<?php" und "?>" steht unverändert weiter, der Text zwischen diesen zwei Elementen muß gültiger php-Code sein, der ausgeführt wird. Wenn der Code nichts ausgibt läuft das unbeobachtet vom Browser ab.

"strftime" ist string formatted time, eine Funktion die die Zeit im angeforderten Format zurück gibt. "echo" gibt diesen String dann aus.

Datenbankabfrageseite

Wir möchten ungefähr so etwas

```
Benutzerliste xyz-Bibliothek                          Seite #

Benutzer Nr   Name
....
...
..

(Druckdatum und Uhrzeit)
```

In HTML

```
Benutzerliste xyz-Bibliothek                          Seite #

Benutzer Nr   Name
....
...
..

(Druckdatum und Uhrzeit)
```

HTML ist nur eine Markup- aber keine Programmiersprache, deshalb brauchen wir php. Folgendes in eine Datei ... "userlist.php" speichern.

```
Benutzerliste xyz-Bibliothek                          Seite #

Benutzer Nr   Name
....
...
..

(erstellt am <?php echo strftime("%Y-%m-%d"); ?> um <?php echo strftime("%H:%M"); ?>)
```

Da HTML die Zeilenwechsel ignoriert, steht alles in einer Zeile, ausser man macht das Fenster schmäler.

http://php.net/manual/en/mysqli.quickstart.php

Um Daten von der Datenbank zu holen macht man das selbe wie auf der Kommandozeile:

```
<?php
// connect to server and database (cmdline : mysql.exe -h localhost bib)
$mysqli = new mysqli("localhost", "", "", "bib");
// hoffentlich ist der Zugriff erlaubt.
// Abfrage abschicken
$res = $mysqli->query("SELECT * FROM user");
// Spaltennamen
printf("Benutzer Nr Name<br />\n");
// Ergebnisdaten zeilenweise (row) holen und ausgeben
while ($row = $res->fetch_assoc()) {
    printf("%s %s %s<br />\n", $row["Nr"], $row["Name"], $row["Vorname"]);
}
?>
```

"printf" (ähnlich wie in C) gibt den als erstes angegeben String aus, nachdem die "%s" Platzhalter durch die weiteren Parameter ersetzt wurden. Im ersten Fall gibt es keine "%"-Zeichen im String "Benutzer Nr Name
\n" also wird der String unverändert ausgegeben.

"\n" wird in einen Zeilenwechsel (newline) umgewandelt.
"
" veranlasst die HTML-Anzeige einen Zeilenwechsel vorzunehmen.

Eingabeformular erstellen

Wenn der Benutzer Daten eingeben soll muss er ein Formular ausfüllen, in english a "form" therefore the html construct is "<form>" ... "</form>", these two group the fields inbetween where data can be entered. These are"<input>" and strangely have no "</input>" ending.
Das einfachste Formular ist also (ausprobieren):

```
<form>
<input>
</form>
```

Dieses Formular zeigt ein Texteingabefeld an. Das Formular wird vom Browser am Client-PC dargestellt, um die eingegebenen Daten an den Server zu schicken braucht es einen Senden-Knopf, english submit.

```
<form>
<input>
<input type="submit">
</form>
```

Wie und wohin werden die Daten geschickt ?
Ist in "<form>" keine action angegeben ist werden die Daten an die selbe Webseite geschickt von der das Formular geladen wurde.

Anders hier

```
<form action="http://somewhereel.se">
</form>
```

Ist in "<form>" keine "method" angegeben werden die Daten als GET-parameter übergeben, das heißt an die Webseitenadresse angehängt: http://server.com/meinformular.html?p1=eins&p2=zwei.

p1 und p2 sind namen die im Formular vergeben werden "eins" und "zwei" die Texte die der Benutzer eingetragen hat.

```
<form>
<input name="p1">
<input name="p2">
<input type="submit">
</form>
```

Obiges in "userentry.php" speichern und im Browser aufrufen.

Wenn man auf "submit" klickt ändert sich die Webadresse zu

```
http://localhost/userentry.php?p1=&p2=
```

In php stehen die eingegebenen Werte in globalen Dictionaries zur Verfügung: "$_GET" oder "$_REQUEST".

php Variablen und dictionaries

Dictionaries sind Arrays die nicht mit Zahlen sondern Strings indiziert werden können.
Das Dollarzeichen "$" vor den Variablennamen ist deshalb notwendig, weil in php die Variable direkt in einer Zeichenkette expandiert werden kann.

```
// einer Variable einen Wert zuweisen. Wenn die Variable noch nicht existiert wird sie erzeug
t.
$name = "alf";
echo "Mein Name ist $name."
// es wird "Mein Name ist alf." ausgegeben
```

Formular V0.1

```
<pre>
<!-- pre = preformatted text = the browser interprets text newline characters as newline-->
<?php
print_r($_GET);
?>
</pre>
<form>
<input name="p1">
<input name="p2">
<input type="submit">
</form>
</pre>
```

Wenn in den Eingabefeldern "eins" und "zwei" eingetragen wird und "submit" geklickt wird zeigt der Browser die Webadresse "http://localhost/userentry.php?p1=eins&p2=zwei" und zeigt

```
Array
(
    [p1] => eins
    [p2] => zwei
)
```

Formular V0.2

Wir benötigen zur Eingabe der Benutzerdaten Nr, Name, Vorname, Geburtsdatum, Emailadresse und Telefonnummer.

```
<pre>
<!-- pre = preformatted text = the browser interprets text newline characters as newline-->
<?php
print_r($_GET);
?>
</pre>
<form>
<input name="Nr">
<input name="Name">
<input name="Vorname">
<input name="GebDatum">
<input name="Email">
<input name="Tel">
<input type="submit">
</form>
```

Das schaut ungefähr so aus.

Formular V0.3

Hmm? Machen wir ein vertikales Formular mit Feldbezeichnnungen !

```
<pre>
<!-- pre = preformatted text = the browser interprets text newline characters as newline-->
<?php
print_r($_GET);
?>
</pre>
<form>
Nr <input name="Nr"><br/>
Name <input name="Name"><br/>
Vorname <input name="Vorname"><br/>
Geb.Datum <input name="GebDatum"><br/>
Email <input name="Email"><br/>
Tel <input name="Tel"><br/>
<input type="submit">
</form>
```

Formular V0.4

Man kann die gesendeten Werte auch im Formular anzeigen, wenn man ein "value"-Attribut zum input angibt.
Und den "submit"-Knopf in "Speichern umbenennen.

```
<form>
Nr <input name="Nr" value="<?php echo $_GET["Nr"]; ?>"><br/>
Name <input name="Name" value="<?php echo $_GET["Name"]; ?>"><br/>
Vorname <input name="Vorname"><br/>
Geb.Datum <input name="GebDatum"><br/>
Email <input name="Email"><br/>
Tel <input name="Tel"><br/>
<input type="submit" value="Speichern">
</form>
```

Formular V0.5

Wie üblich beim Arbeiten fällt Abfall an, bzw. wenn man nicht putzt beginnt es zu stinken. Wir kümmern uns nur um *duplicate code*. der Vorgang heisst refactoring.

```
Nr <input name="Nr" value="<?php echo $_GET["Nr"]; ?>"><br/>
```

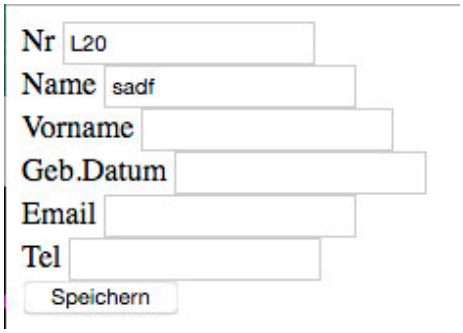
In der Zeile steht dreimal "Nr", wir extrahieren.

```
<?php
$name = "Nr";
printf('%s <input name="%s" value="%s"><br/>\n', $name, $name, $_GET[$name]);
?>
```

Das ist so auch einfacher zu lesen.

```
<form>
<?php
$name = "Nr";
printf('%s <input name="%s" value="%s"><br/>', $name, $name, $_GET[$name]);
?>
Name <input name="Name" value="<?php echo $_GET["Name"]; ?>"><br/>
Vorname <input name="Vorname"><br/>
Geb.Datum <input name="GebDatum"><br/>
Email <input name="Email"><br/>
Tel <input name="Tel"><br/>
<input type="submit" value="Speichern">
</form>
```

Das Formular schaut immer noch gleich (hässlich) aus.



Formular V0.6

Das selbe muss mit den anderen Eingabezeilen gemacht werden. MUSS doppelter Code ist der einzige code smell der entfernt werden muss, aber dieser muss.

```
<?php
$name = "Nr";
printf('%s <input name="%s" value="%s"><br/>', $name, $name, $_GET[$name]);
$name = "Name";
printf('%s <input name="%s" value="%s"><br/>', $name, $name, $_GET[$name]);
?>
```

Die printf-Zeile ist beidesmal die selbe, d.h. wir rollen eine for-Schleife aus mit der dazugehörigen Datenstruktur einem array.
Achtung: vorerst machen wir die Zeilenbeschreibung beim GebDatum ohne ":".

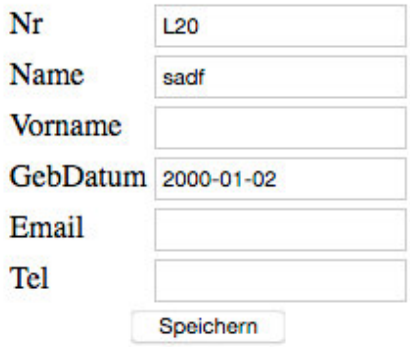
```
<form>
<?php
$fields = array("Nr", "Name", "Vorname", "GebDatum", "Email", "Tel");
foreach ($fields as $name) {
    printf('%s <input name="%s" value="%s"><br/>', $name, $name, $_GET[$name]);
}
?>
<input type="submit" value="Speichern">
</form>
```

Etwas weniger Code leichter wartbar und das ist dann programmieren.

Formular V0.7

Schönere Darstellung in einer Tabelle. Hier hat man dann auch den Vorteil durch das refactoring oben weil nur ein String die Darstellung definiert und man nicht für jedes INPUT eine Tabellenzeile schreiben muss.

```
<form>
<table>
<?php
$fields = array("Nr", "Name", "Vorname", "GebDatum", "Email", "Tel");
foreach ($fields as $name) {
    printf('<tr><td>%s</td><td><input name="%s" value="%s"></td></tr>', $name, $name, $_GET[$
name]);
}
?>
<tr><td colspan="2" align="center"><input type="submit" value="Speichern"></td></tr>
</form>
```



and pack it into a function.

```
<?php
// functions

function build_form($fieldnames) {
// build a html form from fieldnames: array of strings
?>
<form>
<table>
<?php
foreach ($fieldnames as $name) {
    printf('<tr><td>%s</td><td><input name="%s" value="%s"></td></tr>', $name, $name, $_GET[$
name]);
}
?>
<tr><td colspan="2" align="center"><input type="submit" value="Speichern"></td></tr>
</form>
<?php
}

$fields = array("Nr", "Name", "Vorname", "GebDatum", "Email", "Tel");

printf("%s", build_form($fields));
```

Die Funktion ist unabhängig von den Daten bzw durch ihren Parameter fieldnames gesteuert und kann auch für andere Formulare verwendet werden, deshalb lagern wir die Funktion in eine Datei lib.inc.php aus.
Wir führen einen Dateikopf ein, damit wir wissen wo wir sind, weil wir gerade dabei sind mit Versionsnummer (das macht bei mir eigentlich die *Versionskontrollsoftware*).

```
<?php
// file: lib.inc.php
// version: 0.1

// functions

function build_form($fieldnames) {
// build a html form from fieldnames: array of strings
?>
<form>
<table>
<?php
foreach ($fieldnames as $name) {
    printf('<tr><td>%s</td><td><input name="%s" value="%s"></td></tr>', $name, $name, $_GET[$
name]);
}
?>
<tr><td colspan="2" align="center"><input type="submit" value="Speichern"></td></tr>
</form>
<?php
}
```

```
<?php
// file: userentry.php
// version 0.7

$fields = array("Nr", "Name", "Vorname", "GebDatum", "Email", "Tel");

printf("%s", build_form($fields));
```

Formular V0.8 SQL INSERT erzeugen

Um in der Datenbank zu speichern müssen wir einen "INSERT INTO ..." query schicken (prepared statement ein andermal). Ein komplettes INSERT statement enthält zusätzlich zu den VALUES auch ein fieldlist, die Feldnamen auf denen die nachfolgenden VALUES gespeichert werden sollen.

```
INSERT INTO user (Nr,Name,Vorname,Gebdatum,Email,Tel) Values('$_GET["Nr"], ...
```

Aber die Fieldlist ist der Inhalt unseres Arrays "$fields".

Wir prüfen ob "Name" in "$_GET" gesetzt und nicht leer ist.

```
<?php
// file: userentry.php
//version: 0.8a

require("lib.inc.php");

$fields = array("Nr", "Name", "Vorname", "GebDatum", "Email", "Tel");
// save if there is something
if (!empty($_GET["Name"])) {
    $fs = "";
    $vs = "";
    foreach ($fields as $f) {
        $fs .= $f.",";
        $vs .= "'".$_GET[$f]."',";
    }
    echo "INSERT INTO user ($fs) VALUES($vs);<br>";
}

printf("%s", build_form($fields));
```

Der "." hängt zwei Strings aneinander.
".=" ist die Kurzform für "$s = $s . ',';" es wird ein Komman an "$s" angehängt.

```
$vs .= "'".$_GET[$f]."',";
```

Hängt an den String "$vs" ein Apostroph "'" und den Wert der übergeben wurde und ein Apostroph und einen Beistrich.

Wenn wir das Statement "INSERT INTO user (Nr,Name,Vorname,GebDatum,Email,Tel,)
VALUES('L20','sadf','','2000-01-02','','',);" in die mysql-Kommandozeile kopieren, bekommen wir eine Fehlermeldung.
Weil am Ende der fieldlist und der valuelist ein Beistrich zuviel ist, wenn wir den entfernen, funktioniert es.

```
<?php
// file: userentry.php
// version: 0.8b
require("lib.inc.php");

$fields = array("Nr", "Name", "Vorname", "GebDatum", "Email", "Tel");
// save if there is something
if (!empty($_GET["Name"])) {
    $fs = "";
    $vs = "";
    foreach ($fields as $f) {
        $fs .= $f.",";
        $vs .= "'".$_GET[$f]."',";
    }
$fs = substr($fs, 0, -1);
    $vs = substr($vs, 0, -1);
    echo "INSERT INTO user ($fs) VALUES($vs);<br>";
}

printf("%s", build_form($fields));
```

Formular V0.9 zur Datenbank

```
<?php
// file: userentry.php
// version: 0.9a

require("lib.inc.php");

$mysqli = new mysqli("localhost", "", "", "bib");

$fields = array("Nr", "Name", "Vorname", "GebDatum", "Email", "Tel");
// save if there is something
if (!empty($_GET["Name"])) {
    $fs = "";
    $vs = "";
    foreach ($fields as $f) {
        $fs .= $f.",";
        $vs .= "'".$_GET[$f]."',";
    }
    $fs = substr($fs, 0, -1);
    $vs = substr($vs, 0, -1);
$res = $mysqli->query("INSERT INTO user ($fs) VALUES($vs)");
    if ($mysqli->errno) {
        printf("<div>%s</div>\n", $mysqli->error);
    }
}

printf("%s", build_form($fields));
```

Dadurch dass "bib.Nr" eindeutig sein muss passiert uns beim zweiten abschicken keine zweiter INSERT.

Vorher extrahieren wir einmal eine Funktion build_sql_insert.

```
<?php
// file: userentry.php
// version: 0.9b

require("lib.inc.php");

function build_sql_insert( $tablename, $fieldnames, $values )
// build sql insert statement. fieldnames: array of strings, values: dictionary indexed by fe
ldname
{
    $fs = "";
    $vs = "";
    foreach ($fieldnames as $f) {
        $fs .= $f.",";
        $vs .= "'".$values[$f]."',";
    }
    $fs = substr($fs, 0, -1);
    $vs = substr($vs, 0, -1);
    return "INSERT INTO $tablename ($fs) VALUES($vs)";
}

$mysqli = new mysqli("localhost", "", "", "bib");

$fields = array("Nr", "Name", "Vorname", "GebDatum", "Email", "Tel");
// save if there is something
if (!empty($_GET["Name"])) {
    $res = $mysqli->query(build_sql_insert("user", $fields, $_GET));
    if ($mysqli->errno) {
        printf("<div>%s</div>\n", $mysqli->error);
    }
}

printf("%s", build_form($fields));
```

Die Funktion verschieben wir auch nach lib.inc.php version: 0.2.

```
<?php
// file: userentry.php
// version: 0.9c

require("lib.inc.php");
$mysqli = new mysqli("localhost", "", "", "bib");

$fields = array("Nr", "Name", "Vorname", "GebDatum", "Email", "Tel");
// save if there is something
if (!empty($_GET["Name"])) {
    $res = $mysqli->query(build_sql_insert("user", $fields, $_GET));
    if ($mysqli->errno) {
        printf("<div>%s</div>\n", $mysqli->error);
    }
}

printf("%s", build_form($fields));
```

Formular V1.0 UPDATE entry

Um einen Eintrag zu ändern müssen wir wissen welchen. "Welcher" bedeutet in Relationalendatenbanken welche Tabelle welcher PRIMARY KEY.

Die Spalte ID haben wir bis jetzt in den Formularen nicht beachtet. FÜgen wir sie einmal zu den Feldern hinzu.

```
<?php
// file: userentry.php
// version: 1.0a

require("lib.inc.php");
$mysqli = new mysqli("localhost", "", "", "bib");

$fields = array("ID", "Nr", "Name", "Vorname", "GebDatum", "Email", "Tel");
// save if there is something
if (!empty($_GET["Name"])) {
    $res = $mysqli->query(build_sql_insert("user", $fields, $_GET));
    if ($mysqli->errno) {
        printf("<div>%s</div>\n", $mysqli->error);
    }
}

printf("%s", build_form($fields));
```

Wenn man für ID einen bereits existierenden Wert eingibt bekommt man die Fehlermeldung:

```
Duplicate entry '1' for key 'PRIMARY'
```

Wir können bei nicht leerem $_GET["ID"] davon ausgehen, dass es kein INSERT sondern ein update ist.

```
<?php
// file: userentry.php
// version: 1.0b

require("lib.inc.php");
$mysqli = new mysqli("localhost", "", "", "bib");

$fields = array("ID", "Nr", "Name", "Vorname", "GebDatum", "Email", "Tel");
// save if there is something
if (!empty($_GET["Name"])) {
if (empty($_GET["ID"]) {
        $res = $mysqli->query(build_sql_insert("user", $fields, $_GET));
        if ($mysqli->errno) {
            printf("<div>%s</div>\n", $mysqli->error);
        }
}
    else {
        echo "UPDATE";
    }
}

printf("%s", build_form($fields));
```

Das UPDATE Format in SQL ist leider ganz anders als das INSERT, wir benötigen

```
UPDATE user SET Name='def', Tel='123' WHERE ID=2;
```

Analog zu build_sql_insert machen wir uns ein build_sql_update, in lib.inc.php version 0.3

```
function build_sql_update($tablename, $fieldnames, $values) {
// build an SQL UPDATE. Assume PRIMARY KEY is named ID.
    $sql = "UPDATE $tablename ";
    foreach ($fieldnames as $f) {
        $sql .= "SET $f='".$values[$f]."',";
    }
    $sql = substr($sql, 0, -1);  // strip last semicolon
    $sql .= " WHERE ID=".$_GET["ID"];
    return $sql;
}
```

```
<?php
// file: userentry.php
// version: 1.0c

require("lib.inc.php");

$mysqli = new mysqli("localhost", "", "", "test");

$fields = array("ID", "Nr", "Name", "Vorname", "GebDatum", "Email", "Tel");
// save if there is something
if (!empty($_GET["Name"])) {
    if (empty($_GET["ID"])) {
        $res = $mysqli->query(build_sql_insert("user", $fields, $_GET));
        if ($mysqli->errno) {
            printf("<div>%s</div>\n", $mysqli->error);
        }
    }
    else {
        echo build_sql_update("user", $fields, $_GET);
    }
}

printf("%s", build_form($fields));
```

den ausgegebenen SQL UPDATE kopieren wir in die SQL Konsole und probieren ihn aus.

```
UPDATE user SET ID='1',SET Nr='abcd',SET Name='dd',SET Vorname='',SET GebDatum='',SET Email='
',SET Tel='' WHERE ID=1
```

Der funktioniert nicht, es braucht nur ein SET Keyword.

```
UPDATE user SET  ID='1', Nr='L14', Name='dd', Vorname='', GebDatum='', Email='', Tel='' WHERE
ID=1
```

in lib.inc.php version: 0.4

```
function build_sql_update($tablename, $fieldnames, $values) {
// build an SQL UPDATE. Assume PRIMARY KEY is named ID.
    $sql = "UPDATE $tablename SET ";
    foreach ($fieldnames as $f) {
        $sql .= "$f='".$values[$f]."',";
    }
    $sql = substr($sql, 0, -1);  // strip last semicolon
    $sql .= " WHERE ID=".$_GET["ID"];
    return $sql;
}
```

Das Skript mit UPDATE

```
<?php
// file: userentry.php
// version: 1.0d

require("lib.inc.php");

$mysqli = new mysqli("localhost", "", "", "test");

$fields = array("ID", "Nr", "Name", "Vorname", "GebDatum", "Email", "Tel");
// save if there is something
if (!empty($_GET["Name"])) {
    if (empty($_GET["ID"])) {
        $res = $mysqli->query(build_sql_insert("user", $fields, $_GET));
        if ($mysqli->errno) {
            printf("<div>%s</div>\n", $mysqli->error);
        }
    }
    else {
        $res = $mysqli->query(build_sql_update("user", $fields, $_GET));
        if ($mysqli->errno) {
            printf("<div>%s</div>\n", $mysqli->error);
        }
    }
}

printf("%s", build_form($fields));
```

wenn wir die ID eines neuangelegten Eintrags, dann können wir einen Eintrag anlegen und nachbearbeiten:
in mysqli gibt es $insert_id.

```php
<?php
// file: userentry.php
// version: 1.0e

require("lib.inc.php");

$mysqli = new mysqli("localhost", "", "", "test");

$fields = array("ID", "Nr", "Name", "Vorname", "GebDatum", "Email", "Tel");
// save if there is something
if (!empty($_GET["Name"])) {
    if (empty($_GET["ID"])) {
        $res = $mysqli->query(build_sql_insert("user", $fields, $_GET));
        if ($mysqli->errno) {
            printf("<div>%s</div>\n", $mysqli->error);
        }
        else {
            $_GET["ID"] = $mysqli->insert_id;
        }
    }
    else {
        $res = $mysqli->query(build_sql_update("user", $fields, $_GET));
        if ($mysqli->errno) {
            printf("<div>%s</div>\n", $mysqli->error);
        }
    }
}

printf("%s", build_form($fields));
```

Und es fehlt noch das Bearbeiten eines nicht gerade jetzt erstellten Eintrags.
Dazu brauchen wir den PRIMARY KEY "ID" und müssen die Werte aus der Datenbank lesen.

```
<?php
// file: userentry.php
// version: 1.0f

require("lib.inc.php");

$mysqli = new mysqli("localhost", "", "", "test");

$fields = array("ID", "Nr", "Name", "Vorname", "GebDatum", "Email", "Tel");
// save if there is something
if (!empty($_GET["Name"])) {
    if (empty($_GET["ID"])) {
        $res = $mysqli->query(build_sql_insert("user", $fields, $_GET));
        if ($mysqli->errno) {
            printf("<div>%s</div>\n", $mysqli->error);
        }
        else {
            $_GET["ID"] = $mysqli->insert_id;
        }
    }
    else {
        $res = $mysqli->query(build_sql_update("user", $fields, $_GET));
        if ($mysqli->errno) {
            printf("<div>%s</div>\n", $mysqli->error);
        }
    }
}

// load values from db. either the one saved above or an entry to be edited.
if (!empty($_GET["ID"])) {
    // load from database
    $res = $mysqli->query("SELECT * FROM user WHERE ID=".$_GET["ID"]);
    $_GET = $res->fetch_array(MYSQLI_ASSOC);
}

printf("%s", build_form($fields));
```

Wenn wir jetzt Speichern drücken wird im Formular bei Geburtsdatum ein angezeigt "0000-00-00", weil die Werte aus der Datenbank gelesen werden.

Der Aufruf von http://localhost/userentry.php?ID=3 zeigt den aktuellen Eintrag der Datenbank und wir können ihn bearbeiten und speichern.

User report formatted

Damit wir die IDs nicht raten müssen machen wir in den Bericht von ganz vom oben Links.

```
<?php
// file: userlist.php
// version: 0.2

// connect to server and database (cmdline : mysql.exe -h localhost bib)
$mysqli = new mysqli("localhost", "", "", "bib");
// hoffentlich ist der Zugriff erlaubt.
// Abfrage abschicken
$res = $mysqli->query("SELECT * FROM user");
// Spaltennamen
printf("Benutzer Nr Name<br />\n");
// Ergebnisdaten zeilenweise (row) holen und ausgeben
while ($row = $res->fetch_assoc()) {
    printf("<a href=\"userentry.php?ID=%s\">%s</a> %s %s<br />\n", $row["ID"], $row["Nr"], $row["Name"
], $row["Vorname"]);
}
printf("<a href=\"userentry.php\">neuer Leser</a><br />\n");
?>
```

Aufräumen refactoring

- ID im formular als HIDDEN field.
- Tabellenname im userentry nur einmal definieren.
- Link auf userlist.php von userentry.
- die sql-Verbindung in einer Datei db.inc.sql machen und die inkludieren.

Formular V1.1 HIDDEN INPUT

Der Primary Key ID darf nicht verändert werden und soll nicht als input angezeigt werden, aber er muss im Formular mitgeschickt werden, da durch den ID der Eintrag identifizierbar ist. In HTML wird das durch "<INPUT TYPE=HIDDEN>".

Um das in unsere Formulardefinition unterzubringen ühren wir eine optionale zweite Dimension ein.

```
$fields = array(
        array("ID", "HIDDEN"),
        "Nr", "Name", "Vorname", "GebDatum", "Email", "Tel");
```

```php
<?php
// file: userentry.php
// version: 1.1a

require("lib.inc.php");

$mysqli = new mysqli("localhost", "", "", "test");

$fields = array(
array("ID", "HIDDEN"), "Nr", "Name", "Vorname", "GebDatum", "Email", "Tel");
// save if there is something
if (!empty($_GET["Name"])) {
    if (empty($_GET["ID"])) {
        $res = $mysqli->query(build_sql_insert("user", $fields, $_GET));
        if ($mysqli->errno) {
            printf("<div>%s</div>\n", $mysqli->error);
        }
        else {
            $_GET["ID"] = $mysqli->insert_id;
        }
    }
    else {
        $res = $mysqli->query(build_sql_update("user", $fields, $_GET));
        if ($mysqli->errno) {
            printf("<div>%s</div>\n", $mysqli->error);
        }
    }
}

if (!empty($_GET["ID"])) {
    // load from database
    $res = $mysqli->query("SELECT * FROM user WHERE ID=".$_GET["ID"]);
    $_GET = $res->fetch_array(MYSQLI_ASSOC);
}

// display_form
printf("%s", build_form($fields));
```

Das schaut dann so aus

Array
Nr
Name
Vorname
GebDatum
Email
Tel
Speichern

weil die Funktion build_form nichts davon weiß, und die build_sql auch nicht.

```
<?php
// file: lib.inc.php
// version: 0.5a

// functions

function build_form($fieldnames) {
// build a html form from fieldnames: array of strings
?>
<form>
<table>
<?php
foreach ($fieldnames as $name) {
if (! is_array($name)) {
        printf('<tr><td>%s</td><td><input name="%s" value="%s"></td></tr>', $name, $name, $_G
ET[$name]);
}
}
?>
<tr><td colspan="2" align="center"><input type="submit" value="Speichern"></td></tr>
</form>
<?php
}
```

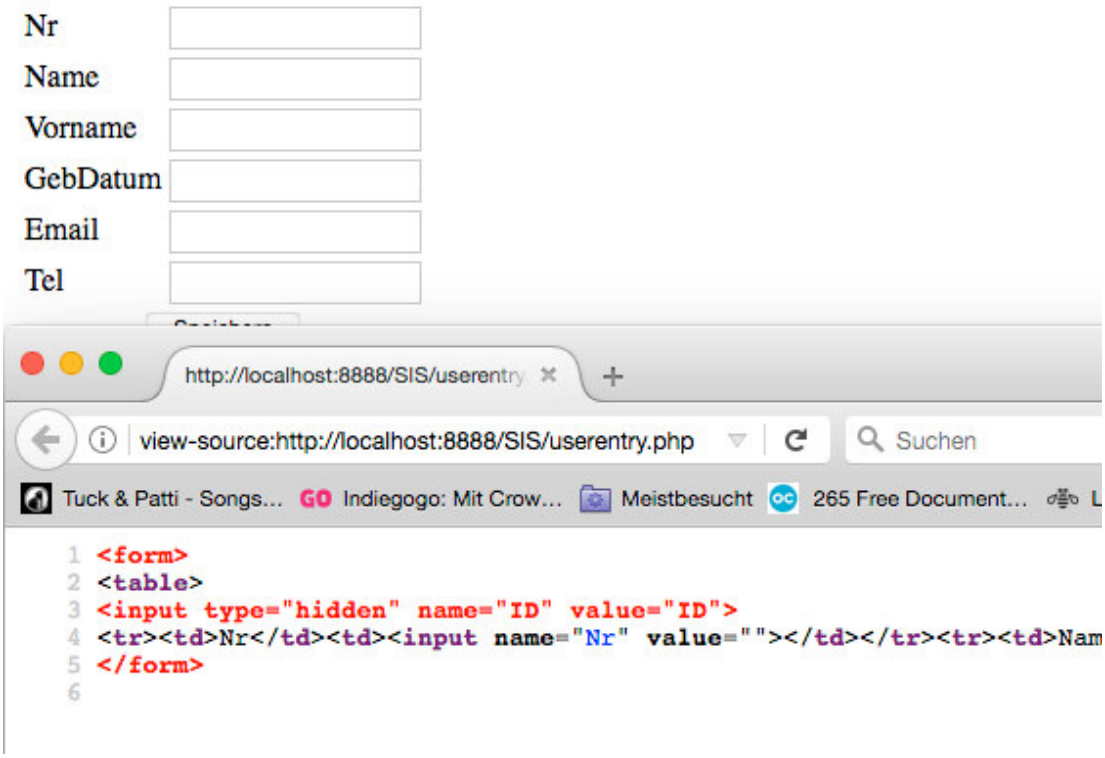
Dann wird ID nicht mehr von build_form ausgegeben.

```
<?php
// file: lib.inc.php
// version: 0.5b

// functions

function build_form($fieldnames) {
// build a html form from fieldnames: array of strings
?>
<form>
<table>
<?php
foreach ($fieldnames as $name) {
    if (! is_array($name)) {
        printf('<tr><td>%s</td><td><input name="%s" value="%s"></td></tr>', $name, $name, $_G
ET[$name]);
    }
    else {
        if ($name[1] == "HIDDEN") {
            printf("<input type=\"hidden\" name=\"%s\" value=\"%s\">\n", $name[0], $name[0],
$_GET[$name]);
        }
    }
}
?>
<tr><td colspan="2" align="center"><input type="submit" value="Speichern"></td></tr>
</form>
<?php
}
```

Jetzt sieht die Ausgabe so aus



Das Feld ID ist da einiges ist rot

1. "<form>" weil wir kein HTML-Dokument gemacht haben sondern nur das Formular ausgeben, es fehlt "<html>",

"<body>".

2. "<input type=hidden" in einem "table". Das heißt die zwei Strukturen form und table sind verschachtelt.

Wirklich falsch ist "value="ID" und dass alles auf einer Zeile steht.

```
<?php
// file: lib.inc.php
// version: 0.5c

// functions

function build_form($fieldnames) {
// build a html form from fieldnames: array of strings
?>
<form>
<table>
<?php
foreach ($fieldnames as $name) {
    if (! is_array($name)) {
        printf('<tr><td>%s</td><td><input name="%s" value="%s"></td></tr>'."\n", $name, $name
, $_GET[$name]);
    }
    else {
        if ($name[1] == "HIDDEN") {
            printf("<input type=\"hidden\" name=\"%s\" value=\"%s\">\n", $name[0], $_GET[$nam
e]);
        }
    }
}
?>
<tr><td colspan="2" align="center"><input type="submit" value="Speichern"></td></tr>
</form>
<?php
}
```

Zwei Probleme

1. Das newline muss in doppelten Anführungszeichen stehen, weil in einfachen die bachslash-expansion nicht durchgeführt wird.
2. wie hebt man hervor, dass etwas gelöscht wurde.

UND falls man Speichern klickt funktioniert es nicht, Error: "Unknown column 'Array' in 'field list'" siehe unten.

Zweites Eingabeformular oder zweiten Bericht wählen und von vorne.

Dabei ergeben sich Gemeinsamkeiten und Unterschiede der Formulare und daraus durch refactoring, sauberer Code und klarere Funktionalität.

Büchertabelle -- Korrektur: Medien Tabelle

Die Bücherei hat nicht nur Bücher sondern auch CDs, DVDs ... BTW wo klebt der Barcode am eBook ?
Eine schwierige Tabelle !

Weil sollen Bücher und CDs in eine Tabelle ? ... Ja ansonst muss man für jede neue Art von Media eine eigene Tabelle und

muss alle Seiten ändern.

Also eine Tabelle für die Typen

```
CREATE TABLE MediaType (
  ID int PRIMARY KEY auto_increment,
  name VARCHAR(32) NOT NULL
);

INSERT INTO MediaType VALUES(1, 'Buch');
INSERT INTO MediaType VALUES(2, 'CD');
INSERT INTO MediaType VALUES(3, 'DVD');
INSERT INTO MediaType VALUES(4, 'Blueray');
```

Frage: bekommt "INSERT INTO MediaType VALUES(NULL, 'eBook')" den ID 5 zugewiesen oder beginnt auto_increment bei 1 ?

Es bleibt schwierig : Bücher haben einen oder mehrere Autoren, Filme einen oder meherer Regisseure und Drehbuchautoren und Buchvorlagen ... hier fangen dann die NoSQL (idiotische Bezeichnung) Datenbanken an.

```
CREATE TABLE Media (
  ID int PRIMARY KEY auto_increment,
  MediaType_ID int NOT NULL,
  title VARCHAR(64),
  publishedYear DATE,
  EAN VARCHAR(16)
);
```

```
CREATE MediaAttribute (
  ID int PRIMARY KEY auto_increment,
  MediaAttributeType_ID int NOT NULL,
  Value VARCHAR(64)
);
```

```
CREATE MediaAttributeType (
  ID int PRIMARY KEY auto_increment,
  Value VARCHAR(64)
);
INSERT INTO MediaAttributeType VALUES(1, 'Autor');
INSERT INTO MediaAttributeType VALUES(2, 'Regisseur');
INSERT INTO MediaAttributeType VALUES(3, 'Verlag');
```

Zum Anfangen mit der zweiten Ausgabe "Medienauflistung" beschränken wir uns auf die table Media.
BUG: Vereinheitlichen der Tabellennamenschreibung mit Großbuchstaben "User".

Medienauflistung

Eine Tabelle mit

```
Nr  Jahr  Typ  Titel
```

Nr fehlt noch in der Tabelle, interne Nummer um das Medium im Bestand zu identifizieren. UNIQUE aber nicht PRIMARY KEY, darf NULL sein dann können neue Medien eingetragen werden bevor man eine Nr hat.

publishedYear brauch kein DATE, int reicht.

```
CREATE TABLE Media (
  ID int PRIMARY KEY auto_increment,
  MediaType_ID int NOT NULL,
  Nr VARCHAR(16) UNIQUE,
  title VARCHAR(64),
  publishedYear int,
  EAN VARCHAR(16)
);
```

```
INSERT INTO Media VALUES(NULL, 1, NULL, 'Invent to Learn', 2013, '9780997554328');
INSERT INTO Media VALUES(NULL, 1, NULL, 'Refactoring', 1999, '9780201485672');
```

BUG: Nr wird nachher in MediaReal (oder so) wandern, weil von einem Buch mehrere in der Bibliothek sein können und jedes eine andere Nr hat.

Wir nehmen userlist.php

```
<?php
// file: userlist.php
// version: 0.2

// connect to server and database (cmdline : mysql.exe -h localhost bib)
$mysqli = new mysqli("localhost", "", "", "test");
// hoffentlich ist der Zugriff erlaubt.
// Abfrage abschicken
$res = $mysqli->query("SELECT * FROM user");
// Spaltennamen
printf("Benutzer Nr Name<br />\n");

// Ergebinsdaten zeilenweise (row) holen und ausgeben
while ($row = $res->fetch_assoc()) {
    printf("<a href=\"userentry.php?ID=%s\">%s</a> %s %s<br />\n", $row["ID"], $row["Nr"], $r
ow["Name"], $row["Vorname"]);
}
printf("<a href=\"userentry.php\">neuer Leser</a><br />\n");
?>

(erstellt am <?php echo strftime("%Y-%m-%d"); ?> um <?php echo strftime("%H:%M"); ?>)
```

kopieren in medialist.php

```
<?php
// file: medialist.php
// version: 0.2

// connect to server and database (cmdline : mysql.exe -h localhost bib)
$mysqli = new mysqli("localhost", "", "", "test");
// hoffentlich ist der Zugriff erlaubt.
// Abfrage abschicken
$res = $mysqli->query("SELECT * FROM Media");
// Spaltennamen
printf("Nr  Jahr  Typ   Titel<br />\n");

// Ergebinsdaten zeilenweise (row) holen und ausgeben
while ($row = $res->fetch_assoc()) {
    printf("<a href=\"mediaentry.php?ID=%s\">%s</a> %s %s<br />\n", $row["ID"], $row["Nr"], $
row["publishedYear"], $row["title"]);
}
printf("<a href=\"mediaentry.php\">neues Medium</a><br />\n");
?>

(erstellt am <?php echo strftime("%Y-%m-%d"); ?> um <?php echo strftime("%H:%M"); ?>)
```

Da ändert sich nicht viel, kann man das extrahieren.

```php
<?php
// file: medialist.php
// version: 0.3

$tablename = "Media";
$editor = "mediaentry.php";
$editorNew = "neues Medium";
$fields = array("Nr", "publishedYear", "title");
$head = array("Nr", "Jahr", "Titel");

// connect to server and database (cmdline : mysql.exe -h localhost bib)
$mysqli = new mysqli("localhost", "", "", "test");
// hoffentlich ist der Zugriff erlaubt.
// Abfrage abschicken
$res = $mysqli->query("SELECT * FROM $tablename");

foreach ($head as $s) {
    printf("%s ", $s);
}
printf("<br />\n");

// Ergebinsdaten zeilenweise (row) holen und ausgeben
while ($row = $res->fetch_assoc()) {
printf("<a href=\"%s?ID=%s\">%s</a> ", $editor, $row["ID"], $row[$fields[0]]);
    for ($i=1; $i<count($fields); $i++) {
        printf("%s ", $row[$fields[$i]]);
    }
    printf("<br />\n");
}
printf("<a href=\"%s\">%s</a><br />\n", $editor, $editorNew);
?>

(erstellt am <?php echo strftime("%Y-%m-%d"); ?> um <?php echo strftime("%H:%M"); ?>)
```

BUG: "Nr" kann NULL sein, dann haben wir keinen Link !

Nr Jahr Typ Titel
2013 Invent to Learn
2013 Invent to Learn
1999 Refactoring
neues Medium
(erstellt am 2017-08-28 um 14:38)

Aber wir verschieben den unabhängigen Teil einmal in eine andere Datei ... tablelist.inc.php

```
<?php
// file: tablelist.php
// version: 0.1

// connect to server and database (cmdline : mysql.exe -h localhost bib)
$mysqli = new mysqli("localhost", "", "", "test");
// hoffentlich ist der Zugriff erlaubt.
// Abfrage abschicken
$res = $mysqli->query("SELECT * FROM $tablename");

foreach ($head as $s) {
    printf("%s ", $s);
}
printf("<br />\n");

// Ergebinsdaten zeilenweise (row) holen und ausgeben
while ($row = $res->fetch_assoc()) {
printf("<a href=\"%s?ID=%s\">%s</a> ", $editor, $row["ID"], $row[$fields[0]]);
    for ($i=1; $i<count($fields); $i++) {
        printf("%s ", $row[$fields[$i]]);
    }
    printf("<br />\n");
}
printf("<a href=\"%s\">%s</a><br />\n", $editor, $editorNew);
?>

(erstellt am <?php echo strftime("%Y-%m-%d"); ?> um <?php echo strftime("%H:%M"); ?>)
```

und verwenden diese in userlist.php

```
<?php
// file: userlist.php
// version: 0.3

$tablename = "user";
$editor = "userentry.php";
$editorNew = "neue/r BenutzerIn";
$fields = array("Nr", "Name", "Vorname");
$head = array("Benutzer", "Name", "Vorname");

require("tablelist.inc.php");
```

und jetzt behübschen

```
<?php
// file: tablelist.php
// version: 0.2

// connect to server and database (cmdline : mysql.exe -h localhost bib)
$mysqli = new mysqli("localhost", "", "", "test");
// hoffentlich ist der Zugriff erlaubt.
// Abfrage abschicken
$res = $mysqli->query("SELECT * FROM $tablename");

printf("<table border=\"1\">\n<tr>\n");
foreach ($head as $s) {
    printf("<th>%s</th>", $s);
}
printf("</tr>\n");

// Ergebinsdaten zeilenweise (row) holen und ausgeben
while ($row = $res->fetch_assoc()) {
printf("<tr>\n");
    printf("  <td><a href=\"%s?ID=%s\">%s</a></td>\n", $editor, $row["ID"], $row[$fields[0]])
;
    for ($i=1; $i<count($fields); $i++) {
        printf("  <td>%s</td>\n", $row[$fields[$i]]);
    }
printf("</tr>\n");
}
printf("<tr><td colspan=\"%d\"><a href=\"%s\">%s</a></td></tr>\n", count($fields), $editor, $
editorNew);
printf("</table>\n");
?>

(erstellt am <?php echo strftime("%Y-%m-%d"); ?> um <?php echo strftime("%H:%M"); ?>);
```

"td colspan" beduetet diese Spalte geht über mehrere Spalten der anderen Zeilen.

Und das verwenden wir jetzt auch bei

```
<?php
// file: medialist.php
// version: 0.4

$tablename = "Media";
$editor = "mediaentry.php";
$editorNew = "neues Medium";
$fields = array("Nr", "publishedYear", "title");
$head = array("Nr", "Jahr", "Titel");

require("tablelist.inc.php");
```

Nr	Jahr	Titel
	2013	Invent to Learn
	2013	Invent to Learn
	1999	Refactoring
neues Medium		

(erstellt am 2017-08-28 um 14:34)

Bei Nr brauchen wir noch ein klickbares Zeichen.

```
<?php
// file: tablelist.php
// version: 0.2

// connect to server and database (cmdline : mysql.exe -h localhost bib)
$mysqli = new mysqli("localhost", "", "", "test");
// hoffentlich ist der Zugriff erlaubt.
// Abfrage abschicken
$res = $mysqli->query("SELECT * FROM $tablename");

printf("<table border=\"1\">\n<tr>\n");
foreach ($head as $s) {
    printf("<th>%s</th>", $s);
}
printf("</tr>\n");

// Ergebinsdaten zeilenweise (row) holen und ausgeben
while ($row = $res->fetch_assoc()) {
    printf("<tr>\n");
    if (empty($row[$fields[0]])) {
        $row[$fields[0]] = "?";
    }
    printf("  <td><a href=\"%s?ID=%s\">%s</a></td>\n", $editor, $row["ID"], $row[$fields[0]])
;
    for ($i=1; $i<count($fields); $i++) {
        printf("  <td>%s</td>\n", $row[$fields[$i]]);
    }
    printf("</tr>\n");
}
printf("<tr><td colspan=\"%d\"><a href=\"%s\">%s</a></td></tr>\n", count($fields), $editor, $
editorNew);
printf("</table>\n");
?>

(erstellt am <?php echo strftime("%Y-%m-%d"); ?> um <?php echo strftime("%H:%M"); ?>)
```

Medieneingabe

Wir kopieren userentry.php und ändern

```php
<?php
// file: mediaentry.php
// version: 0.1

require("lib.inc.php");

$mysqli = new mysqli("localhost", "", "", "test");

$tablename = "Media";
$fields = array(        array("ID", "HIDDEN"), "MediaType_ID", "Nr", "title", "publishedYear"
, "EAN");

// save if there is something
if (!empty($_GET["Name"])) {
    if (empty($_GET["ID"])) {
        $res = $mysqli->query(build_sql_insert($tablename, $fields, $_GET));
        if ($mysqli->errno) {
            printf("<div>%s</div>\n", $mysqli->error);
        }
        else {
            $_GET["ID"] = $mysqli->insert_id;
        }
    }
    else {
        $res = $mysqli->query(build_sql_update($tablename, $fields, $_GET));
        if ($mysqli->errno) {
            printf("<div>%s</div>\n", $mysqli->error);
        }
    }
}

if (!empty($_GET["ID"])) {
    // load from database
    $res = $mysqli->query("SELECT * FROM $tablename WHERE ID=".$_GET["ID"]);
    $_GET = $res->fetch_array(MYSQLI_ASSOC);
}

// display_form
printf("%s", build_form($fields));
```

der Teil nach "save if there ..." ist ident zu dem in userentry.php

leider funktioniert er nur dort, weil in dieser Tabelle kein Feld "Name" vorhanden ist.

wenn wir den Teil verallgemeinern wollen brauchen wir ein Feld, dass immer vorhanden ist

"ID" kann nicht verwendet werden, weil es auch aus der Auflistung übergeben wird ... aber wir können ein weiteres HIDDEN-INPUT ins Formular geben (oder dem Knopf einen Namen)

```
<?php
// file: lib.inc.php
// version: 0.6

// functions

function build_form($fieldnames) {
// build a html form from fieldnames: array of strings
?>
<form>
<table>
    <input type="HIDDEN" name="cmd" value="save">
<?php
foreach ($fieldnames as $name) {
    if (! is_array($name)) {
        printf('<tr><td>%s</td><td><input name="%s" value="%s"></td></tr>'."\n", $name, $name
, $_GET[$name]);
    }
    else {
        if ($name[1] == "HIDDEN") {
            printf("<input type=\"hidden\" name=\"%s\" value=\"%s\">\n", $name[0], $_GET[$nam
e]);
        }
    }
}
?>
<tr><td colspan="2" align="center"><input type="submit" value="Speichern"></td></tr>
</form>
<?php
}
...
```

```php
<?php
// file: mediaentry.php
// version: 0.2

require("lib.inc.php");

$mysqli = new mysqli("localhost", "", "", "test");

$tablename = "Media";
$fields = array(
        array("ID", "HIDDEN"), "MediaType_ID", "Nr", "title", "publishedYear", "EAN");

// save if there is something
if (!empty($_GET["cmd"])) {
    if (empty($_GET["ID"])) {
        $res = $mysqli->query(build_sql_insert($tablename, $fields, $_GET));
        if ($mysqli->errno) {
            printf("<div>%s</div>\n", $mysqli->error);
        }
        else {
            $_GET["ID"] = $mysqli->insert_id;
        }
    }
    else {
        $res = $mysqli->query(build_sql_update($tablename, $fields, $_GET));
        if ($mysqli->errno) {
            printf("<div>%s</div>\n", $mysqli->error);
        }
    }
}

if (!empty($_GET["ID"])) {
    // load from database
    $res = $mysqli->query("SELECT * FROM $tablename WHERE ID=".$_GET["ID"]);
    $_GET = $res->fetch_array(MYSQLI_ASSOC);
}

// display_form
printf("%s", build_form($fields));
```

BUG: beim Speichern wird über dem Formular "Unknown column 'Array' in 'field list'" ausgegeben.

Debugging mit printf.

```
<?php
// file: mediaentry.php
// version: 0.3a

require("lib.inc.php");

$mysqli = new mysqli("localhost", "", "", "test");

$tablename = "Media";
$fields = array(
        array("ID", "HIDDEN"), "MediaType_ID", "Nr", "title", "publishedYear", "EAN");
echo "A";
// save if there is something
if (!empty($_GET["cmd"])) {
echo "B";
    if (empty($_GET["ID"])) {
echo "C";
        $res = $mysqli->query(build_sql_insert($tablename, $fields, $_GET));
        if ($mysqli->errno) {
            printf("<div>%s</div>\n", $mysqli->error);
        }
        else {
            $_GET["ID"] = $mysqli->insert_id;
        }
    }
    else {
echo "D";
        $res = $mysqli->query(build_sql_update($tablename, $fields, $_GET));
        if ($mysqli->errno) {
            printf("<div>%s</div>\n", $mysqli->error);
        }
    }
}

if (!empty($_GET["ID"])) {
echo "E";
    // load from database
    $res = $mysqli->query("SELECT * FROM $tablename WHERE ID=".$_GET["ID"]);
    $_GET = $res->fetch_array(MYSQLI_ASSOC);
}

// display_form
printf("%s", build_form($fields));
```

"Speichern" gibt "ABC" aus.
"A" ist klar, das Script läuft. "lib.inc.php" compiliert und die Datenbankverbindung funktioniert.

"B" bedeutet das Script wurde vom Formular aufgerufen "cmd" ist vorhanden.

"C" heißt einen neuen Eintrag anlegen ... das sollte nicht sein ... "ID" ist nicht gesetzt.

```
<?php
// file: lib.inc.php
// version: 0.7a

// functions

function build_form($fieldnames) {
// build a html form from fieldnames: array of strings
?>
<form>
<table>
    <input type="HIDDEN" name="cmd" value="save">
<?php
foreach ($fieldnames as $name) {
    if (! is_array($name)) {
        printf('<tr><td>%s</td><td><input name="%s" value="%s"></td></tr>'."\n", $name, $name
, $_GET[$name]);
    }
    else {
        if ($name[1] == "HIDDEN") {
            printf("<input type=\"hidden\" name=\"%s\" value=\"%s\">\n", $name[0], $_GET[$nam
e[0]]);
        }
    }
}
?>
<tr><td colspan="2" align="center"><input type="submit" value="Speichern"></td></tr>
</form>
<?php
}
```

Beim Prüfen ob der Eintrag in der fieldlist ein Array ist haben den Index beim Auslesen des Feldnamens vergessen.

Jetzt bekommen wir

```
ABD
```

Unknown column 'Array' in 'field list'
E
das heißt: es wird erkannt , dass es ein UPDATE sein soll, aber dann kommt wieder die selbe Meldung ... da wir schon ziemlich lange hier am Keyboard sitzen und denkmüde werden lassen wir uns das SQL-Statement ausgeben.

```
<?php
// file: mediaentry.php
// version: 0.3a

require("lib.inc.php");

$mysqli = new mysqli("localhost", "", "", "test");

$tablename = "Media";
$fields = array(
        array("ID", "HIDDEN"), "MediaType_ID", "Nr", "title", "publishedYear", "EAN");
 echo "A";
// save if there is something
if (!empty($_GET["cmd"])) {
echo "B";
    if (empty($_GET["ID"])) {
echo "C";
        $res = $mysqli->query(build_sql_insert($tablename, $fields, $_GET));
        if ($mysqli->errno) {
            printf("<div>%s</div>\n", $mysqli->error);
        }
        else {
            $_GET["ID"] = $mysqli->insert_id;
        }
    }
    else {
echo "D".build_sql_update($tablename, $fields, $_GET);
        $res = $mysqli->query(build_sql_update($tablename, $fields, $_GET));
        if ($mysqli->errno) {
            printf("<div>%s</div>\n", $mysqli->error);
        }
    }
}

if (!empty($_GET["ID"])) {
echo "E";
    // load from database
    $res = $mysqli->query("SELECT * FROM $tablename WHERE ID=".$_GET["ID"]);
    $_GET = $res->fetch_array(MYSQLI_ASSOC);
}

// display_form
printf("%s", build_form($fields));
```

Fehler

```
ABDUPDATE Media SET  Array='', MediaType_ID='1', Nr='', title='Invent to
Learn', publishedYear='2013', EAN='978-0-9975543-2' WHERE ID=1
```

Unknown column 'Array' in 'field list'
BUG: Wir haben weder in build_sql_insert noch in _update etwas geändert nachdem wir die fieldlist eine optionale zweite Dimension eingeführt haben.

```
<?php
// file: lib.inc.php
// version: 0.7a

// functions

function build_form($fieldnames) {
// build a html form from fieldnames: array of strings
?>
<form>
<table>
    <input type="HIDDEN" name="cmd" value="save">
<?php
foreach ($fieldnames as $name) {
    if (! is_array($name)) {
        printf('<tr><td>%s</td><td><input name="%s" value="%s"></td></tr>'."\n", $name, $name
, $_GET[$name]);
    }
    else {
        if ($name[1] == "HIDDEN") {
            printf("<input type=\"hidden\" name=\"%s\" value=\"%s\">\n", $name[0], $_GET[$nam
e[0]]);
        }
    }
}
?>
<tr><td colspan="2" align="center"><input type="submit" value="Speichern"></td></tr>
</form>
<?php
}

function build_sql_insert( $tablename, $fieldnames, $values )
// build sql insert statement.
// fieldnames: array of strings, values: dictionary indexed by fieldname
{
    $fs = "";
    $vs = "";
    foreach ($fieldnames as $f) {
        if (is_array($f)) {
            $f = $f[0];
        }
        $fs .= $f.",";
        $vs .= "'".$values[$f]."',";
    }
    $fs = substr($fs, 0, -1);
    $vs = substr($vs, 0, -1);
    return "INSERT INTO $tablename ($fs) VALUES($vs)";
}

function build_sql_update($tablename, $fieldnames, $values) {
// build an SQL UPDATE. Assume PRIMARY KEY is named ID.
    $sql = "UPDATE $tablename SET ";
    foreach ($fieldnames as $f) {
        if (is_array($f)) {
```

```
            $f = $f[0];
        }
        $sql .= " $f='".$values[$f]."',";
    }
    $sql = substr($sql, 0, -1);  // strip last semicolon
    $sql .= " WHERE ID=".$_GET["ID"];
    return $sql;
}
```

und wir probieren auch "Neues "medium" anlegen und das selbe bei userentry

UND: erkennen die Notwendigkeit automatisierter Tests.

Vorerst extrahieren wir den gemeinsamen Code aus user/mediaentry.php nach entry.inc.php

```
<?php
// file: entry.inc.php
// version: 0.1

// save if there is something
if (!empty($_GET["cmd"])) {
    if (empty($_GET["ID"])) {
        $res = $mysqli->query(build_sql_insert($tablename, $fields, $_GET));
        if ($mysqli->errno) {
            printf("<div>%s</div>\n", $mysqli->error);
        }
        else {
            $_GET["ID"] = $mysqli->insert_id;
        }
    }
    else {
        $res = $mysqli->query(build_sql_update($tablename, $fields, $_GET));
        if ($mysqli->errno) {
            printf("<div>%s</div>\n", $mysqli->error);
        }
    }
}

if (!empty($_GET["ID"])) {
    // load from database
    $res = $mysqli->query("SELECT * FROM $tablename WHERE ID=".$_GET["ID"]);
    $_GET = $res->fetch_array(MYSQLI_ASSOC);
}

// display_form
printf("%s", build_form($fields));
```

```
<?php
// file: mediaentry.php
// version: 0.4

require("lib.inc.php");

$mysqli = new mysqli("localhost", "", "", "test");

$tablename = "Media";
$fields = array(
        array("ID", "HIDDEN"), "MediaType_ID", "Nr", "title", "publishedYear", "EAN");

require("entry.inc.php");
```

```
<?php
// file: userentry.php
// version: 1.3

require("lib.inc.php");

$tablename = "user";
$mysqli = new mysqli("localhost", "", "", "test");

$fields = array(
        array("ID", "HIDDEN"), "Nr", "Name", "Vorname", "GebDatum", "Email", "Tel");

require("entry.inc.php");
```

und probieren es aus.

- userlist
- userentry neu
- userentry bearbeiten
- menulist
- menuentry neu
- menuentry bearbeiten

Testen

testen ist anders, aber testen ist sehr viel unittests, Funktionen aufrufen und ausprobieren.

phpunit herunterladen und ausführbar machen oder mit php aufrufen

```
php.exe phpunit.phar --bootstrap lib.inc.php tests/libTest
```

das funktioniert nicht weil test/libTest nicht existiert

```
<?php
// file: test_lib.php
// version: 0.1

use PHPUnit\Framework\TestCase;

final class libTest extends TestCase
{
    public function testBuildSqlInsert()
    {
        $this->assertEquals(
            'user@example.com',
            build_sql_insert("tbl", array("af", "bf"), array())
        );
    }
}
```

Das Ergebnis

```
PHPUnit 5.7.21 by Sebastian Bergmann and contributors.

E                                                                   1 / 1 (100%)

Time: 244 ms, Memory: 12.25MB

There was 1 error:

1) libTest::testBuildSqlInsert
Undefined index: af

/Users/engelbert/public_html/bibs/lib.inc.php:41
/Users/engelbert/public_html/bibs/tests/libTest.php:13

ERRORS!
Tests: 1, Assertions: 0, Errors: 1.
```

"undefined index: af" ist wenig überraschend, weil wir für Werte ein leeres array angegeben haben.

phpunit ist etwas "picky", diese Meldungen werden unterdrückt indem man ein "@" vor den Befehl setzt.

```
<?php
// file: lib.inc.php
// version: 0.8

// ...

function build_sql_insert( $tablename, $fieldnames, $values )
// build sql insert statement.
// fieldnames: array of strings, values: dictionary indexed by fieldname
{
    $fs = "";
    $vs = "";
    foreach ($fieldnames as $f) {
        if (is_array($f)) {
            $f = $f[0];
        }
        $fs .= $f.",";
        $vs .= "'".@$values[$f]."',";
    }
    $fs = substr($fs, 0, -1);
    $vs = substr($vs, 0, -1);
    return "INSERT INTO $tablename ($fs) VALUES($vs)";
}
```

jetzt bekommen wir

```
PHPUnit 5.7.21 by Sebastian Bergmann and contributors.

F                                                                   1 / 1 (100%)

Time: 250 ms, Memory: 12.25MB

There was 1 failure:

1) libTest::testBuildSqlInsert
Failed asserting that two strings are equal.
--- Expected
+++ Actual
@@ @@
-'user@example.com'
+'INSERT INTO tbl (af,bf) VALUES('','')'

/Users/engelbert/public_html/bibs/tests/libTest.php:14

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

auch wenig überraschend, wir haben ja einen Blödsinn "user@example.com" aus dem Beispiel übernommen, probieren kann man ja.

Korrigieren und zwei Tests dazu bauen.

```
<?php
// file: test_lib.php
// version: 0.2

use PHPUnit\Framework\TestCase;

final class libTest extends TestCase
{
    public function testBuildSqlInsert1()
    {
        $this->assertEquals(
            "INSERT INTO tbl (af,bf) VALUES('','')",
            build_sql_insert("tbl", array("af", "bf"), array())
        );
    }
    public function testBuildSqlInsert2()
    {
        $this->assertEquals(
            "INSERT INTO tbl (af,bf) VALUES('12','')",
            build_sql_insert("tbl", array("af", "bf"), array("af"=>12))
        );
    }
    public function testBuildSqlInsert3()
    {
        $this->assertEquals(
            "INSERT INTO tbl (af,bf) VALUES('14','help')",
            build_sql_insert("tbl", array(array("af", "HIDDEN"), "bf"),
                    array("af"=>14, "bf"=>"help"))
        );
    }
}
```

Weitere Tests wenn wir Änderungen machen.

Tabellenverbindungen Primary-Foreign-Key

Die Medieneingabe ist noch unvollständig, weil wir den Medientyp nicht aus der dazugehörigen Tabellen holen.

Und für eine Leihbibliothek brauchen wir einen Ausleihtabelle, die Bücher mit Benutzer verbindet

Und für eine Leihbibliothek muss angenommen werden, dass es von einem Buch mehr als ein Exemplar gibt.

Objektorientierte Implementierung

Warum objektorientiert ? Ein Eintrag in der Datenbank ist dann ein Objekt identifiziert über tablename-primaryKey und das Objekt weiß wie es gespeichert wird. In weiterer Folge kann ein Objekt auch verbundene Einträge lesen und sichern, ohne dass der aufrufende Code das wissen muss.

Das heißt

- Eine Class enthält den Tabellenname, die Feldliste, die Werte.
- Die Class hat eine Methode save die abhängig davon ob der PrimaryKey (ID) schon einen Wert hat oder nicht ein update oder ein insert ausführt.
- Die Class braucht eine Load Methode die per PrimaryKey lädt und eine um ein neues Objekt anzulegen und Werte einzufüllen.
- Ob die Datenbankverbindung in der Class steht oder eine globale (Singleton) Variable ist ?

- Eine Delete Methode
- Eine Abfragemethode, die eine Liste von Objekten zurückgibt ... ist ein interessantes Problem: Wo steht der Tabellenname ?