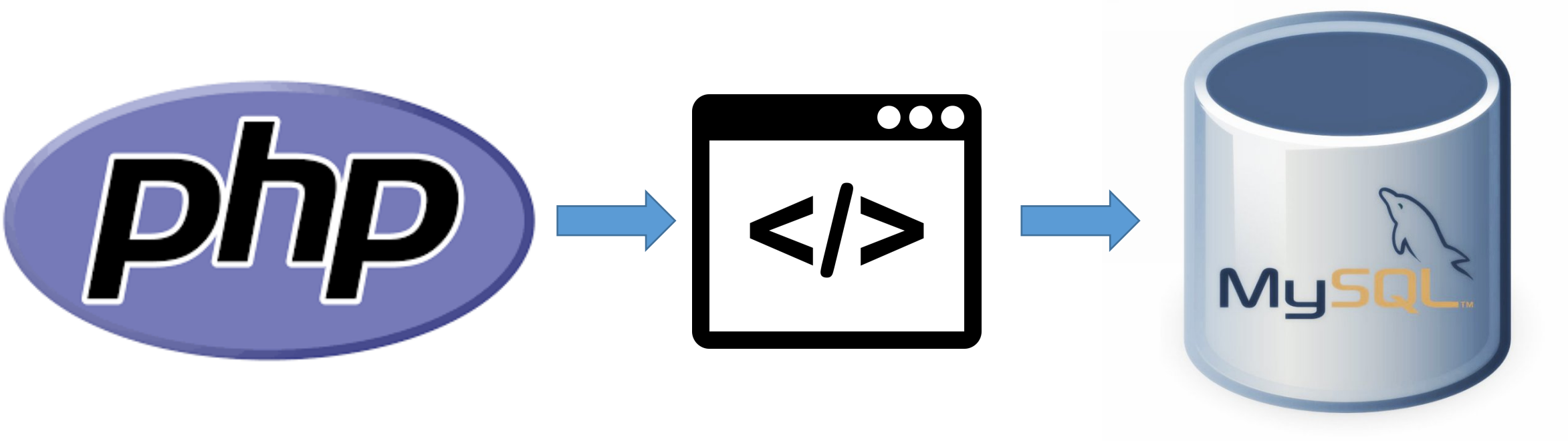
Datenbankanbindung

PHP mit einer Datenbank verbinden!



API

PHP‘s MySQL Extension

PHP‘s mysqli Extension

PHP Data Objects (PDO)

© GIBM, Daniel Brodbeck

API: Application Programming Interface

- **PHP‘s MySQL Extension**
  prozedural
  für MySQL Versionen älter 4.1.3
  ALT und BÖSE!!!
- **PHP‘s mysqli Extension**
  prozedural und objektorientiert
  für MySQL Versionen neuer 4.1.3
  für PHP-Versionen älter 5
- **PHP Data Objects (PDO)**
  ermöglicht einen einheitlichen Zugang von PHP auf unterschiedlichste SQL-basierte Datenbanken wie
  MySQL, PostgreSQL, SQLite
  ab PHP 5.1

© GIBM, Daniel Brodbeck

Vorteile mysqli vs. MySQL

+ **Objektorientierter Ansatz**
+ Unterstützt **Prepared Statements**
+ Es können **mehrere SQL-Statements gleichzeitig** ausgeführt werden
+ Unterstützt **Transaktionen**

Da wir hier mit MySQL-Datenbanken arbeiten, werden wir uns mit **mysqli** befassen. Sie dürfen aber **PDO** für Ihre Projekt-Arbeiten verwenden.

© GIBM, Daniel Brodbeck

mysqli-Objekt instanziieren, mit DB verbinden

```
$host = 'localhost';                    // host
$username = 'root';                     // username
$password = '';                         // password
$database = '133_Praktikum';            // database

// mit datenbank verbinden
$mysqli = new mysqli($host, $username, $password, $database);

//fehlermeldung, falls verbindung fehl schlägt.
if ($mysqli->connect_error) {
        die('Connect Error (' . $mysqli->connect_errno . ') '. $mysqli->
        connect_error);
}
```


© GIBM, Daniel Brodbeck

SQL-Injection, was ist das?

SQL-Injection ist eine Technik, mit der ein Angreifer versucht, bestehende Datenbank-Abfragen so zu manipuliere dass er

- versteckte Daten sichtbar machen
- bestehende Daten überschreiben
- gefährliche Kommandos auf Systemeben des Hosts ausführen

kann.

© GIBM, Daniel Brodbeck

SQL-Injection, wie funktioniert das?

Über ein Formular kann der User nach einem bestimmten Namen in der Datenbank suchen.

Die Eingabe des Users wird in der Variable `$_POST[‚firstname‘]` an das Script übergeben.

Wir erstellen eine Query mit folgendem Inhalt:

```
$query = „SELECT * FROM tbl_person where id_person = $_POST[‚id‘]“;
```

Der User kann nun seine Eingabe folgendermaßen gestalten und alle Personen aus der Datenbank auslesen:

```
1 or 1=1
```

Das Query lautet dann wie folgt:

```
SELECT * FROM tbl_person where id_person = 1 or 1=1
```


© GIBM, Daniel Brodbeck

Verhindern von SQL-Injection

Ohne **Prepared Statement** werden SQL Code und die Daten als String an den Datenbankserver gesendet, der Datenbankserver kann nicht unterscheiden, welche Teile zum SQL-Query gehören und welche Daten der Benutzer eingegeben hat.

```
„SELECT * FROM tbl_person where id_person = 1 or 1=1“
```

Beispiel:
http://praktikum.localhost/datenbankzugriff/injection/
http://praktikum.localhost/datenbankzugriff/injection/index_save.php

© GIBM, Daniel Brodbeck

Verhindern von SQL-Injection

Mit Prepared Statements wird zuerst der SQL-Code an den Server gesendet (`prepare`), das SQL-Statement kann nachträglich nicht mehr geändert werden.
Jeder variable Wert wird im Statement mit einem **?** gekennzeichnet.

Anschliessend werden die Benutzerdaten an den Datenbankserver gesendet und bei den ? in den SQL-Code eingebaut. (`bind_param`) Alle Benutzerdaten werden als Daten und nicht mehr als Code angesehen. Das ursprüngliche SQL-Statement kann nicht mehr verändert werden.

Zum Schluss wird das SQL-Statement ausgeführt (`execute`).

© GIBM, Daniel Brodbeck

Verhindern von SQL-Injection

```
// Script-Injection verhindern
$username = htmlspecialchars(trim($_POST[‚username‘]));
// SQL-Statement
$query = „SELECT * FROM tbl_person where id_person= ?“;
// Statement an den Server senden
$stmt = $mysqli->prepare($query);
// Benutzerdaten an Statememtn binden
$stmt->bind_param(„i“, $username);
// Statement ausführen
$stmt->execute();
```


© GIBM, Daniel Brodbeck

bind_param()

Für jedes Fragezeichen wir eine Wert oder eine Variable an das Statement gebunden.

```
$stmt->bind_params(„is“, $id, $username);
```

Jeder Wert wir mit einem Buchstaben deklariert:

- **s** steht für **s**tring
- **i** steht für **i**nteger
- **d** steht für **d**ouble
- **b** steht für **b**lob (**B**inary **L**arge **Ob**ject)

© GIBM, Daniel Brodbeck

Daten auslesen und ausgeben

Mit `$result = $stmt->get_result()` kann das Resultat der Abfrage gelesen werden.

Die Daten können dann folgendermassen ausgegeben werden:

```
while($row = $result->fetch_assoc()){
    echo $row['username'];
}
```


© GIBM, Daniel Brodbeck

Links

- MySQL, mysqli, PDO
  http://php.net/manual/de/mysqli.overview.php
- mysqli
  http://php.net/manual/de/book.mysqli.php
- SQL-Injection:
  http://www.w3schools.com/sql/sql_injection.asp
  http://php.net/manual/de/security.database.sql-injection.php
  http://www.unixwiz.net/techtips/sql-injection.html

© GIBM, Daniel Brodbeck

Anwenden

Lösen Sie nun die Aufgabe im Ordner:

- BSCW > 133 > 4_Aufgabe_Datenbank > 4_SQL-Injection

Zusätzliche Aufgaben zu Datenbanken finden Sie in den Ordnern:

- BSCW > 133 > 4_Aufgabe_Datenbank > 1_Einführung_Datenbankanbindung
- BSCW > 133 > 4_Aufgabe_Datenbank > 2_SQL-Test_Zugriff
- BSCW > 133 > 4_Aufgabe_Datenbank > 3_Unterschied_POST_GET
- BSCW > 133 > 4_Aufgabe_Datenbank > 5_Hilfestellung

© GIBM, Daniel Brodbeck