

Zugriff auf Datenbanken mit PHP Data Objects (PDO)

Verbindung aufbauen

Eine Verbindung zur Datenbank kann wie folgt aufgebaut werden:

```
$pdo = new PDO('mysql:host=localhost;dbname=test', 'username', 'password');
```

Einfache Abfragen

Nachdem eine Verbindung zur Datenbank hergestellt wurde, können SQL Statements wie folgt ausgeführt werden:



```
$sql = "SELECT vorname, name FROM doktoren";
$query_rows = $pdo->query($sql); // liefert ein assoziatives array zurück.
keys sind die felddnamen.

foreach ($query_rows as $row) {
    echo $row['vorname'] . " " . $row['name'] . "<br />";
}
```



(A1)

Importiere die

Tabellen der Schuldatenbank

in deine Übungsdatenbank.

Frage die folgenden Infos in deinem PHP Skript ab und gib sie in einer HTML-Tabelle aus:

- Erstelle eine Klassenliste der 7a
- Erstelle eine Liste aller Schüler, die Salvador Dali als Betreuer haben.
- Wieviele Schüler befinden sich in der Jahrgangsstufe 10?

Dynamische Abfragen

Mit über Formulare kann man nun auch Eingaben des Benutzers in Abfragen einbauen, auf diese

Weise werden die Abfragen dynamisch.



Grundregel der Webentwicklung: Vertraue keinem Datum, das dir ein Benutzer gibt. **Auf keinen Fall sollte man in einem produktiven System Benutzereingaben direkt in SQL Statements übernehmen.**¹⁾

Eine **schlechte Idee** ist es also, das naheliegende zu tun:

```
// id wird in einem Formular vom Benutzer erfragt
if(isset($_POST['id'])) {
    $id = $_POST['id'];
} else {
    die(" Es muss eine Datensatz ID angegeben werden!");
}

echo "Datensatz mit der ID $id: <br>";
$sql = "SELECT * FROM schueler WHERE id = $id";
$rows = $pdo->query($sql) ;
foreach ($rows as $row) {
    echo $row['id'] . " " . $row['vorname'] . " " . $row['nachname'] . "<br />";
}
```

Dies funktioniert zwar, ist aber anfällig für sogenannte SQL Injections. Ein Angreifer kann über den POST-Parameter die SQL-Abfrage manipulieren und weiteren SQL-Code einschleusen. Im schlimmsten Fall werden dadurch sensible Daten ausgegeben, Tabelle verändert oder gar ganze Tabellen gelöscht. Gibt der Anwender nämlich ins Formularfeld beispielsweise folgendes ein: 1 OR id > 1

Werden alle Datensätze ausgegeben, denn an die Datenbank wird die Abfrage SELECT * FROM schueler WHERE id = 1 OR id > 1 geschickt.



(Quelle: <https://xkcd.com/327/>, Lizenz [Creative Commons Attribution-NonCommercial 4.0 License](https://creativecommons.org/licenses/by-nc/4.0/)).

Prepared Statements

Um SQL-Injections zu unterbinden, sollte man prepared Statements verwenden. In dem Moment, in

dem ihr Daten von Benutzern an die Datenbank übergibt, solltet ihr stets auf prepared Statements zurückgreifen.

```
// id wird in einem Formular vom Benutzer erfragt
if(isset($_POST['id'])) {
    $id = $_POST['id'];
} else {
    die(" Es muss eine Datensatz ID angegeben werden!");
}

echo "Datensatz mit der ID $id: <br>";

$stmt = $pdo->prepare("SELECT * FROM schueler WHERE id = ?");
$stmt->execute(array($id));

foreach ($row = $stmt->fetch()) {
    echo $row['id'] . " " . $row['vorname'] . " " . $row['nachname'] . "<br />";
}
```

1)

<https://www.ionos.de/digitalguide/server/sicherheit/sql-injection-grundlagen-und-schutzmassnahmen/>

From:
<https://www.info-bw.de/> -

Permanent link:
<https://www.info-bw.de/faecher:informatik:oberstufe:datenbanken:projekt:dbphp:phppdo:start?rev=1606345728>

Last update: **25.11.2020 23:08**

