# **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 feldnamen.

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: Vetraue keinem Datum, das dir ein Benutzer gibt. Auf keinen Fall sollte man in einem produktiven System Benutzereingaben direkt in SQL Statements übernehmen.<sup>1)</sup>

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

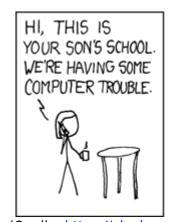
```
// id wird in einem Formular vom Benutzer erfragt
if(isset($_POST['id'])) {
        $id = $_POST['id'];
} else {
        die(" Es muss eine Datensatz ID angegeben werden!");
}

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

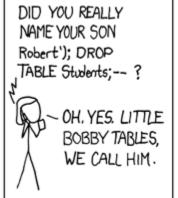
echo "Datensatz mit der ID $id: <br>";
$sql = "SELECT * FROM schueler WHERE id = $id";
$rows = $pdo->query($sql);
foreach ($rows as $row) {
        echo $row['id'] . " " . $row['vorname']." ".$row['nachname']."<br/>br />";
}
```

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 = OR id > 1 geschickt.









(Quelle: https://xkcd.com/327/, Lizenz Creative Commons Attribution-NonCommercial 2.5 License.

https://info-bw.de/ Printed on 05.08.2025 16:43

### **Prepared Statements**

Um SQL-Injections zu unterbinden, sollte man prepared Statements verwenden. In dem Moment, in dem ihr Daten von Benutzern an die Datenbank übergebt, 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/>
$statement = $pdo->prepare("SELECT * FROM schueler WHERE id = ?");
$statement->execute(array($id));

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



#### (A2)

Eine Vorlage für ein HTML Formular mit angebundenem PHP Skript findest du hier.

- Vollziehe die Beispiele oben nach, bestätige die SQL Injection.
- Erweitere das Formular so, dass man mehrere Parameter der Abfrage dynamisiseren kann. Verwende prepared statements.
- Mache Klassenlisten über ein Dropdown Formularfeld zugänglich.

1

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

From:

https://info-bw.de/ -

Permanent link:

https://info-bw.de/faecher:informatik:oberstufe:php:phppdo:start?rev=1619446269

Last update: 26.04.2021 14:11

