Prepared Statements - PDO

Um SQL-Injections zu verhindern empfiehlt sich der Einsatz von **prepared statements**. Sobald man irgendwelche Daten vom Benutzer an die Datenbank übergibt, sollte man stets auf prepared Statements zurückgreifen. Mittels prepared Statements ist man gegen SQL-Injections geschützt.

SQL-Injections:
Ein Angreifer kann über den GET-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.

Bei „prepared statements“ wird ein Platzhalter anstelle von dem aktuellen Parameter verwendet. Dieser Platzhalter wird durch den aktuellen Wert ersetzt, wenn das Statement ausgeführt (executed) wird.

Es besteht aus zwei Stufen, dem „prepare“ und dem „execute“.

Unterschied ob man MySQLi oder PDO benutzt:

MySQLi supports the use of anonymous positional placeholder (`?`), as shown below:

```
INSERT INTO persons (first_name, last_name, email) VALUES (?, ?, ?);
```

While, PDO supports both anonymous positional placeholder (`?`), as well as the named placeholders. A named placeholder begins with a colon (`:`) followed by an identifier, like this:

```
INSERT INTO persons (first_name, last_name, email)
VALUES (:first_name, :last_name, :email);
```

Bei PDO-Verbindung:
Neben anonymen Parametern, die man mittels *?* angibt, kann man auch benannte Parameter nutzen. Statt dem Fragezeichen schreibt man dann
:name
sprich, man beginnt mit einem Doppelpunkt und dann dem Namen. Ein Minus darf im Namen nicht enthalten sein, nutzt dort stattdessen den Unterstrich.

Die Benennung von Parametern ist besonders praktisch, wenn man mehr als nur einen Parameter in dem SQL-Query hat. So läuft man nicht Gefahr, aus Versehen die falschen Werte zu übergeben. Eure Parameternamen müssen nicht so heißen wie die Spaltennamen, auch wenn dies stark zu empfehlen ist.

Advantages of Using Prepared Statements

A prepared statement can execute the same statement repeatedly with high efficiency, because the statement is parsed only once again, while it can be executed multiple times. It also minimize bandwidth usage, since upon every execution only the placeholder values need to be transmitted to the database server instead of the complete SQL statement.

Prepared statements also provide strong protection against SQL injection, because parameter values are not embedded directly inside the SQL query string. The parameter values are sent to the database server separately from the query using a different protocol and thus cannot interfere with it. The server uses these values directly at the point of execution, after the statement template is parsed. That's why the prepared statements are less error-prone, and thus considered as one of the most critical element in database security.

The type definition string specify the data types of the corresponding bind variables and contains one or more of the following four characters:

- **b** — binary (such as image, PDF file, etc.)
- **d** — double (floating point number)
- **i** — integer (whole number)
- **s** — string (text)

The number of bind variables and the number of characters in type definition string must match the number of placeholders in the SQL statement template.

```
* s is for strings
* d is for decimals
* i is for integers
* b is for blob
```

https://www.tutorialrepublic.com/php-tutorial/php-mysql-prepared-statements.php

Beispiel – später im Webshop:

Dies soll um die Sicherheit zu erhöhen mit „prepaired statements“ erfolgen. Daher muss man Platzhalter (mit Doppelpunkt am Beginn) arbeiten. Das verhindert schädliche SQL-Injections.

Zuerst muss man aber eine neue SQL-Query erzeugen mit „INSERT INTO“.

Erstelle eine Variabel „$sql“ mit einem SQL-Befehl INSERT INTO

```
33      $productId = (int)$routeTeile[3];
34      $sql = "INSERT INTO korb SET
35          anzahl=1,
36          user_id = :userId,
37          product_id = :productId";
38  }
```

Man kann, wie hier, statt „value“ auch „set“ verwenden.

Für das „prepaired statement“ verwende folgenden Code:

```
35          user_id = :userId,
36          product_id = :productId";
37      $statement = getDB()->prepare($sql);
38
39      $statement->execute([]);
40  }
```

```
$statement->execute([
    ':userId'=> $userId,
    ':productId'=> $productId
]);
```

In Zeile 39 muss das Statement ausgeführt werden. Die Details folgen in den Klammern: dabei wird der Datenbank mitgeteilt, wodurch die Platzhalter ersetzt werden sollen, so z.B. der Platzhalter :userId mit der Variablen „$userId“.

```
39      $statement->execute([
40          ':userId'=> $userId,
41          ':productId'=> $productId
42      ]);
```

Gesamtes Beispiel:

```
1   <?php
2   function addProductToKorb(int $userId, int $productId){
3       $sql = "INSERT INTO korb SET
4           anzahl=1,
5           user_id = :userId,
6           product_id = :productId";
7       $statement = getDB()->prepare($sql);
8
9       $statement->execute([
10          ':userId'=> $userId,
11          ':productId'=> $productId
12      ]);
13  }
```

Beispiel mit MySQLi:

https://www.tutorialrepublic.com/php-tutorial/php-mysql-prepared-statements.php

```php
<?php

$vorname = $_POST["vorname"];
$nachname = $_POST["nachname"];
$geschlecht = $_POST["geschlecht"];
$klasse = $_POST["klasse"];
$geb = $_POST["geb"];
$bild = addslashes(file_get_contents($_FILES['bild']['tmp_name']));

require 'config/verbindung.php';

//prepare an insert statement
$sql = "INSERT INTO schueler (vorname, nachname, geschlecht, klasse,  geb, bild) VALUES (?,?,?,?,?,?)";
//soviele Fragezeichen wie Elemente

$stmt = mysqli_prepare($con, $sql);

//soviele Platzhalter s wie Elemente
mysqli_stmt_bind_param($stmt, "ssssss", $vorname, $nachname, $geschlecht, $klasse, $geb, $bild);

mysqli_stmt_execute($stmt);
```


```php
<?php

$vorname = $_POST["vorname"];
$nachname = $_POST["nachname"];
$geschlecht = $_POST["geschlecht"];
$klasse = $_POST["klasse"];
$geb = $_POST["geb"];
$bild = addslashes(file_get_contents($_FILES['bild']['tmp_name']));

require 'config/verbindung.php';

//prepare an insert statement
$sql = "INSERT INTO schueler (vorname, nachname, geschlecht, klasse,  geb, bild) VALUES (?,?,?,?,?,?)";
//soviele Fragezeichen wie Elemente

$stmt = mysqli_prepare($con, $sql);

//soviele Platzhalter s wie Elemente
mysqli_stmt_bind_param($stmt, "ssssss", $vorname, $nachname, $geschlecht, $klasse, $geb, $bild);

mysqli_stmt_execute($stmt);
```


https://www.php-einfach.de/mysql-tutorial/crashkurs-pdo/#Verbindung_aufbauen

Prepared Statements

Prepared Statement mit anonymen Parametern:

```php
<?php
$pdo = new PDO('mysql:host=localhost;dbname=databasename', 'username', 'password');

$statement = $pdo->prepare("SELECT * FROM users WHERE vorname = ? AND nachname = ?");
$statement->execute(array('Max', 'Mustermann'));
while($row = $statement->fetch()) {
   echo $row['vorname']." ".$row['nachname']."<br />";
   echo "E-Mail: ".$row['email']."<br /><br />";
}
?>
```

Prepared Statement mit benannten Parametern:

```php
<?php
$pdo = new PDO('mysql:host=localhost;dbname=databasename', 'username', 'password');

$statement = $pdo->prepare("SELECT * FROM users WHERE vorname = :vorname AND nachname = :nac
$statement->execute(array(':vorname' => 'Max', ':nachname' => 'Mustermann'));
while($row = $statement->fetch()) {
   echo $row['vorname']." ".$row['nachname']."<br />";
   echo "E-Mail: ".$row['email']."<br /><br />";
}
?>
```

Typischen „Holen“ der Daten
z.B. ‚mit einer while:
$row = $statement->fetch()