PDO – PHP Data Object

ERWEITERUNG ZU PHP

1

Erweiterung von PHP mit Datenbanken

- Datenzugriff auf jede beliebige Datenbank einheitlich gestaltet seit PHP 5.0
- Einheitliche Funktionen
- Funktionsbezeichnungen ähnlich wie in "reinem" PHP MySQLi
- Verbindungsaufbau muss durch NULL Zuweisung geschlossen werden – bzw. wird beim Beenden des Skripts beendet
- PDO erlaubt aber auch persistente Verbindungen – eine geöffnet Verbindung bleibt solange erhalten bis sie explizit geschlossen wird. Dies kann die Geschwindigkeit beim Zugriff erhöhen.
- Für MariaDB gibt es keine extra Verbindung, hier wird noch MySQL verwendet (Stand Oktober 2019)

2

Klassischer DB Zugriff

```
$server = "localhost";
$user = "root";
$pwd = "";
$db = "multiple_choice";

try {
    $con = new PDO('mysql:host='.$server.';dbname='.$db.';charset=utf8',
                   $user, $pwd);
    // Exception Handling "einschalten":
    $con->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch(Exception $e) {
    echo $e->getMessage();
}
```

3

Exceptions

- Error Nummer – Exception $e - $e->getCode():
  - 2002: Serververbindung fehlgeschlagen.
  - 1044: Benutzerberechtigung oder Benutzer nicht vorhanden.
  - 1045: Passwort fehlerhaft.
  - 1049: Datenbank nicht vorhanden.

4

Zugriff auf Tabellen

- Zur Verhinderung von SQL-Injection wird ausschließlich mit prepare-Statements gearbeitet.
- Vorteile:
  - Es werden keine Strings übermittelt, die gehackt werden können.
  - Bei einem prepare-Statement,
    - wird zuerst nur geprüft, ob die Aktion auf die Datenbank überhaupt möglich ist (z.B. Überprüfung, ob Tabellen und Attribute vorhanden sind).
    - erfolgt die Datenübermittlung (auch das Auslesen von Daten) erst in einem zweiten Schritt, dabei wird überprüft, ob die Anfrage vom gleichen Server stammt.

5

prepare Methode

- Über die Methode prepare, wird eine Anfrage auf die DB abgesetzt, ob das SQL-Statement überhaupt möglich ist.
- Dabei wird die zuvor erstellte Verbindungsvariable ($db) genutzt:
  - `$select = $db->prepare('select * from tabelle where attribut=?');`
  - Das Fragezeichen ist ein Platzhalter, da noch keine Daten übermittelt werden, ist die Datenbank nicht zum Lesen oder Schreiben geöffnet und nicht angreifbar.
- Alternative Schreibweise:
  - `$select = $db->prepare('select * from tabelle where attribut =:wert');`
  - Anstelle des Fragezeichens als Platzhalter wird hier ein textueller Platzhalter gesetzt. Dies erhöht die Lesbarkeit des Source-Codes.

6

bindParam

- Nach erfolgreicher Anfrage durch die prepare-Methode, können nun die Werte für den Platzhalter mittels bindParam übermittelt werden. Die Abfrage selbst wird immer noch nicht durchgeführt und die Datenbank ist nach wie vor geschützt.
- Platzhalter Fragezeichen:
  - $select->bindParam(1, $variable);
  - Es können mehrere Variable übermittelt werden (je nach Anzahl der Fragezeichen), daher muss bei 1 beginnend durchnummeriert werden.
- Platzhalter :wert:
  - $select->bindParam(:wert, $variable);

7

execute

- Nachdem die Werte für die Platzhalter erfolgreich übermittelt wurden, kann die Abfrage durchgeführt werden.
- execute nach Verwendung von bindParam
  - `$select->execute();`
- execute ohne bindParam
  - Es ist möglich, auf die Methode bindParam gänzlich zu verzichten und diese Aktion gleichzeitig mit execute auszuführen:
    - `$select->execute([$variable]);`
  - Anmerkung: laut PDO Manual sind beide Varianten gleich sicher.

8

Aufgabe

- Unter Verwendung der DB schule erstellen Sie folgende Abfrage:
  - Alle Personen und Jobzuordnung – Spaltenausgabe:
    - PERSON | JOB

9

Datensätze ausgeben

- FETCH Methode
  - Wandelt die ausgelesenen Daten in ein Array um.
    - platzbasierendes Array: PDO::FETCH_NUM
    - assoziatives Array: PDO::FETCH_ASSOC
    - platzbasierend und assoziatives Array (Standard): PDO::FETCH_BOTH
  - Anmerkung: Wird die Standard-Variante genommen, scheinen die Datensätze/Attribute doppelt vorhanden zu sein, da diese in beiden Varianten vorliegen! Daher Methode immer angeben, da sonst die Verwendung von FOREACH nicht möglich ist!
- Beispiel:
- `while($row = $select->fetch(PDO::FETCH_NUM)`
- `{`
  - `echo $row[0].'<br>';`
- `}`

10

real_escape_string mit PDO

- Um SQL-Injection durch Übergabe einer Variablen zu verhindern, ist der übergebene String zu überprüfen. Bei MySQLi war dies die Methode real_escape_string.
- Mit PDO ist dies nicht mehr nötig, sondern wird in den Methoden bindParam und/oder execute implizit durchgeführt.
- PDO bietet aber die Methode quote() an.
  - Beispiel:
  - $con->quote($vname)
    - laut PDO Manual nicht mehr im Zusammenhang mit prepare zu verwenden

11

META Data - Attributeigenschaften

getColumnMeta(ColumnPos)

- Mit dieser Methode können die Eigenschaften (z.B. Datentyp, Name) jedes einzelnen Attributs ausgelesen werden
- Nützlich ist in diesem Zusammenhang auch die Methode columnCount(), die Anzahl der gewählten Attribute aus dem Select-Statement zurückliefert.
- Die Ausgabe erfolgt am leichtesten mit der foreach-Schleife

Beispiel:

```
$ct = $stmt->columnCount();
$meta = array(); // leeres Array erstellen
for($i = 0; $i < $ct; $i++)
{
  $meta[] = $stmt->
        getColumnMeta($i);
}

Zugriff auf Attributnamen: $meta[$i]['name']
```

12

Quelle: http://php.net/manual/de/pdostatement.getcolumnmeta.php

Name	Value
native_type	The PHP native type used to represent the column value.
driver:decl_type	The SQL type used to represent the column value in the database. If the column in the result set is the result of a function, this value is not returned by **PDOStatement::getColumnMeta()**.
flags	Any flags set for this column.
name	The name of this column as returned by the database.
table	The name of this column's table as returned by the database.
len	The length of this column. Normally -1 for types other than floating point decimals.
precision	The numeric precision of this column. Normally 0 for types other than floating point decimals.
pdo_type	The type of this column as represented by the PDO::PARAM_* constants.

13

Sonstige Methoden

- Anzahl der selektierten Datensätze:
  - $stmt->rowCount();
- Letzte eingefügte ID
  - $con->lastInsertId();

14

PDO Exception Handling

- Hier wird anstelle von Exception PDOException geschrieben
- Beim Verbindungsaufbau zur DB muss auch die Verwendung von PDOException "eingeschaltet" werden:
  - $con->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
- Beispiel:
- Eine benutzerdefinierte SQLSTATE Exception (beginnt mit 02) wird geworfen:
- `if($e->getCode() == '02001')`
- `echo '<br>NR: '.$e->getCode().': '.$e->getMessage().' '.$e->getFile().' '.$e->getLine();`

15

Transaktionen

- PDO unterstützt Transaction Control Language.
  - PDO::beginTransaction()
  - PDO::commit()
  - PDO::rollBack()
- Wurde ein PDO::beginTransaction ausgeführt, wird ein rollback aber bei Abbruch eines Skripts immer automatisch ausgeführt!
- Probleme können bei MySQL auftreten, da hier ein implizites COMMIT verwendet wird und ein vollständiger ROLLBACK daher nicht möglich ist.

16

TCL

- beginTransaction()
- commit()
- Rollback(
- Voraussetzung für Exception-Handling: PDO::ERRMODE_EXCEPTION

```
try {

$connect>beginTransaction();
    $stmt = $pdo-
>prepare("INSERT INTO users
(name) VALUES (?)");
   $stmt->execute($name);
    $connect>commit();
} catch (Exception $e){
    $connect>rollback();
    throw $e;
}
```

17

- Quellen:
- https://phpdelusions.net/pdo#transactions

18

DB Zugriff persistent

- `$db = new PDO('mysql:host='.$server.';dbname='.$db, $user, $pwd, array(PDO::ATTR_PERSISTENT => true));`
- Hier muss später, wenn die Verbindung nicht mehr benötigt wird, diese explizit freigegeben werden:
  - $db = null;
- Anmerkung: wird PDO::ATTR_PERSISTENT erst nachträglich mit PDO::setAttribute() gesetzt, ist die Verbindung NICHT persistent!
- In Zusammenhang mit ODBC wird empfohlen auf persistente Verbindungen zu verzichten, da hier mehrere Module bedient werden müssen und es dadurch zu Fehlern kommt.

19