Probeprojekt: Adresslistenabfrage

Vertiefung Datenbank-Klasse

Zunächst machen wir uns noch ein paar Gedanken darüber, wie wir die Datenbank Klasse erweitern können, so dass diese nicht nur "Objekte mit Datenbankverbindung" instanziieren kann, sondern mit dieser Datenbankverbindung auch noch was anfangen kann, beispielsweise eine Abfrage ausführen.

Beispielsweise könnte man eine "Getter"-Methode implementieren, die das Datenbankhandle zurückliefert, so dass man dieses in anderen Klassen (die es noch nicht gibt) weiter verwenden könnte. Eventuell wäre aber auch Vererbung besser geeignet, das überlegen wir uns zu gegebener Zeit - schaden tuts ja erst man nichts? Also los...

Hilfreich ist vielleicht auch ein Attribut, das die abzufragende Tabelle(n) beinhaltet, ein passender Setter wäre da auch günstig.

```
class mysqldb {

    /**
     * Connect to db, set connection attribute to pdo object
     *
     * @param string        $dbusername   DB username
     * @param string        $dbpassword   DB password
     * @param string        $dbname       Database to connect to
     * @param string        $host         Database host to connect to
(optional)
     */
    function __construct($dbusername, $dbpassword, $dbname,
$host="localhost" ) {

    try {
        $pdo = new PDO("mysql:host=$host;dbname=$dbname", "$dbusername",
"$dbpassword");
        $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    } catch ( PDOException $e ) {
        echo 'Verbindung zur Datenbank fehlgeschlagen: ' . $e->getMessage();
        exit();
    }

    $this->connection = $pdo;

    }

    /**
    * Get pdo object for usage in other classes
    */
    function getConnection() {
```

```
        return $this->connection;
    }

    /**
    * Set attribute query table to given value
    *
    * @param string        $tables   Tables
    */
    function setQueryTable($tables) {
        $this->qtable = $tables;
    }

    /**
     * Make query, return array with results
     *
     * @param string        $searchnane  Name to search
     *
     * @return array        Query result
     */
    function searchName($searchname) {
        $statement= $this->connection->prepare("SELECT * FROM $this->qtable
WHERE Nachname LIKE :nachname");
        //$statement->execute(array($table,$searchname));
        $statement->execute(array('nachname' => "%$searchname%"));

        // Besonderheit, weil wir ein Array wollen - Kapselung!
        // $statement->fetch() liefert das Ergebnis der Abfrage zeilenweise
        // https://www.php.net/manual/de/pdostatement.fetch.php
        // Für uns ist es meist besser ein assoziatives Array
        // mit den Ergebnissen zurückzugeben das dann gerendert werden
        // kann
        // Leeres Array definieren
        $resultArray = array();
        while($row = $statement->fetch(PDO::FETCH_ASSOC)) {
            // Die Zeilen an das Array anhängen (Push)
            $resultArray[] = $row;
        }

        // For Debugging
        // print_r($resultArray);

        // Ergebnisarray zurückgeben
        return $resultArray;
    }
```

Aufgaben

(A1)

Erweitere die bisherige Arbeit am Plugin auf ein "Adresslistenausgabewerkzeug":

(1) Lege die fiktive Adressdatenbank zugrunde, importiere die Adresstabelle in deine mysql-DB um sie dann abzufragen.

(2) Ergänze dein Plugin um eine Konfigurations-Option "databasetable", damit du in der Konfiguration festlegen kannst, welche Tabelle abgefragt werden soll.

(3) Erstelle ein – zunächst einfaches – Abfrageformular, mit dem du z.B. anhand des Vor- und/oder Nachnamens nach der Adresse einer Person suchen kannst. Implementiere die Abfrage als Methode in der mysqldb-Klasse, die das DB-Handle und die Abfrageparameter erhält und ein Array mit dem Ergebnis der Abfrage zurückgibt. Dieses Array kann die render Funktion dann als hübsche Tabelle ausgeben.

(4) Nachdem die Grundfunktionalität sichergestellt ist, kannst du das Abfrageformular und die Programmlogik um weitere Funktionen erweitern. (Sortierung, weitere Felder, optionale Suche nach Teilstrings in Datenbankfeldern u.v.m.)

(A2)

Ergänze bei deinem Plugin ein Formular, mit dem ein neuer Datensatz an die Tabelle angefügt werden kann. Erweitere deine DB Klasse um eine entsprechende Methode. Hinweise zur Datenmanipulation mit Prepared Statements findest du z.B. hier.

(1) Welches der Formulare angezeigt werden soll, kann z.B. durch den Parameter nach dem > gesteuert werden:

```
{{projekt>abfrage}} // rendert und bearbeitet die Abfrage
{{projekt>eingabe}} // rendert und bearbeitet das Eingabeformular
```

(2) Beachte Sicherheitsaspekte und führe eine Eingabeüberprüfung durch, bevor du die Eingabedaten an deine Abfrage übergibst.

Du hängst fest? → Hilfestellung