PDO/DBAccess als neue Datenbankschnittstelle

Was ist PDO?

PHP Data Objects (PDO) ist eine Datenbankschnittstelle für PHP, genau wie die `mysql_*`-Funktionen und `mysqli`. PDO ist für Webanwendungen „State of the Art“, weil es die modernste DB-Schnittstelle ist, eine schöne und konsistente API bietet. PDO ist voll objektorientiert. PDO ist nicht auf MySQL-Datenbanken beschränkt, sondern dient als Schnittstelle zu beliebigen Datenbanken, wobei aber nicht auf datenbank-spezifische Features verzichtet werden muss.

Wie ist das Design von diesem objektorientierten PDO?

Es gibt zwei Klassen: `PDO` und `PDOStatement`. Ähnlich wie bei `mysqli` repräsentieren Instanzen der Klasse `PDO` eine Datenbankverbindung. Durch Methoden wie `query` und `exec` bzw. `prepare` können dann Datenbankabfragen abgesetzt bzw. vorbereitet (→ Prepared Statements) werden.
Objekte der Klasse `PDOStatement` werden von `query` und `prepare` zurückgegeben. Sie repräsentieren den „Result Set“ der Abfrage. Bei Prepared Statements müssen – bevor die Ergebnisse bereitstehen – natürlich noch die Parameter gesetzt und die Abfrage ausgeführt werden. Auch dafür hat PDOStatement entsprechende Methoden: `bindParam`, `bindValue`, `execute`.

Was ist DBAccess?

DBAccess ist eine eigene Klasse (von uns), die von PDO abgeleitet ist (OOP). Das heißt sie kann alles, was PDO kann aber noch mehr. Dabei ging es aber nicht darum viele Convencience Methoden hinzuzufügen! Denn auch DBAccess soll nicht mehr sein, als unsere grundlegende DB-Schnittstelle.

Was kann DBAccess mehr als PDO?

Zum einen fügt DBAccess eine Methode `count` hinzu. Damit lässt sich die Anzahl von Datensätzen bestimmen, die bestimmte Kriterien erfüllen.
Des Weiteren werden folgende Methoden „überladen“ um Variablen sicher in SQL-Statements einzusetzen. Damit wird SQL-Injection verhindert.[1]

1. `query` → `pquery`
2. `exec` → `pexec`
3. `count` → `pcount`

Das „p“ steht für „parameterized“ und bedeutet eben, dass die SQL-Statements in der Form `SELECT * FROM tbl WHERE id = ?` eingegeben werden. Die Variablen, die für die Fragezeichen (?) eingesetzt werden sollen, werden einfach als weitere Parameter übergeben.

Wo ist die Dokumentation zu DBAccess?

Die Dokumentation ist direkt im Quellcode DBAccess.php als PHPDoc. Diese Dokumentation bezieht sich häufig auf die Dokumentation von PDO: http://php.net/pdo

[1] SQL-Statements dürfen nie so aussehen: `'SELECT * FROM tbl WHERE id = ' . $id`

Darf ich der Klasse DBAcces noch eine Methode hinzufügen?

Nein. Die Idee ist, dass DBAccess wie PDO eine **grundlegende** DB-Schnittstelle ist und bleibt.[2]

Wie benutzt man DBAccess?

Ganz einfach! Beispiel:

```
function daysUntilPasswordExpires($id)
{
    $dbh = new DBAccess();
    $result = $dbh->pquery("SELECT
        (TO_DAYS(pw_expire) - TO_DAYS(NOW())) AS days
        FROM mitarbeiter WHERE id = ?", $id);
    return $result->fetchColumn();
}
```

Für weitere Beispiele kann man einfach im Internet nach PDO suchen.

Wie kann ich DBAccess in einer anderen Webanwendung verwenden?

Abgesehen vom entsprechenden PDO-Treiber (standardmäßig richtig installiert, siehe `phpinfo()`): Das einzige was DBAccess braucht um zu funktionieren sind die Logindaten zur Datenbank. Dazu muss ganz oben beim `include` der Pfad angepasst werden.[3] In der `include`-PHP-Datei müssen folgende Konstanten definiert sein: `DB_HOST`, `DB_NAME`, `DB_USER`, `DB_PASSWORD` und zu guter Letzt:

```
define('PDO_DSN', 'mysql:host='.DB_HOST.';dbname='.DB_NAME);
```

[2] Na gut, wenn du z.B. eine **super API** anzubieten hättest um **ganz allgemein und datenbankunabhängig** Stored Procedures/Functions aufzurufen, dann kann man vielleicht darüber reden.

[3] Es ist am sichersten, die PHP-Datei mit den Logindaten außerhalb des Document Root zu platzieren.