PHP
Datenbankzugriff

Verschiedene Arten des Datenbankzugriffs

- PHP bietet fertige Schnittstellen zu vielen Datenbanken:
  - MySQL: kompliziertes Lizensierungsmodell mit der Alternative MariaDB
  - PostgreSQL: kostenlos unter BSD-Lizenz
  - SQLite: Open Source
  - viele kommerzielle Datenbanken wie von IBM, MS oder SAP

- Datenbankzugriff via MySQL-Extension:
  - dies ist die historisch älteste Schnittstelle
  - sie ist rein prozedural und verwendet keine Objekte
  - sie wird empfohlen für alte MySQL-Versionen (<4.1.3)
  - die Erweiterung wurde in PHP 5.5.0 als veraltet markiert und in PHP 7.0.0 komplett entfernt
  - aus diesem Grund wird diese Erweiterung hier nicht genauer betrachtet

Datenbankzugriff via MySQLi-Extension

- dieser Zugriff wird oft auch als MySQL improved (da MySQLi) bezeichnet
- sie bietet eine prozedurale und objektorientierte Schnittstelle
- für neuere MySQL-Versionen (>= 4.1.3) empfohlen
- viele Verbesserungen:
  - Prepared Statements
  - Multiple Statements
  - Transaktionen
  - verbessertes Debugging
  - uvm.

Datenbankzugriff via MySQLi-Extension

- Verwendung von MySQLi:
  - Verbindung zur Datenbank aufbauen:
    - `$con=new MySQLi($host, $user, $pwd, $dbName);`
    - `$con->set_charset("UTF-8");`
  - SQL-Statement vorbereiten:
    - `$pstm = $con->prepare(...);`
  - Parameter und Rückgabevariablen binden:
    - `$pstm->bind_param("s", $name);`
    - `$pstm->bind_result($id, $nachname, $vorname);`
  - Statement ausführen:
    - `$pstm->execute();`

Datenbankzugriff via MySQLi-Extension

- Verwendung von MySQLi:
  - Über die Ergebnisliste iterieren:
    - `while ($pstm->fetch()) { ... }`
  - Statement schließen:
    - `$pstm->close();`
  - Verbindung schließen:
    - `$con->close();`

Datenbankzugriff via MySQLi-Extension

- Lesendes Beispiel:

```
// Verbindung zur Datenbank herstellen
$con = new MySQLi("localhost", "user", "password", "database");

// Zeichensatz auf UTF-8 setzen
$con->set_charset("UTF-8");

// SQL-Statement vorbereiten
$pstm = $con->prepare("SELECT id, nachname, vorname FROM studenten WHERE nachname = ?");

// Parameter binden
$name = "Meier";
$pstm->bind_param("s", $name); // "s" steht für exakt einen String

// Ergebnisvariablen binden und Statement ausführen
$pstm->bind_result($id, $nachname, $vorname);
$pstm->execute();
```

Datenbankzugriff via MySQLi-Extension

- Lesendes Beispiel:

```
// Über die Ergebnisliste iterieren und Daten ausgeben
while ($pstm->fetch()) {
        echo "$id, $nachname, $vorname"; echo "<br>";
}

// Statement schließen
$pstm->close();

// Verbindung schließen
$con->close();
```

Datenbankzugriff via MySQLi-Extension

- Schreibendes Beispiel:

```
// Verbindung zur Datenbank herstellen und Zeichensatz auf UTF-8 setzen
$con = new MySQLi("localhost", "ichbinphp", "php", "deinedatenbank");
$con->set_charset("UTF-8");

// SQL-Statement vorbereiten
$pstm = $con->prepare("INSERT INTO eingaben (user, pass) VALUES (?, ?)");

// Parameter binden und Statement ausführen
$pstm->bind_param("ss", $benutzername, $kennwort);
$pstm->execute();

// Statement und Verbindung schließen
$pstm->close();
$con->close();
```

Datenbankzugriff via PDO-Klassen

- PHP Data Objects (PDO):
  - abstrahieren vollständig von der Datenbank
  - bilden eine einheitliche Schnittstelle unabhängig von der Datenbank
  - erlauben den Austausch der Datenbank mit geringem Aufwand
  - schränken die verfügbare Funktionalität durch kleinsten gemeinsamen Nenner ein

Datenbankzugriff via PDO-Klassen

- PDO Klassen:
  - PDO
    - Verwaltung und Aufbau von Verbindungen
    - Verbindungskonfiguration (z.B. SSL)
  - PDOStatement
    - Verwaltung von Abfragen (prepared statements)
    - Verknüpfung von Variablen mit Abfragen
    - verwaltet die Ergebnisse einer Anfrage
    - muss nach Verwendung wieder freigegeben werden

Datenbankzugriff via PDO-Klassen

- Verwendung von PDO:
  - Verbindung zur Datenbank aufbauen:
    - `$dbh=new PDO("mysql:host=$host;dbname=$dbName", $user,$pwd);`
  - SQL-Statement vorbereiten:
    - `$stmt = $dbh->prepare(...);`
  - Parameter setzen:
    - `$stmt->bindValue(1, "Meier");`
  - Statement ausführen:
    - `$stmt->execute();`

Datenbankzugriff via PDO-Klassen

- Verwendung von PDO:
  - Über die Ergebnisliste iterieren:
    - `while ($row = $stmt->fetch()) { ... }`
  - Statement schließen:
    - `$stmt = null;`
  - Verbindung schließen:
    - `$dbh = null;`

Datenbankzugriff via PDO-Klassen

- Lesendes Beispiel:

```
// Verbindung zur Datenbank herstellen
$dbh = new PDO('mysql:host=localhost;dbname=database‘, 'user', 'password‘);
// SQL-Statement vorbereiten
$stmt = $dbh->prepare("SELECT id, nachname, vorname FROM studenten WHERE nachname = ?");
// Parameter binden
$stmt->bindValue(1, "Meier");
// Statement ausführen
$stmt->execute();
// Über die Ergebnisliste iterieren und Daten ausgeben
while ($row = $stmt->fetch()) {
        echo $row["id"] . " " . $row["nachname"] . " ";
        echo $row["vorname"] . "<br>";
}
// Statement und Verbindung schließen
$stmt = null; $dbh = null;
```

Datenbankzugriff via PDO-Klassen

- Schreibendes Beispiel:

```
// Verbindung zur Datenbank herstellen
$dbh = new PDO('mysql:host=localhost;dbname=deinedatenbank', 'ichbinphp', 'php');
$dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

// SQL-Statement vorbereiten
$stmt = $dbh->prepare('INSERT INTO eingaben (user, pass) VALUES (:user, :pass)');

// Parameter binden und Statement ausführen
$stmt->bindParam(':user', $benutzername);
$stmt->bindParam(':pass', $kennwort);
$stmt->execute();
```