

PHP und MySQL

PHP bietet eingebaute Funktionen, die es erlauben, mit einer MySQL-Datenbank Daten auszutauschen. Der Inhalt einer PHP-Seite kann damit abhängig vom Inhalt der Datenbank sein, man kann auch den Inhalt der Datenbank aufgrund von Benutzereingaben ändern.

Ein Schema für die Datenbankkommunikation

Der folgende Programmausschnitt ist ein Muster, wie die Kommunikation mit einer MySQL-Datenbank abläuft (die doppelten Schrägstriche und die Zahlen sind Kommentare ohne Bedeutung für das Programm):

```
$conn = mysql_connect('localhost','root','')
        or die ('keine Verbindungs zum Datenbankserver');    // (1)

mysql_select_db('schueler',$conn)
        or die ('Fehler beim Auswählen einer Datenbank');    // (2)

$sql = 'SELECT * FROM schueler WHERE SKLasse = 9';           // (3)

$ergebnis = mysql_query($sql,$conn)
        or die('Fehler bei der Abfrage: '.mysql_error());    // (4)

while ($zeile = mysql_fetch_row($ergebnis))                 // (5)
{
    echo $zeile[0]. '<br>';                                   // (6)
}
```

Wir schauen uns die einzelnen Teile genauer an:

1. `mysql_connect('Rechneradresse','Benutzername','Passwort')`:

Die Funktion stellt die Verbindung der Seite mit dem Server her, auf dem die MySQL-Datenbank läuft. Als Rückgabewert bekommt man ein Objekt, das man zur Kommunikation mit der Datenbank verwenden kann.

Wenn Webserver (Apache) und Datenbankserver auf demselben Rechner laufen (was bei xampp häufig der Fall ist), kann man wie im Beispiel 'localhost' als Rechneradresse angeben, andernfalls gehört hier eine IP-Adresse oder ein Hostname hin.

Benutzername und Passwort müssen in der Datenbank definiert worden sein. Im Beispiel verwenden wir den Benutzer `root`, der alle Rechte besitzt und kein Passwort braucht. Da dies ein potentielles Sicherheitsproblem ist, sollte man bei einer öffentlichen Seite einen Benutzer mit genau den Rechten verwenden, die er braucht, um seine Aufgaben zu erfüllen.

Aus verschiedenen Gründen kann es passieren, dass die Verbindung zur Datenbank nicht klappt: Der Rechnername kann falsch geschrieben sein, es kann sein, dass die Datenbank gerade nicht läuft oder die Verbindung unterbrochen ist, Benutzername und Passwort können falsch sein usw. Für diesen Fall gibt die Funktion `mysql_connect` den Wert `false` zurück. Da der erste Befehl ein `or`-Term ist, wird der zweite Teil (die Funktion `die`) nur dann ausgewertet, wenn der

erste `false` ist. Die Funktion die sorgt dafür, dass die Ausführung des PHP-Programms mit einer Fehlermeldung beendet wird.

2. `mysql_select_db('Datenbankname', Verbindung) :`

Auf einem Datenbankserver liegen üblicherweise mehrere Datenbanken. Man muss dem Verbindungsobjekt sagen, welche Datenbank man verwenden will. Der Rückgabewert ist `true` oder `false`. Für den Fall, dass `false` zurückgegeben wurde, sollte man das Programm mit `die` abbrechen.

3. Hier wird eine SQL-Abfrage formuliert und die Zeichenkette wird in einer Variablen gespeichert. Die SQL-Abfrage kann natürlich auch von einer anderen Variable abhängig gemacht werden, dadurch wird die Seite dynamisch:

```
$Nr = $_GET[ 'Nummer' ] ;
```

```
$sql = "SELECT * FROM schueler WHERE SNr = $Nr" ;
```

Wichtig: Hier doppelte Anführungszeichen verwenden. Außerdem sollte man zu Beginn des Programms prüfen, ob der Parameter in der URL gesetzt wurde, andernfalls kann es zu Fehlern kommen.

4. `mysql_query(sql-Abfrage, Verbindung) :`

Stellt die Anfrage an die Datenbank. Wenn die Anfrage korrekt und erfolgreich war, bekommt man als Rückgabewert die Ergebnisdatensätze (im Fall einer `SELECT`-Abfrage). Im Fehlerfall kommt `false` zurück, man kann also auch hier mit `die` das Programm beenden oder den Fehler irgendwie anders abfangen.

Die Funktion `mysql_error()` liefert die letzte Fehlermeldung der Datenbank als Zeichenkette zurück. Man kann damit also herausfinden, warum eine Anfrage gescheitert ist.

5. `mysql_fetch_row(Datensätze) :`

Diese Funktion holt den nächsten Datensatz aus der gegebenen Datensatzliste. Man kann sich das so vorstellen. `mysql_query(sql-Abfrage, Verbindung)` liefert die komplette Tabelle zurück. Ich kann aber nur immer zeilenweise auf die Daten zugreifen. Deswegen benötigen wir die `while`-Schleife. So lange ich noch nicht am Ende meiner Datensätze angekommen bin, springe ich bei jedem Schleifendurchlauf zum nächsten Datensatz und speichere diesen in dem Array `$zeile`.

6. Mit `$zeile[i]` kann ich dann auf die `i`-te Spalte meiner Tabelle zugreifen. Achtung: Nummerierung beginnt bei 0.