Einführung: Datenbankanbindung in PHP

PHP + Datenbank: Das Dream-Team

PHP hilft dabei, die Webseite zu bauen und mit dem Benutzer zu kommunizieren. SQL-Datenbanken speichern alle Daten der Website, wie zum Beispiel Benutzerinformationen oder Produktdetails. Wenn jemand eine Website besucht, arbeitet PHP mit der SQL-Datenbank zusammen, um die benötigten Informationen schnell anzuzeigen, wie z.B. die Liste der Produkte in einem Online-Shop.

In unserem ersten Beispiel haben wir eine SQL-Datenbank namens *beach.sqlite*, die in der Tabelle *produkte* Badezubehör enthält:

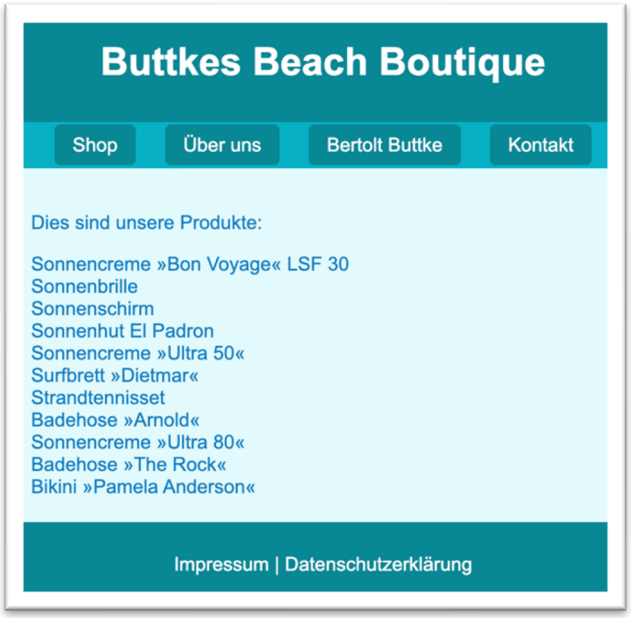
	id	bezeichnung	verkaufspreis	herstellungsland
1	1	Sonnencreme »Bon Voyage« LSF 30	6.95	F
2	2	Sonnenbrille	11.95	D
3	3	Sonnenschirm	29.95	CN
4	4	Sonnenhut El Padron	4.95	ES
5	5	Sonnencreme »Ultra 50«	9.95	F
6	6	Surfbrett »Dietmar«	14.95	CN
7	7	Strandtennisset	3.49	CN
8	8	Badehose »Arnold«	9.5	NULL
9	9	Sonnencreme »Ultra 80«	18.95	F
10	10	Badehose »The Rock«	9.9	F
11	11	Bikini »Pamela Anderson«	24.95	CN

Aufgabe:
Öffnen Sie die Datenbank *beach.sqlite* mit SQLiteStudio und wärmen Sie Ihre Finger mit ein paar SQL-Befehlen auf:

1. Lassen Sie sich alle Artikel ausgeben mit `SELECT * FROM produkte p;`
2. Lassen Sie sich alle Artikel ausgeben, die mehr als 10 Euro kosten.
3. Lassen Sie sich nur die Bezeichnung aller Artikel ausgeben. Sortieren Sie die Liste alphabetisch.
4. Lassen Sie sich alle Artikel ausgeben, die in Spanien oder Frankreich hergestellt wurden.
5. PRO: Was ist der höchste und der niedrigste Verkaufspreis?
6. PRO: Wie viele Artikel sind in der Tabelle?
7. SUPER-PRO: Wie viele Artikel aus den einzelnen Herstellungsländern gibt es? (Tipp: Gruppieren Sie nach Land.)

Diese Tabelle wollen wir von einer Website aus ansteuern – wir wollen die SQL-Befehle, die wir gerade ins SQLiteStudio eingegeben haben, via PHP ausführen. Und das ist einfacher als gedacht.

Unsere Beispielwebseite sieht so aus:

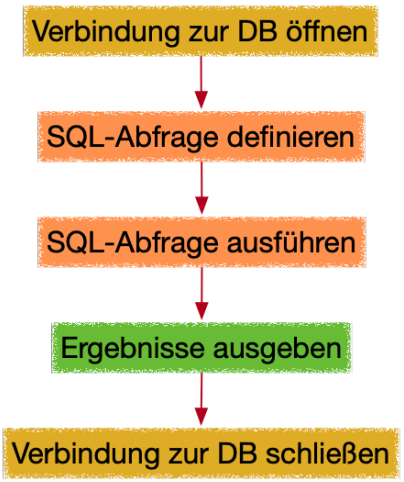


Hier der Code, der die Datenbank ansteuert:

```php
<?php

// Verbindung zur SQLite-Datenbank herstellen
$verbindung = new PDO('sqlite:beach.sqlite'); 1

// SQL-Abfrage definieren
$abfrage = "SELECT * FROM produkte p;"; 2

// SQL-Abfrage ausführen
$ergebnis = $verbindung->query($abfrage); 3

// Ergebnisse durchlaufen
foreach ($ergebnis as $zeile) {
    $bezeichnung = $zeile['bezeichnung']; 4
    echo $bezeichnung . "<br>";

    /* oder mit print_r($zeile) alle Daten des Arrays $zeile
    ausgeben … oder etwas versuchen wie
    if($zeile['verkaufspreis'] > 10) { echo " - uih, teuer …"; } */
}

// Datenbankverbindung schließen
$verbindung = null; 5
?>
```

Nicht so kompliziert, oder?

Schritte der Datenbankabfrage




```php
// Verbindung zur SQLite-Datenbank herstellen
$verbindung = new PDO('sqlite:beach.sqlite');

// SQL-Abfrage definieren
$abfrage = "SELECT * FROM produkte p;";

// SQL-Abfrage ausführen
$ergebnis = $verbindung->query($abfrage);

// Ergebnisse durchlaufen
foreach ($ergebnis as $zeile) {
    $bezeichnung = $zeile['bezeichnung'];
    echo $bezeichnung . "<br>";
}

// Datenbankverbindung schließen
$verbindung = null;
```

(1) Verbindung zur Datenbank herstellen

```php
// Verbindung zur SQLite-Datenbank herstellen
$verbindung = new PDO('sqlite:beach.sqlite'); 1
```

Technisch gesprochen erzeugen wir ein neues Objekt der Klasse PDO (siehe unten) und geben diesem Objekt die Information, wie die Datenbank heißt (der Parameter ist ein String, also Anführungszeichen nicht vergessen). Das Objekt bekommt den Namen **`$verbindung`**.

*Beachte, dass der Name des Objekts (im Beispiel oben: **`$verbindung`**) frei wählbar ist.*

PRO-Wissen: Das Schöne an SQLite ist, dass wir keine Userdaten brauchen. In MySQL wäre der einfachste vorstellbare Code:
`$verbindung = new PDO('mysql:host=localhost;dbname=db1', 'username', 'passwort');`
Dazu muss dann ein Server laufen (z.B. xampp) und die Datenbank eingespielt sein. In SQLite brauchen wir einfach nur die Datei.

(2) SQL-Abfrage definieren

```php
// SQL-Abfrage definieren
$abfrage = "SELECT * FROM produkte p;"; 2
```

Wir definieren eine SQL-Abfrage als String und speichern diesen in einer Variablen – im Beispiel **`$abfrage`**.

Beachte, dass wir diesen Schritt weglassen könnten und den String direkt in (3) schreiben könnten. Das ist aber sehr fehleranfällig, vor allem wenn wir kompliziertere Abfragen bauen. Deshalb werden wir Schritt (2) IMMER schön einzeln ausführen.

(3) SQL-Abfrage ausführen

```php
// SQL-Abfrage ausführen
$ergebnis = $verbindung->query($abfrage); 3
```

Wir schnappen uns das PDO-Objekt aus Schritt (1) (**`$verbindung`**) und wenden auf dieses Objekt die Methode **`query(…)`** an. Das Ergebnis speichern wir in einer Variablen **`$ergebnis`** (die könnte natürlich, auch anders heißen, zum Beispiel **`$alfons`** - das wäre aber nicht so gut verständlich).

*Beachte: Was in Java der Punkt macht (**`this.getWorld().getWidth()`**), ist in PHP der Pfeil **`->`** (**`$this->getWorld()->getWidth()`**)*

*PRO-Wissen: Nachdem wir eine Verbindung zur Datenbank hergestellt haben, verhält sich PDO für alle Datenbanken gleich. Gleich ob MySQL, SQLite oder Sybase – wir schreiben hier immer **`$verbindung->query(…)`** . Wir haben von der Datenbank "abstrahiert" – deshalb ist PDO eine "Abstraktionsschicht".*

(4) Ergebnisse durchlaufen

```php
// Ergebnisse durchlaufen
foreach ($ergebnis as $zeile) {
    $bezeichnung = $zeile['bezeichnung'];
    echo $bezeichnung . "<br>";
}
```

`$ergebnis` enthält nach Schritt (3) das Ergebnis der ausgeführten SQL-Abfrage.

In unserer Schleife verhält sich **`$ergebnis`** wie ein Array. Was hier konkret passiert: Wir holen die Daten zeilenweise aus der Datenbank und nennen sie jeweils **`$zeile`**.
`$zeile` ist ein Array, in dem die Daten gespeichert sind:

```
$zeile['id']                => 1
$zeile['bezeichnung']       => Sonnencreme
$zeile['verkaufspreis']     => 6.95
$zeile['herstellungsland']  => F
```

Die Spaltenüberschriften können wir also als Indexwert verwenden, etwa

```php
$bezeichnung      = $zeile['bezeichnung'];
$land             = $zeile['land'];
echo "<p>" . $bezeichnung . " wird in " . $land . " hergestellt.</p>";
```

TIPP: Am einfachsten ist es, wenn wir die benötigten Arrayfelder immer zuerst in einer Variablen speichern. Das gewöhnen wir uns an:

```php
foreach ($ergebnis as $zeile) {
    $bezeichnung    = $zeile['bezeichnung'];
    $verkaufspreis  = $zeile['verkaufspreis'];
    echo "$bezeichnung kostet $verkaufspreis<br>";
}
```

ist einfacher als

```php
foreach ($ergebnis as $zeile) {
    echo $zeile['bezeichnung'] . " kostet " . $zeile['verkaufspreis'] . "<br>";
}
```

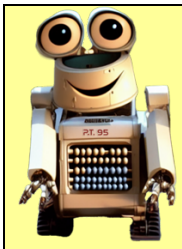
PRO-Wissen: `$ergebnis` *ist ein Objekt der Klasse PDOStatement. Diese Klasse bietet einige Methoden für den Zugriff auf Daten an, z.B.* `fetch()` *(nächste Zeile holen) oder* `fetchColumn()` *(Wert von nur einer Spalte holen) .*

(5) (Optional): Datenbankverbindung schließen

```php
// Datenbankverbindung schließen
$verbindung = null;
```

5

Das müssen wir nicht machen, PHP schließt die Verbindung normalerweise automatisch irgendwann. Es ist aber gute Praxis, das zu machen, um Ressourcen freizugeben oder Probleme mit zu vielen offenen Verbindungen zu vermeiden.



PDO

PDO (engl. für ***P**HP **D**ata **O**bjects*) ist eine sog. »Abstraktionsschicht« - ein universelles Werkzeug, das es erlaubt, auf verschiedene Arten von Datenbanken (MySQL, SQLite, PostgreSQL etc.) zuzugreifen und mit ihnen zu arbeiten, ohne den Code groß zu ändern. Deshalb enthält der Parameter beim Öffnen der DB den Namen des Datenbanksystems:

(`$verbindung = new PDO('sqlite:name.sqlite')`) oder

```
$verbindung = new PDO('mysql:host=localhost;dbname=db1', alf',
'1234');
```

Es macht die Datenbankarbeit einfacher und sicherer, indem es eine gemeinsame Methode zur Kommunikation mit verschiedenen Datenbanken bietet.

Aufgabe 1: DB-Verbindung fixen

Im folgenden Code sind die Schritte durcheinandergeraten. Bringen Sie sie in eine sinnvolle Reihenfolge.

A
```php
$verbindung = new PDO('sqlite:beach.sqlite');
```

B
```php
$ergebnis = $verbindung->query($abfrage);
```

C
```php
$abfrage = "SELECT * FROM produkte p;";
```

D
```php
$verbindung = null;
```

E
```php
foreach ($ergebnis as $zeile) {
    $artikelname = $zeile['bezeichnung'];
    echo "$artikelname<br>";
}
```

Aufgabe 2: Buttkes Shopartikel ausgeben

Öffnen Sie das Projekt *1-buttke-shop* und ergänzen Sie die Datei index.php so, dass die Inhalte der Datenbank wie folgt ausgegeben werden:

Dies sind unsere Produkte
NACH PREIS ABSTEIGEND SORTIERT:

Sonnenschirm: 29.95€
Bikini »Pamela Anderson«: 24.95€
Sonnencreme »Ultra 80«: 18.95€
Surfbrett »Dietmar«: 14.95€
Sonnenbrille: 11.95€
Sonnencreme »Ultra 50«: 9.95€
Badehose »The Rock«: 9.9€
Badehose »Arnold«: 9.5€
Sonnencreme »Bon Voyage« LSF 30: 6.95€
Sonnenhut El Padron: 4.95€
Strandtennisset: 3.49€

Zusatzaufgabe I

Die Inhalte sollen als unnummerierte Liste (`<ul><li>` …) ausgegeben werden.
Tipp: Der HTML-Tag für die Liste kommt außerhalb der Schleife, jeder Schleifendurchlauf gibt einen einzelnen Listenpunkt aus:

Dies sind unsere Produkte
NACH PREIS ABSTEIGEND SORTIERT:

- Sonnenschirm: 29.95€
- Bikini »Pamela Anderson«: 24.95€
- Sonnencreme »Ultra 80«: 18.95€
- Surfbrett »Dietmar«: 14.95€
- Sonnenbrille: 11.95€
- Sonnencreme »Ultra 50«: 9.95€
- Badehose »The Rock«: 9.9€
- Badehose »Arnold«: 9.5€
- Sonnencreme »Bon Voyage« LSF 30: 6.95€
- Sonnenhut El Padron: 4.95€
- Strandtennisset: 3.49€

Zusatzaufgabe II

Prüfen Sie in der Schleife mit PHP, ob der Preis mehr als 20 Euro beträgt. Wenn ja, soll hinter dem Preis "(teuer)" stehen. Wenn der Preis weniger als 5 Euro beträgt, soll hinter dem Preis "(billig)" stehen.
Die Texte in den Klammern könnte man noch einfärben, etwa durch

```
<span style='color: red'>texttext</span>
```

Das würde dann so aussehen:

Dies sind unsere Produkte
NACH PREIS ABSTEIGEND SORTIERT:

Sonnenschirm: 29.95€ (teuer)
Bikini »Pamela Anderson«: 24.95€ (teuer)
Sonnencreme »Ultra 80«: 18.95€
Surfbrett »Dietmar«: 14.95€
Sonnenbrille: 11.95€
Sonnencreme »Ultra 50«: 9.95€
Badehose »The Rock«: 9.9€
Badehose »Arnold«: 9.5€
Sonnencreme »Bon Voyage« LSF 30: 6.95€
Sonnenhut El Padron: 4.95€ (billig)
Strandtennisset: 3.49€ (billig)

Zusatzaufgabe IV

Spielen Sie mit den SQL-Abfragen herum, zum Beispiel:

- Lassen Sie sich nur alle Artikel anzeigen, die weniger als 28 Euro kosten.
- Sortieren Sie die Ergebnisliste nach Preis absteigend.
- Alle Artikel anzeigen, deren Bezeichnung mit einem S beginnt und die in Frankreich oder Spanien hergestellt wurden.

Aufgabe 3: All-In-One Shop

Öffnen Sie im Projekt *2-aio-shop* die Datei *aio-shop.php*.

Aufgabe 3a

Entwerfen Sie dort den Code für diese Anforderungen:

- Greifen Sie auf die Datenbank *shop.db* zu.
- Ermitteln Sie mit einer SQL-Abfrage alle Waren, deren Preis über 400 Euro liegt.
- Geben Sie alle Datensätze auf der php-Seite aus wie hier dargestellt:

```
Rüttelplatten-Verdichter 196ccm / 414.99 / 1
Asus GT-AX11000 ROG Rapture Gaming Router / 418.18 / 1
Sony LED TV 85" / 419 / 1
Asus Laptop / 419 / 1
VU+ Ultimo 4K Digitaler Sat-Receiver / 419 / 1
Laptop / 419 / 1
Laptop mit installiertem Office / 449 / 1
Garmin Fenix 7 Sportuhr / 499 / 1
Freistehende Badewanne mit Wasserhahn / 549 / 1
```

Aufgabe 3b

Lassen Sie sich unter der Ausgabe der Artikel die Anzahl der ermittelten Artikel ausgeben:

```
-------
Anzahl der Artikel: 9
```

Aufgabe 3c

Erstellen Sie eine weitere Abfrage, mit der folgende Artikel ausgewählt werden:

- Preis > 100 Euro
- Verkaufseinheit = „Stück“

Die Datensätze müssen Sie nicht ausgeben, wir werden diese aber in der nächsten Aufgabe verwenden!

Aufgabe 3d

Berechnen Sie den durchschnittlichen Preis der Artikel, die Sie in Aufgabe 3 ermittelt haben.
Zur Kontrolle:

```
=========

Artikel: Preis > 100, Einheit: Stück
-----------------------------------------
Durchschnittlicher Preis: 236.15 Euro
```