SQLITE 3 & PHP

Adress Datenbank Beispiel

Ich möchte hier ein paar Dinge zusammen tragen welche man beim Erstellen einer Datenbankverwaltung braucht. Anhand einer einfachen Datenbankverwaltung für Adressen. Genaue Sytax und Hilfe gibt es auf vielen anderen Seiten.

SQLite ist eine Datenbank, die mit einer Datei und einem Treiber anstelle eines Datenbankprozess arbeitet. SQLite ist eine vollständige Implementierung von ANSI Standard-SQL.
Als schnelle, zuverlässige und relationale Datenbank ist SQLite bestens für Web-Anwendungen geeignet und optimiert für mobile Geräte.

Für die Sicherheit ist es wichtig sich ein Konzept zu überlegen um SQL Injections zu vermeiden. Dafür gibt es verschiedene Ansätze welche auch teilweise in diesem Beitrag beschrieben werden.
Man sollte sich überlegen bei Datenbankzugriffen in Zukunft auf PDO zu setzen, weil hier standardmäßig Prepared Statements unterstützt werden. Hiermit ist man wesentlich sicherer.
Seite mit PDO Script:
http://www.nof-tutorials.com/Datenbank-SQLite/tutorial.html

PHP und SQLite3 für Webseiten

SQLite ist eine Datenbank, welche keine komplizierten Datenbankverbindungen etc. braucht. Es wird einfach eine Datei im Verzeichnis erstellt in welcher sich die Tabellen befinden. SQLite ist eine vollständige Implementierung von ANSI Standard-SQL.

Als schnelle, zuverlässige und relationale Datenbank ist SQLite bestens für Web-Anwendungen geeignet und optimiert für mobile Geräte.

SQLite – Datenbank in einer Datei

SQLite nimmt eine Datei und nutzt sie als Datenbank. SQLite 3.6 war in den meisten PHP-Installation ab PHP 5.3 schon vertreten, im PHP 7 sitzt SQLite 3.30 (2019-10-11) oder neuer. SQLite ist der zuverlässige Langzeitspeicher in allen Mobilgeräten und damit die meist genutzte Datenbank.

PHP stellt zwei Schnittstellen für SQLite3 bereit:

Natives SQLite3

Das native PHP-SQLite3-Interface ist stark an SQLite3 in C ausgerichtet. Allerdings ist PHP SQLite3 nicht die vollständige Umsetzung von SQLite3 in C, unterstützt Exceptions nicht vollständig und die Fehlerbehandlung ist spartanisch.

PDO – PHP Data Objects

PDO ist eine normalisierte Schnittstelle zu Datenbanken auf verschiedenen Plattformen und stellt in den meisten Fällen das bevorzugte Interface zu SQLite3 dar.

SQLite Datentypen („Storage Classes“)

Datentyp	Wert	(Priorität) Typ-Affinität
NULL	NULL	
INTEGER	Eine Ganzzahl, gespeichert in 1, 2, 3, 4, 6, oder 8 Byte je nach Größe des Wertes (signed 64-bit)	(1) INT, INTEGER, TINYINT, SMALLINT, MEDIUMINT, BIGINT, UNSIGNED BIG INT, INT2, INT8, FLOATING POINT
TEXT	Ein String, gespeichert unter Verwendung der Datenbank-Kodierung (UTF-8, UTF-16BE oder UTF-16LE) (Max Grösse Vorgabe 1,000,000,000 bytes)	(2) CHARACTER(20), VARCHAR(255), VARYING CHARACTER(255), NCHAR(55), NATIVE CHARACTER(70), NVARCHAR(100), TEXT, CLOB
BLOB	Eine Masse von Daten, gespeichert wie eingegeben	(3) BLOB
REAL	Eine Fließkommazahl gespeichert als IEEE Gleitkommazahl von 8 Byte (64-bit (double))	(4) REAL, DOUBLE, DOUBLE PRECISION, FLOAT
NUMERIC		(5) NUMERIC, DECIMAL(10,5), BOOLEAN, DATE, DATETIME, STRING

SQLite Vor- und Nachteile

In SQLite fehlen einige SQL-Funktionen, so z.B. Right Join, for-each triggers, drop column, alter column.

Es gibt keinen Mechanismus, der einen falschen Datentyp in einem Feld verhindert – ich kann einen String in ein Integer-Feld setzen.

Bei hochfrequenten Schreibvorgängen wird SQLite langsam.

Auf der Pro-Seite wird MacOS seit 10.4 schon mit SQLite ausgeliefert, unter Windows ist SQLite schnell

installiert und einfach zu initialisieren. Die Portierung auf einen anderen Server ist denkbar einfach: kopieren und fertig. Und ein Backup: kopieren und fertig.

SICHERHEIT

Für SQLite-Datenbankdateien wird in der Regel die Endung SQLITE verwendet.
Sie sollten, müssen aber diese Endung nicht verwenden, da Sie immer den vollen Dateinamen angeben müssen. Jedoch sollten Sie darauf achten, die Endung entweder in der Apache-Konfiguration so einzustellen, dass darauf nicht direkt zugegriffen werden kann und sie nicht in einer Verzeichnisauflistung erscheinen, oder aber die Datenbankdatei außerhalb des Root-Verzeichnisses des Webservers abzulegen. Um die Dateien zu verstecken, legen Sie in dem Verzeichnis eine .htaccess-Datei an und fügen Sie folgende Zeilen ein:

```
<Files ~ "\.(sqlite)$">
Order allow,deny
Deny from all
</Files>
```

Beispiele

Anhand einer kleinen Adressverwaltung zeige ich hier den Umgang mit PHP und SQLite3

Einfaches Beispiel

Öffnet eine Datenbank, legt die Tabelle „foo“ an, schreibt einen Datensatz hinein und gibt ihn anschließend wieder aus

```
<?php
$db = new SQLite3('mysqlitedb.db');               // öffnet eine Datenbank

$db->exec('CREATE TABLE foo (bar STRING)');       // erstellt eine Tabelle
mit dem Namen "foo" und dem Feld "bar" als String
$db->exec("INSERT INTO foo (bar) VALUES ('Dies ist ein Test')");   // legt
einen Datensatz an und schreibt in das Feld "bar" den Wert "Dies ist ein
Test"

$result = $db->query('SELECT bar FROM foo');      // ließt das Feld "bar"
aus dem Datensatz der Datenbank
var_dump($result->fetchArray());                  // Ausgabe der
ausgelesenen Datensätze
?>
```

Datenbank öffnen

Datenbank öffnen

```
$db = new SQLite3(<Datenbank-Name>);
```

Datenbank schließen

```
$db->close();
```

Tabelle für eine Datenbank anlegen wenn nicht vorhanden

Zuerst muss die Datenbank geöffnet sein, dann kann man hiermit eine Tabelle anlegen. Mit „IF NOT EXISTS“ wird geprüft ob die Tabelle schon existiert.

```
$db-> exec("CREATE TABLE IF NOT EXISTS <Tabelle>(
    id INTEGER PRIMARY KEY AUTOINCREMENT,      // hier wird dem Datensatz
autom. eine eindeutige Nummer zugewiesen
    name TEXT NOT NULL DEFAULT '0',            // weitere Felder wie name,
strasse, plz und ort
        strasse TEXT NOT NULL DEFAULT '0',
        plz TEXT NOT NULL DEFAULT '0',
        ort TEXT NOT NULL DEFAULT '0',
)");
```

Datensatz anlegen

<exec> wird für INSERT, UPDATE und DELETE benutzt.
Also dann wenn keine Rückgabe Wert erwartet wird. Es wird nur true oder false zurück gegeben.

<QUERY> sollte dann benutzt werden, wenn man ein Ergebnis erwartet - wie z.B. das auslesen von Datensätzen.

```
$db->exec("INSERT INTO <Tabelle> (name, strasse, plz, ort) VALUES ('$_name',
'$_strasse','$_plz', '$_ort')");
```

alle Datensätze auslesen und anzeigen

```
$results = $db->query("SELECT *
   FROM <Tabelle>
   ORDER BY name");    // sortiert nach Name
// alle eingelesenen Datensätze ausgeben
while ($row = $results->fetchArray()) {
   echo $row['name'];
   echo $row['strasse'];
   echo $row['plz'];
   echo $row['ort'];
}
```

Datensatz ändern

exec wird für INSERT, UPDATE und DELETE benutzt.

```
$db->exec("UPDATE <Tabelle> SET name='$_name', strasse='$_strasse',
plz='$_plz', ort='$_ort' WHERE ID = 6");
```

Datensatz löschen

exec wird für INSERT, UPDATE und DELETE benutzt.

```
$db->exec("DELETE FROM <Tabelle> WHERE ID = 6");
```

Wenn Sie alle Datensätze aus der Tabelle löschen möchten, brauchen Sie keine WHERE-Klausel zu verwenden, DELETE-Abfragen wie folgt:

```
$db->exec("DELETE FROM <Tabelle>");
```

komplette Tabelle löschen

```
$db->exec("DROP TABLE $db_table");
```

einzelnen Datensatz ausgeben

Alle Felder des Datensatzes mit der ID = 23 auslesen.
Mit extract() werden aus dem Array wieder normale Variablen wie die Datenfelder heißen.
Z.B. $id und $name

Mit var_dump($row) kann man sich das Array anzeigen lassen.

```
$results = $db->query("SELECT * FROM <Tabelle> WHERE id='23' LIMIT 1");
    $row = $results->fetchArray();
        extract($row);
        echo $id." ";
                echo $name." ";
                echo $strasse." ";
                echo $plz." ";
                echo $ort."<br>";
```

ODER ALS ARRAY AUSGEBEN

```
$results = $db->query("SELECT * FROM <Tabelle> WHERE id='23' LIMIT 1");
    $row = $results->fetchArray();
        echo $row["id"]." ";
```

```
                echo $row["name"]." ";
                echo $row["strasse"]." ";
                echo $row["plz"]." ";
                echo $row["ort!]."<br>";
```

so kann man Fehler abfangen

```
$results= $db->query("SELECT * FROM <Tabelle>");
            if($results==FALSE) {
                echo "Error in fetch ".$db->lastErrorMsg();
            } else {
                while ($row= $results->fetchArray())
                ...
            }
```

letzten Datensatz ermitteln

Zuletzt eingefügter Record

```
$lstrack = $db->lastInsertRowid();
```

Bei Tabellen mit Primary Key gibt $db→lastInsertRowid() die id des zuletzt eingefügten Datensatz zurück.
Funktioniert aber nur nach INSERT.

Andere Möglichkeit:
Durch sortieren nach ID und auslesen des 1. Datensatzes haben wir die letzte Datensatz ID

```
// letzte ID ermitteln = $row['id']
$results = $db->query("SELECT id FROM <Tabelle> ORDER BY id DESC LIMIT 1");
$row = $results->fetchArray();
echo "Letzte ID: ".$row["id"]."<br>";
```

LIKE und Wild-Card-Suche

Mittel der Anweisung LIKE können wir nach einfachen Mustern in unserer Datenbank suchen, beispielsweise nach allen Personen deren Vorname mit Ma beginnt. Dazu verwenden wir LIKE in Kombination mit der Wild-Card %. % steht dabei für beliebig viele Zeichen:

```
// Findet alle User deren Name mit Ma beginnt (z.B. Max, Mark, Markus,
Matthias...)
$results = $db->query("SELECT * FROM <tabelle> WHERE vorname LIKE 'Ma%'");
```

Neben % können wir auch _ verwenden. Dies entspricht genau einem unbekannten Zeichen. 'De_' würde also auf die Wörter Der, Den, Des usw. passen.

Hiermit können wir eine einfache Suche für unsere Tabelle bauen.

```
// Findet alle Einträge die 'Suchwort' in der entsprechenden Spalte
enthalten
$results = $db->query("SELECT * FROM <tabelle> WHERE spalte LIKE
'%Suchwort%'");
```

GLOB und Wild-Card-Suche

Der GLOB Operator ist vergleichbar mit dem LIKE Operator, aber er benutzt die Unix File Syntax (*?). Auch ist GLOB „case sensitive“ (Beachtet die Groß- und Kleinschreibung) was LIKE nicht macht.

```
$results = $db->query("SELECT trackid, name FROM <Tabelle> WHERE name GLOB
'?ere*");
```

Menge der Datensätze ermitteln

```
$results = $db->query("SELECT * FROM <Tabelle>");
echo "Gesamt: ".sqlite_num_rows($result)
```

oder

```
$results = $db->query("SELECT * FROM <Tabelle>");
$row = $results->fetchArray();
echo "Gesamt: ".count($row)."<br>";
```

Doppelte Einträge verhindern

Mittels der DISTINCT-Anweisung können wir in einem SELECT doppelte Einträge vermeiden, sprich, so erhalten wir nur die eindeutigen Einträge zurück. Dabei muss das DISTINCT ganz am Anfang der Spaltenauswahl stehen.

Nachfolgend ein Beispiel, zum einen um die eine Liste aller (eindeutigen) Vornamen und aller (eindeutigen) Vor- und Nachnamen zu erhalten:

```
//Liste mit eindeutigen Vornamen, alphabetisch sortiert
$results = $db->query("SELECT DISTINCT vorname FROM <tabelle> ORDER BY
vorname");

//Liste mit eindeutigen Namen (Vor- und Nachname(, alphabetisch sortiert
$results = $db->query("SELECT DISTINCT vorname, nachname FROM <tabelle>
ORDER BY vorname, nachname");
```

wiederholtes ändern / speichern vermeiden

Bei der Übergabe der Variablen mit $_POST oder $_GET wird durch aktualisieren der Seite im Browser die Daten erneut versendet.

Damit deshalb z.B. nicht jedesmal erneut ein Datensatz angelegt wird, arbeite ich mit Session.
Ich merke mir die letzte Datensatz ID und wenn diese nochmal angelegt werden soll, dann führe ich das INSERT nicht aus.

```
// am Anfang muss die Session gestartet werden
session_start();

// letzte Datensatz ID ermitteln - lastinsertrow() funktioniert hier nicht
$results = $db->query("SELECT id FROM <Tabelle> ORDER BY id DESC LIMIT 1");
$row = $results->fetchArray();

// prüfen ob gemerkte ID mit letzter ID übereinstimmt
IF ($_SESSION["lastrec"] == $row["id"]){
        echo "Fehler: Daten wurden bereits angelegt";
} else {
        $db->exec("INSERT INTO <Tabelle> ( name, strasse, plz, ort ) VALUES
('$_name', '$_strasse','$_plz', '$_ort' )");
    $_SESSION["lastrec"] = $db->lastInsertRowid(); // letzte Datensatz ID in
Session merken
    echo "Daten wurden gespeichert";
}
```

```
$db-> exec("DROP TABLE $db_table);
```

Guter Beitrag: https://wiki.selfhtml.org/wiki/PHP/Tutorials/Reloadsperre

CSV Import

Daten aus der Datei mediennews.csv einlesen und in die Tabelle übertragen. Die einzelnen Felder sind mit einem Semikolon separiert. Dies ist wichtig, weil in der entsprechenden PHPFunktion zum Einlesen der Daten dieser Parameter angegeben werden muss.

```
("mediennews.csv","r");
while ( ($data = fgetcsv ($handle, 1000, ";")) !== FALSE ) {
     $results = $db->exec("INSERT INTO <Tabelle> VALUES (NULL,'$data[0]',
'$data[1]', '$data[2]')");
}
fclose ($handle);
```

Benutzerdefinierte Funktionen

Quasi als Ausgleich für, dass SQLite nicht alle Befehle von z.B. MySQL unterstützt, bietet Ihnen SQLite jedoch die Möglichkeit diese Defizite auszugleichen, durch eigene Funktionen zu schreiben und diese in Ihren SQL-Abfragen zu verwenden.

Mit dem Befehl sqlite_create_function() können Sie eine PHP-Funktion als so genannte User Defined Function (UDF) erzeugen und diese dann direkt in SQL-Befehlen nutzen. Im folgenden Beispiel wird eine Funktion definiert, die die md5-Summe eines Strings berechnet und dann rückwärts

ausliefert.Wenn der SQL-Befehl durchgeführt wird, liefert er den Wert der Spalte filename durch die Funktion transformiert zurück. Die Daten, die in $rows stehen, enthalten also die bereits gewandelten Daten:

```
<?php
function md5_rueckwaerts($string) {
      return strrev(md5($string));
}

sqlite_create_function($db, 'md5rw','md5_rueckwaerts', 1);
$rows = sqlite_array_query($db,'SELECT md5rw(filename) from files');
?>
```

SICHERHEIT

SQL-Injections

Hacker kann es durch SQL Injections gelingen, Daten zu klauen, manipulieren oder zu löschen. Um SQL-Injections zu verhindern empfiehlt sich der Einsatz von prepared statements . Sobald ihr irgendwelche Daten vom Benutzer an die Datenbank übergebt, sollte ihr stets auf prepared Statements zurückgreifen.

```
$db = new SQLite3('school.db');
if($stmt = $db->prepare('SELECT id,student_name FROM classTen '))
{
         $result = $stmt->execute();
         $names=array();
         while($arr=$result->fetchArray(SQLITE3_ASSOC))
         {
          $names[$arr['id']]=$arr['student_name'];
         }
}
```

```
$smt = $db->prepare("insert into names (name, email) values (':name',
':email')");
$smt->bindValue(':name', $_POST['post_name'], SQLITE3_TEXT);
$smt->bindValue(':email', $_POST['post_email'], SQLITE3_TEXT);

$smt->execute();
```

escapeString

Gibt eine passend maskierte Zeichenkette zurück - entfernt Sonderzeichen und erzeugt eine Zeichenkette welche ohne Bedenken in einer SQL-Anfrage benutzt werden kann.

```
$unsafe = $_GET['nastyvar'];
$safe = SQLite3::escapeString($unsafe);
$sql = "INSERT INTO table (field) VALUES ($safe);";
```

```
echo ($sql);
```

```
 $url = $db->escapeString($url);
```

Variablen prüfen

Variablen sollten auch geprüft werden ob sie den gewünschten Inhalt Typ haben.
Hierfür gibt es in PHP die CTYPE Funktionen

XXX

Wandelt alle geeigneten Zeichen in entsprechende HTML-Codes um
https://www.php.net/manual/de/function.htmlentities.php

posted sollten so behandelt werden

```
        $name=htmlspecialchars(strip_tags($_POST['name']));
```

Links

- http://www.w3big.com/de/sqlite/default.html
- https://www.mediaevent.de/tutorial/php-sqlite3.html
- SQLite Dokumentation und Download
- DB Browser for SQLite