SQLite PHP: Connecting to SQLite Database From PHP Using PDO

Summary: in this tutorial, we will show you how to establish a connection to an SQLite database from PHP using PDO.

PHP includes the SQLite extension by default so you don't need to perform any configuration to make it works with the SQLite.

Setup PHP project structure with Composer

First, create a project folder named `phpsqliteconnect` and another subfolder name `app`. The `app` folder is used to store all classes that deal with application logic and database.

Next, create a new `composer.json` file with the following code:

```
{
    "autoload": {
        "psr-4": {
            "App\\": "app/"
        }
    }
}
```

If you are not familiar with Composer, you can check it out here (https://getcomposer.org/) . In PHP, we use the composer as a tool for dependency management. The composer allows you to declare the library in our project and manage the updates automatically.

In the composer file, you map the `App` namespace with the `/app` folder.

Then, create another subfolder name `db` to store the SQLite database file.

After that, open the command tool, navigate to the `phpsqliteconnect`, and type the following command:

```
composer update
```

The following message will display:

```
>composer update
Loading composer repositories with package information
Updating dependencies (including require-dev)
Nothing to install or update
Generating autoload files
```

In addition, Composer creates the new folder named `vendor` as shown in the following screenshot:

Finally, create a file named `index.php` in the root folder and add the following code:

```
<?php
require 'vendor/autoload.php';
```

From now on, if you want to use any classé in the `app` folder, you just simply declare and use it.

Establish database connection to an SQLite database

First, create a new file `Config.php` inside the `app` folder and add a new class named `Config` as follows:

```
<?php

namespace App;

class Config {
   /**
    * path to the sqlite file
    */
    const PATH_TO_SQLITE_FILE = 'db/phpsqlite.db';

}
```

The constant `PATH_TO_SQLITE_FILE` is used to store the path to the sqlite database file which resides in the `db` folder.

Second, create a new `SQLiteConnection.php` file and add the `SQLiteConnection` class as follows:

```
<?php
namespace App;

/**
 * SQLite connnection
 */
class SQLiteConnection {
    /**
     * PDO instance
     * @var type
     */
    private $pdo;

    /**
```

```
      * return in instance of the PDO object that connects to the SQLite
      * @return \PDO
      */
     public function connect() {
         if ($this->pdo == null) {
             $this->pdo = new \PDO("sqlite:" . Config::PATH_TO_SQLITE_FI
         }
         return $this->pdo;
     }
 }
```

To establish a database connection to an SQLite database, you need to create a new instance of the `PDO` class and pass a connection string to the constructor of the `PDO` object.

Suppose the SQLite database file is in the `db` folder, you use the following connection string:

```
sqlite:db/phpsqlite.db
```

Because you store the path to the sqlite database file in the `Config` class, you just simply use it to construct the connection string.

Notice that when PHP connects to an SQLite database that does not exist, PHP will create a new SQLite database file.

The `$pdo` is used to store the instance of the `PDO` object. In the `connect()` method, you check if a database connection has been established. If not, you create a new instance of the PDO object otherwise, you return the PDO object.

After having all classes in places, you use the following command to generate the autoload file:

```
composer dump-autoload -o
```

To establish a connection to the SQLite database, you use the following code in the `index.php` file:

```
<?php
```

```
require 'vendor/autoload.php';

use App\SQLiteConnection;

$pdo = (new SQLiteConnection())->connect();
if ($pdo != null)
    echo 'Connected to the SQLite database successfully!';
else
    echo 'Whoops, could not connect to the SQLite database!';
```

If you check the `db` folder, you will see a file with the name `phpsqlite.db` created.

When you create a new instance of the PDO, it will always throw a PDOException (http://php.net/manual/en/class.pdoexception.php) if the connection fails.

To handle the exception, you can use the `try catch` block as follows:

```
try {
   $this->pdo = new \PDO("sqlite:" . Config::PATH_TO_SQLITE_FILE);
} catch (\PDOException $e) {
   // handle the exception here
}
```

In this tutorial, you have learned how to set up the PHP project structure and establish a database connection to the SQLite database.