PDO и MySQLi

Как правило, в качестве хранилища данных используются базы данных. PHP позволяет использовать различные системы управления базами данных, но наиболее популярной на сегодняшний день в связке с PHP является СУБД MySQL.

MySQL бесплатную систему управления базами данных, которая разрабатывается компанией Oracle и которая позволяет взаимодействовать с базой данных с помощью команд языка SQL. MySQL относительно легко устанавливать и настраивать. Более того эта СУБД может работать на всех популярных ОС - на Windows, MacOS, Linux. MySQL вполне подходит как для маленьких, так и для больших проектов.

Формально есть два способа для подключения к MySQL из PHP:

Библиотека MySQLi (Improved MySQL)

Библиотека PDO (PHP Data Objects)

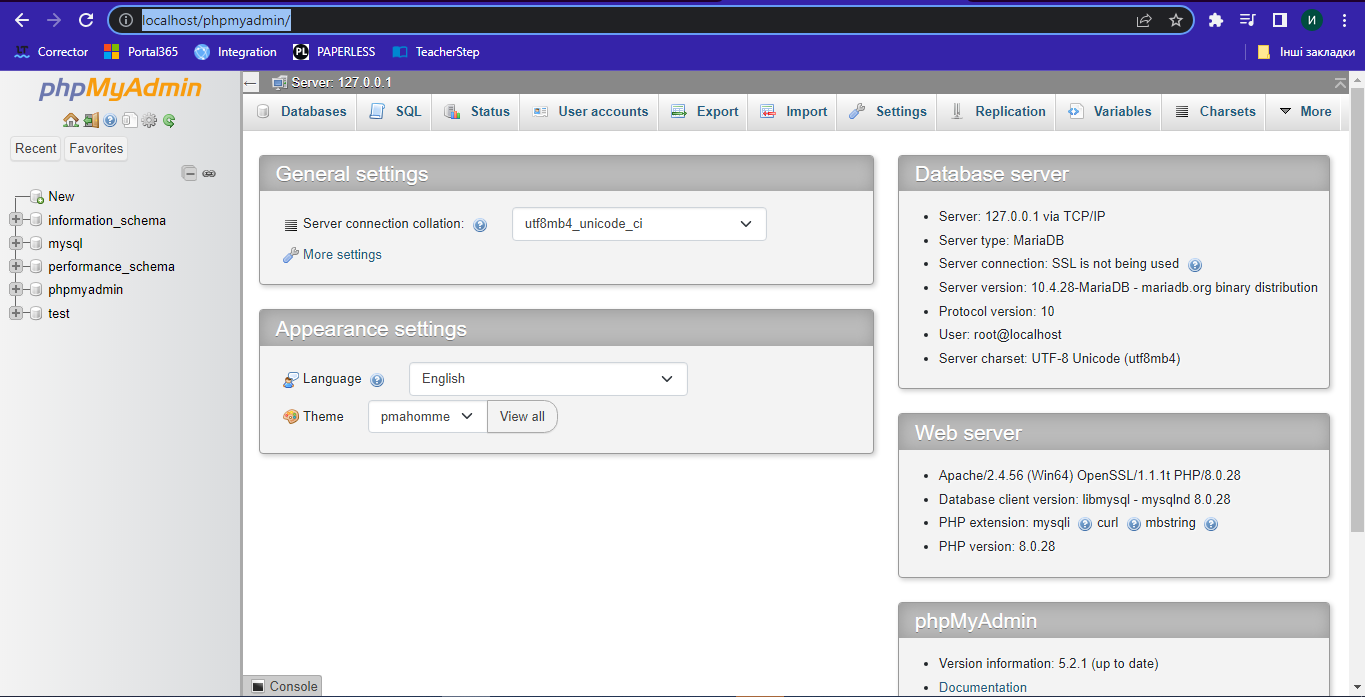
Преимуществом pdo является то, что оно позволяет работать не только с mysql, но и с еще рядом систем баз данных - Firebird, PostreSQL, SQLite, Oracle, MS SQL Server и т.д. С помощью PDO можно использовать общий подход для подключения к поддерживаемым системам баз данных, где часто достаточно изменить строку подключения, что естественно добавляет гибкости. Также особенностью PDO является то, что эта библиотека представляет объектно-ориентированный подход для работы с базами данных.

Расширение MySQLi ограничено только одной СУБД - MySQL. MySQLi предоставляет два способа взаимодействия с базами данных: объектно-ориентированный и процедурный. К преимуществам MySQLi нередко относят то, что она больше ориентирована на специфику MySQL, специфичные для этой СУБД возможности и быстрее применяет новвоведения, которая появляются с новыми выпусками MySQL. Кроме того, также в качестве преимущества MySQLi нередко заявляют о большей производительности и скорости по сравнению с PDO.

Как бы то ни было, в настоящее время распространены оба подхода. Поэтому далее мы рассмотрим, как работать с MySQL как через PDO, так и через MySQLi.

Обе библиотеки - и mysqli и pdo\_mysql по умолчанию включены в базовый комплект PHP. И нам только надо немного изменить файл конфигурации php.ini, чтобы начать работу с этими библиотеками.

Для начала проверим работу phpmyadmin:

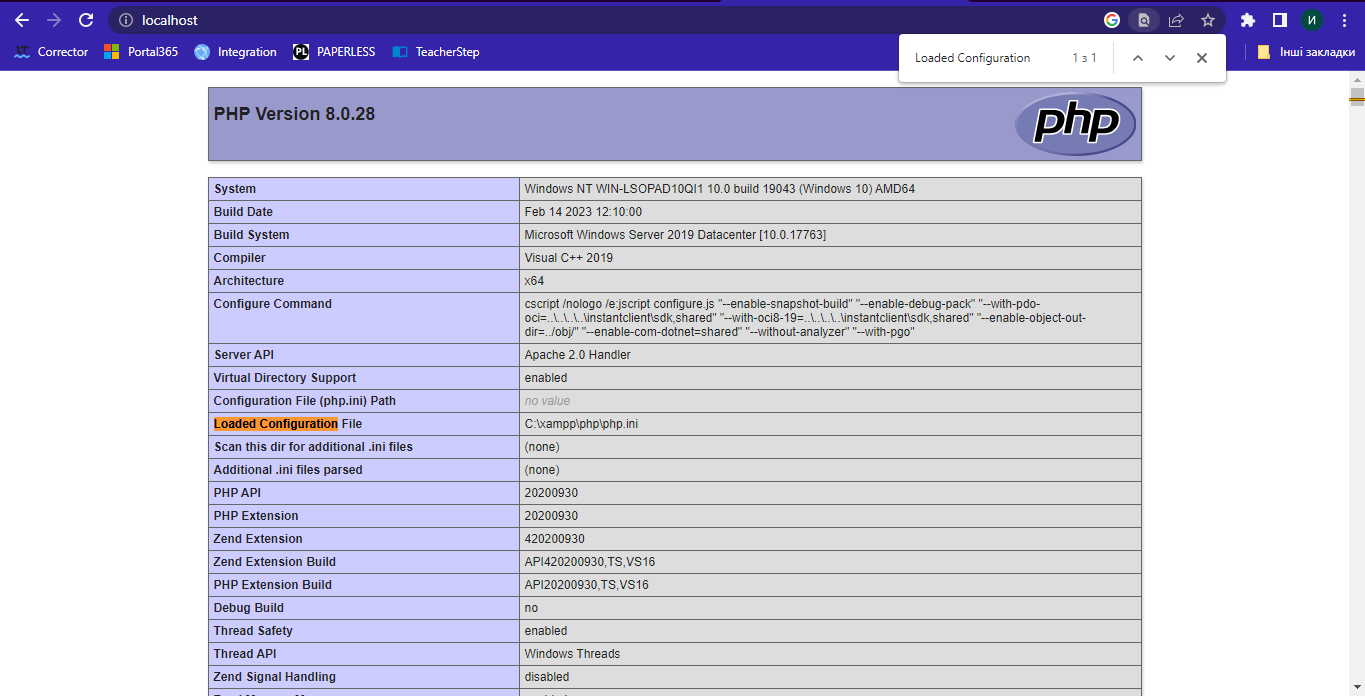


Если на сервере используются несколько версий PHP или просто не можете найти, где находится php. ini, то это можно посмотреть через phpinfo() . В строке Loaded Configuration File будет указан путь к php.

<?php

phpinfo();

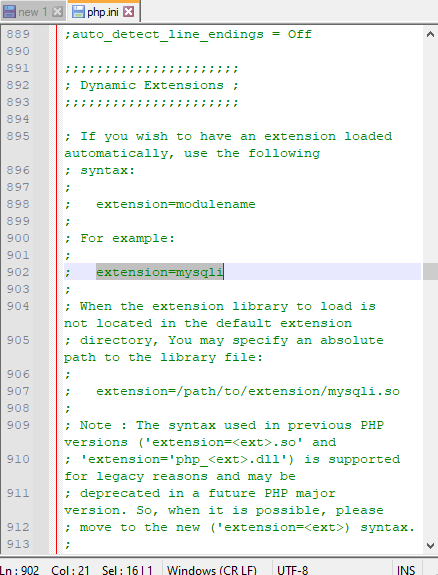
?>



Подключение mysqli

Чтобы использовать библиотеку mysqli для работы с MySQL, нам надо указать соответствующее расширение в файле php.ini. По умолчанию оно уже имеется в файле, только оно закомментировано. Так, найдем в файле php.ini следующую строку:

;extension=mysqli



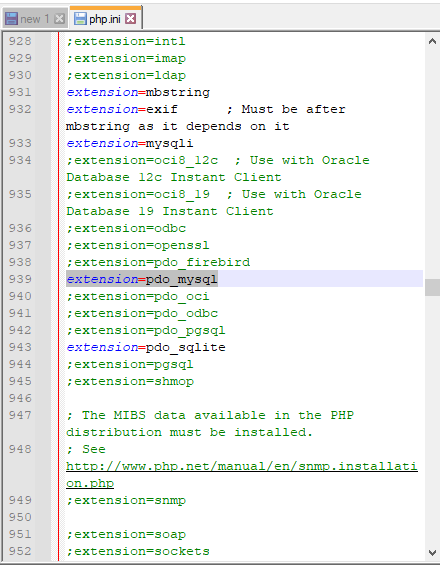
Раскомментируем ее, убрав знак точки с запятой:

extension=mysqli

Подключение pdo\_mysql

Чтобы использовать библиотеку pdo\_mysql найдеем в файле php.ini следующую строку:

;extension=pdo\_mysql



Раскомментируем ее, убрав точку с запятой:

extension=pdo\_mysql

Теперь мы можем использовать оба расширения для работы с базой данных MySQL.

PDO. Создание подключения

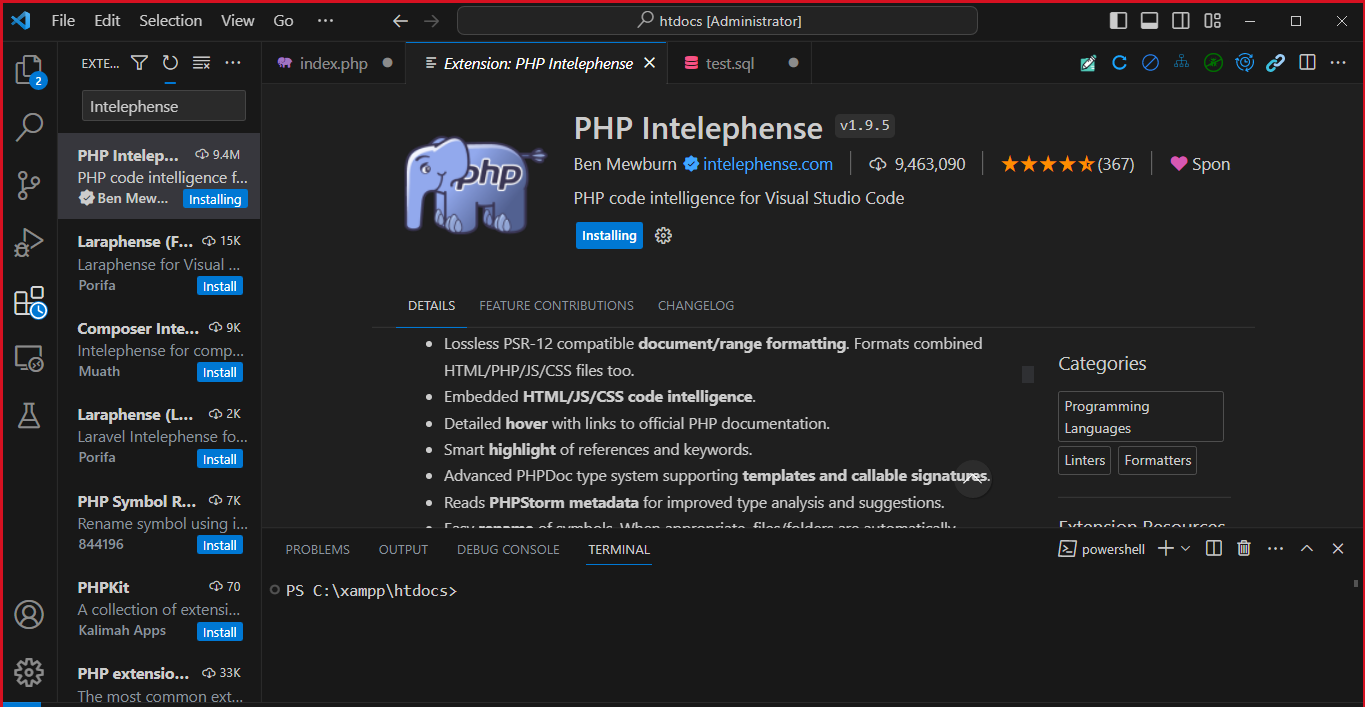
Для создания подключения к серверу базы данных в PDO применяется конструктор new PDO(), который принимает в качестве параметров настройки подключения:

<?php

new PDO("mysql:host=адрес\_сервера;port=номер\_порта;dbname=имя\_базы\_данных", "имя\_пользователя", "пароль")

?>

Установка плагина для синтаксиса в mysql:



Принимаемые параметры:

Сначала указывается строка подключечения, которая состоит из ряда настроек. Первая настройка - название драйвера базы данных. Так, в данном случае подключение осуществляется к MySQL, то тип баз данных будет mysql:.

Далее идет настройка host, которая задает хост сервера, например, host=localhost (если сервер MySQL запущен локально).

Затем дополнительно можно указать номер порта через параметр port. Если он не указан, то используется порт по умолчанию - для mysql это 3306.

И далее идет настройка dbname, которая устанавливает имя базы данных.

Кроме этих настроек строка подключения может включать еще ряд других, но это самые основные.

Второй параметр задает имя пользователя MySQL

Третий параметр устанавливает пароль для выше указанного пользователя

При успешном подключении вызов конструктора new PDO() возвращает созданный объект PDO, который представляет установленное подключение и через который мы сможем взавмодействовать с базой данных. Однако если установка подключения прошла неудачно (например, сервер базы данных недоступен, указаны неправильные имя пользователя и/или пароль, какая-то еще ошибка), то вызов конструктора генерирует исключение. Соответственно вызов данного конструктора лучше помещать в конструкцию try..catch

Определим простейший скрипт для подключения к серверу базы данных MySQL:

<?php

try {

    // подключаемся к серверу

    $conn = new PDO("mysql:host=127.0.0.1", "root", "");

    echo "Database connection established";

}

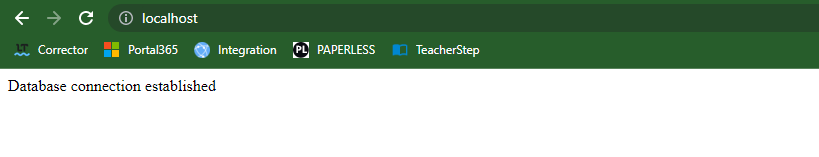
catch (PDOException $e) {

    echo "Connection failed: " . $e->getMessage();

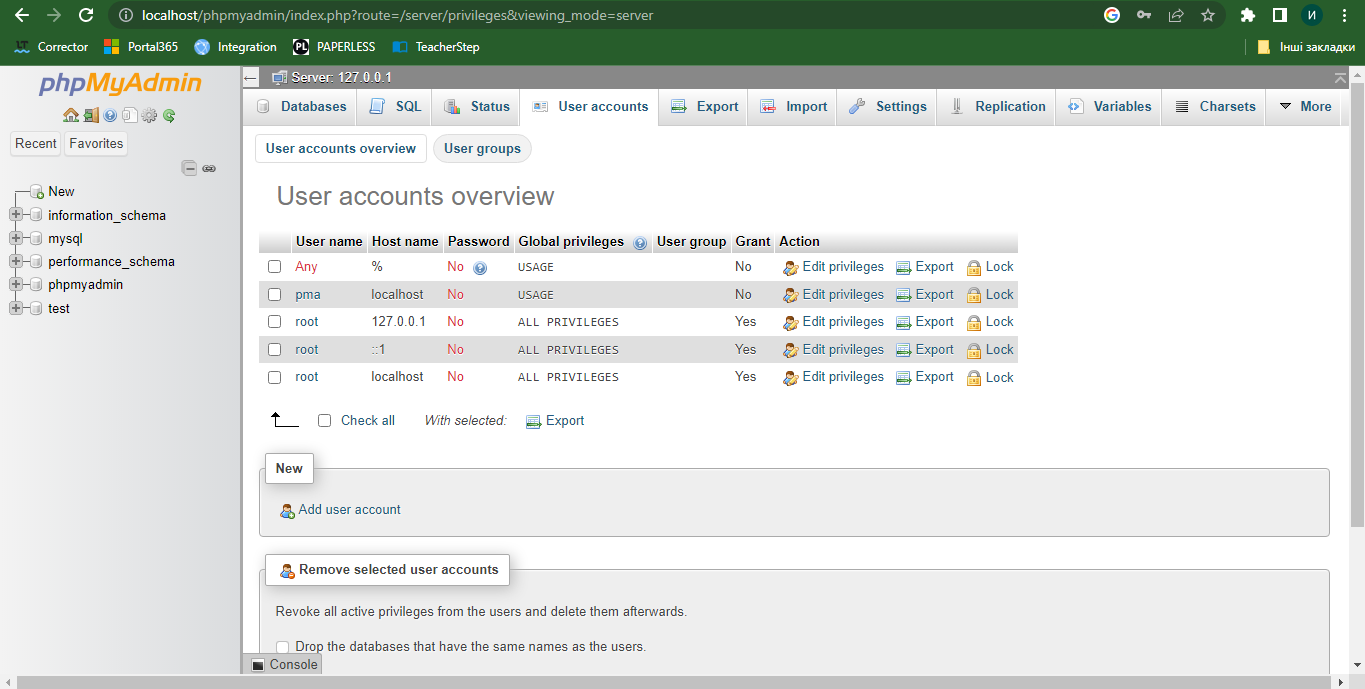
}

?>

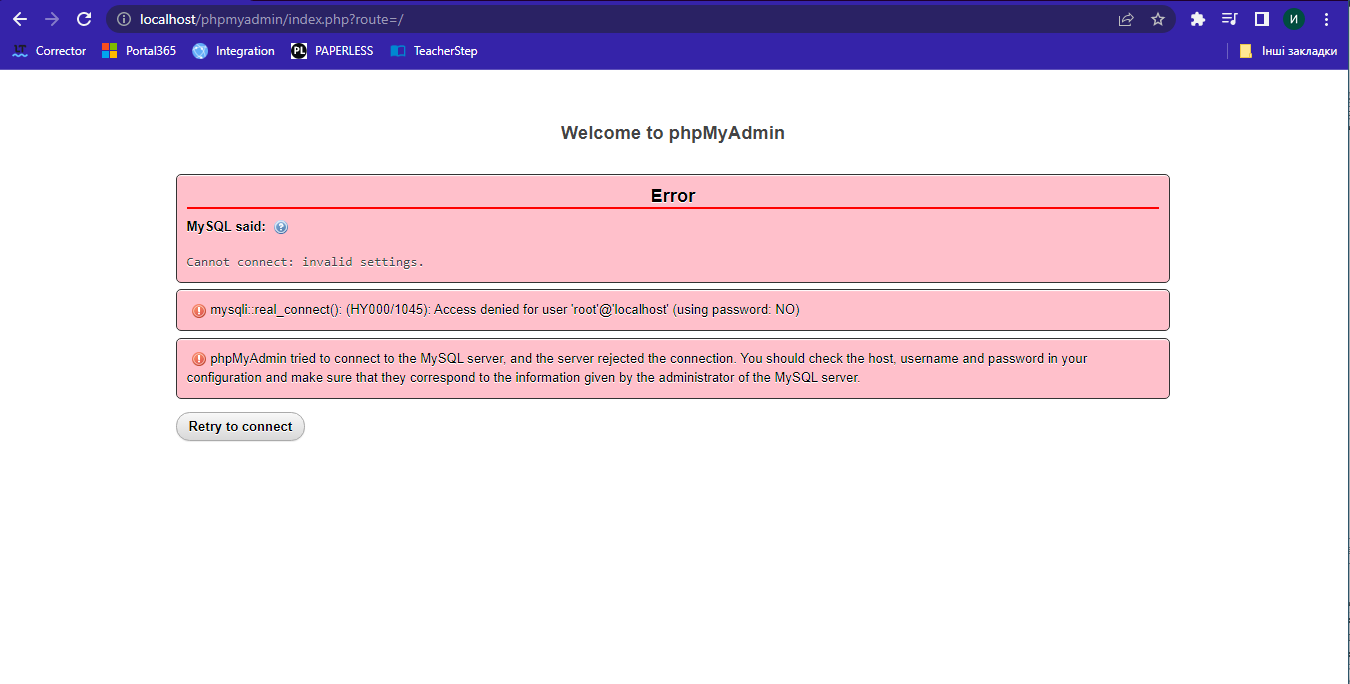
Результат должен быть именно такой:



Пароль оставляем пустым. Он по умолчанию у пользователя root не установлен

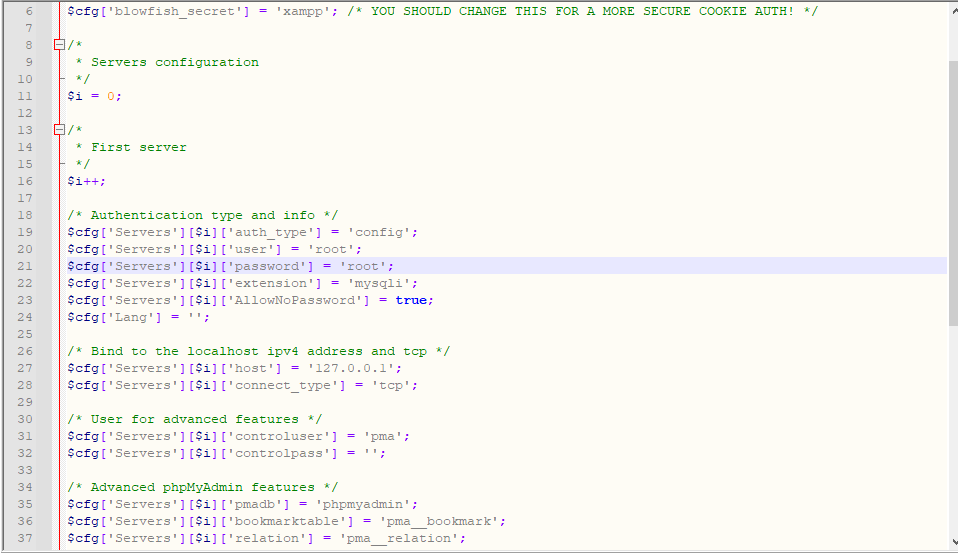


Если у Вас такая ошибка:



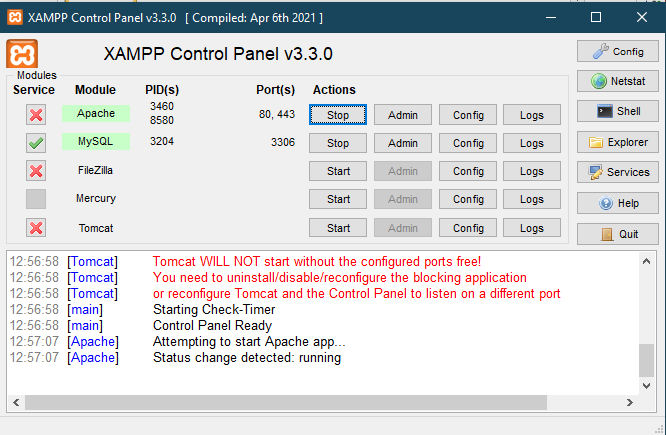
Для начала нужно проверить все параметры в конфиг файле mysql

Xampp -> phpmyadmin -> config.inc



Здесь могут быть установлены пароли, а значит их нужно или убрать, или попробовать с тем что указано в файле.

Если не помогло, попробуйте в control pannel – удаление программ проверить, возможно у Вас уже установлен mysql тогда проблема в этом, т.к. он мешает запуску служб phpmyadmin и порту 3306

  
Переустановить полностью xampp, если непомогло, то можно тогда при повторной переустановке xampp не ставить галочку mysql, а саму базу поставить через урок установка бд mysql этот урок нужно запросить у преподавателя.

Выполнение запросов в PDO. Создание базы данных и таблиц

Для выполнения запросов к серверу базы данных у объекта PDO вызывается метод exec(), в который передается выполняемое выражение SQL.

<?php

try {

    // подключаемся к серверу

    $conn = new PDO("mysql:host=127.0.0.1", "root", "");

    echo "Database connection established";

    $conn->exec('команда\_sql');

}

catch (PDOException $e) {

    echo "Connection failed: " . $e->getMessage();

}

?>

Создание базы данных

Для создания базы данных применяется SQL-команда CREATE DATABASE, после которой указывается имя создаваемой базы данных.

Создадим базу данных через PHP:

<?php

try {

    // подключаемся к серверу

    $conn = new PDO("mysql:host=127.0.0.1", "root", "");

    // SQL-выражение для создания базы данных

    $sql = "CREATE DATABASE testdb1";

    // выполняем SQL-выражение

    $conn->exec($sql);

    echo "Database has been created";

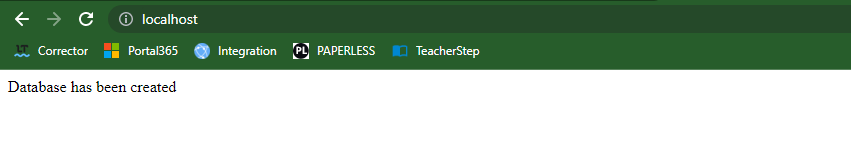
}

catch (PDOException $e) {

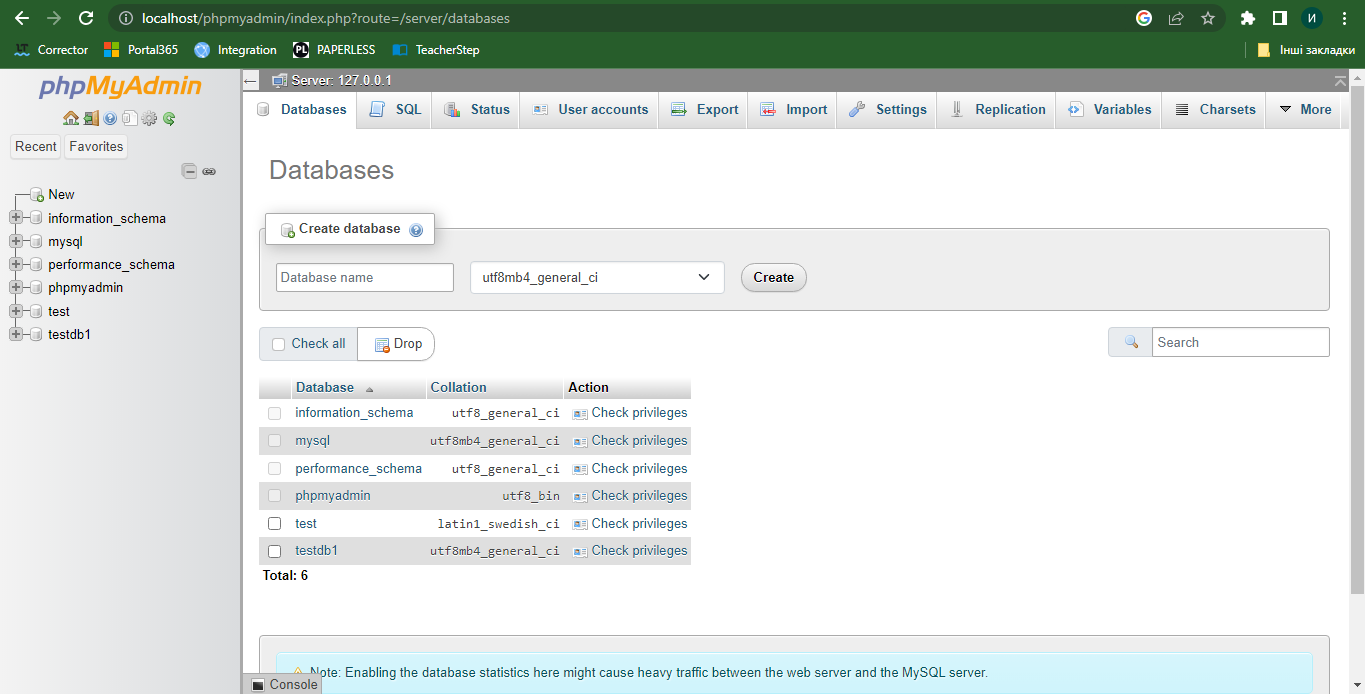
    echo "Connection failed: " . $e->getMessage();

}

?>



Проверим результат через phpmyadmin:



testdb1 должна быть в списке баз данных.

В данном случае после подключения к серверу определяется переменная $sql, которая хранит команду на создание бд с именем "textdb1":

<?php

$sql = "CREATE DATABASE testdb1";

?>

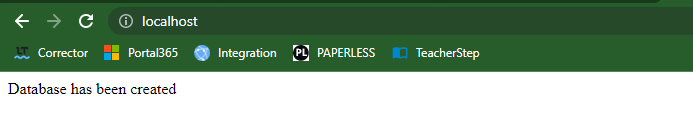
Далее для выполнения этой команды передаем ее в метод exec():

<?php

$conn->exec($sql);

?>

В итоге при успешном создании базы данных мы увидим в браузере сообщение об создании БД:



Создание таблицы

Подобным образом можно выполнять запросы на создание таблиц в базе данных. Для создания таблиц применяется SQL-команда CREATE TABLE, после которой указывается имя создаваемой таблицы и в скобках определения столбцов.

Так, возьмем выше созданную базу данных "testdb1". И создадим в ней таблицу, которая описывается следующим кодом

CREATE TABLE Users (id INTEGER AUTO\_INCREMENT PRIMARY KEY, name VARCHAR(30), age INTEGER);

Здесь создается таблица под названием "users". Она будет хранить условных пользователей. В ней будет три столбца: id, name и age. Столбец id представляет числовой уникальный идентификатор строки - или идентификатор пользователя. Столбец name представляет строку - имя пользователя. А столбец age представляет число - возраст пользователя.

Для создания таблицы определим следующий скрипт php:

<?php

try {

    // подключаемся к серверу

    $conn = new PDO("mysql:host=127.0.0.1;dbname=testdb1", "root", "");

    // SQL-выражение для создания таблицы

    $sql = "create table users (id integer auto\_increment primary key, name varchar(30), age integer);";

    // выполняем SQL-выражение

    $conn->exec($sql);

    echo "Table Users has been created";

}

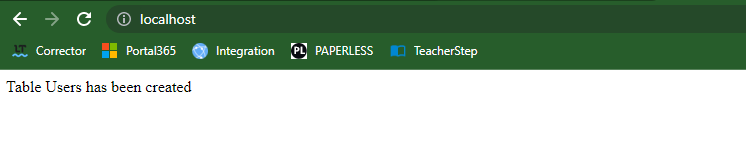
catch (PDOException $e) {

    echo "Database error: " . $e->getMessage();

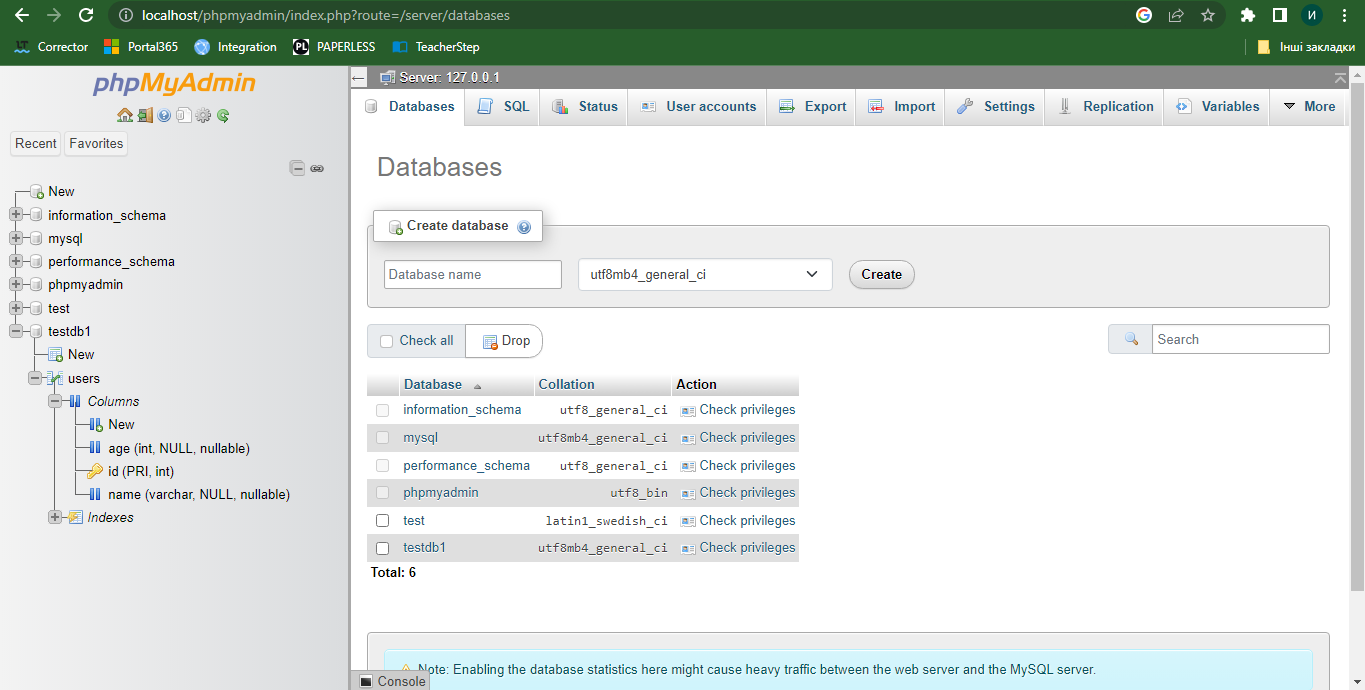
}

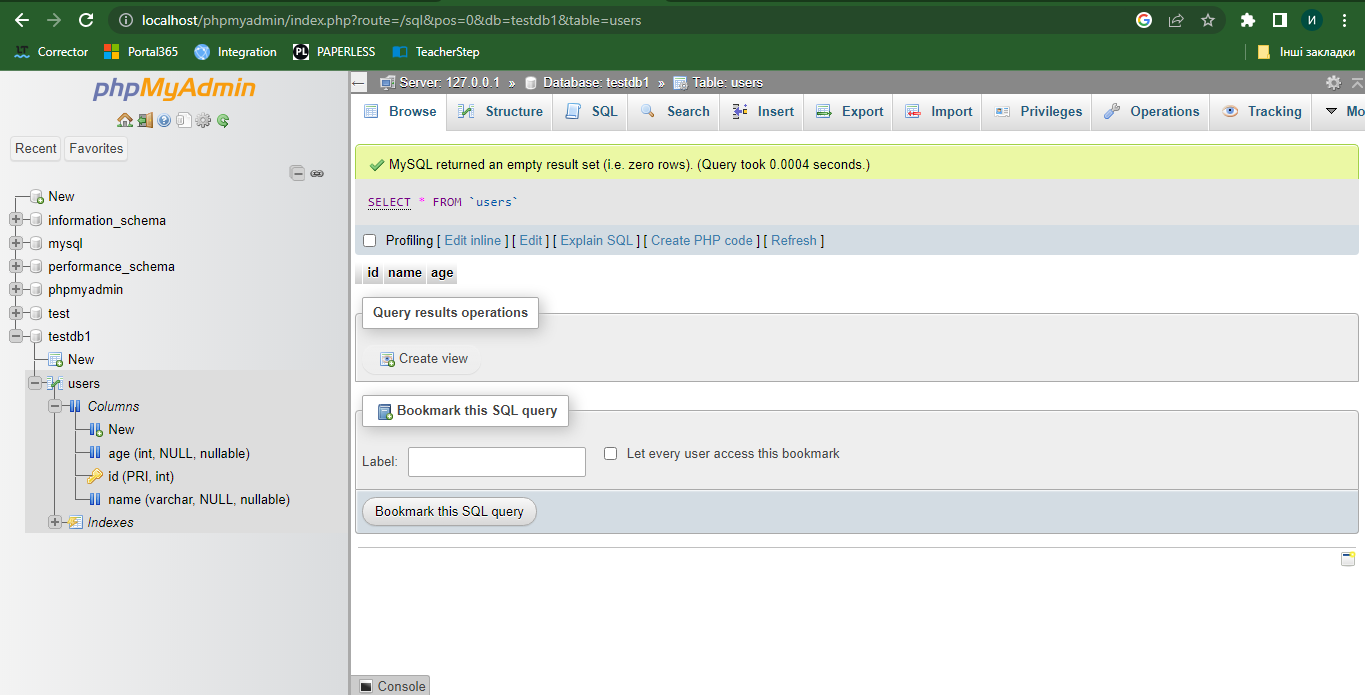
?>

Result:



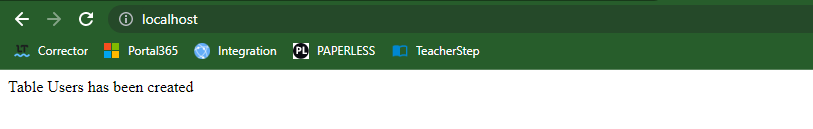
Phpmyadmin:





Обратите внимание, что по сравнению с предыдущим примером здесь в строке подключения указана база данных, в которой создается таблица: "mysql:host=127.0.0.1;dbname=testdb1"

И после успешного выполнения запрос мы увидим в браузере сообщение об создании таблицы:



Добавление данных в PDO и параметризация запросов

Для добавления данных в БД MySQL применяется sql-команда INSERT, которая имеет следующий синтаксис:

INSERT INTO название\_таблицы (столбец1, столбец2, столбецN) VALUES ( значение1, значение2, значениеN)

Данная команда также выполняется методом exec() объекта PDO. Стоит отметить, что для sql-команд INSERT, UPDATE и DELETE метод exec() возвращает количество затронутных командой строк (добавленных, измененных или удаленных). Таким образом, мы можем узнать сколько строк было добавлено.

Сначала рассмотрим простейшее добавление одного объекта в БД. Для примера возьмем созданную в прошлой теме базу данных "testdb1" и созданную в ней таблицу Users со следующим определением:

CREATE TABLE Users (id INTEGER AUTO\_INCREMENT PRIMARY KEY, name VARCHAR(30), age INTEGER)

И для добавления определим следующий скрипт PHP:

<?php

try {

    $conn = new PDO("mysql:host=localhost;dbname=testdb1", "root", "");

    // SQL-выражение для добавления данных

    $sql = "INSERT INTO Users (name, age) VALUES ('Tom', 37)";

    $affectedRowsNumber = $conn->exec($sql);

    echo "В таблицу Users добавлено строк: $affectedRowsNumber";

}

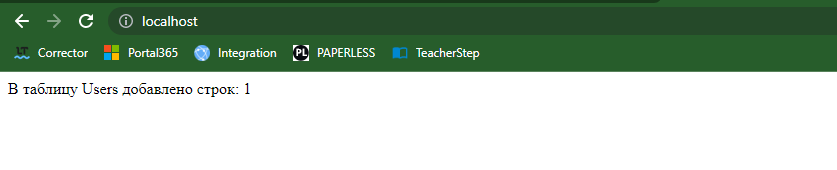
catch (PDOException $e) {

    echo "Database error: " . $e->getMessage();

}

?>

Можно заменить 127.0.0.1 на localhost (этот момент не у всех работает, причин может быть очень много разных например в php.ini или в конфиг файле бд)



Команда на добавление здесь выглядит следующим образом:

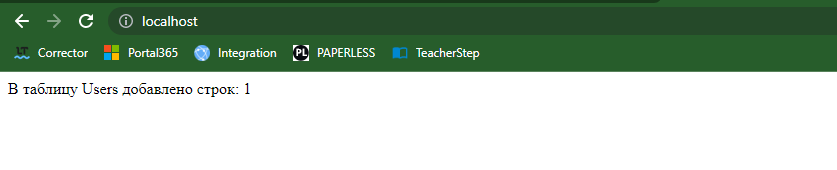
<?php

"INSERT INTO Users (name, age) VALUES ('Tom', 37)"

?>

То есть в столбец name добавляется строка "Tom", а в столбец age - число 37. Для столбца id не добавляется никакого значения, потому что при создании таблицы для него указан параметр AUTO\_INCREMENT - то есть значение этого столбца у каждой добавляемой строки будет автоматически увеличиваеться по сравнению с предыдущей на единицу.

При добавлении мы получаем количество добавленных строк в переменую $affectedRowsNumber и затем выводим ее значение в браузере. Поэтому при успешном добавлении мы увидим



Множественное добавление

Также мы можем добавить сразу несколько объектов:

<?php

try {

    $conn = new PDO("mysql:host=localhost;dbname=testdb1", "root", "");

    $sql = "INSERT INTO Users (name, age) VALUES

            ('Sam', 41),

            ('Bob', 29),

            ('Alice', 32)";

    $affectedRowsNumber = $conn->exec($sql);

    echo "В таблицу Users добавлено строк: $affectedRowsNumber";

}

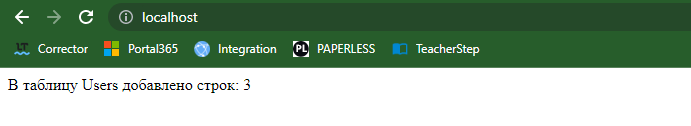
catch (PDOException $e) {

    echo "Database error: " . $e->getMessage();

}

?>

Здесь в таблицу добавляется три строки. Соответственно в браузере мы увидим:



Добавление данных из формы HTML

В большинстве случаев добавляемые данные будут приходить из вне, например, присылаться в запросе пользователя. Рассмотрим добавление данных, отправленных из формы HTML. Для этого определим следующий скрипт:

<!DOCTYPE html>

<html>

<head>

<title>Example PHP&MySql</title>

<meta charset="utf-8" />

</head>

<body>

<?php

if (isset($\_POST["username"]) && isset($\_POST["userage"])) {

    $username = $\_POST["username"];

    $userage = $\_POST["userage"];

    try {

        $conn = new PDO("mysql:host=localhost;dbname=testdb1", "root", "");

        $sql = "INSERT INTO Users (name, age) VALUES ('$username', $userage)";

        $affectedRowsNumber = $conn->exec($sql);

        // если добавлена как минимум одна строка

        if($affectedRowsNumber > 0 ){

            echo "Data successfully added: name=$username  age= $userage";

        }

    }

    catch (PDOException $e) {

        echo "Database error: " . $e->getMessage();

    }

}

?>

<h3>Create a new User</h3>

<form method="post">

    <p>User Name:

    <input type="text" name="username" /></p>

    <p>User Age:

    <input type="number" name="userage" /></p>

    <input type="submit" value="Save">

</form>

</body>

</html>

Здесь мы проверяем, пришли ли с сервера данные в POST-запросе, которые имеют ключи "username" и "userage":

<?php

if (isset($\_POST["username"]) && isset($\_POST["userage"])) {

?>

Если эти данные имеются, то есть был отправлен post-запрос с данными на добавление, то мы получаем эти данные в переменные и добавляем их в бд.

<?php

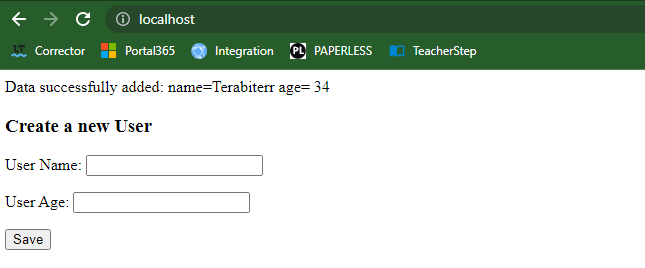
$sql = "INSERT INTO Users (name, age) VALUES ('$username', $userage)";

?>

Если была добавлена строка, то есть метод exec() возвратил число больше нуля, то выводим пользователю соответствующее сообщение.

После кода php собственно определена форма на добавление данных с помощью post-запроса.

Запустим скрипт. Введем в форму данные и нажмем на кнопку:



И данные будут добавлены. Вроде все хорошо. Однако здесь есть большой недостаток.

Параметризация запросов

Недостаток выше приведенного скрипа заключается в том, что мы никак не констролируем присылаемые данные и сохраняем их в базу данных как есть. Что несет потенциальную угрозу безопасности, особенно при добавлении строк типа "; DELETE FROM `Users`; --. Кроме того, в ряде случае может быть проблематично добавить даже безопасные данные, например, строку, которая содержит одинарную кавычку, типа "Tom O'Brian".

Для решения этих проблем PDO предлагает параметризацию запросов с помощью применения заранее подготовленных выражений - prepared statement. Выражения prepared statement вместо жестко установленных значений или переменных принимают параметры, которые не привязаны к конкретным значениям. Эти выражения prepared statement посылаются серверу базы данных до того, как станут известны используемые данные, что позволяет серверу приготовить их к выполнению, но при этом они не выполняются. А когда пользователь присылает данные - параметры заменяются пришедшими данными, и выражение prepared statement выполняется.

Перепишем предыдущий пример с использованием параметров:

<!DOCTYPE html>

<html>

<head>

<title>Example PHP&MySql</title>

<meta charset="utf-8" />

</head>

<body>

<?php

if (isset($\_POST["username"]) && isset($\_POST["userage"])) {

    try {

        $conn = new PDO("mysql:host=localhost;dbname=testdb1", "root", "mypassword");

        $sql = "INSERT INTO Users (name, age) VALUES (:username, :userage)";

        // определяем prepared statement

        $stmt = $conn->prepare($sql);

        // привязываем параметры к значениям

        $stmt->bindValue(":username", $\_POST["username"]);

        $stmt->bindValue(":userage", $\_POST["userage"]);

        // выполняем prepared statement

        $affectedRowsNumber = $stmt->execute();

        // если добавлена как минимум одна строка

        if($affectedRowsNumber > 0 ){

            echo "Data successfully added: name=" . $\_POST["username"] ."  age= " . $\_POST["userage"];

        }

    }

    catch (PDOException $e) {

        echo "Database error: " . $e->getMessage();

    }

}

?>

<h3>Create a new User</h3>

<form method="post">

    <p>User Name:

    <input type="text" name="username" /></p>

    <p>User Age:

    <input type="number" name="userage" /></p>

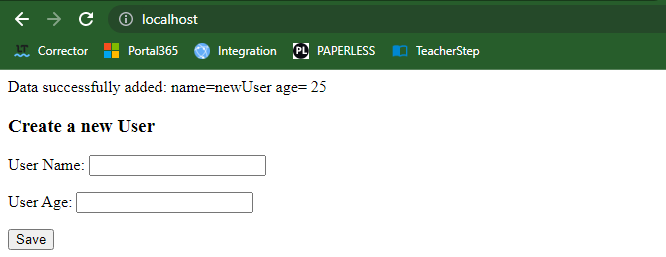
    <input type="submit" value="Save">

</form>

</body>

</html>

Result:



В SQL-выражении теперь применяются параметры:

<?php

$sql = "INSERT INTO Users (name, age) VALUES (:username, :userage)";

?>

:username и :userage - это названия параметров. Причем они начинаются с символа двоеточия :.

Само выражение prepared statement создается с помощью метода prepare() объекта PDO, в который передается выполняемая sql-команда:

<?php

$stmt = $conn->prepare($sql);

?>

Фактически здесь создается объект PDOStatement, который сохраняется в переменную $stmt.

Чтобы связать параметр с конкретным значением у объекта PDOStatement вызывается метод bindValue(). Первый параметр этого метода - собственно параметр из sql-команды, а второй параметр - передаваемое ему значение.

<?php

$stmt->bindValue(":username", $\_POST["username"]);

?>

Так, в данном случае параметр :username привязывается к значению из $\_POST["username"]

Причем привязка может производиться и к конкретным значениям и обычным переменным, например:

<?php

$user = "Tom"

// привязка к переменной $user

$stmt->bindValue(":username", $user);

?>

Для выполнения sql-выражения у объекта PDOStatement вызывается метод execute(), который для команды INSERT возвращает число добавленных строк.

Передача значений параметрам через массив по имени

В примере выше для параметризации применялся метод bindValue():

<?php

$sql = "INSERT INTO Users (name, age) VALUES (:username, :userage)";

$stmt = $conn->prepare($sql);

// привязываем параметры к значениям

$stmt->bindValue(":username", $\_POST["username"]);

$stmt->bindValue(":userage", $\_POST["userage"]);

// выполняем prepared statement

$affectedRowsNumber = $stmt->execute();

?>

Но есть и другой способ привязки параметров к значениям - мы можем передать в метод execute() параметры и их значения в виде ассоциативного массива:

<?php

$sql = "INSERT INTO Users (name, age) VALUES (:username, :userage)";

$stmt = $conn->prepare($sql);

// через массив передаем значения параметрам по имени

$rowsNumber = $stmt->execute(array(":username" => $\_POST["username"], ":userage" => $\_POST["userage"]));

?>

В этом случае названия параметров являются ключами.

Передача значений параметрам через массив по позиции

Третий способ привязки значений к параметрам представляет передачу значений по позиции:

<?php

$sql = "INSERT INTO Users (name, age) VALUES (?, ?)";

$stmt = $conn->prepare($sql);

// через массив передаем значения параметрам по позиции

$rowsNumber = $stmt->execute(array($\_POST["username"], $\_POST["userage"]));

?>

В этом случае вместо названий параметров применяются знаки вопроса ?. Для передачи этим параметрам значений в метод execute() также передается массив. Первое значение массива привязывается к первому параметру (условно добавляется вместо первого знака вопроса), второе значение привязывается ко второму параметру и т.д.

Получение данных в PDO

На уровне кода SQL получение данных осуществляется с помощью команды SELECT. Например, получение всех данных из таблицы Users:

SELECT \* FROM Users

В библиотеке pdo для получения данных у объекта PDO вызывается метод query(), в который передается команда SQL. Метод query() возвращает объект PDOStatement, который представляет набор всех полученных из базы данных строк.

<?php

$sql = "SELECT \* FROM Users";

$result = $conn->query($sql);

?>

Получив объект PDOStatement, мы можем извлечь данные. В частности, его метод fetch() при первом обращении первую строку (если в наборе есть строки):

<?php

$row = $result->fetch();

?>

При последующих обращениях метод fetch() возвращает следующие строки, пока в наборе не останется строк. Если строк в наборе больше нет, то метод возвращает false. Поэтому для получения всех строк удобно использовать циклы. Например, цикл while:

<?php

while($row = $result->fetch()){

    // обработка строк

}

?>

Таким образом, при каждой итерации цикл while будет получать новую строку из набора в переменную $row, пока метод $result->fetch() не возвратит false - после чего произойдет выход из цикла.

Строка возвращается в виде ассоциативного массива, где отдельные значения - это столбцы строки, а ключи этих значений - названия столбцов таблицы. Например, получение значения столбца "name" в переменную:

<?php

while($row = $result->fetch()){

    $username = $row["name"];

    // операции с $username

}

?>

Вместо цикла while можно использовать цикл for/foreach. Например:

<?php

foreach($result as $row){

    $username = $row["name"];

    // операции с $username

}

?>

Здесь явным образом не вызывается метод $result->fetch(). Формально мы просто перебираем переменную $result как обычный массив, также помещая каждую строку в переменную $row.

Теперь все объединим и получим данные из таблицы Users из прошлых тем, которая имеет следующее определение:

CREATE TABLE Users (id INTEGER AUTO\_INCREMENT PRIMARY KEY, name VARCHAR(30), age INTEGER)

Для этого определим следующий простенький скрипт:

<?php

try {

    $conn = new PDO("mysql:host=localhost;dbname=testdb1", "root", "mypassword");

    $sql = "SELECT \* FROM Users";

    $result = $conn->query($sql);

    echo "<table><tr><th>Id</th><th>Name</th><th>Age</th></tr>";

    while($row = $result->fetch()){

        echo "<tr>";

            echo "<td>" . $row["id"] . "</td>";

            echo "<td>" . $row["name"] . "</td>";

            echo "<td>" . $row["age"] . "</td>";

        echo "</tr>";

    }

    echo "</table>";

}

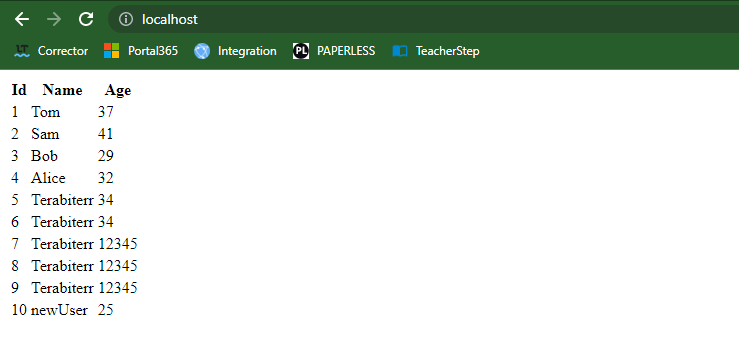
catch (PDOException $e) {

    echo "Database error: " . $e->getMessage();

}

?>

В данном случае полученные данные будут выводиться в таблицу, создаваемую элементом <table>:



Фильтрация данных в PDO

В прошлой статье применялся метод query() для получения всех данных из БД. Но что, если нам надо получить не все, а какие-то определенные данные, которые овечают некоторому критерию, иными словами, произвести фильтрацию данных. Например, возьмем использовавшуюся в прошлых темах таблицу Users:

CREATE TABLE Users (id INTEGER AUTO\_INCREMENT PRIMARY KEY, name VARCHAR(30), age INTEGER)

Она имеет столбец id, и мы хотим получить определенный объект по id. На первый взгляд мы могли бы определить следующий код:

<?php

$sql = "SELECT \* FROM Users WHERE id = 1";

$result = $conn->query($sql);

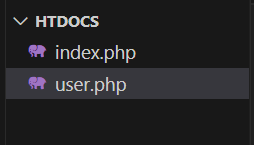
?>

Для фильтрации команде SELECT передается выражение WHERE, которая принимает названия столбцов их значения в качестве критерия фильтрации. То есть, здесь мы получаем строке, где

id = 1.

Однако если данные для фильтрации приходят извне, например, значение для столбца id, то опять же, как и в случае с добавлением, мы сталкиваемся с потенциальной уязвимостью кода. И также, как и при добавлении, в этом случае лучше использовать параметризацию и prepared statements.

Например, мы хотим получать в GET-запросе значение для id и по нему получть из базы данных нужные данные. Определим для этого следующий скрипт user.php:



<!DOCTYPE html>

<html>

<head>

<title>Test db PDO</title>

<meta charset="utf-8" />

</head>

<body>

<?php

if(isset($\_GET["id"]))

{

    try {

        $conn = new PDO("mysql:host=localhost;dbname=testdb1", "root", "");

        $sql = "SELECT \* FROM Users WHERE id = :userid";

        $stmt = $conn->prepare($sql);

        // привязываем значение параметра :userid к $\_GET["id"]

        $stmt->bindValue(":userid", $\_GET["id"]);

        // выполняем выражение и получаем пользователя по id

        $stmt->execute();

        if($stmt->rowCount() > 0){

            foreach ($stmt as $row) {

              $username = $row["name"];

              $userage = $row["age"];

              echo "<div>

                    <h3>Информация о пользователе</h3>

                    <p>Имя: $username</p>

                    <p>Возраст: $userage</p>

                </div>";

            }

        }

        else{

            echo "Пользователь не найден";

        }

    }

    catch (PDOException $e) {

        echo "Database error: " . $e->getMessage();

    }

}

?>

</body>

</html>

Для выполнения запроса к БД вначале создаем prepared statement, которое использует параметр userid, привязанный к значению $\_GET["id"]:

<?php

$sql = "SELECT \* FROM Users WHERE id = :userid";

$stmt = $conn->prepare($sql);

$stmt->bindValue(":userid", $\_GET["id"]);

Далее у полученного объекта PDOStatement вызываем метод execute(), который выполняет запрос к бд:

<?php

$stmt->execute();

После выполнения команды SELECT этот объект содержит полученные из БД данные, которые мы можем перебрать с помощью цикла:

<?php

if($stmt->rowCount() > 0){

    foreach ($stmt as $row) {

        $username = $row["name"];

        $userage = $row["age"];

    }

При этом с помощью метода rowCount() мы можем узнать количество возвращенных строк. Получение данных столбцов строки производится как и было описано выше для простого запроса SELECT. Получив данные столбцов в переменные, мы можем затем что-то с ними сделать, например, вывести их значения на страницу.

Чтоб было проще обращаться к скрипту user.php и передавать ему id, определим скрипт index.php, который будет выводить список объектов:

<!DOCTYPE html>

<html>

<head>

<title>Test PDO</title>

<meta charset="utf-8" />

</head>

<body>

<h2>Список пользователей</h2>

<?php

try {

    $conn = new PDO("mysql:host=localhost;dbname=testdb1", "root", "");

    $sql = "SELECT id, name FROM Users";

    $result = $conn->query($sql);

    echo "<table><tr><th>Имя</th><th></th></tr>";

    foreach($result as $row){

        echo "<tr>";

            echo "<td>" . $row["name"] . "</td>";

            echo "<td><a href='user.php?id=" . $row["id"] . "' >Посмотреть</a></td>";

        echo "</tr>";

    }

    echo "</table>";

}

catch (PDOException $e) {

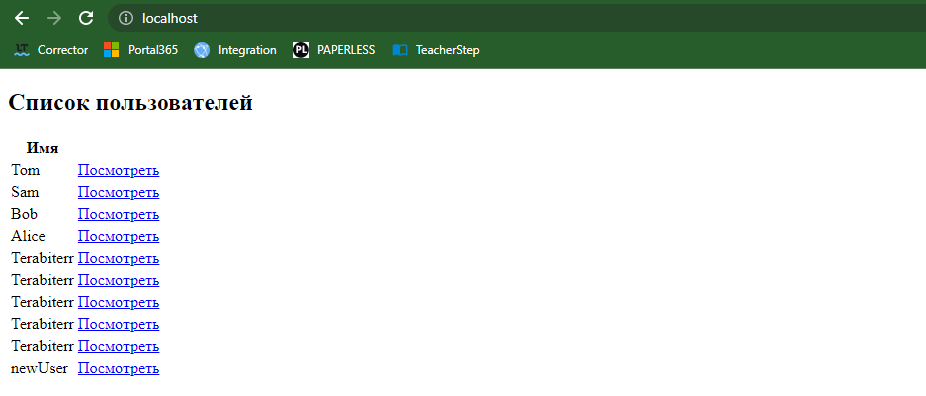
    echo "Database error: " . $e->getMessage();

}

?>

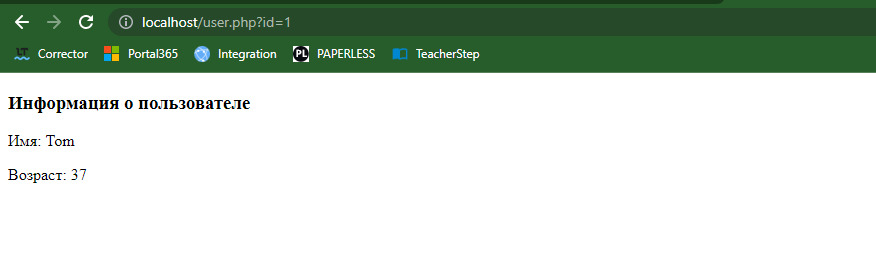
</body>

</html>



Здесь все объекты из базы данных выводятся в таблицу, где второй столбец содержит ссылку на скрипт user.php, которому передается соответствующее значение id. В итоге по нажатию на эту ссылку мы перейдем к описанию объекта по id:

При клике на пользователя обращаемся к файлу user.php



Обновление данных в PDO

Для обновления применяется sql-команда UPDATE:

UPDATE Таблица

SET столбец1 = значение1, столбец2 = значение2,...

WHERE столбец = значение

В библиотеке pdo для обновления данных может применяться тот же метод exec() объекта PDO, который применяется при добавлении. Например, возьмем использованную в прошлых темах таблицу Users со следующим определением:

CREATE TABLE Users (id INTEGER AUTO\_INCREMENT PRIMARY KEY, name VARCHAR(30), age INTEGER)

Изменим в этой таблице поле age для строки, которая имеет id = 1:

<?php

try {

    $conn = new PDO("mysql:host=localhost;dbname=testdb1", "root", "");

    $sql = "UPDATE Users SET age = 22 WHERE id = 1";

    $affectedRowsNumber = $conn->exec($sql);

    echo "Обновлено строк: $affectedRowsNumber";

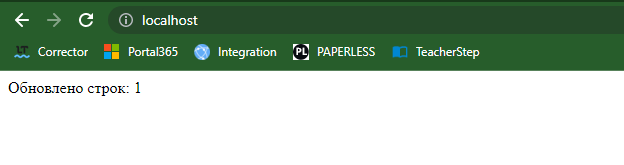
}

catch (PDOException $e) {

    echo "Database error: " . $e->getMessage();

}

?>



Результат метода $conn->exec() в данном случае количество обновленных строк.

Однако если данные на обновление приходят извне, то мы опять как и при добавлении сталкиваемся с потенциальной уязвимостью подобного подхода. Поэтому в этом случаае опять же лучше использовать параметризацию и prepared statements.

Отправка данных из формы и обновление

Сначала определим файл index.php, который будет выводить список пользователей:

<!DOCTYPE html>

<html>

<head>

<title>Test PDO</title>

<meta charset="utf-8" />

</head>

<body>

<h2>Список пользователей</h2>

<?php

try {

    $conn = new PDO("mysql:host=localhost;dbname=testdb1", "root", "");

    $sql = "SELECT \* FROM Users";

    $result = $conn->query($sql);

    echo "<table><tr><th>Имя</th><th>Возраст</th><th></th></tr>";

    foreach($result as $row){

        echo "<tr>";

            echo "<td>" . $row["name"] . "</td>";

            echo "<td>" . $row["age"] . "</td>";

            echo "<td><a href='update.php?id=" . $row["id"] . "'>Обновить</a></td>";

        echo "</tr>";

    }

    echo "</table>";

}

catch (PDOException $e) {

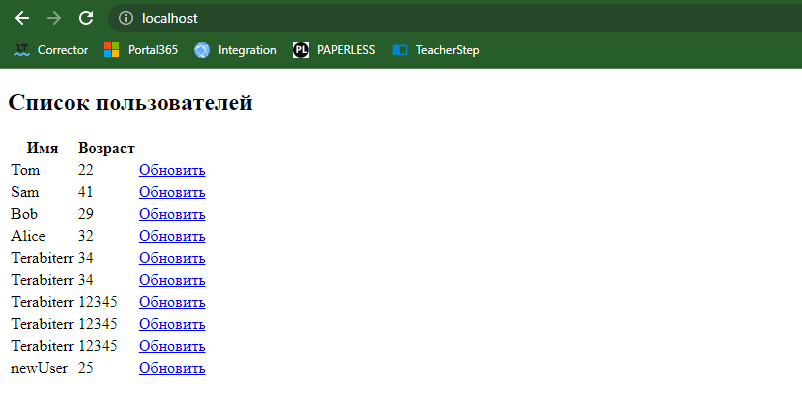
    echo "Database error: " . $e->getMessage();

}

?>

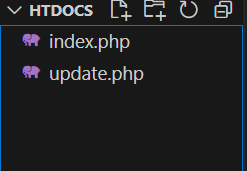
</body>

</html>



Здесь используется команда SELECT, которая получает всех пользователей из таблицы Users. В таблице третий столбец хранит ссылку на скрипт update.php, который мы далее создадим и которому передается параметр id с идентификатором пользователя, которого надо изменить.

Теперь определим файл update.php для редактирования пользователей:



<?php

try {

    $conn = new PDO("mysql:host=localhost;dbname=testdb1", "root", "");

}

catch (PDOException $e) {

    die("Database error: " . $e->getMessage());

}

?>

<!DOCTYPE html>

<html>

<head>

<title>Test PDO</title>

<meta charset="utf-8" />

</head>

<body>

<?php

// если запрос GET

if($\_SERVER["REQUEST\_METHOD"] === "GET" && isset($\_GET["id"]))

{

    $userid = $\_GET["id"];

    $sql = "SELECT \* FROM Users WHERE id = :userid";

    $stmt = $conn->prepare($sql);

    $stmt->bindValue(":userid", $userid);

    // выполняем выражение и получаем пользователя по id

    $stmt->execute();

    if($stmt->rowCount() > 0){

        foreach ($stmt as $row) {

            $username = $row["name"];

            $userage = $row["age"];

        }

        echo "<h3>Обновление пользователя</h3>

                <form method='post'>

                    <input type='hidden' name='id' value='$userid' />

                    <p>Имя:

                    <input type='text' name='name' value='$username' /></p>

                    <p>Возраст:

                    <input type='number' name='age' value='$userage' /></p>

                    <input type='submit' value='Сохранить' />

            </form>";

    }

    else{

        echo "Пользователь не найден";

    }

}

elseif (isset($\_POST["id"]) && isset($\_POST["name"]) && isset($\_POST["age"])) {

    $sql = "UPDATE Users SET name = :username, age = :userage WHERE id = :userid";

    $stmt = $conn->prepare($sql);

    $stmt->bindValue(":userid", $\_POST["id"]);

    $stmt->bindValue(":username", $\_POST["name"]);

    $stmt->bindValue(":userage", $\_POST["age"]);

    $stmt->execute();

    header("Location: index.php");

}

else{

    echo "Некорректные данные";

}

?>

</body>

</html>

Весь код обновления структурно делится на две части. В первой части мы обрабатываем запрос Get. Когда пользователь нажимает на ссылку "Обновить" на странице index.php, то отправляется запрос GET, в котором передается id редактируемого пользователя. Поэтому мы сначала смотрим, представляет ли запрос GET-запрос и имеет ли он параметр id.

<?php

if($\_SERVER["REQUEST\_METHOD"] === "GET" && isset($\_GET["id"]))

И если это запрос GET, то нам надо вывести данные редактируемого пользователя в поля формы. Для этого отправляем базе данных запрос

<?php

$sql = "SELECT \* FROM Users WHERE id = :userid";

$stmt = $conn->prepare($sql);

$stmt->bindValue(":userid", $userid);

$stmt->execute();

Далее получаем полученные данные и, если они имеются, выводим их в поля формы. Таким образом, пользователь увидит на форме данные редактируемого объекта.

Вторая часть скрипта представляет обработку POST-запроса - когда пользователь нажимает на кнопку на форме, то будет отправляться POST-запрос с отправленными данными. Мы получаем эти данные и отправляем базе данных команду UPDATE с этими данными, используя при этом параметризацию запроса:

<?php

$sql = "UPDATE Users SET name = :username, age = :userage WHERE id = :userid";

$stmt = $conn->prepare($sql);

$stmt->bindValue(":userid", $\_POST["id"]);

$stmt->bindValue(":username", $\_POST["name"]);

$stmt->bindValue(":userage", $\_POST["age"]);

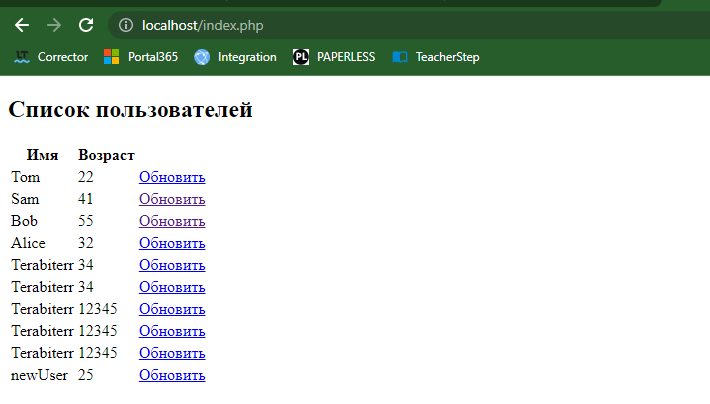
$stmt->execute();

После выполнения запроса к БД перенаправляем пользователя на скрипт index.php с помощью функции

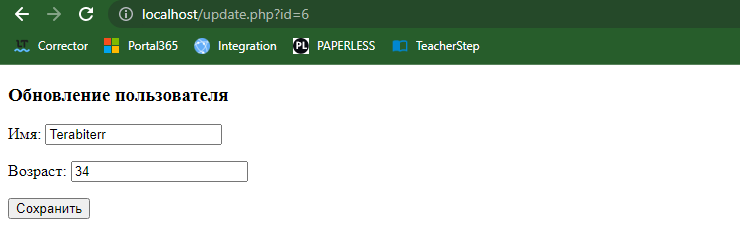
<?php

header("Location: index.php");

Таким образом, пользователь обращается к скрипту index.php, видит таблицу с данными и нажимает на ссылку "Обновить" в одной из строк.



Можем нажать на кнопку обновить и обновить результаты:



Удаление данных в PDO

Для удаления данных применяется sql-команда DELETE:

DELETE FROM Таблица

WHERE столбец = значение

Для удаления данных также может применяться метод exec() объекта PDO. Например, возьмем использованную в прошлых темах таблицу Users со следующим определением:

CREATE TABLE Users (id INTEGER AUTO\_INCREMENT PRIMARY KEY, name VARCHAR(30), age INTEGER)

Удалим из этой таблицы строку, где id = 5:

<?php

try {

    $conn = new PDO("mysql:host=localhost;dbname=testdb1", "root", "");

    $sql = "DELETE FROM Users WHERE id = 5";

    $affectedRowsNumber = $conn->exec($sql);

    echo "Удалено строк: $affectedRowsNumber";

}

catch (PDOException $e) {

    echo "Database error: " . $e->getMessage();

}

?>

Результат метода $conn->exec() в данном случае количество удаленых строк. Однако опять же поскольку значение столбца, на основе которого происходит удаление, нередко приходит извне, то в этом случае лучше использовать параметризацию.

Итак, определим для вывода всех объектов из БД скрипт index.php:

<!DOCTYPE html>

<html>

<head>

<title>Test PDO</title>

<meta charset="utf-8" />

</head>

<body>

<h2>Список пользователей</h2>

<?php

try {

    $conn = new PDO("mysql:host=localhost;dbname=testdb1", "root", "mypassword");

    $sql = "SELECT \* FROM Users";

    $result = $conn->query($sql);

    echo "<table><tr><th>Имя</th><th>Возраст</th><th></th></tr>";

    foreach($result as $row){

        echo "<tr>";

            echo "<td>" . $row["name"] . "</td>";

            echo "<td>" . $row["age"] . "</td>";

            echo "<td><form action='delete.php' method='post'>

                        <input type='hidden' name='id' value='" . $row["id"] . "' />

                        <input type='submit' value='Удалить'>

                    </form></td>";

        echo "</tr>";

    }

    echo "</table>";

}

catch (PDOException $e) {

    echo "Database error: " . $e->getMessage();

}

?>

</body>

</html>

В таблицы для каждой строки определена форма, которая посылает данные в POST-запросе скрипту delete.php. Чтобы передать в delete.php идентификатор удаляемого объекта, на форме определено скрытое поле для хранения id объекта.

Обратите внимание, что в данном случае применяется не ссылка для удаления типа

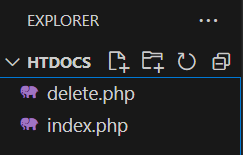
<a href="http://адрес\_нашего\_сайта/delete.php?id=1">Удалить<a/>

которая оправляет данные в GET-запросе, а именно форма, которая отправляет данные в POST-запросе. Почему? Подобные GET-запросы потенциально небезопасны. Допустим, нам пришло электронное письмо, в которое была внедрена картинка посредством тега:

<img src="http://адрес\_нашего\_сайта/delete.php?id=1" />

В итоге при открытии письма 1-я запись в таблице может быть удалена. Уязвимость касается не только писем, но может проявляться и в других местах, но смысл один - GET-запрос к скрипту, который удаляет данные, несет потенциальную уязвимость.

Теперь определим сам скрипт delete.php, который будет выполнять удаление:



<?php

if(isset($\_POST["id"]))

{

    try {

        $conn = new PDO("mysql:host=localhost;dbname=testdb1", "root", "");

        $sql = "DELETE FROM Users WHERE id = :userid";

        $stmt = $conn->prepare($sql);

        $stmt->bindValue(":userid", $\_POST["id"]);

        $stmt->execute();

        header("Location: index.php");

    }

    catch (PDOException $e) {

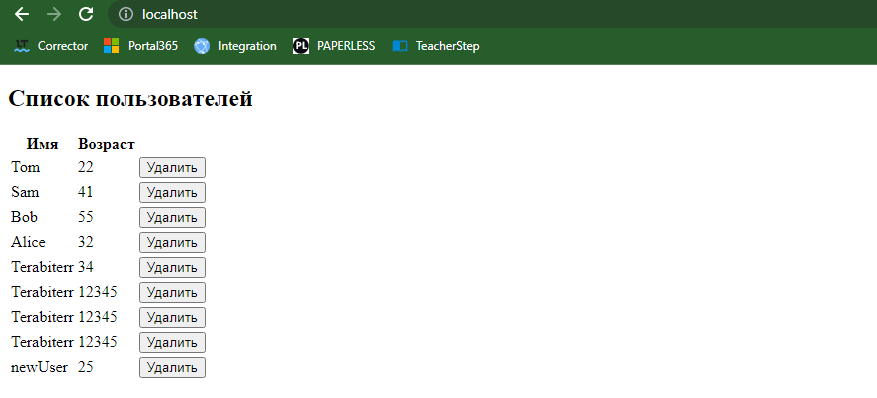
        echo "Database error: " . $e->getMessage();

    }

}

?>

В данном случае скрипт получает через POST-запрос значение id и по этому идентификатору выполняет удаление. После чего происходит переадресация на скрипт index.php.



Практика PDO

Пример на PHP с использованием PDO для CRUD операций и вывода результатов в браузер:

Устанавливаются методами **PDOObj->setAttribute() или PDOStatement->setAttribute();** или вообще указываются как параметры при создании нового объекта PDO:

[?](http://php-zametki.ru/php-prodvinutym/58-pdo-konstanty-atributy.html)

|  |  |
| --- | --- |
| 1  2  3 | $DB = new PDO('mysql:dbname=my\_test;host=localhost',                'root','----',                array(PDO::MYSQL\_ATTR\_INIT\_COMMAND => 'SET NAMES \'UTF8\'')); |

**PDO::ATTR\_AUTOCOMMIT ( 0 )**

Доступно для драйверов: OCI, Firebird и MySQL Если её значение - FALSE, PDO пытается отключить автоматическую фиксацию так, что бы подключение автоматом начало транзакцию.

**PDO::ATTR\_PREFETCH ( 1 )**

Установка максимального размера данных возвращаемых запросом. Позволяет Вам балансировать скорость и использование памяти для Вашего приложения. Не все драйвера и не всех баз данных поддерживают данную установку. При установке большого значения – увеличится скорость работы, но за счет более высокого потребления памяти.

**PDO::ATTR\_TIMEOUT ( 2 )**

Устанавливает таймаут в секундах для соединения с базой данных.

**PDO::ATTR\_ERRMODE ( 3 )**

Установка способа обработки ошибок:

PDO::ERRMODE\_SILENT (0) – Это режим по умолчанию. PDO будет просто устанавливать код ошибки. Для получения более подробной информации вам потребуется обращаться к методам: PDO::errorCode() и PDO::errorInfo(), если ошибку вызвал объект класса PDO, или к PDOStatement::errorCode() или PDOStatement::errorInfo(), если косяк в объекте класса PDOStatement.

PDO::ERRMODE\_WARNING (1) - В дополнение к установкам вывода ошибок, PDO будет выводить традиционные сообщение E\_WARNING. Этот параметр полезен во время отладки - тестирования, если вы просто хотите видеть возникшие проблемы, не прерывая поток приложения.

PDO::ERRMODE\_EXCEPTION (2) - В дополнение к установкам уровня вывода ошибок, PDO выбросит PDOException - исключение и задаст его свойства, так чтобы отразить код ошибки и сведения об ошибке. Этот параметр также полезен при отладке, так как он будет эффективно "останавливать" сценарий в точке ошибки, очень быстро указывая «пальцем» на потенциальные проблемы в вашем коде (Кстати, транзакции автоматически откатываются, если было выброшено исключение).

Режим исключений также полезен ещё и, потому, что вы можете структурировать обработку ошибок более четко, чем при традиционном стиле PHP - предупреждений и с меньшим количеством кода / вложенности, чем при дефолтном режиме, где требуется явно городить проверки возвращаемого значения при каждом обращении к базе. Вообще работа с исключениями это отдельная тема, о ней позже. На закуску: простая конструкция отлова PDO исключений:

[?](http://php-zametki.ru/php-prodvinutym/58-pdo-konstanty-atributy.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | try  {    // Здесь потенциально опасный код... использующий PDO    $dbh = new PDO($dsn, $user, $password);    $dbh->setAttribute(PDO::ATTR\_ERRMODE, PDO::ERRMODE\_EXCEPTION);  }  //Какая прелесть: PHP5 позволяет указать некоторые типы входных параметров!  //Здесь мы поставили ловушку на отлов экземпляров класса PDOException:  catch(PDOException $e)  {    // Здесь выводим сообщение на экран:    echo ‘Соединение с БД, с треском провалилось!          Посмотри что ты наделал: ’.$e->getMessage();  } |

**PDO::ATTR\_SERVER\_VERSION ( 4 )**

**Только для чтения.**

Выводит информацию о версии сервера базы данных, к которому подключен PDO.

Типа того: 5.0.51a

**PDO::ATTR\_CLIENT\_VERSION ( 5 )**

**Только для чтения;**

Выводит информацию о версии клиентских библиотек, которые использует драйвер PDO.

Тоже, типа того: 5.0.51a

**PDO::ATTR\_SERVER\_INFO ( 6 )**

**Только для чтения;**

Выводит краткую метаинформацию о сервере базы данных, к которому подключен PDO.

Типа того: Uptime: 26804 Threads: 1 Questions: 1185 Slow queries: 2 Opens: 224 Flush tables: 1 Open tables: 1 Queries per second avg: 0.44

**PDO::ATTR\_CONNECTION\_STATUS ( 7 )**

**Только для чтения;**

Статус соединения.

Типа того: localhost via TCP/IP

**PDO::ATTR\_CASE ( 8 )**

Приводит имена столбцов выборки к конкретному виду/регистру, определенному константами:

PDO::CASE\_LOWER (2): Приводит имена к нижнему регистру.  
PDO::CASE\_NATURAL (0): Оставляет без изменений.  
PDO::CASE\_UPPER (1): Приводит имена к верхнему регистру.

**PDO::ATTR\_DRIVER\_NAME ( 16 )**

Возвращает имя драйвера.

Строка типа: mysql

Пример , как можно использовать PDO::ATTR\_DRIVER\_NAME

[?](http://php-zametki.ru/php-prodvinutym/58-pdo-konstanty-atributy.html)

|  |  |
| --- | --- |
| 1  2  3  4 | if ($db-> getAttribute (PDO::ATTR\_DRIVER\_NAME) == 'mysql')  {    echo "Запущен драйвер mysql; здесь какие нибудь специфичные mysql функции\n";  } |

**PDO::ATTR\_PERSISTENT ( 12 )**

Запрашивает постоянное подключение, вместо того, чтобы при каждом запросе создавать новое подключение. Если вы включили эту опцию, то не сможете использовать PDO::ATTR\_STATEMENT\_CLASS и наоборот.

**PDO::ATTR\_STATEMENT\_CLASS ( 13 )**

С помощью этой константы (желательно сразу после получения объекта PDO) можно указать свой класс, который должен расширять PDOStatement:

$this->setAttribute( PDO::ATTR\_STATEMENT\_CLASS, array('DB\_statement', array($param1, [$param2, [...]])) );

В качестве второго параметра PDOStatement::setAttribute() вы должны будете указать индексный массив, первое значение которого - строковое представление имени вашего класса, расширяющего PDOStatement, второе - опять же массив в котором вы можете указать параметры для конструктора Вашего класса.

Кстати, конструктор вашего класса, расширяющего PDOStatement, обязательно **должен быть как минимум защищённым – protected**

Ниже пример реализации своего функционала общения с БД, который имеет счётчик запросов:

[?](http://php-zametki.ru/php-prodvinutym/58-pdo-konstanty-atributy.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | class DB extends PDO  {    public static $query\_count = 0; //счётчик запросов      public function \_\_construct(/\*string\*/$dsn, /\*string\*/$user, /\*string\*/$pass, $driver\_option = array())    {      parent::\_\_construct($dsn,$user,$pass,$driver\_option);      $this->setAttribute(PDO::ATTR\_STATEMENT\_CLASS, array('DB\_statement', array($this)));    }    public function DBexec($statement)    {      $rows = parent::exec($statement);      self::$query\_count ++;      return $rows;    }    public function DBquery($sql)    {      self::$query\_count ++;      return parent::query($sql);    }  }    class DB\_statement extends PDOStatement  {    protected $db;      protected function \_\_construct(DB $db)    {      $this->db = $db;    }    public function DBexecute($parameters = array())    {      DB::$query\_count ++;      return parent::execute($parameters);    }  } |

**PDO::ATTR\_FETCH\_CATALOG\_NAMES ( 15 )**

**Не поддерживается драйвером mysql!** Добавляет имя каталога к каждому имени столбца, возвращенному в наборе результатов. Имя каталога и имя столбца разделены десятичным (.) символом. Поддержка этого атрибута на уровне драйвера; он может не поддерживаться Вашим драйвером.

**PDO::ATTR\_FETCH\_TABLE\_NAMES ( 14 )**

**Не поддерживается драйвером mysql!** Добавляет имя таблицы к каждому имени столбца, возвращенному в наборе результатов. Имя таблицы и имя столбца разделены десятичным (.) символом. Поддержка этого атрибута на уровне драйвера; он может не поддерживаться Вашим драйвером.

**PDO::ATTR\_STRINGIFY\_FETCHES ( 17 )**

**Не поддерживается драйвером mysql!** Требует указания значения типа boolean. При получении результата конвертирует числовые значения в строковые

**PDO::ATTR\_MAX\_COLUMN\_LEN ( 18 )**

**Не поддерживается драйвером mysql!**

**PDO::ATTR\_DEFAULT\_FETCH\_MODE ( 19 )**

**Доступный начиная с PHP 5.2.0**

Задаёт тип получаемого результата по-умолчанию:

[?](http://php-zametki.ru/php-prodvinutym/58-pdo-konstanty-atributy.html)

|  |  |
| --- | --- |
| 1 | $db->setAttribute(PDO::ATTR\_DEFAULT\_FETCH\_MODE,PDO::FETCH\_ASSOC); |

**PDO::ATTR\_EMULATE\_PREPARES ( 20 )**

**Доступный начиная с PHP 5.1.3.**Эмуляция подготовленных запросов. Если честно не понял, для чего нужна эта опция, у PDO есть прекрасные механизмы для осуществления безопасных подготовленных запросов.

**PDO::ATTR\_ORACLE\_NULLS ( 11 )**

**(Доступно со всеми драйверами, не только с Oracle)**

Конвертация NULL и пустых строк при получении результата запроса:

PDO::NULL\_NATURAL (0) - Не конвертируется.  
PDO::NULL\_EMPTY\_STRING (1) - Пустые строки конвертируются в NULL.  
PDO::NULL\_TO\_STRING (2)- NULL конвертируется в пустую строку.

**PDO::ATTR\_CURSOR\_NAME ( 9 )**

**(Специфично для Firebird и ODBC)**

Получить или определить имя курсора для UPDATE ... WHERE CURRENT OF. Полезно при использовании прокручиваемых курсоров и позиционированных обновлений.

**PDO::ATTR\_CURSOR ( 10 )**

Выбирает тип курсора. PDO в настоящий момент поддерживает:

**PDO::CURSOR\_FWDONLY ( 0 )**

Создаёт объект PDOStatement с однонаправленным курсором. Это - вариант курсора по умолчанию, поскольку он - самый быстрый и самый распространенный объект доступа к данным в PHP.

**PDO:: CURSOR\_SCROLL ( 1 )**

Создаёт объект PDOStatement с прокручиваемым курсором. Передайте PDO:: FETCH\_ORI\_\* константы, чтобы управлять строками, принесенными от набора результатов.

Используйте PDO:: CURSOR\_FWDONLY, если Вы точно не знаете, нужен ли Вам прокручиваемый курсор:

**PDO::FETCH\_ORI\_NEXT (0 )**

Получает следующую строку в наборе результатов. Работает только с прокручиваемыми курсорами.

**PDO::FETCH\_ORI\_PRIOR ( 1 )**

Получает предыдущую строку в наборе результатов. Работает только с прокручиваемыми курсорами.

**PDO::FETCH\_ORI\_FIRST ( 2 )**

Получает первую строку в наборе результатов Работает только с прокручиваемыми курсорами.

**PDO::FETCH\_ORI\_LAST ( 3 )**

Получает последнюю строку в наборе результатов. Работает только с прокручиваемыми курсорами.

**PDO::FETCH\_ORI\_ABS ( 4 )**

Получает запрашиваемую строку номером ряда из набора результатов. Работает только с прокручиваемыми курсорами.

**PDO::FETCH\_ORI\_REL ( 5 )**

Получает запрашиваемую строку с относительным положением от настоящего положения курсора в наборе результатов. Работает только с прокручиваемыми курсорами.

[?](http://php-zametki.ru/php-prodvinutym/58-pdo-konstanty-atributy.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | $sql = 'SELECT \* FROM `gp\_cities` ORDER BY city DESC';    $stmt = $dbh->prepare($sql,array(PDO::ATTR\_CURSOR => PDO::CURSOR\_SCROLL));    $stmt->execute();    while($row1 = $stmt->fetch(PDO::FETCH\_OBJ, PDO::FETCH\_ORI\_NEXT) )  {    print\_r($row1);  } |

**PDO::ERR\_NONE ( 00000 )**

**Удобно при PDO::ATTR\_ERRMODE => PDO::ERRMODE\_SILENT**

Эта константа создана для удобства, с ней можно сравнивать значения методов: PDO::errorCode () или PDOStatement::errorCode (), чтобы определить, встретилась ли ошибка. Соответствует SQLSTATE '00000', что означает, что SQL-оператор был успешно выполнен без ошибок или предупреждений:

[?](http://php-zametki.ru/php-prodvinutym/58-pdo-konstanty-atributy.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | $dbh = new PDO( 'mysql:host=localhost;dbname=shop;',                  'root', '',                  array(PDO::MYSQL\_ATTR\_INIT\_COMMAND => 'SET NAMES UTF8',                        PDO::ATTR\_ERRMODE => PDO::ERRMODE\_SILENT));    $sql = 'SELECT \* FROM `gp\_cities` ORDER BY city DESC';    $stmt = $dbh->prepare($sql,array(PDO::ATTR\_CURSOR => PDO::CURSOR\_SCROLL));    if( $dbh->errorCode() != PDO::ERR\_NONE) exit("Проблемы PDO !!!");    $stmt->execute();    if( $stmt->errorCode() != PDO::ERR\_NONE) exit("Проблемы PDOStatement!!!");    print\_r($stmt->fetchAll(PDO::FETCH\_OBJ)); |

<?php

// Класс для установления подключения к базе данных

class Database {

    private $host = "localhost";

    private $username = "root";

    private $password = "password";

    private $database = "mydb";

    protected $conn;

    public function \_\_construct() {

        try {

            // Создаем подключение к базе данных с опцией обработки ошибок

            $this->conn = new PDO("mysql:host=$this->host;dbname=$this->database", $this->username, $this->password, [

                PDO::ATTR\_ERRMODE => PDO::ERRMODE\_EXCEPTION

            ]);

            // Вызываем метод для создания таблицы users

            $this->createTable();

        } catch(PDOException $e) {

            echo "Ошибка подключения: " . $e->getMessage();

        }

    }

    // Метод для создания таблицы users

    private function createTable() {

        try {

            $query = "CREATE TABLE IF NOT EXISTS users (

                id INT AUTO\_INCREMENT PRIMARY KEY,

                name VARCHAR(255) NOT NULL,

                email VARCHAR(255) NOT NULL

            )";

            $this->conn->exec($query);

            echo "Таблица users успешно создана или уже существует.<br>";

        } catch(PDOException $e) {

            echo "Ошибка при создании таблицы: " . $e->getMessage();

        }

    }

}

// Класс для выполнения CRUD операций

class CRUD extends Database {

    // Метод для создания пользователя

    public function create($name, $email) {

        try {

            $stmt = $this->conn->prepare("INSERT INTO users (name, email) VALUES (:name, :email)");

            $stmt->bindParam(":name", $name);

            $stmt->bindParam(":email", $email);

            $stmt->execute();

            echo "Пользователь успешно добавлен: $name ($email)<br>";

        } catch(PDOException $e) {

            echo "Ошибка: " . $e->getMessage();

        }

    }

    // Метод для получения списка пользователей

    public function read() {

        try {

            $stmt = $this->conn->prepare("SELECT \* FROM users");

            $stmt->execute();

            $users = $stmt->fetchAll(PDO::FETCH\_ASSOC);

            return $users;

        } catch(PDOException $e) {

            echo "Ошибка: " . $e->getMessage();

        }

    }

    // Метод для обновления пользователя

    public function update($id, $name, $email) {

        try {

            $stmt = $this->conn->prepare("UPDATE users SET name = :name, email = :email WHERE id = :id");

            $stmt->bindParam(":name", $name);

            $stmt->bindParam(":email", $email);

            $stmt->bindParam(":id", $id);

            $stmt->execute();

            echo "Пользователь успешно обновлен: ID $id, новое имя - $name, новый email - $email<br>";

        } catch(PDOException $e) {

            echo "Ошибка: " . $e->getMessage();

        }

    }

    // Метод для удаления пользователя

    public function delete($id) {

        try {

            $stmt = $this->conn->prepare("DELETE FROM users WHERE id = :id");

            $stmt->bindParam(":id", $id);

            $stmt->execute();

            echo "Пользователь успешно удален: ID $id<br>";

        } catch(PDOException $e) {

            echo "Ошибка: " . $e->getMessage();

        }

    }

}

// Пример использования

$crud = new CRUD();

// Создание пользователя

$crud->create("Иван", "ivan@example.com");

// Получение списка пользователей

$users = $crud->read();

echo "Список пользователей:<br>";

foreach ($users as $user) {

    echo "ID: " . $user['id'] . ", Имя: " . $user['name'] . ", Email: " . $user['email'] . "<br>";

}

// Обновление пользователя

$crud->update(1, "Петр", "peter@example.com");

// Удаление пользователя

$crud->delete(1);

?>

В этой практике мы изучим, как работать с базой данных MySQL на PHP с помощью PDO (PHP Data Objects) и выполним базовые операции CRUD (Create, Read, Update, Delete) - создание, чтение, обновление и удаление данных.

Что такое PDO и зачем он нужен:

PDO - это интерфейс доступа к данным для различных баз данных на PHP. Он предоставляет унифицированный способ взаимодействия с различными СУБД, такими как MySQL, PostgreSQL, SQLite и другие. Использование PDO обеспечивает безопасность от SQL-инъекций и упрощает процесс работы с базой данных.

Создание подключения к базе данных:

Сначала мы создаем подключение к базе данных с использованием класса Database, который содержит логику подключения и метод для создания таблицы users.

Класс CRUD:

Класс CRUD наследует функциональность класса Database и содержит методы для выполнения CRUD операций.

Метод create(): Этот метод добавляет нового пользователя в таблицу users. Мы используем подготовленные запросы, чтобы обезопасить наш код от SQL-инъекций.

Метод read(): Этот метод извлекает все записи из таблицы users и возвращает результат в виде массива.

Метод update(): Этот метод обновляет информацию о пользователе по его id.

Метод delete(): Этот метод удаляет пользователя по его id.

Пример использования:

Мы создаем объект класса CRUD и демонстрируем выполнение операций:

Создание пользователя с именем "Иван" и email "ivan@example.com".

Получение списка пользователей и вывод в браузер.

Обновление данных первого пользователя (изменение имени и email).

Удаление первого пользователя.

Примеры из жизни:

Учет сотрудников: Представьте, что вы разрабатываете систему для учета сотрудников в компании. Таблица users может хранить данные о каждом сотруднике, его имени и email.

Интернет-магазин: При создании интернет-магазина нужно хранить информацию о товарах, заказах и клиентах. Таблица users может хранить данные о зарегистрированных клиентах.

Социальная сеть: В социальной сети таблица users может хранить информацию о пользователях - их имена, email и другие данные.

Домашнее задание:

Добавьте методы для работы с другой таблицей, например, таблицей products для магазина.

Создайте страницу для регистрации пользователей, используя метод create() для добавления новых пользователей.

Создайте страницу для просмотра списка заказов из другой таблицы.

Заключение:

В этом уроке мы изучили основы работы с базой данных MySQL на PHP с использованием PDO. Мы научились выполнять CRUD операции, используя безопасные подготовленные запросы, и применили полученные знания на примере разработки функциональности для учета сотрудников в компании.

Практика MySQLi и CRUD операции

<!DOCTYPE html>

<html>

<head>

    <title>Пример CRUD с MySQLi</title>

</head>

<body>

<?php

// Параметры подключения к базе данных

$host = "localhost";

$username = "root";

$password = "password";

$database = "mydb";

// Подключение к базе данных

$conn = new mysqli($host, $username, $password, $database);

// Проверка на успешное подключение

if ($conn->connect\_error) {

    die("Ошибка подключения: " . $conn->connect\_error);

}

// Создание таблицы users

$createTableQuery = "CREATE TABLE IF NOT EXISTS users (

    id INT AUTO\_INCREMENT PRIMARY KEY,

    name VARCHAR(255) NOT NULL,

    email VARCHAR(255) NOT NULL

)";

if ($conn->query($createTableQuery) === TRUE) {

    echo "Таблица users успешно создана или уже существует.<br>";

} else {

    echo "Ошибка при создании таблицы: " . $conn->error;

}

// Вставка данных

$name = "Иван";

$email = "ivan@example.com";

$insertQuery = "INSERT INTO users (name, email) VALUES ('$name', '$email')";

if ($conn->query($insertQuery) === TRUE) {

    echo "Пользователь успешно добавлен: $name ($email)<br>";

} else {

    echo "Ошибка: " . $conn->error;

}

// Вывод списка пользователей

$selectQuery = "SELECT \* FROM users";

$result = $conn->query($selectQuery);

echo "Список пользователей:<br>";

if ($result->num\_rows > 0) {

    while ($row = $result->fetch\_assoc()) {

        echo "ID: " . $row['id'] . ", Имя: " . $row['name'] . ", Email: " . $row['email'] . "<br>";

    }

} else {

    echo "Нет пользователей.";

}

// Закрытие соединения с базой данных

$conn->close();

?>

</body>

</html>

Введение:

В этом практическом примере мы изучим работу с базой данных MySQL на PHP с помощью расширения MySQLi (MySQL Improved). Мы познакомимся с базовыми операциями CRUD (Create, Read, Update, Delete) для работы с данными в базе данных и научимся использовать MySQLi для их выполнения. Примеры и комментарии помогут лучше понять каждый шаг.

Что такое MySQLi:

MySQLi - это улучшенное расширение для взаимодействия с MySQL базами данных на PHP. Оно предоставляет более удобные и безопасные функции для работы с базой данных, включая подготовленные запросы, что снижает риск SQL-инъекций.

Создание подключения к базе данных:

Для начала работы с MySQLi, мы создаем соединение с базой данных. Параметры подключения, такие как хост, имя пользователя, пароль и имя базы данных, передаются при создании объекта MySQLi.

<?php

$host = "localhost";

$username = "root";

$password = "password";

$database = "mydb";

// Подключение к базе данных

$conn = new mysqli($host, $username, $password, $database);

// Проверка на успешное подключение

if ($conn->connect\_error) {

    die("Ошибка подключения: " . $conn->connect\_error);

}

?>

Создание таблицы:

Мы можем использовать SQL запросы для создания таблицы в базе данных. Если таблица уже существует, этот запрос не повлияет на нее.

<?php

$createTableQuery = "CREATE TABLE IF NOT EXISTS users (

    id INT AUTO\_INCREMENT PRIMARY KEY,

    name VARCHAR(255) NOT NULL,

    email VARCHAR(255) NOT NULL

)";

if ($conn->query($createTableQuery) === TRUE) {

    echo "Таблица users успешно создана или уже существует.<br>";

} else {

    echo "Ошибка при создании таблицы: " . $conn->error;

}

?>

Выполнение CRUD операций:

Create (Создание): Добавление новых записей в таблицу.

<?php

$name = "Иван";

$email = "ivan@example.com";

$insertQuery = "INSERT INTO users (name, email) VALUES ('$name', '$email')";

if ($conn->query($insertQuery) === TRUE) {

    echo "Пользователь успешно добавлен: $name ($email)<br>";

} else {

    echo "Ошибка: " . $conn->error;

}

?>

Read (Чтение): Извлечение данных из таблицы.

<?php

$selectQuery = "SELECT \* FROM users";

$result = $conn->query($selectQuery);

echo "Список пользователей:<br>";

if ($result->num\_rows > 0) {

    while ($row = $result->fetch\_assoc()) {

        echo "ID: " . $row['id'] . ", Имя: " . $row['name'] . ", Email: " . $row['email'] . "<br>";

    }

} else {

    echo "Нет пользователей.";

}

?>

Update (Обновление): Изменение данных в записи.

<?php

$idToUpdate = 1;

$newName = "Петр";

$newEmail = "peter@example.com";

$updateQuery = "UPDATE users SET name='$newName', email='$newEmail' WHERE id=$idToUpdate";

if ($conn->query($updateQuery) === TRUE) {

    echo "Пользователь успешно обновлен: ID $idToUpdate, новое имя - $newName, новый email - $newEmail<br>";

} else {

    echo "Ошибка: " . $conn->error;

}

?>

Delete (Удаление): Удаление записи из таблицы.

<?php

$idToDelete = 1;

$deleteQuery = "DELETE FROM users WHERE id=$idToDelete";

if ($conn->query($deleteQuery) === TRUE) {

    echo "Пользователь успешно удален: ID $idToDelete<br>";

} else {

    echo "Ошибка: " . $conn->error;

}

?>

Примеры из жизни:

Социальная сеть: Представьте, что вы создаете социальную сеть. Таблица users может хранить данные о пользователях - их имена, email и другие данные.

Онлайн-магазин: Если вы создаете интернет-магазин, таблица products может хранить информацию о товарах, а таблица orders - о заказах клиентов.

Учетная система: При разработке системы учета в компании, таблица employees может хранить данные о сотрудниках.

Домашнее задание:

Создайте дополнительную таблицу, например categories для категорий товаров.

Реализуйте страницу регистрации пользователей с возможностью добавления новых записей.

Создайте страницу просмотра списка заказов из таблицы orders.

Заключение:

В этом методическом материале мы изучили основы работы с базой данных MySQL на PHP с использованием MySQLi. Мы познакомились с выполнением CRUD операций (Create, Read, Update, Delete), создали подключение к базе данных, создали таблицу и выполнили операции над данными. Эти навыки могут быть полезными при разработке веб-приложений, интернет-магазинов и других проектов, где требуется взаимодействие с базой данных.

Практика PDO (10 простых примеров)

Создание подключения к базе данных

<?php

// Класс для работы с базой данных

class Database {

    private $host = "localhost";

    private $username = "root";

    private $password = "password";

    private $database = "mydb";

    protected $conn;

    // Конструктор, создающий подключение

    public function \_\_construct() {

        $this->conn = new PDO("mysql:host=$this->host;dbname=$this->database", $this->username, $this->password);

    }

}

// Создаем объект класса Database

$db = new Database();

// Выводим сообщение в браузер

echo "Подключение к базе данных успешно установлено.";

?>

Пример 2: Вставка данных в базу данных

<?php

// Класс для работы с базой данных

class Student {

    private $conn;

    public function \_\_construct($conn) {

        $this->conn = $conn;

    }

    // Метод для вставки данных

    public function insert($name, $age) {

        $query = "INSERT INTO students (name, age) VALUES (:name, :age)";

        $stmt = $this->conn->prepare($query);

        $stmt->bindParam(":name", $name);

        $stmt->bindParam(":age", $age);

        return $stmt->execute();

    }

}

// Создаем объект класса Database

$db = new Database();

// Создаем объект класса Student

$student = new Student($db->conn);

// Вставляем данные и выводим сообщение

if ($student->insert("John", 20)) {

    echo "Данные успешно добавлены в базу данных.";

} else {

    echo "Ошибка при добавлении данных.";

}

?>

Пример 3: Получение данных из базы данных

<?php

// Класс для работы с базой данных

class Student {

    private $conn;

    public function \_\_construct($conn) {

        $this->conn = $conn;

    }

    // Метод для получения всех данных

    public function getAll() {

        $query = "SELECT \* FROM students";

        $stmt = $this->conn->query($query);

        return $stmt->fetchAll(PDO::FETCH\_ASSOC);

    }

}

// Создаем объект класса Database

$db = new Database();

// Создаем объект класса Student

$student = new Student($db->conn);

// Получаем данные и выводим список студентов

$students = $student->getAll();

echo "Список студентов:<br>";

foreach ($students as $row) {

    echo "Имя: " . $row['name'] . ", Возраст: " . $row['age'] . "<br>";

}

?>

Пример 4: Обновление данных в базе данных

<?php

// Класс для работы с базой данных

class Student {

    private $conn;

    public function \_\_construct($conn) {

        $this->conn = $conn;

    }

    // Метод для обновления возраста

    public function updateAge($id, $newAge) {

        $query = "UPDATE students SET age = :age WHERE id = :id";

        $stmt = $this->conn->prepare($query);

        $stmt->bindParam(":age", $newAge);

        $stmt->bindParam(":id", $id);

        return $stmt->execute();

    }

}

// Создаем объект класса Database

$db = new Database();

// Создаем объект класса Student

$student = new Student($db->conn);

// Обновляем возраст и выводим сообщение

if ($student->updateAge(1, 21)) {

    echo "Возраст успешно обновлен.";

} else {

    echo "Ошибка при обновлении возраста.";

}

?>

Пример 5: Удаление данных из базы данных

<?php

// Класс для работы с базой данных

class Student {

    private $conn;

    public function \_\_construct($conn) {

        $this->conn = $conn;

    }

    // Метод для удаления данных

    public function delete($id) {

        $query = "DELETE FROM students WHERE id = :id";

        $stmt = $this->conn->prepare($query);

        $stmt->bindParam(":id", $id);

        return $stmt->execute();

    }

}

// Создаем объект класса Database

$db = new Database();

// Создаем объект класса Student

$student = new Student($db->conn);

// Удаляем данные и выводим сообщение

if ($student->delete(1)) {

    echo "Данные успешно удалены.";

} else {

    echo "Ошибка при удалении данных.";

}

?>

Пример 6: Подготовленные запросы

<?php

// Класс для работы с базой данных

class Student {

    private $conn;

    public function \_\_construct($conn) {

        $this->conn = $conn;

    }

    // Метод для получения данных по имени

    public function getByName($name) {

        $query = "SELECT \* FROM students WHERE name = :name";

        $stmt = $this->conn->prepare($query);

        $stmt->bindParam(":name", $name);

        $stmt->execute();

        return $stmt->fetchAll(PDO::FETCH\_ASSOC);

    }

}

// Создаем объект класса Database

$db = new Database();

// Создаем объект класса Student

$student = new Student($db->conn);

// Получаем данные по имени и выводим список студентов

$students = $student->getByName("John");

echo "Студенты с именем John:<br>";

foreach ($students as $row) {

    echo "Имя: " . $row['name'] . ", Возраст: " . $row['age'] . "<br>";

}

?>

Пример 7: Использование транзакций

<?php

// Класс для работы с базой данных

class TransactionExample {

    private $conn;

    public function \_\_construct($conn) {

        $this->conn = $conn;

    }

    // Метод для выполнения транзакции

    public function performTransaction() {

        try {

            $this->conn->beginTransaction();

            // Выполняем операции

            $query1 = "INSERT INTO table1 (column1) VALUES (:value1)";

            $stmt1 = $this->conn->prepare($query1);

            $stmt1->bindParam(":value1", $value1);

            $stmt1->execute();

            $query2 = "UPDATE table2 SET column2 = :value2 WHERE column3 = :value3";

            $stmt2 = $this->conn->prepare($query2);

            $stmt2->bindParam(":value2", $value2);

            $stmt2->bindParam(":value3", $value3);

            $stmt2->execute();

            $this->conn->commit();

            return true;

        } catch (PDOException $e) {

            $this->conn->rollBack();

            return false;

        }

    }

}

// Создаем объект класса Database

$db = new Database();

// Создаем объект класса TransactionExample

$transaction = new TransactionExample($db->conn);

// Выполняем транзакцию и выводим сообщение

if ($transaction->performTransaction()) {

    echo "Транзакция успешно выполнена.";

} else {

    echo "Ошибка при выполнении транзакции.";

}

?>

Пример 8: Использование транзакций

<?php

// Класс для работы с базой данных

class TransactionExample {

    private $conn;

    public function \_\_construct($conn) {

        $this->conn = $conn;

    }

    // Метод для выполнения транзакции

    public function performTransaction() {

        try {

            $this->conn->beginTransaction();

            // Выполняем операции

            $query1 = "INSERT INTO table1 (column1) VALUES (:value1)";

            $stmt1 = $this->conn->prepare($query1);

            $stmt1->bindParam(":value1", $value1);

            $stmt1->execute();

            $query2 = "UPDATE table2 SET column2 = :value2 WHERE column3 = :value3";

            $stmt2 = $this->conn->prepare($query2);

            $stmt2->bindParam(":value2", $value2);

            $stmt2->bindParam(":value3", $value3);

            $stmt2->execute();

            $this->conn->commit();

            return true;

        } catch (PDOException $e) {

            $this->conn->rollBack();

            return false;

        }

    }

}

// Создаем объект класса Database

$db = new Database();

// Создаем объект класса TransactionExample

$transaction = new TransactionExample($db->conn);

// Выполняем транзакцию и выводим сообщение

if ($transaction->performTransaction()) {

    echo "Транзакция успешно выполнена.";

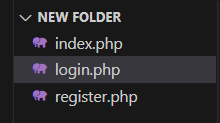
} else {

    echo "Ошибка при выполнении транзакции.";

}

?>

Простой пример регистрации и авторизации пользователя на PDO



index.php:

<!DOCTYPE html>

<html>

<head>

    <title>Пример авторизации и регистрации</title>

</head>

<body>

<h2>Регистрация</h2>

<form method="post" action="register.php">

    <input type="text" name="username" placeholder="Имя пользователя" required>

    <input type="password" name="password" placeholder="Пароль" required>

    <button type="submit">Зарегистрироваться</button>

</form>

<h2>Авторизация</h2>

<form method="post" action="login.php">

    <input type="text" name="username" placeholder="Имя пользователя" required>

    <input type="password" name="password" placeholder="Пароль" required>

    <button type="submit">Войти</button>

</form>

</body>

</html>

register.php:

<?php

$db = new PDO("mysql:host=localhost;dbname=users", "root", "password");

if ($\_SERVER["REQUEST\_METHOD"] == "POST") {

    $username = $\_POST["username"];

    $password = password\_hash($\_POST["password"], PASSWORD\_DEFAULT);

    $stmt = $db->prepare("INSERT INTO users\_info (username, password) VALUES (:username, :password)");

    $stmt->bindParam(":username", $username);

    $stmt->bindParam(":password", $password);

    if ($stmt->execute()) {

        echo "Регистрация успешна!";

    } else {

        echo "Ошибка при регистрации.";

    }

}

?>

login.php:

<?php

$db = new PDO("mysql:host=localhost;dbname=users", "root", "password");

if ($\_SERVER["REQUEST\_METHOD"] == "POST") {

    $username = $\_POST["username"];

    $password = $\_POST["password"];

    $stmt = $db->prepare("SELECT id, password FROM users\_info WHERE username = :username");

    $stmt->bindParam(":username", $username);

    $stmt->execute();

    $user = $stmt->fetch();

    if ($user && password\_verify($password, $user["password"])) {

        echo "Авторизация успешна!";

    } else {

        echo "Неверное имя пользователя или пароль.";

    }

}

?>

Этот пример объединяет код регистрации, авторизации и выводит результат на экран. Пользователи могут вводить данные в формы для регистрации и авторизации, а затем видеть соответствующие сообщения о результатах на экране.

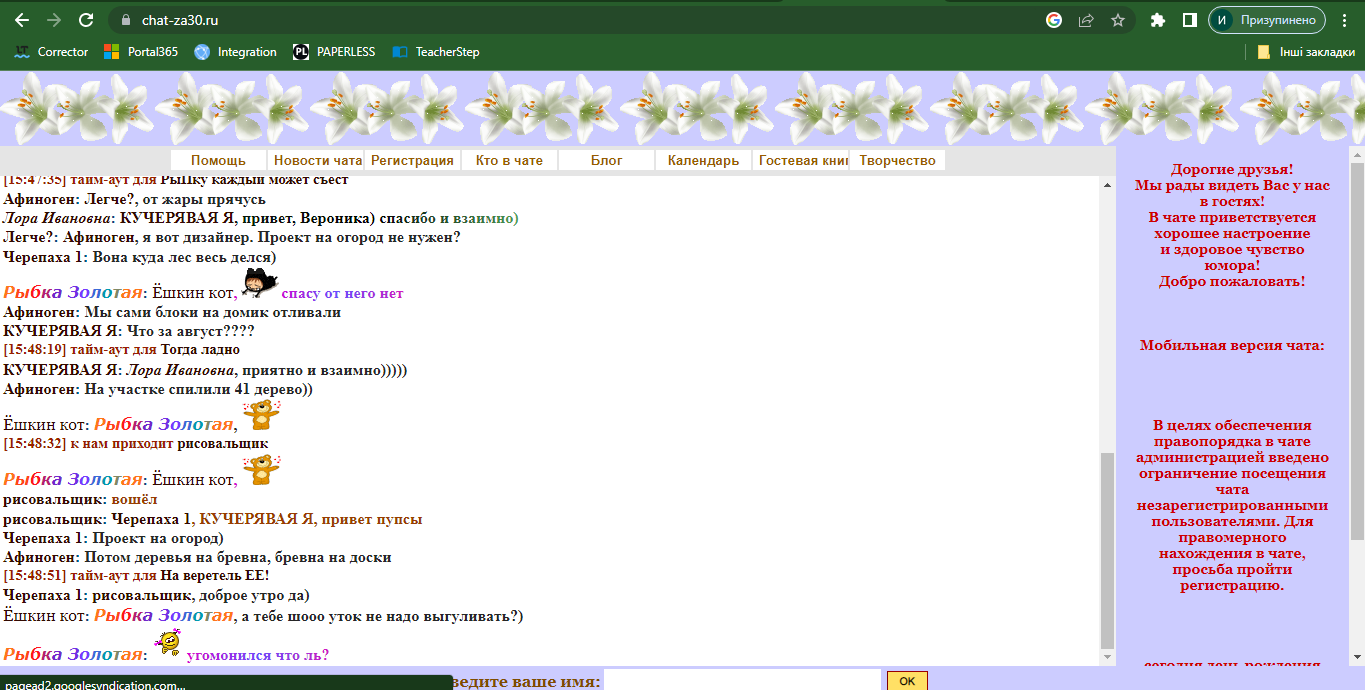
Материалы к уроку:

1. <https://www.hostedpmm.com/mysql-php-on-backend-best-practices/>
2. <https://www.computer-pdf.com/exercises/php-and-mysql>
3. <https://www.amazon.com/MySQL-Practice-Learn-Jitendra-Patel-ebook/dp/B00BRTV1JC>

Домашнее задание:

1. Создайте WEBSITE chat, пользователи могут заходить и проходить регистрацию, авторизацию, а так же есть категория пользователей, для быстрого сообщения. Если пользователь авторизован, то он оставляет сообщение в чате, а так же получает рейтинг за определенное количество сообщений. Все данные нужно хранить в базе данных, использовать можно подключение любое PDO или MySqlI

Примерный вид чата:



1. Создайте класс "Toy" для работы с игрушками. Реализуйте методы для добавления новой игрушки в базу данных и изменения ее статуса (в наличии/продана).
2. Разработайте класс "Pet" для управления домашними питомцами. Создайте методы для добавления нового питомца и обновления его информации (имя, возраст и т.д.).
3. Напишите класс "Recipe" для работы с рецептами блюд. Реализуйте методы для добавления нового рецепта и вывода списка всех рецептов.
4. Создайте класс "Game" для управления компьютерными играми. Разработайте методы для добавления новой игры, изменения ее жанра и получения информации о ней.
5. Реализуйте класс "Friend" для работы с друзьями. Напишите методы добавления нового друга и изменения его контактной информации.
6. Создайте класс "Plant" для управления растениями. Разработайте методы добавления нового растения и обновления его состояния (растет/засохло).
7. Напишите класс "Goal" для работы с личными целями. Реализуйте методы добавления новой цели и изменения ее статуса (достигнута/не достигнута).
8. Создайте класс "Movie" для управления фильмами. Разработайте методы добавления нового фильма, изменения его рейтинга и получения списка фильмов.
9. Реализуйте класс "Holiday" для работы с праздниками. Напишите методы добавления нового праздника и изменения его даты или типа.
10. Создайте класс "Achievement" для управления достижениями. Разработайте методы добавления нового достижения и изменения его описания.

