Работа с MySQL через PDO

PDO. Создание подключения

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

SQL

```
NEW
PDO("mysql:host=адрес_сервера;port=номер_порта;dbname=имя_базы_данных",
"имя_пользователя", "пароль")
```

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

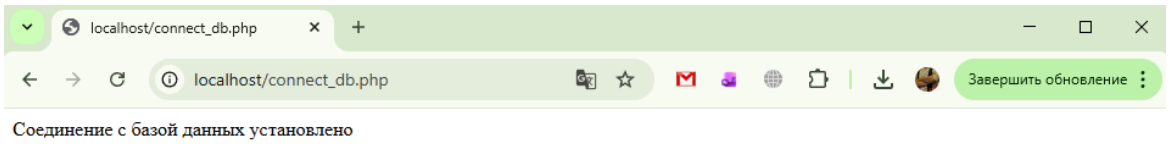
При успешном подключении вызов конструктора new PDO() возвращает созданный объект PDO, который представляет установленное подключение и через который мы сможем взавмодействовать с базой данных. Однако если установка подключения прошла неудачно (например, сервер базы данных недоступен, указаны неправильные 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:

connect_db.php

```
<?php
try {
    // подключаемся к серверу
    $conn = NEW PDO("mysql:host=localhost", "root", "");
    echo "Соединение с базой данных установлено";
}
catch (PDOException $e) {
    echo "Соединение не удалось: " . $e->getMessage();
}
```

```
?>
```

http://localhost/connect_db.php



Здесь производится подключение к локальному серверу, поэтому в строке подключения параметр host имеет значение «localhost». Поскольку база данных пока не важна, то параметр dbname не указан. Подключение производится для пользователя - пользователя «root», для которого установлен пароль «mypassword».

«root» - это пользователь по умолчанию, который существует для сервера MySQL. А «mypassword» - пароль, установленный для этого пользователя. Естественно в каждом конкретном случае пароль для этого пользователя может отличаться. Однако если на сервере MySQL созданы другие пользователи, то можно указывать этих пользователей и их пароли.

При успешном подключении созданный объект PDO будет сохранен в переменную $conn, через которую мы затем сможем взаимодействовать с MySQL:

connect_db.php

```
$conn = NEW PDO("mysql:host=localhost", "root", "");
```

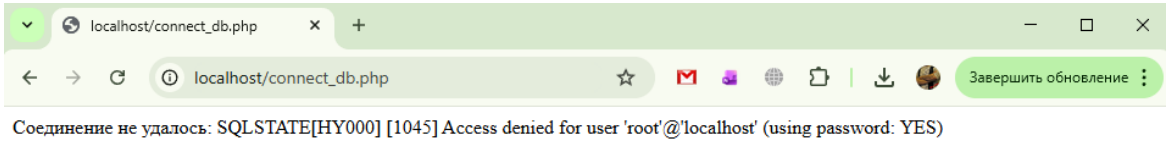
Если произойдет ошибка, то будет сгенерировано исключение типа PDOException. Как и у других классов исключений с помощью метода getMessage() мы можем получить сообщение об ошибке.

И при успешном подключении мы увидим в браузере следующее сообщение:

> Соединение с базой данных установлено

А если произойдет ошибка, то браузер выведет сообщение об ошибке. Например, сообщение об ошибке при некорректном пароле:

> Connection failed: SQLSTATE[HY000] [1045] Access denied for user 'root'@'localhost' (using password: YES)



Установка режима вывода ошибок

Если при взаимодействии с MySQL произойдет ошибка, то, как правило, ожидается, что мы получим сообщение об ошибке. Однако реальное поведение зависит от редима вывода ошибок, который установлен для объекта PDO. Режим вывода ошибок задается с помощью атрибута **PDO::ATTR_ERRMODE**, который может принимать следующие значения:

- **PDO::ERRMODE_SILENT:** PDO просто устанавливает код ошибки. Для получения которого и для получения информации об ошибке по которому необходимо было вызывать специальные методы. Поскольку при этом режиме необходимо вызывать дополнительные методы, то этот способ обычно рассматривался как не самый удобный. Он был значением по умолчанию до версии PHP 8.0.
- **PDO::ERRMODE_WARNING:** PDO генерирует сообщение типа E_WARNING. Обычно применяется при отладке или тестировании
- **PDO::ERRMODE_EXCEPTION:** PDO передает информацию об ошибке в объект PDOException, благодаря чему через блок catch в конструкции try..catch мы можем отловить ошибку и получить информацию об этом исключении. Этот режим применяется как режим по умолчанию начиная с версии PHP 8.0.

Если мы хотим получать информацию об ошибке через исключение PDOException и обрабатывать его в блоке catch, то нам нужно значение **PDO::ERRMODE_EXCEPTION.** В PHP 8.0 и выше это значение применяется по умолчанию, однако, если версия ниже 8.0, то необходимо это значение установить явным образом с помощью метода setAttribute() объекта PDO:

connect_db.php

```
try {
    // подключаемся к серверу
    $conn = NEW PDO("mysql:host=localhost", "root", "mypassword");
    // установка режима вывода ошибок
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    echo "Соединение с базой данных установлено";
}
catch (PDOException $e) {
    echo "Соединение не удалось: " . $e->getMessage();
}
```

Закрытие подключения

После завершения работы скрипта PHP автоматически закрывает открытые подключения к базе данных. Но может потребоваться закрыть подключение еще в процессе работы скрипта. В этом случае объекту PDO можно присвоить значение null:

SQL

```
$conn = NULL;   // отключаемся от сервера базы данных
```

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.

SQL

```
$conn = NEW PDO("mysql:host=localhost", "root", "mypassword");
$conn->EXEC(команда_sql);
```

Создание базы данных

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

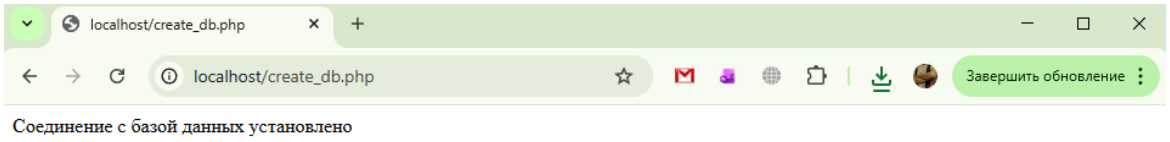
Создадим базу данных через PHP:

create_db.php

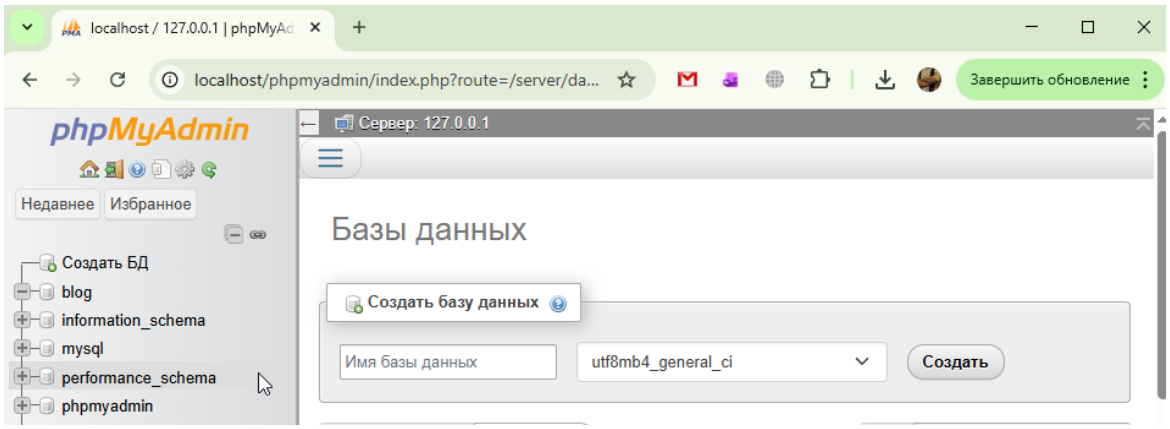
```
<?php
 try {
    // подключаемся к серверу
    $conn = NEW PDO("mysql:host=localhost", "root", "");

    //  SQL-выражение для создания базы данных
    $sql = "CREATE DATABASE blog";
    // выполняем  SQL-выражение
    $conn->EXEC($sql);
    echo "Соединение с базой данных установлено";
}
```

```
catch (PDOException $e) {
    echo "Соединение не удалось: " . $e->getMessage();
}
?>
```

http://localhost/create_db.php



http://localhost/phpmyadmin/index.php?route=/server/databases



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

SQL

```
$sql = "CREATE DATABASE blog";
```

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

SQL

```
$conn->EXEC($sql);
```

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

База данных создана

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 «blog». И создадим в ней таблицу, которая описывается следующим кодом

SQL

```
<?php
CREATE TABLE Users (id INTEGER AUTO_INCREMENT PRIMARY KEY, name
VARCHAR(30), pass VARCHAR(30), STATUS VARCHAR(30));
?>
```

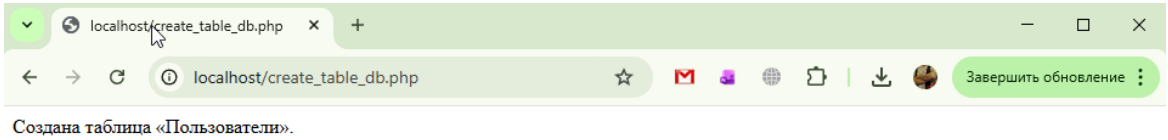
Здесь создается таблица под названием «users». Она будет хранить условных пользователей. В ней будет три столбца: id, name, pass и status. Столбец id представляет числовой уникальный идентификатор строки - или идентификатор пользователя. Столбец name представляет строку - имя пользователя. Столбец pass - соответственно пароль. А столбец status определяет, что это администратор, пользователь или гость.

Для создания таблицы определим следующий скрипт php:

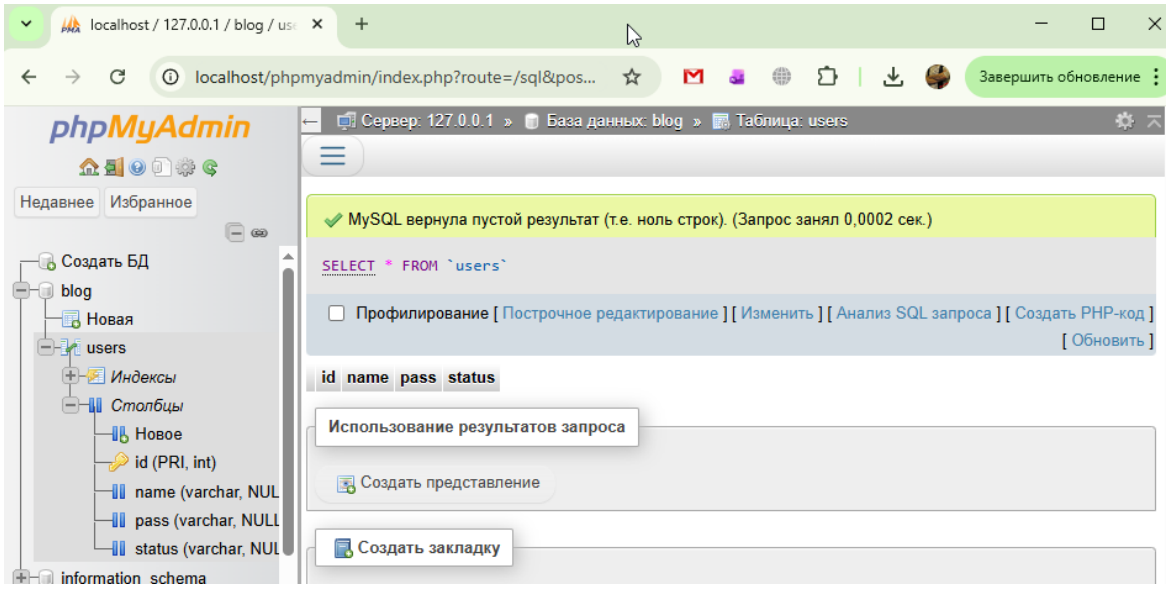
create_table_db.php

```
<?php
try {
    // подключаемся к серверу
    $conn = NEW PDO("mysql:host=localhost;dbname=blog", "root", "");

    // SQL-выражение для создания таблицы
    $sql = "create table users (id integer auto_increment primary key,
name varchar(30), pass varchar(30),status varchar(30));";
    // выполняем SQL-выражение
    $conn->EXEC($sql);
    echo "Создана таблица «Пользователи».";
}
catch (PDOException $e) {
    echo "Соединение не удалось: " . $e->getMessage();
}
?>
```

http://localhost/create_table_db.php



http://localhost/phpmyadmin/index.php?route=/sql&pos=0&db=blog&table=users



Обратите внимание, что по сравнению с предыдущим примером здесь в строке подключения указана база данных, в которой создается таблица: «mysql:host=localhost;dbname=blog»

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

Создана таблица «Пользователи».

Добавление данных в PDO и параметризация запросов

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

SQL

```
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 затронутых 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 «blog» и созданную в ней таблицу Users со следующим определением:

SQL

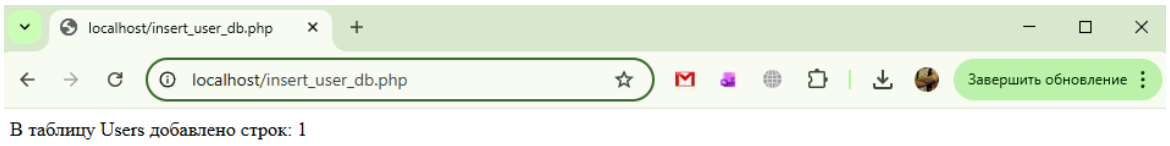
```
CREATE TABLE Users (id INTEGER AUTO_INCREMENT PRIMARY KEY, name
VARCHAR(30), pass VARCHAR(30), STATUS VARCHAR(30));
```

И для добавления определим следующий скрипт PHP:

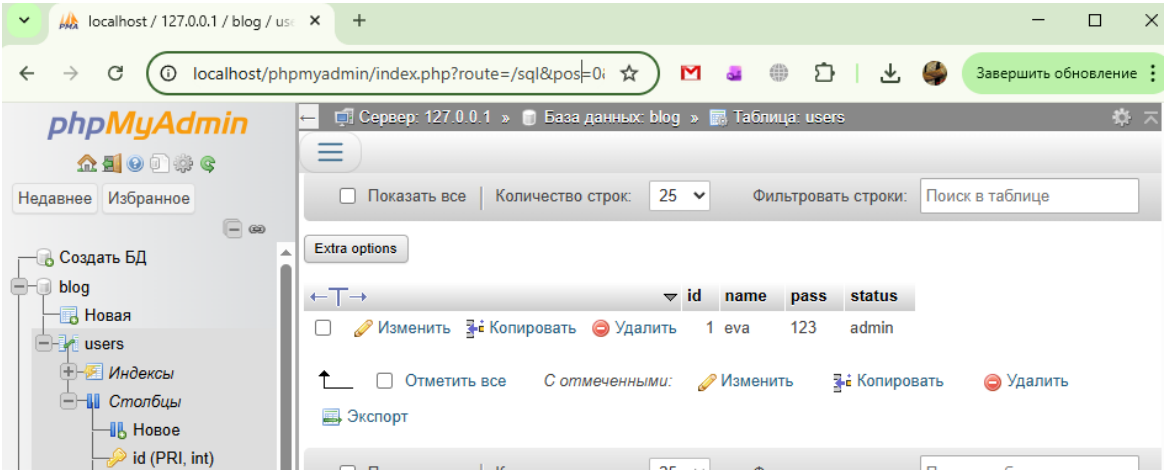
insert_user_db.php

```
<?php
try {
    $conn = NEW PDO("mysql:host=localhost;dbname=blog", "root", "");

    // SQL-выражение для добавления данных
    $sql = "INSERT INTO Users (name, pass, status) VALUES ('eva', 123,
'admin')";

    $affectedRowsNumber = $conn->EXEC($sql);
    echo "В таблицу Users добавлено строк: $affectedRowsNumber";
}
catch (PDOException $e) {
    echo "Соединение не удалось:" . $e->getMessage();
}
?>
```

http://localhost/insert_user_db.php



http://localhost/phpmyadmin/index.php?route=/sql&pos=0&db=blog&table=users



Команда на добавление здесь выглядит следующим образом:

SQL

```
$sql = "INSERT INTO Users (name, pass, status) VALUES ('eva', 123,
'admin')";
```

То есть в столбец **name** добавляется строка «**eva**», в **pass** - **123** а в столбец **status** - значение **admin**. 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 увеличиваться по сравнению с предыдущей на единицу.

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

> В таблицу **Users** добавлено строк: 1

Множественное добавление

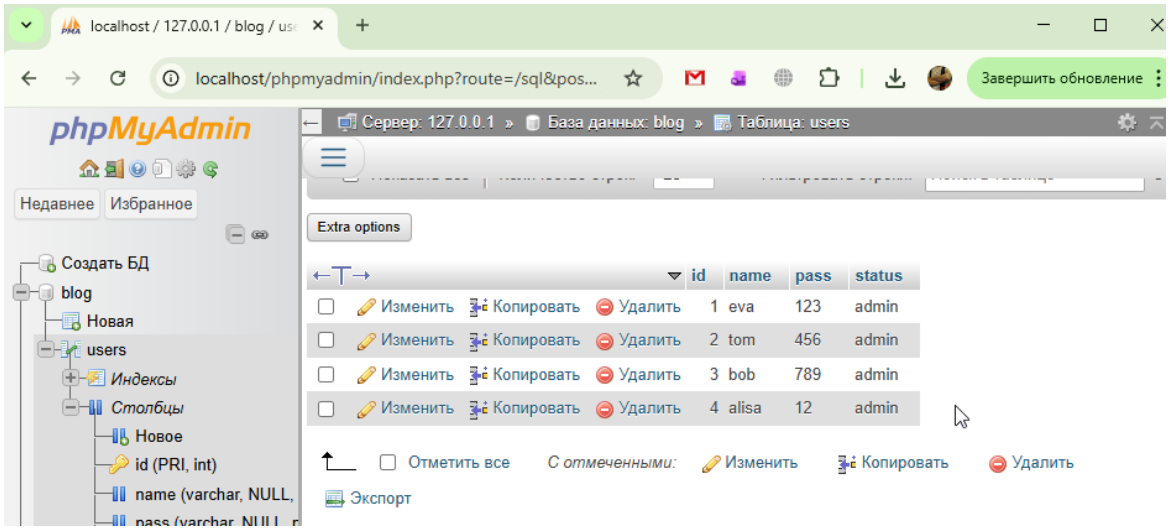
Также мы можем добавить сразу несколько объектов:

insert_user_db.php

```
<?php
try {
    $conn = NEW PDO("mysql:host=localhost;dbname=blog", "root", "");

    // SQL-выражение для добавления данных
    $sql = "INSERT INTO Users (name, age) VALUES ('Tom', 37)";
    $sql = "INSERT INTO Users (name, pass, status) VALUES ('eva', 123,
'admin')";
```

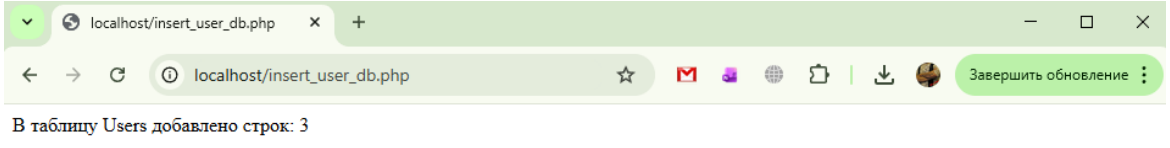
```
    $sql = "INSERT INTO Users (name, pass, status) VALUES
            ('tom', 456, 'admin'),
            ('bob', 789, 'admin'),
            ('alisa', 012, 'admin')";

    $affectedRowsNumber = $conn->EXEC($sql);
    echo "В таблицу Users добавлено строк: $affectedRowsNumber";
}
catch (PDOException $e) {
    echo "Соединение не удалось: " . $e->getMessage();
}
?>
```

http://localhost/phpmyadmin/index.php?route=/sql&pos=0&db=blog&table=users



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

> В таблицу Users добавлено строк: 3



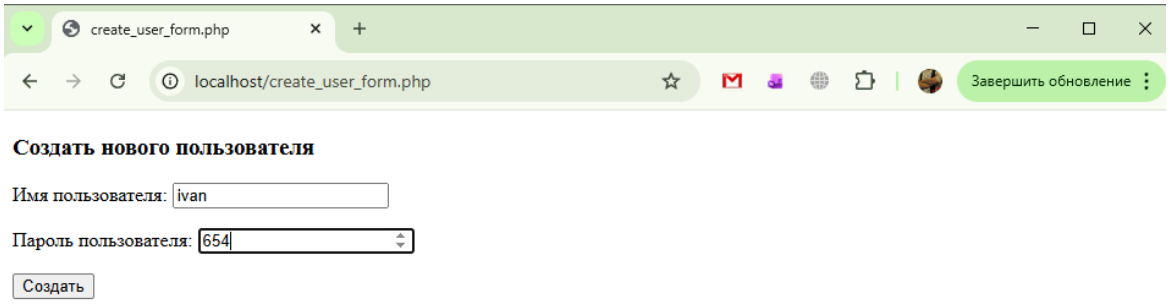
Добавление данных из формы HTML

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:

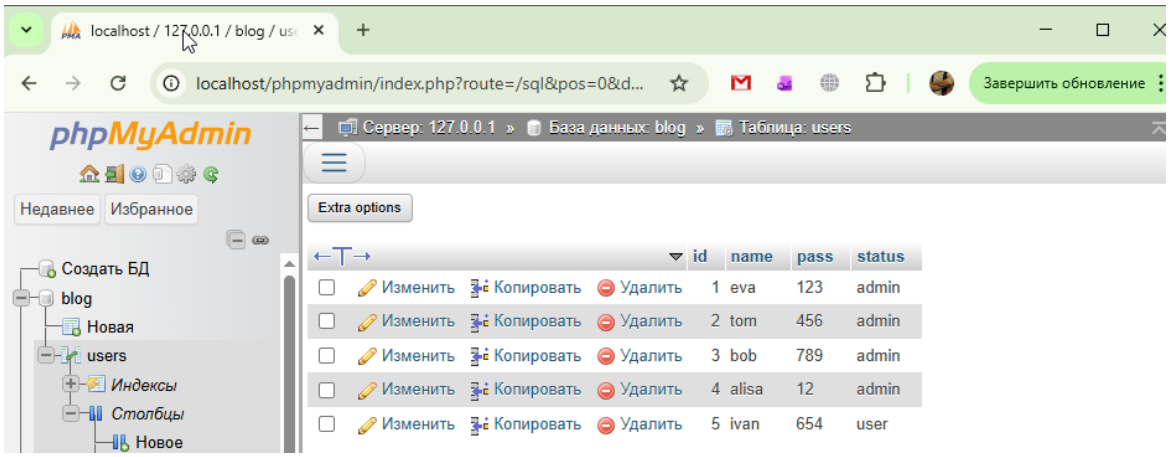
create_user_form.php

```
<!DOCTYPE html>
<html>
<head>
<title>create_user_form.php</title>
<meta charset="utf-8" />
</head>
<body>
<?php
if (isset($_POST["username"]) && isset($_POST["userpass"])) {

    $username = $_POST["username"];
    $userpass = $_POST["userpass"];
    try {
        $conn = new PDO("mysql:host=localhost;dbname=blog", "root",
"");
        $sql = "INSERT INTO Users (name, pass, status) VALUES
('$username', $userpass, 'user')";
        $affectedRowsNumber = $conn->exec($sql);
        // если добавлена как минимум одна строка
        if($affectedRowsNumber > 0 ){
            echo "Данные успешно добавлены: name=$username  pass=
$userpass status= user";
        }
    }
    catch (PDOException $e) {
        echo "Ошибка базы данных: " . $e->getMessage();
    }
}
?>
<h3>Создать нового пользователя</h3>
<form method="post">
    <p>Имя пользователя:
    <input type="text" name="username" /></p>
    <p>Пароль пользователя:
    <input type="number" name="userpass" /></p>
    <input type="submit" value="Создать">
</form>
</body>
</html>
```

http://localhost/create_user_form.php



http://localhost/phpmyadmin/index.php?route=/sql&pos=0&db=blog&table=users



Здесь мы проверяем, пришли ли с сервера данные в POST-запросе, которые имеют ключи **«username»** и **«userpass»**:

SQL

```
IF (isset($_POST["username"]) && isset($_POST["userpass"])) {
```

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

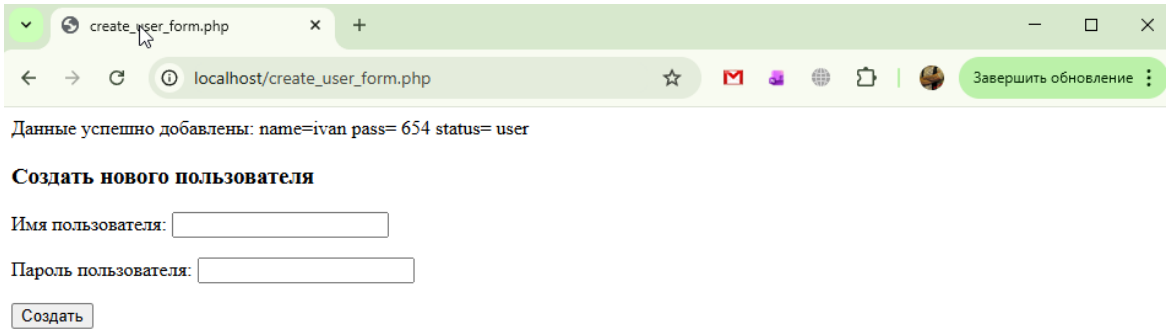
SQL

```
$sql = "INSERT INTO Users (name, pass, status) VALUES ('$username',
$userpass, 'user')";
```

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. Здесь в таблицу добавляется новая строка строки. Соответственно в браузере мы увидим:

Данные успешно добавлены: name=ivan pass= 654 status= user



Параметризация запросов

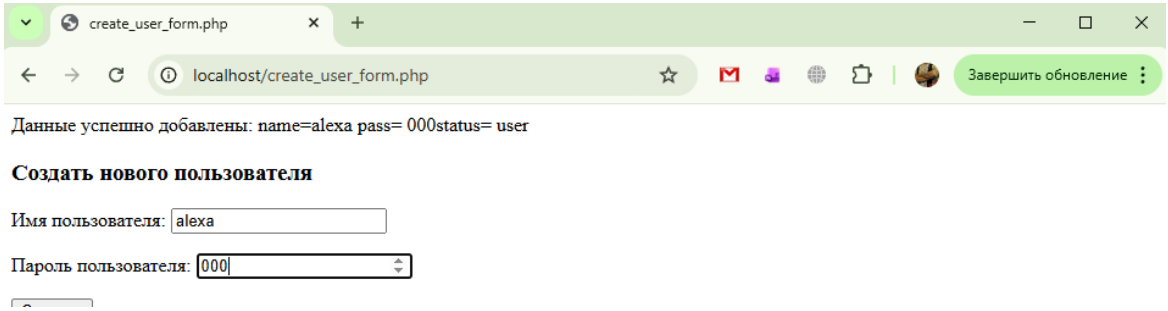
Недостаток выше приведенного скрипа заключается в том, что мы никак не констролируем присылаемые данные и сохраняем их в базу данных как есть. Что несет потенциальную угрозу безопасности, особенно при добавлении строк типа **"; DELETE FROM `Users`; --.** Кроме того, в ряде случае может быть проблематично 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:

create_user_form.php

```
<!DOCTYPE html>
<html>
<head>
<title>create_user_form.php</title>
<meta charset="utf-8" />
</head>
<body>
<?php
IF (isset($_POST["username"]) && isset($_POST["userpass"])) {

    try {
        $conn = NEW PDO("mysql:host=localhost;dbname=blog", "root",
"");
        $user = 'user';
        $sql = "INSERT INTO Users (name, pass, status) VALUES
(:username, :userpass, :user)";
```

```
        // определяем prepared statement
        $stmt = $conn->PREPARE($sql);
        // привязываем параметры к значениям
        $stmt->bindValue(":username", $_POST["username"]);
        $stmt->bindValue(":userpass", $_POST["userpass"]);
        $stmt->bindValue(":user", $user);
        // выполняем prepared statement
        $affectedRowsNumber = $stmt->EXECUTE();
        // если добавлена как минимум одна строка
        IF($affectedRowsNumber > 0 ){
            echo "Данные успешно добавлены: name=" . $_POST["username"]
."  pass= " . $_POST["userpass"] . "status= user";
        }
    }
    catch (PDOException $e) {
        echo "Database error: " . $e->getMessage();
    }
}
?>
<h3>Создать нового пользователя</h3>
<form method="post">
    <p>Имя пользователя:
    <INPUT TYPE="text" name="username" /></p>
    <p>Пароль пользователя:
    <INPUT TYPE="number" name="userpass" /></p>
    <INPUT TYPE="submit" VALUE="Создать">
</form>
</body>
</html>
```



В SQL-выражении теперь применяются параметры:

SQL

```
$sql = "INSERT INTO Users (name, pass, status) VALUES (:username,
:userpass, :user)";
```

:username и **:userpass** - 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:

SQL

```
$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.

SQL

```
$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:

SQL

```
$user = "alexa"
// 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()**:

SQL

```
$sql = "INSERT INTO Users (name, pass, status) VALUES (:username,
:userpass, :user)";
// определяем prepared statement
$stmt = $conn->PREPARE($sql);
// привязываем параметры к значениям
$stmt->bindValue(":username", $_POST["username"]);
$stmt->bindValue(":userpass", $_POST["userpass"]);
```

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

SQL

```
$sql = "INSERT INTO Users (name, pass, status) VALUES (:username,
:userpass, :user)";
$stmt = $conn->PREPARE($sql);
// через массив передаем значения параметрам по имени
$rowsNumber = $stmt->EXECUTE(array(":username" => $_POST["username"],
":userpass" => $_POST["userpass"], ":user" => $user));
```

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:

SQL

```
$sql = "INSERT INTO Users (name, pass, status) VALUES (?, ?, ?)";
// определяем prepared statement
$stmt = $conn->PREPARE($sql);
// привязываем параметры к значениям
$rowsNumber = $stmt->EXECUTE(array($_POST["username"],
$_POST["userpass"], $user));
```

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:

SQL

```
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.

SQL

```
$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):

SQL

```
$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**:

SQL

```
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:

SQL

```
while($row = $result->fetch()){
    $username = $row["name"];
    // операции с $username
}
```

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

SQL

```
foreach($result AS $row){
    $username = $row["name"];
    // операции с $username
}
```

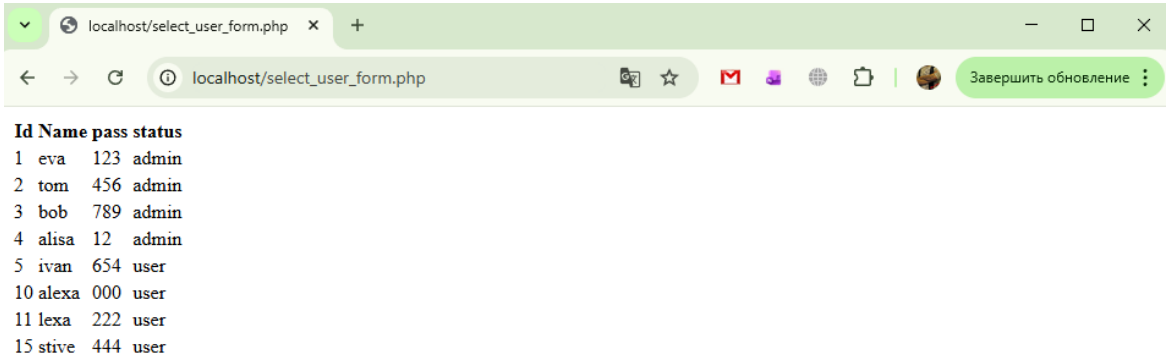
Здесь явным образом не вызывается метод **$result->fetch()**. Формально мы просто перебираем переменную $result как обычный массив, также помещая каждую строку в переменную $row.

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

select_user_form.php

```
<?php
try {
    $conn = NEW PDO("mysql:host=localhost;dbname=blog", "root", "");
    $sql = "SELECT * FROM Users";
    $result = $conn->query($sql);
    echo
"<table><tr><th>Id</th><th>Name</th><th>pass</th><th>status</th></tr>";
    while($row = $result->fetch()){
        echo "<tr>";
            echo "<td>" . $row["id"] . "</td>";
            echo "<td>" . $row["name"] . "</td>";
            echo "<td>" . $row["pass"] . "</td>";
            echo "<td>" . $row["status"] . "</td>";
        echo "</tr>";
    }
    echo "</table>";
}
catch (PDOException $e) {
    echo "Ошибка базы данных: " . $e->getMessage();
}
?>
```

В данном случае полученные данные будут выводиться в таблицу, создаваемую элементом **<table>**:
http://localhost/select_user_form.php



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:

SQL

```
CREATE TABLE Users (id INTEGER AUTO_INCREMENT PRIMARY KEY, name
VARCHAR(30), pass VARCHAR(30),STATUS VARCHAR(30))
```

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

SQL

```
$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**.

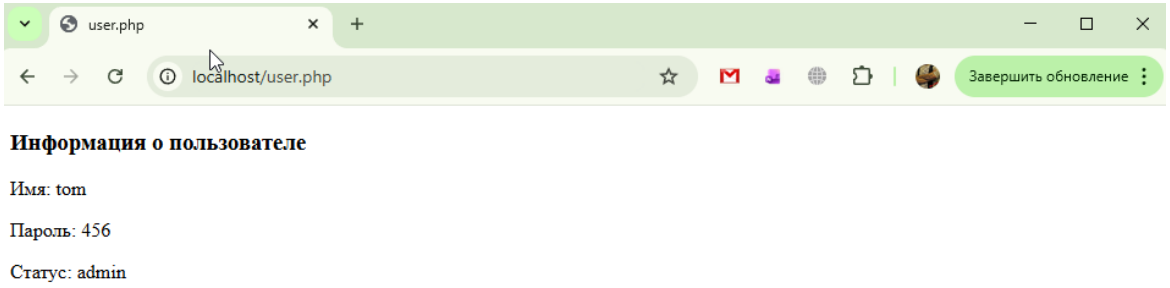
Например, мы хотим получать в **GET**-запросе значение для **id** и по нему получть из базы данных нужные данные. Определим для этого следующий скрипт **user.php**:

user.php

```
<!DOCTYPE html>
```

```
<html>
<head>
<title>USER.php</title>
<meta charset="utf-8" />
</head>
<body>
<?php
// зададим переменную для вывода строки
$_GET["id"] = 2;
IF(isset($_GET["id"]))
{
    try {
        $conn = NEW PDO("mysql:host=localhost;dbname=blog", "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"];
              $userpass = $row["pass"];
              $userstatus = $row["status"];
              echo "<div>
                    <h3>Информация о пользователе</h3>
                    <p>Имя: $username</p>
                    <p>Пароль: $userpass</p>
                    <p>Статус: $userstatus</p>
                </div>";
            }
        }
        ELSE{
            echo "Пользователь не найден";
        }
    }
    catch (PDOException $e) {
        echo "Соединение не удалось: " . $e->getMessage();
    }
}
?>
</body>
</html>
```

http://localhost/user.php



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

SQL

```
$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:

SQL

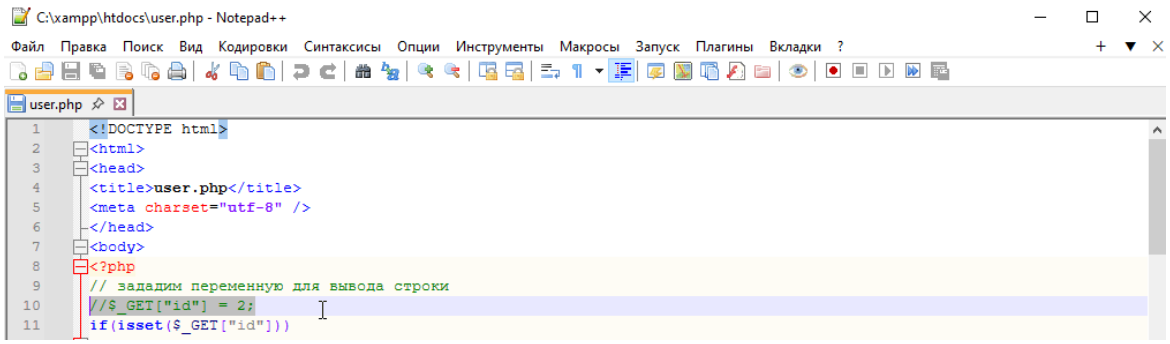
```
$stmt->EXECUTE();
```

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

SQL

```
IF($stmt->rowCount() > 0){
    foreach ($stmt AS $row) {
        $username = $row["name"];
        $userpass = $row["pass"];
        $userstatus = $row["status"];
    }
```

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.
Закоментируем явное определение **$_GET["id"] = 2;**

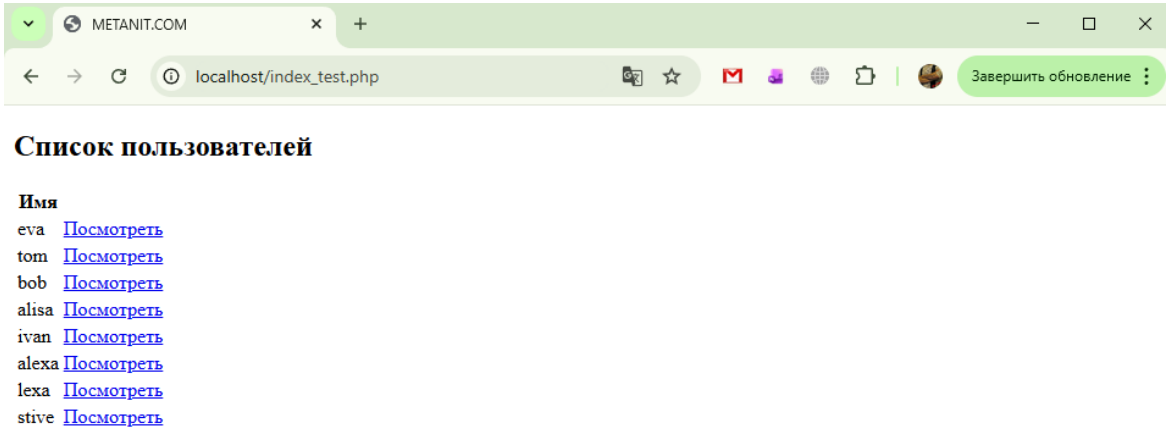


Чтоб было проще обращаться к скрипту **user.php** и передавать ему **id**, определим скрипт **index_test.php**, который будет выводить список объектов:

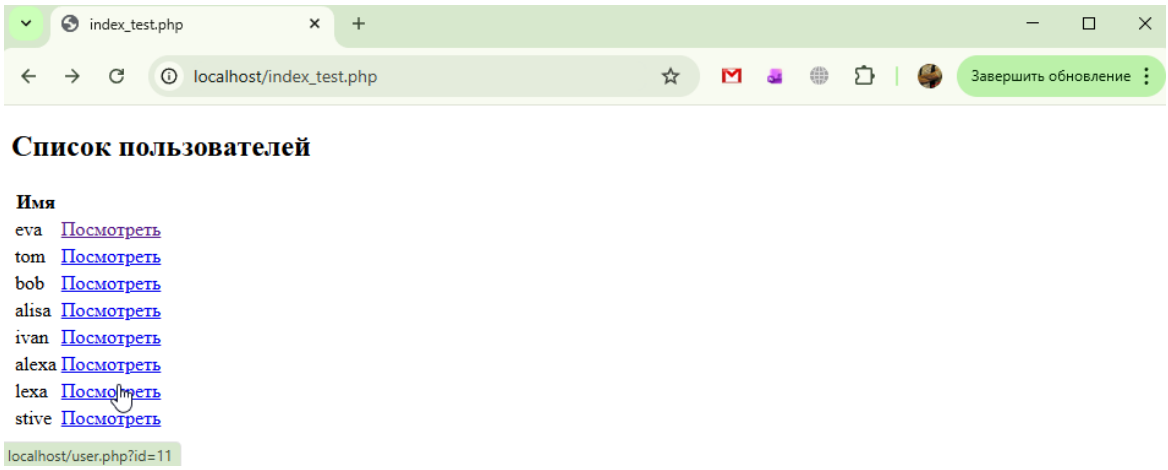
index_test.php

```
<!DOCTYPE html>
<html>
<head>
<title>index_test.php</title>
<meta charset="utf-8" />
</head>
<body>
<h2>Список пользователей</h2>
<?php
try {
    $conn = new PDO("mysql:host=localhost;dbname=blog", "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 "Ошибка базы данных: " . $e->getMessage();
}
?>
</body>
</html>
```

http://localhost/index_test.php



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**:



Выведем значение, нажатием на ссылку

Обновление данных в PDO

Для обновления применяется **sql**-команда **UPDATE**:

SQL

```
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:

SQL

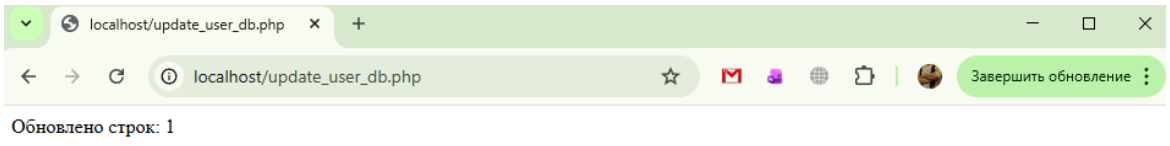
```
CREATE TABLE Users (id INTEGER AUTO_INCREMENT PRIMARY KEY, name
VARCHAR(30), pass VARCHAR(30), STATUS VARCHAR(30))
```

Изменим в этой таблице поле **pass** для строки, которая имеет **id = 11**:

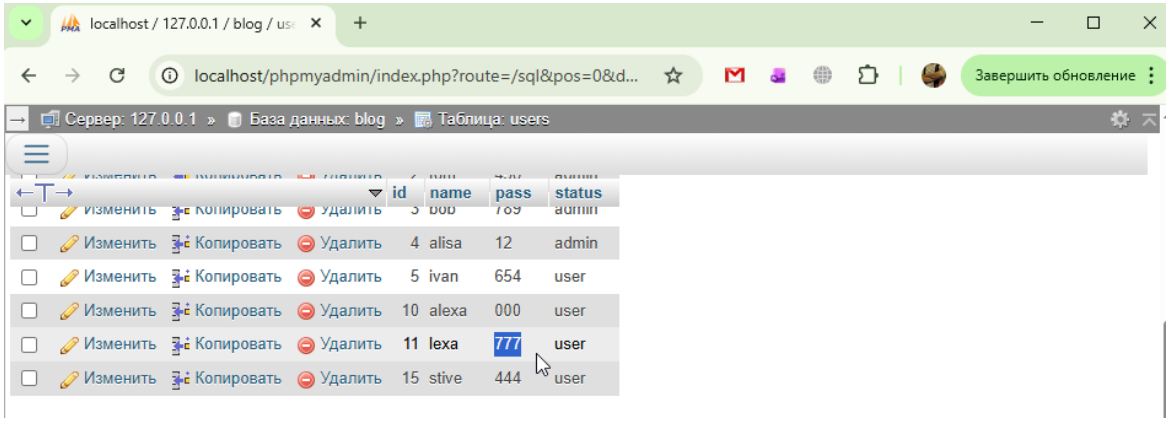
update_user_db.php

```
<?php
try {
    $conn = NEW PDO("mysql:host=localhost;dbname=blog", "root", "");
    $sql = "UPDATE Users SET pass = 777 WHERE id = 11";
    $affectedRowsNumber = $conn->EXEC($sql);
    echo "Обновлено строк: $affectedRowsNumber";
}
catch (PDOException $e) {
    echo "Database error: " . $e->getMessage();
}
?>
```

http://localhost/update_user_db.php



Результат метода $conn→exec() в данном случае количество обновленных строк.
http://localhost/phpmyadmin/index.php?route=/sql&pos=0&db=blog&table=users



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_test.php**, который будет выводить список пользователей:

index_test.php

```
<!DOCTYPE html>
<html>
<head>
<title>index_test.php</title>
<meta charset="utf-8" />
</head>
<body>
<h2>Список пользователей</h2>
<?php
try {
    $conn = new PDO("mysql:host=localhost;dbname=blog", "root", "");
    $sql = "SELECT * FROM Users";
    $result = $conn->query($sql);
    echo "<table><tr><th>Имя</th><th>Пароль</th><th></th></tr>";
    foreach($result as $row){
        echo "<tr>";
            echo "<td>" . $row["name"] . "</td>";
            echo "<td>" . $row["pass"] . "</td>";
            echo "<td><a href='update_user.php?id=" . $row["id"] .
"'>Обновить</a></td>";
        echo "</tr>";
    }
    echo "</table>";
}
catch (PDOException $e) {
    echo "Ошибка базы данных: " . $e->getMessage();
}
?>
</body>
```

```
</html>
```

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

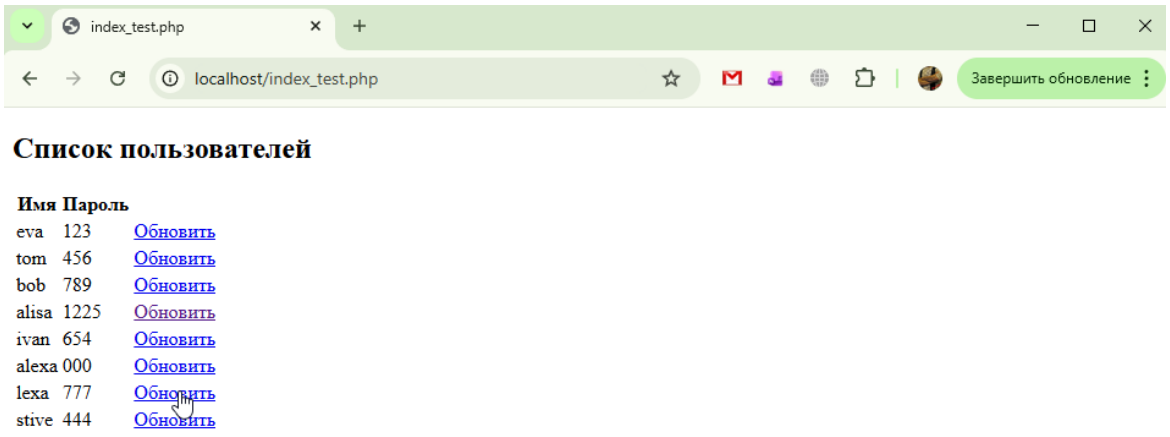
Теперь определим файл **update_user.php** для редактирования пользователей:

update_user.php

```
<?php
try {
    $conn = new PDO("mysql:host=localhost;dbname=blog", "root", "");
}
catch (PDOException $e) {
    die("Database error: " . $e->getMessage());
}
?>
<!DOCTYPE html>
<html>
<head>
<title>update_user.php</title>
<meta charset="utf-8" />
</head>
<body>
<?php
// если запрос  GET
if($_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"];
            $userpass = $row["pass"];
            $userstatus = $row["status"];
        }
        echo "<h3>Обновление пользователя</h3>
                <form method='post'>
                    <input type='hidden' name='id' value='$userid' />
                    <p>Имя:
                    <input type='text' name='name' value='$username'
/></p>
                    <p>Пароль:
```

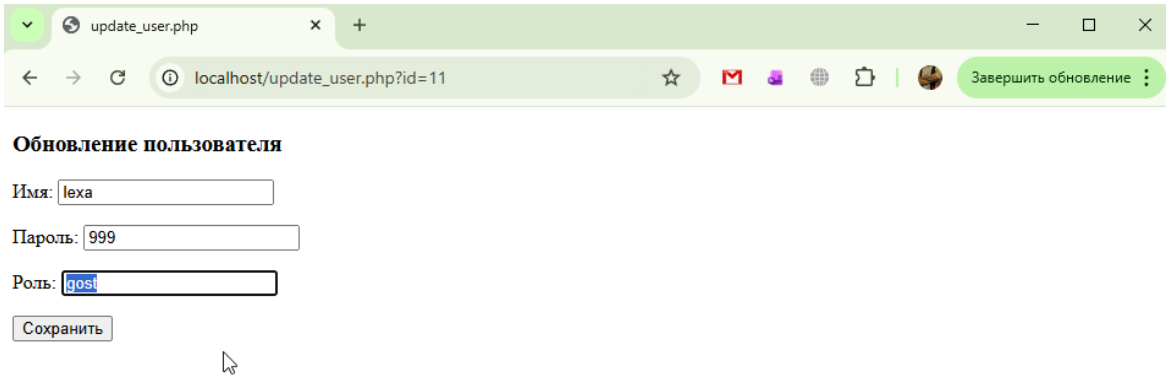
```
                    <input type='text' name='pass' value='$userpass'
/></p>
                    <p>Роль:
                    <input type='text' name='status'
value='$userstatus' /></p>
                    <input type='submit' value='Сохранить'/>
            </form>";
    }
    else{
        echo "Пользователь не найден";
    }
}
elseif (isset($_POST["id"]) && isset($_POST["name"]) &&
isset($_POST["pass"]) && isset($_POST["status"])) {

    $sql = "UPDATE Users SET name = :username, pass = :userpass, status
= :userstatus WHERE id = :userid";
    $stmt = $conn->prepare($sql);
    $stmt->bindValue(":userid", $_POST["id"]);
    $stmt->bindValue(":username", $_POST["name"]);
    $stmt->bindValue(":userpass", $_POST["pass"]);
    $stmt->bindValue(":userstatus", $_POST["status"]);

    $stmt->execute();
    header("Location: index_test.php");
}
else{
    echo "Некорректные данные";
}
?>
</body>
</html>
```

http://localhost/index_test.php

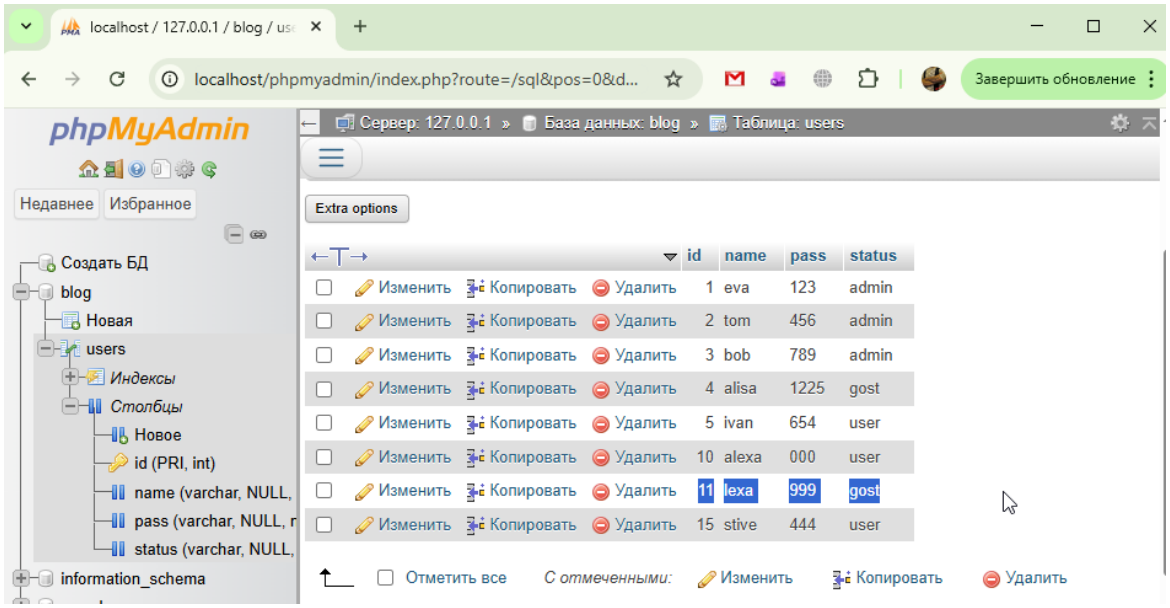


При нажатии на кнопку **Обновить** браузер переместит нас на страницу обновления данных

пользователя **update_user.php**



http://localhost/phpmyadmin/index.php?route=/sql&pos=0&db=blog&table=users



Весь код обновления структурно делится на две части. В первой части мы обрабатываем запрос **Get**. Когда пользователь нажимает на ссылку "**Обновить**" на странице **index_test.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**.

SQL

```
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

SQL

```
$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:

SQL

```
$sql = "UPDATE Users SET name = :username, pass = :userpass, status =
:userstatus WHERE id = :userid";
$stmt = $conn->PREPARE($sql);
$stmt->bindValue(":userid", $_POST["id"]);
$stmt->bindValue(":username", $_POST["name"]);
$stmt->bindValue(":userpass", $_POST["pass"]);
$stmt->bindValue(":userstatus", $_POST["status"]);
$stmt->EXECUTE();
```

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

update_user.php

```
header("Location: index_test.php");
```

Удаление данных в PDO

Для удаления данных применяется sql-команда DELETE:

delete_user.php

```
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:

SQL

```
CREATE TABLE Users (id INTEGER AUTO_INCREMENT PRIMARY KEY, name
VARCHAR(30), pass VARCHAR(30), STATUS VARCHAR(30))
```

Рассмотрим следующий пример удаления пользователя со значением **id = 5**

delete_user.php

```
<?php
try {
    $conn = NEW PDO("mysql:host=localhost;dbname=blog", "root", "");
    $sql = "DELETE FROM Users WHERE id = 5";
    $affectedRowsNumber = $conn->EXEC($sql);
    echo "Удалено строк: $affectedRowsNumber";
}
catch (PDOException $e) {
    echo "Ошибка базы данных: " . $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.

</code> Итак, определим для вывода всех объектов из БД скрипт **index_test.php**:

index_test.php

```
<!DOCTYPE html>
<html>
<head>
<title>index_test.php</title>
<meta charset="utf-8" />
</head>
<body>
<h2>Список пользователей</h2>
<?php
try {
    $conn = new PDO("mysql:host=localhost;dbname=blog", "root", "");
    $sql = "SELECT * FROM Users";
    $result = $conn->query($sql);
    echo "<table><tr><th>Имя</th><th>Пароль</th><th></th></tr>";
    foreach($result as $row){
        echo "<tr>";
            echo "<td>" . $row["name"] . "</td>";
            echo "<td>" . $row["pass"] . "</td>";
            echo "<td><form action='delete_user.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 "Ошибка базы данных: " . $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

exsample.html

```
<a href="http://адрес_нашего_сайта/delete_user.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:

exsample.html

```
<img src="http://адрес_нашего_сайта/delete_user.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.

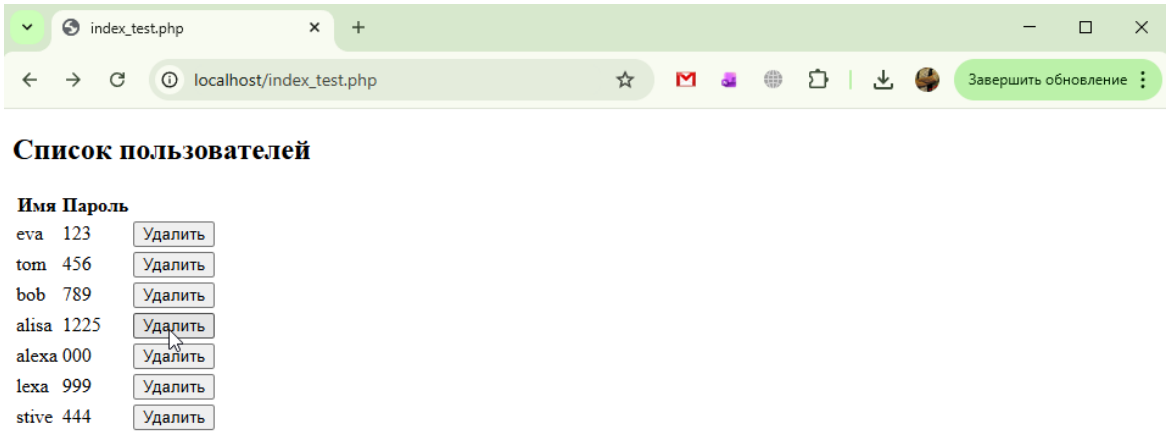
Теперь определим сам скрипт **delete_user.php**, который будет выполнять удаление:

delete_user.php

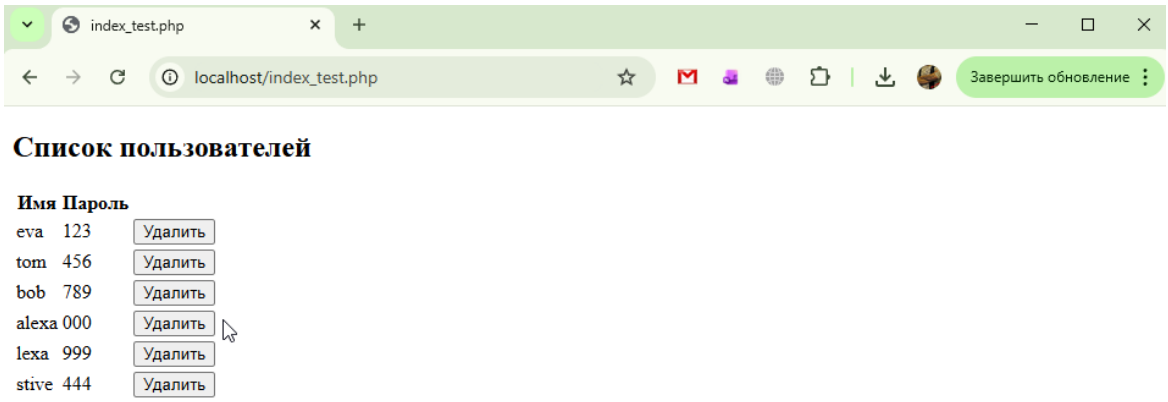
```
<?php
IF(isset($_POST["id"]))
{
    try {
    $conn = NEW PDO("mysql:host=localhost;dbname=blog", "root", "");
        $sql = "DELETE FROM Users WHERE id = :userid";
        $stmt = $conn->PREPARE($sql);
        $stmt->bindValue(":userid", $_POST["id"]);
        $stmt->EXECUTE();
```

```
        header("Location: index_test.php");
    }
    catch (PDOException $e) {
        echo "Ошибка базы данных: " . $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_test.php.
http://localhost/index_test.php



Результат работы скрипта, удаление записи



Дополнения и Файлы