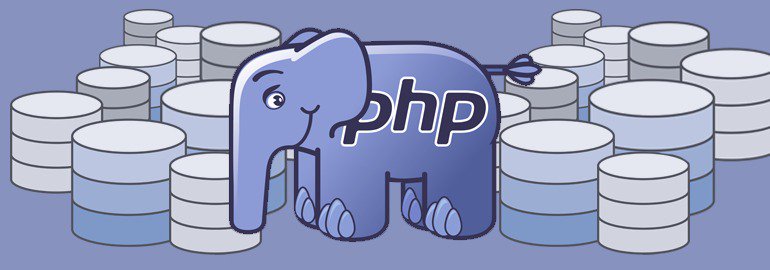
Настройка и использование PDO — расширения PHP Data Objects для работы с базами данных

* 5 октября 2017 в 16:08, [Переводы](https://tproger.ru/category/translations/)
* 13 минут
* 3191



PDO ([PHP Data Objects](https://php.net/manual/ru/intro.pdo.php)) — расширение PHP, которое реализует взаимодействие с базами данных при помощи объектов. Профит в том, что отсутствует привязка к конкретной системе управления базами данных.

Предоставляемый интерфейс поддерживает, среди прочих, такие популярные СУБД:

* MySQL;
* SQLite;
* PostgreSQL;
* Microsoft SQL Server.

В этом руководстве представлен обзор PDO:

1. [Пошаговое описание работы с базами данных](https://tproger.ru/translations/how-to-configure-and-use-pdo/#pdo_usage), начиная с установки соединения до выполнения выборки.
2. [Порядок использования подготовленных запросов](https://tproger.ru/translations/how-to-configure-and-use-pdo/#prepared_statements).
3. [Настройка обработки ошибок](https://tproger.ru/translations/how-to-configure-and-use-pdo/#pdo_errors).

Для работы потребуются:

* базовые знания MySQL и опыт использования команды mysql в консоли;
* понимание основ объектно-ориентированного программирования;
* PHP >= 5.1;
* рабочая СУБД MySQL/MariaDB.

Создание тестовой базы данных и таблицы

Для начала создадим базу данных для этого руководства:

CREATE DATABASE solar\_system;

GRANT ALL PRIVILEGES ON solar\_system.\* TO 'testuser'@'localhost'

IDENTIFIED BY 'testpassword';

Пользователю с логином testuser и паролем testpassword предоставили полные права доступа к базе solar\_system.

Теперь создадим таблицу и заполним данными, астрономическая точность которых не подразумевается:

USE solar\_system;

CREATE TABLE planets (

id TINYINT(1) UNSIGNED NOT NULL AUTO\_INCREMENT,

PRIMARY KEY(id),

name VARCHAR(10) NOT NULL,

color VARCHAR(10) NOT NULL

);

INSERT INTO planets(name, color) VALUES('earth', 'blue'), ('mars', 'red'), ('jupiter', 'strange');

Описание соединения

Теперь, когда создана база, определим DSN ([Data Source Name](https://en.wikipedia.org/wiki/Data_source_name" \t "_blank)) — сведения для подключения к базе, представленные в виде строки. Синтаксис описания отличается в зависимости от используемой СУБД. В примере работаем с MySQL/MariaDB, поэтому указываем:

* [тип драйвера](https://php.net/manual/ru/pdo.drivers.php);
* имя хоста, где расположена СУБД;
* порт (необязательно, если используется стандартный порт 3306);
* имя базы данных;
* кодировку (необязательно).

Строка DSN в этом случае выглядит следующим образом:

$dsn = "mysql:host=localhost;port=3306;dbname=solar\_system;charset=utf8";

Первым указывается database prefix. В примере — mysql. Префикс отделяется от остальной части строки двоеточием, а каждый следующий параметр — точкой с запятой.

Создание PDO-объекта

Теперь, когда строка DSN готова, создадим PDO-объект. Конструктор на входе принимает следующие параметры:

1. Строку DSN.
2. Имя пользователя, имеющего доступ к базе данных.
3. Пароль этого пользователя.
4. Массив с дополнительными параметрами (необязательно).

$options = [

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

PDO::ATTR\_DEFAULT\_FETCH\_MODE => PDO::FETCH\_ASSOC

];

$pdo = new PDO($dsn, 'testuser', 'testpassword', $options);

Дополнительные параметры можно также определить после создания объекта с помощью метода SetAttribute:

$pdo->SetAttribute(PDO::ATTR\_ERRMODE, PDO::ERRMODE\_EXCEPTION);

Определение метода выборки по умолчанию

PDO::DEFAULT\_FETCH\_MODE — важный параметр, который определяет метод выборки по умолчанию. Указанный метод используется при получении результата выполнения запроса.

PDO::FETCH\_BOTH

Режим по умолчанию. Результат выборки индексируется как номерами (начиная с 0), так и именами столбцов:

$stmt = $pdo->query("SELECT \* FROM planets");

$results = $stmt->fetch(PDO::FETCH\_BOTH);

После выполнения запроса с этим режимом к тестовой таблице планет получим следующий результат:

Array

(

[id] => 1

[0] => 1

[name] => earth

[1] => earth

[color] => blue

[2] => blue

)

PDO::FETCH\_ASSOC

Результат сохраняется в ассоциативном массиве, в котором ключ — имя столбца, а значение — соответствующее значение строки:

$stmt = $pdo->query("SELECT \* FROM planets");

$results = $stmt->fetch(PDO::FETCH\_ASSOC);

В результате получим:

Array

(

[id] => 1

[name] => earth

[color] => blue

)

PDO::FETCH\_NUM

При использовании этого режима результат представляется в виде массива, индексированного номерами столбцов (начиная с 0):

Array

(

[0] => 1

[1] => earth

[2] => blue

)

PDO::FETCH\_COLUMN

Этот вариант полезен, если нужно получить перечень значений одного поля в виде одномерного массива, нумерация которого начинается с 0. Например:

$stmt = $pdo->query("SELECT name FROM planets");

В результате получим:

Array

(

[0] => earth

[1] => mars

[2] => jupiter

)

PDO::FETCH\_KEY\_PAIR

Используем этот вариант, если нужно получить перечень значений двух полей в виде ассоциативного массива. Ключи массива — это данные первого столбца выборки, значения массива — данные второго столбца. Например:

$stmt = $pdo->query("SELECT name, color FROM planets");

$result = $stmt->fetchAll(PDO::FETCH\_KEY\_PAIR);

В результате получим:

Array

(

[earth] => blue

[mars] => red

[jupiter] => strange

)

PDO::FETCH\_OBJECT

При использовании PDO::FETCH\_OBJECT для каждой извлеченной строки создаётся анонимный объект. Его общедоступные ([public](https://php.net/manual/ru/language.oop5.visibility.php" \t "_blank)) свойства — имена столбцов выборки, а результаты запроса используются в качестве их значений:

$stmt = $pdo->query("SELECT name, color FROM planets");

$results = $stmt->fetch(PDO::FETCH\_OBJ);

В результате получим:

stdClass Object

(

[name] => earth

[color] => blue

)

PDO::FETCH\_CLASS

В этом случае, как и в предыдущем, значения столбцов становятся свойствами объекта. Однако требуется указать существующий класс, который будет использоваться для создания объекта. Рассмотрим это на примере. Для начала создадим класс:

class Planet

{

private $name;

private $color;

public function setName($planet\_name)

{

$this->name = $planet\_name;

}

public function setColor($planet\_color)

{

$this->color = $planet\_color;

}

public function getName()

{

return $this->name;

}

public function getColor()

{

return $this->color;

}

}

Обратите внимание, что у класса Planet закрытые (private) свойства и нет конструктора. Теперь выполним запрос.

Если используется метод fetch с PDO::FETCH\_CLASS, перед отправкой запроса на получение данных нужно применить метод setFetchMode:

$stmt = $pdo->query("SELECT name, color FROM planets");

$stmt->setFetchMode(PDO::FETCH\_CLASS, 'Planet');

Первый параметр, который передаем методу setFetchMode, — константа PDO::FETCH\_CLASS. Второй параметр — имя класса, который будет использоваться при создании объекта. Теперь выполним:

$planet = $stmt->fetch();

var\_dump($planet);

В результате получим объект Planet:

Planet Object

(

[name:Planet:private] => earth

[color:Planet:private] => blue

)

Значения, полученные в результате запроса, назначены соответствующим свойствам объекта, даже закрытым.

Определение свойств после выполнения конструктора

В классе Planet нет явного конструктора, поэтому проблем при назначении свойств не будет. При наличии у класса конструктора, в котором свойство было назначено или изменено, они будут перезаписаны.

При использовании константы FETCH\_PROPS\_LATE значения свойств будут присваиваться после выполнения конструктора:

class Planet

{

private $name;

private $color;

public function \_\_construct($name = moon, $color = grey)

{

$this->name = $name;

$this->color = $color;

}

public function setName($planet\_name)

{

$this->name = $planet\_name;

}

public function setColor($planet\_color)

{

$this->color = $planet\_color;

}

public function getName()

{

return $this->name;

}

public function getColor()

{

return $this->color;

}

}

Мы изменили класс Planet, добавив конструктор, который принимает на входе два аргумента: name (имя) и color (цвет). Значения этих полей по умолчанию: moon (луна) и gray (серый) соответственно.

Если не использовать FETCH\_PROPS\_LATE, при создании объекта свойства будут перезаписаны значениями по умолчанию. Проверим это. Сначала выполним запрос:

$stmt = $pdo->query("SELECT name, color FROM solar\_system WHERE name = 'earth'");

$stmt->setFetchMode(PDO::FETCH\_CLASS, 'Planet');

$planet = $stmt->fetch();

var\_dump($planet);

В результате получим:

object(Planet)#2 (2) {

["name":"Planet":private]=>

string(4) "moon"

["color":"Planet":private]=>

string(4) "gray"

}

Как и ожидалось, извлеченные из базы данных значения перезаписаны. Теперь рассмотрим решение задачи с помощью FETCH\_PROPS\_LATE (запрос аналогичный):

$stmt->setFetchMode(PDO::FETCH\_CLASS|PDO::FETCH\_PROPS\_LATE, 'Planet');

$planet = $stmt->fetch();

var\_dump($planet);

В результате получим то, что нужно:

object(Planet)#4 (2) {

["name":"Planet":private]=>

string(5) "earth"

["color":"Planet":private]=>

string(4) "blue"

}

Если у конструктора класса нет значений по умолчанию, а они нужны, параметры конструктора задаются  при вызове метода setFetchMode третьим аргументом в виде массива. Например:

class Planet

{

private $name;

private $color;

public function \_\_construct($name, $color)

{

$this->name = $name;

$this->color = $color;

}

[...]

}

Аргументы конструктора обязательны, поэтому выполним:

$stmt->setFetchMode(PDO::FETCH\_CLASS|PDO::FETCH\_PROPS\_LATE, 'Planet', ['moon', 'gray']);

Входящие параметры выступают также в роли значений по умолчанию, которые нужны для инициализации. В дальнейшем они будут перезаписаны значениями из базы данных.

Получение нескольких объектов

Множественные результаты извлекаются в виде объектов с помощью метода fetchвнутри цикла while:

while ($planet = $stmt->fetch()) {

// обработка результатов

}

Или путём выборки всех результатов сразу. Во втором случае используется метод fetchAll, причём режим указывается в момент вызова:

$stmt->fetchAll(PDO::FETCH\_CLASS|PDO\_FETCH\_PROPS\_LATE, 'Planet', ['moon', 'gray']);

PDO::FETCH\_INTO

При выборе этого варианта выборки PDO не создаёт новый объект, а обновляет свойства существующего. Однако это возможно только для общедоступных (public) свойств или при использовании в объекте [«магического» метода](https://php.net/manual/ru/language.oop5.magic.php) \_\_set.

Подготовленные и прямые запросы

В PDO два способа выполнения запросов:

* прямой, который состоит из одного шага;
* подготовленный, который состоит из двух шагов.

Прямые запросы

Существует два метода выполнения прямых запросов:

* query используется для операторов, которые не вносят изменения, например SELECT. Возвращает объект PDOStatemnt, из которого с помощью методов fetch или fetchAll извлекаются результаты запроса;
* exec используется для операторов вроде INSERT, DELETE или UPDATE. Возвращает число обработанных запросом строк.

Прямые операторы используются только в том случае, если в запросе отсутствуют переменные и есть уверенность, что запрос безопасен и правильно экранирован.

Подготовленные запросы

PDO поддерживает подготовленные запросы ([prepared statements](https://php.net/manual/ru/pdo.prepared-statements.php" \t "_blank)), которые полезны для защиты приложения от [SQL-инъекций](https://tproger.ru/translations/webapp-security/#sql_injection): метод prepare выполняет необходимые экранирования.

Рассмотрим пример. Требуется вставить свойства объекта Planet в таблицу Planets. Сначала подготовим запрос:

$stmt = $pdo->prepare("INSERT INTO planets(name, color) VALUES(?, ?)");

Используем метод prepare, который принимает как аргумент SQL-запрос с псевдопеременными ([placeholders](https://php.net/manual/ru/pdostatement.bindvalue.php" \t "_blank)). Псевдопеременные могут быть двух типов: неименнованые и именованные.

Неименованные псевдопеременные

Неименованные псевдопеременные ([positional placeholders](https://php.net/manual/ru/pdostatement.bindvalue.php" \t "_blank)) отмечаются символом ?. Запрос в результате получается компактным, но требуется предоставить значения для подстановки, размещенные в том же порядке. Они передаются в виде массива через метод execute:

$stmt->execute([$planet->name, $planet->color]);

Именованные псевдопеременные

При использовании именованных псевдопеременных ([named placeholders](https://php.net/manual/ru/pdostatement.bindvalue.php" \t "_blank)) порядок передачи значений для подстановки не важен, но код в этом случае становится не таким компактным. В метод execute данные передаются в виде ассоциативного массива, в котором каждый ключ соответствует имени псевдопеременной, а значение массива — значению, которое требуется подставить в запрос. Переделаем предыдущий пример:

$stmt = $pdo->prepare("INSERT INTO planets(name, color) VALUES(:name, :color)");

$stmt->execute(['name' => $planet->name, 'color' => $planet->color]);

Методы prepare и execute используются как при выполнении запросов на изменение, так и при выборке.

А информацию о количестве обработанных строк при необходимости предоставит метод rowCount.

Управление поведением PDO при ошибках

Параметр выбора режима ошибок PDO::ATTR\_ERRMODE используется для определения поведения PDO в случае ошибок. Доступно три варианта: PDO::ERRMODE\_SILENT, PDO::ERRMODE\_EXCEPTION и PDO::ERRMODE\_WARNING.

PDO::ERRMODE\_SILENT

Вариант по умолчанию. PDO просто запишет информацию об ошибке, которую помогут получить методы [errorCode](https://php.net/manual/ru/pdo.errorcode.php" \t "_blank) и [errorInfo](https://php.net/manual/ru/pdo.errorinfo.php" \t "_blank).

PDO::ERRMODE\_EXCEPTION

Это предпочтительный вариант, при котором в дополнение к информации об ошибке PDO выбрасывает исключение ([PDOException](https://php.net/PDOException" \t "_blank)). Исключение прерывает выполнение скрипта, что полезно при использовании транзакций PDO. Пример приведён [ниже](https://tproger.ru/translations/how-to-configure-and-use-pdo/#transaction) при описании транзакций.

PDO::ERRMODE\_WARNING

В этом случае PDO также записывает информацию об ошибке. Поток выполнения скрипта не прерывается, но выдаются предупреждения.

Методы bindValue и bindParam

Для подстановки значений в запросе можно также использовать методы bindValue и bindParam. Первый связывает значение переменной с псевдопеременной, которая использована при подготовке запроса:

$stmt = $pdo->prepare("INSERT INTO planets(name, color) VALUES(:name, :color)");

$stmt->bindValue('name', $planet->name, PDO::PARAM\_STR);

Связали значение переменной $planet->name с псевдопеременной :name. Обратите внимание, что при использовании методов bindValue и bindParam как третий аргумент указывается тип переменной, используя соответствующие константы PDO. В примере — PDO::PARAM\_STR.

Метод bindParam привязывает переменную к псевдопеременной. В этом случае переменная связана с псевдопеременной ссылкой, а значение будет подставлено в запрос только после вызова метода execute. Рассмотрим на примере:

$stmt->bindParam('name', $planet->name, PDO::PARAM\_STR);

Транзакции в PDO

Транзакции позволяют сохранить на некоторое время и организовать выполнение нескольких запросов «пакетом». Запросы, включённые в транзакцию, применяются только в том случае, если при выполнении отсутствуют ошибки. Транзакции поддерживаются не всеми СУБД и работают не со всеми SQL-конструкциями, так как некоторые из них вызывают неявное выполнение. Список таких конструкций можно найти на [сайте MariaDB](https://mariadb.com/kb/en/library/sql-statements-that-cause-an-implicit-commit/).

Представим необычный пример. Пользователю требуется выбрать список планет, причём каждый раз при выполнении запроса текущие данные удаляются из базы, а потом вставляются новые. Если после удаления произойдёт ошибка, то следующий пользователь получит пустой список. Чтобы этого избежать, используем транзакции:

$pdo->beginTransaction();

try {

$stmt1 = $pdo->exec("DELETE FROM planets");

$stmt2 = $pdo->prepare("INSERT INTO planets(name, color) VALUES (?, ?)");

foreach ($planets as $planet) {

$stmt2->execute([$planet->getName(), $planet->getColor()]);

}

$pdo->commit();

} catch (PDOException $e) {

$pdo->rollBack();

}

Метод beginTransaction отключает автоматическое выполнение запросов, а внутри конструкции try-catch запросы выполняются в нужном порядке. Если не возникнет исключений PDOException, запросы выполнятся с помощью метода commit. В противном случае откатятся с помощью метода rollback, а автоматическое выполнение запросов восстановится.

Таким образом появилась согласованность выполнения запросов. Очевидно, что для этого параметру PDO::ATTR\_ERRMODE необходимо установить значение PDO::ERRMODE\_EXCEPTION.

Заключение

Теперь, когда работа с PDO описана, отметим его основные преимущества:

* с PDO легко перенести приложение на другие СУБД;
* поддерживаются все популярные СУБД;
* встроенная система управления ошибками;
* разнообразные варианты представления результатов выборки;
* поддерживаются подготовленные запросы, которые сокращают код и делают его устойчивым к SQL-инъекциям;
* поддерживаются транзакции, которые помогают сохранить целостность данных и согласованность запросов при параллельной работе пользователей.