PHP PDO MySQL Примеры. Как использовать?

22 Ноябрь 2012, 12:32

Я думаю настало время повысить свой экспириенс, и перейти при работе с базой данных с mysql\_ функций на PDO. Эта библиотека является мощным и быстрым дополнением к PHP. Одно из ее преимуществ – работа со многими базами данных (MS SQL , MySQL , PostgreSQL , Oracle и др.). Также отличительной характеристикой этой библиотеки являются подготовленные выражения, так называемые prepared statements, которые должны ускорить работу с базой, а главное сделать обмен данным безопасными и забыть о таких уязвимостях как sql-enjection. Кроме того, есть и другие очень полезные возможности. В этой статье я попытался собрать часто используемые примеры работ, по которым можно сразу понять суть работы PDO.

В PHP существуют три расширения для работы с базой MySQL: mysql, mysqli и PDO. PHP PDO (PHP Data Objects) включено в PHP 5.1 и выше. Как я понял, на сегодняшний момент, функции для работы с базой данных mysql\_ не рекомендуют к использованию, так как разработка php\_mysql остановилась на поддержке функционала MySQL 4.1.3. и также не поддерживает транзакции, объектный интерфейс и подвержено уязвимостям при подстановке значений в запрос. После mysql\_ появилось расширение mysqli (MySQL Improved в 5 версии), которое поддерживает новые возможности MySQL и использует как ОПП синтаксис так и процедурное программирование. Все эти библиотеки используют стандартную клиентскую библиотеку MySQL (libmysql). В этой же заметке давайте на живых примерах посмотрим как осуществляется работа с mysql, при помощи самого свежего расширения – **PDO**.

PDO Соединение с БД

|  |  |
| --- | --- |
| 1  2 | //пример соединения с MySQL при помощи PDO  $db = new PDO('mysql:host=localhost;dbname=test', $user, $pass); |

Как только будет установлено успешное соединение с конкретным сервером базы данных, то будет возвращен объект PDO. Этот объект позволяет выполнять самые разнообразные задачи баз данных.

Если есть какие-либо ошибки подключения, то сработает механизм исключений – PDOException. Нужно всегда оборачивать операции PDO в блок try/catch. Вы можете поймать исключение, если вы хотите обрабатывать ошибки, или вы можете оставить его для глобального обработчика исключений (exception), которые вы создали с помощью set\_exception\_handler (). В PDO есть специальные функции для ошибок: errorCode() – вернет номер ошибки, errorInfo() – вернет массив с номером ошибки и описанием. Они нужны так как по умолчанию режим работы с ошибками стоит ERRMODE\_SILENT. В этом случае чтобы увидеть эти ошибки придется их вызвать:

|  |  |
| --- | --- |
| 1  2 | echo $conn->errorCode();  echo $conn->errorInfo(); |

Чтобы этого не делать, в режиме разработке проще сразу выставить нужный режим работы с ошибками: ATTR\_ERRMODE и ERRMODE\_EXCEPTION. Можно также прописать кодировку работы с базой. В итоге у нас получится такой вот код подключения:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | try {    $db = new PDO("mysql:host=$host;dbname=$dbname", $user, $password);    $db->setAttribute(PDO::ATTR\_ERRMODE, PDO::ERRMODE\_EXCEPTION);    $db->exec("set names utf8");  }  catch(PDOException $e) {      echo $e->getMessage();  } |

После успешного подключения к базе данных, экземпляр класса PDO возвращается в сценарий. $db содержит дескриптор базы данных. Соединение остается активным в течение всей жизни объекта PDO. Чтобы закрыть соединение, вам нужно уничтожить объект с гарантией, что все остальные ссылки на него будут удалены. Сделать это можно путем присвоения переменной, которая содержит объект, значения NULL. Если вы не сделаете этого явно, PHP будет автоматически закрывать соединение после завершения работы скрипта.

|  |  |
| --- | --- |
| 1  2 | //Закрытие соединения  $db = null; |

Многие веб-приложения выиграют от создания постоянных подключений к серверам баз данных. Постоянные соединения не закрываются в конце скрипта, а кэшируются и повторно используются, когда другой сценарий запрашивает соединение, используя те же учетные данные. Постоянное соединение позволяет уменьшить ресурсы на создание нового соединения каждый раз, когда сценарий должен обратиться к базе данных, что приводит к более быстрой работе веб-приложения.

|  |  |
| --- | --- |
| 1  2  3  4 | //Постоянное соединение  $dbh = new PDO('mysql:host=localhost;dbname=test', $user, $pass, array(  PDO::ATTR\_PERSISTENT => true  )); |

Теперь, когда вы увидели, как открывать и закрывать соединение, давайте рассмотрим другие примеры работы с PDO. В данном случае я собираюсь показать вам, как выполнять запросы к конкретной базе данных. Запросы можно делать 3 функциями: exec(), query() и prepare+execute.

Exec()

Первый – exec вернет только кол-во задействованных строк или FALSE при ошибке и используется там, где не возвращаются данные, например при удалении:

|  |  |
| --- | --- |
| 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 | //использование метода exec()  try{  $db = new PDO('mysql:host=localhost;dbname=test','user','password');  $delrows=$db->exec('DELETE FROM users WHERE id>20');  echo 'Количество удаленных строк: '.$delrows;  }  catch(PDOException $e){  echo 'Error : '.$e->getMessage();  exit();  }    //Еще  $db->exec('DELETE FROM folks WHERE 1');    //или  $db->exec("SET time\_zone = '-8:00'");    //или  $db->exec("CREATE TABLE `test`(      id INT PRIMARY KEY AUTO\_INCREMENT,      name VARCHAR(20) NOT NULL DEFAULT '',      email VARCHAR(50) NOT NULL DEFAULT '')");    //или  $db->exec("SET CHARACTER SET utf8"); |

Query()

Второй – query() вернет результат в объекте PDOStatement. Также возвращает результат или FALSE при ошибке. Ему можно доверить простые запросы. Можно использовать query() с условием (я правда не знаю зачем кому то это может понадобиться), но тогда все равно придется экранировать данные методом PDO::quote

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | //Простые запросы  $db->query("SET CHARACTER SET utf8");  $db->query("SELECT \* FROM users");    //Можно вычислить количество строк  $stmt = $db->query('SELECT \* FROM table');  $row\_count = $stmt->rowCount();  echo $row\_count.' rows selected';    //Еще вариант с количеством  $stmt = $db->query('SELECT \* from users');  $rows = $stmt->fetchAll();  $count = count($rows);  foreach($rows as $row)  {  print\_r($row);  }    //Запрос с условием и экранированием  $conn->query('SELECT \* FROM table WHERE id = ' . $conn->quote($id)); |

**lastInsertId()** возвращает идентификатор последней вставленной строки базы данных.

|  |  |
| --- | --- |
| 1  2  3 | //метод lastInsertId() возвращает id последней записи  $db->query("INSERT INTO users SET name='Vasya',address='Here',email='vasya@test.com'");  $insertId=$db->lastInsertId(); |

Подготовленные выражения – prepared statments.

Третий способ**-**prepare+execute - **Подготовленные выражения**, они же подготовленные инструкции они же плейсхолдеры они же **prepared statments**или связываемые переменные, позволяют определить выражение один раз, а затем многократно его выполнять с разными параметрами. Также они позволяют отделить переменные от запроса, что делает код безопаснее и повышает скорость выполнения. В вашем коде больше не надо будет пытаться очистить передаваемые данные. Мы сделаем это только один раз перед выполнением запроса к БД. Для этого используем функцию **Prepare()**; В качестве параметра она принимает SQL запрос, но в нем, вместо переменных используются метки, в виде знака вопроса ‘?’ или номеров ‘:1′, или переменой, имя которой начинается с двоеточия ‘:’. Если вы остановились на знаках вопроса (:цифрах), то вам надо в функцию execute передать массив значений, в соответствующей последовательности. Если ваш выбор именованные переменные, то надо назначить каждой переменной значение через одну из двух функций: либо bindValue(), которая присваивает псевдо-переменной значение, либо bindParam(), которая связывает псевдо-переменную с настоящей переменной. Третьим параметром можно указать тип переменной, например $db->bindParam(‘:id’,$id, PDO::PARAM\_INT).

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | //Не именованные метки  try {  $stmt = $db->prepare("INSERT INTO test (label,color) VALUES (?,?)");  $stmt -> execute(array('perfect','green'));  }  catch(PDOException $e){  echo 'Error : '.$e->getMessage();  exit();  }  //stmt - это "дескриптор состояния"    //Именованные метки  $stmt = $db->prepare("INSERT INTO test (label,color) VALUES (:label,:color)");  $stmt -> execute(array('label'=>'perfect', 'color'=>'green'));    //Еще вариант  $stmt = $db->prepare("INSERT INTO users (firstname, lastname, email) VALUES (:firstname, :lastname, :email)");  $stmt->bindParam(':firstname', $firstname);  $stmt->bindParam(':lastname', $lastname);  $stmt->bindParam(':email', $email);  $firstname = "John";  $lastname = "Smith";  $email = "john@test.com";  $stmt->execute(); |

Еще раз напомню, что если вы не используете подготовленные выражения но все-таки хотите обезопасить передаваемые данные, то это можно сделать при помощи функции PDO:quote.

PDO SELECT данных

Для выборки данных используются методы fetch() или fetchAll(). Перед вызовом функции нужно указать PDO как Вы будете доставать данные из базы. PDO::FETCH\_ASSOC вернет строки в виде ассоциативного массива с именами полей в качестве ключей. PDO::FETCH\_NUM вернет строки в виде числового массива. По умолчанию выборка происходит с PDO::FETCH\_BOTH, который дублирует данные как с численными так и с ассоциативными ключами, поэтому рекомендуется указать один способ, чтобы не иметь дублирующих массивов:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | $stmt = $db->query('SELECT \* from users');  //Установка fetch mode  $stmt->setFetchMode(PDO::FETCH\_ASSOC);  while($row = $stmt->fetch())  {      echo "<p>" . $row['firstname'] . "&nbsp;" . $row['lastname'] . "</p>";      echo "<p>" . $row['email'] . "</p><br />";  } |

Либо можно указать метод выборки в самом методе ->fetch()

|  |  |
| --- | --- |
| 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 | $stmt = $db->query('SELECT \* FROM table');  while($row = $stmt->fetch(PDO::FETCH\_ASSOC)) {  echo $row['field1'].' '.$row['field2']; //и т.д...  }    //Внимание, не работающий пример с LIKE!  $stmt = $db->prepare("SELECT field FROM table WHERE field LIKE %?%");  $stmt->bindParam(1, $search, PDO::PARAM\_STR);  $stmt->execute();  //Нужно так:  $stmt = $db->prepare("SELECT field FROM table WHERE field LIKE ?");  $stmt->bindValue(1, "%$search%", PDO::PARAM\_STR);  $stmt->execute();    //Еще пример  $stmt = $db->prepare("SELECT \* FROM table WHERE id=? AND name=?");  $stmt->bindValue(1, $id, PDO::PARAM\_INT);  $stmt->bindValue(2, $name, PDO::PARAM\_STR);  $stmt->execute();  $rows = $stmt->fetchAll(PDO::FETCH\_ASSOC);    //Еще вариант  $stmt = $db->prepare('SELECT \* from users');  $stmt -> execute();  while($row = $stmt->fetch()) {  print\_r($row);  } |

PDO UPDATE данных

Происходит по существу также как и INSERT и SELECT (в данном случае опять же будем использовать именованные метки (placeholders)):

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | $stmt = $db->prepare("UPDATE users set email = :email where lastname=:lastname");  $stmt->bindParam(':lastname', $lastname);  $stmt->bindParam(':email', $email);  $lastname = "Smith";  $email = "newmail@test.com";  $stmt->execute(); |

**DELETE** Удаление происходит простейшим образом:

|  |  |
| --- | --- |
| 1 | $db->exec("DELETE FROM users"); |

Конечно вы можете также использовать именованные параметры (placeholders) при удалении.

Если у вас есть свои часто используемые заготовки при работе с расширением PHP PDO, буду признателен если вы ими поделитесь. Ссылка на [мануал](http://www.php.net/manual/ru/book.pdo.php)

Если у вас возникла [ошибка Undefined variable: DBH](http://labdes.ru/undefined-variable-dbh)… то можете почитать как это исправить.

Google помог найти шпаргалку по PDO, может кому пригодится: