# Trabalhando com PDO no PHP

Alisson G. Chiquitto

20 de março de 2016

## 1 Introdução

PDO (PHP Data Objects) é uma extensão que fornece uma interface padronizada para trabalhar com bancos de dados, cuja finalidade é abstrair a conexão e interações com os bancos, ou seja, independente do banco de dados que estiver sendo utilizado os métodos executados serão os mesmos, mas isso não significa que seu sistema será portável entre diversos bancos de dados, por mais que o uso do PDO facilite a portabilidade, esta interface significa apenas que você se comunicará com qualquer banco de dados através de um determinado conjunto de métodos e classes.

Não é possível executar funções de interação com o banco de dados utilizando somente a extensão PDO, é preciso utilizar um driver específico do PDO para acessar um determinado banco de dados. Cada banco de dados pode prover um driver para PDO, porém nem todos os recursos são suportados em todos os bancos, por exemplo, no MySQL, tabelas do tipo MyISAM não suportam transações, impossibilitando o funcionamento dos métodos PDO::beginTransaction(), PDO::commit() e PDO::rollBack().

PDO não é uma abstração de banco de dados e não reescreve SQL.

## 2 Vantagens de usar PDO

Abstração de conexão e interação com banco de dados Segurança Suporte a diversos drivers Ainda hoje muitas pessoas consideram o uso do PDO uma opção, analisando a possibilidade de um determinado projeto mudar de banco, utilizando o PDO somente se essa possibilidade existir. O grande perigo nessa análise superficial é que nem sempre no início de um projeto temos essa visão e se tiver que mudar não adianta lamentar, então em cima disso particularmente recomendo a utilização deste recurso, principalmente se o projeto necessitar trabalhar com mais de um tipo de banco.

Confira todos os drivers suportados pelo PDO.

## 3 Manipulando conexões

Toda conexão com banco de dados é realizada ao criar uma instância da classe PDO, ou seja, independente do driver utilizado sempre iremos instanciar a classe PDO.

O construtor da classe PDO recebe as informações do banco como parâmetro obrigatório, conhecido como dsn(Data Source Name), além dos parâmetros opcionais username, password e driver_options.

```
__construct (string $dsn, string $username, string $password,
        array $driver_options )
```

No exemplo abaixo realizamos uma conexão com o banco de dados mysql.

```
<?php
$conn = new PDO(
    'mysql:host=localhost;dbname=example', 'usuario', '123456'
);
```

Após abrir uma conexão podemos interagir com o banco utilizando 3 métodos da classe PDO:

**exec**

(int) Utilizado para insert, update e delete.

**query**
(PDOStatement) Utilizado para resultados tabulares, comando select.

**prepare**
(PDOStatement) Cria um prepared statement, utilizado para dados variáveis..

Normalmente para fechar uma conexão é preciso destruir o objeto, assim como suas referências, para isso atribuímos o valor NULL a variável que contém o objeto. Se isso não for feito o PHP irá fechar automaticamente a conexão quando o script terminar, caso não seja uma conexão persistente.

# 4 Prepared statements

Os prepared statements oferecem dois ótimos benefícios:

- A query só precisa ser preparada uma vez, mas pode ser executada várias vezes;
- Os parâmetros não precisam ser escapados, pois o driver cuida disso automaticamente.

Esses benefícios significam duas coisas, agilidade e segurança, confira a criação de um prepared statement.

```php
<?php
$stmt = $conn->prepare(
    'INSERT INTO posts (title, content) VALUES (:title, :content)'
);
```

Com nosso prepared statement criado, precisamos informar os valores para compor a query, confira os métodos que podem ser utilizados:

- bindValue()

- bindParam()

```php
<?php
$stmt = $conn->prepare(
    'INSERT INTO posts (title, content) VALUES (:title, :content)'
);

$title = 'Titulo do post';
$content = 'Conteudo do post';

$stmt->bindValue(':title', $title);
$stmt->bindValue(':content', $content);
```

Após informar os dados necessários para o prepared statement, precisamos executar o método execute() para realizar a query no banco.

Para resgatar resultados de um comando select temos algumas alternativas.

**fetch**
: Retorna a próxima linha do resultado.

**fetchAll**
: Retorna um array com todos os resultados.

**fetchObject**
: Retorna a próxima linha do resultado como objeto.

**fetchColumn**
: Retorna uma coluna da próxima linha do resultado.

```php
<?php
$stmt = $conn->prepare("SELECT * FROM posts");
while($row = $stmt->fetch()) {
    print_r($row);
}
```

Um recurso interessante na hora de resgatar valores é o método bindColumn(), que tem a função de vincular o valor de uma coluna do resultado do prepared statment à uma variável.

# 5 Transações

Uma transação é um conjunto de procedimentos executados no banco de dados como uma única operação. Na prática, indicamos o início de uma transação utilizando o comando start transaction ou begin no MySQL, em seguida realizamos algumas tarefas, inserção, alteração ou remoção de registro(s), no termino desses procedimentos caso tudo ocorra bem, informamos através do comando commit que as mudanças podem ser aplicadas de fato no banco, mas caso ocorra algo de errado em algum dos procedimentos podemos utilizar o comando rollback, garantindo que todos os procedimentos realizados desde o início da transação sejam desfeitos.

A integridade de uma transação depende de quatro propriedades, conhecidas como ACID.

1. Atomicidade: Uma transação deve ser uma unidade atômica de trabalho; ou todas as suas modificações de dados são executadas ou nenhuma delas é executada.

2. Consistência: Regras de integridade dos dados são asseguradas, ou seja, as transações não podem quebrar as regras do banco de dados.

3. Isolamento: O resultado de uma transação executada concorrentemente a outra deve ser o mesmo que o de sua execução de forma isolada. Operações exteriores a uma dada transação jamais verão esta transação em estados intermediários.

4. Durabilidade: Depois que uma transação tiver sido concluída, seus efeitos ficam permanentemente no sistema.

No PDO utilizamos três métodos para trabalhar com transações, beginTransaction() para iniciar uma transação, commit() para que as tarefas realizadas sejam mantidas e rollback() para desfazer caso ocorra algum problema.

# 6 Tratamento de erros no PDO

O PDO oferece 3 alternativas para manipulação de erros.

**PDO::ERRMODE_SILENT** Esse é o tipo padrão utilizado pelo PDO, basicamente o PDO seta internamente o código de um determinado erro, podendo ser resgatado através dos métodos PDO::errorCode() e PDO::errorInfo().

**PDO::ERRMODE_WARNING** Além de armazenar o código do erro, este tipo de manipulação de erro irá enviar uma mensagem E_WARNING, sendo este muito utilizado durante a depuração e/ou teste da aplicação.

**PDO::ERRMODE_EXCEPTION** Além de armazenar o código de erro, este tipo de manipulação de erro irá lançar uma exceção PDOException, esta alternativa é recomendada, principalmente por deixar o código mais limpo e legível.

```php
<?php
$dsn = 'mysql:host=localhost;dbname=example';
$user = 'root';
$password = '123456';
try {
    $conn = new PDO($dsn, $user, $password);
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch (PDOException $e) {
    echo $e->getMessage();
}
```

# 7 Leia mais

- http://php.net/manual/en/book.pdo.php

- http://www.php.net/manual/en/pdo.drivers.php
- http://www.slideshare.net/wezfurlong/php-data-objects
- http://www.diogomatheus.com.br/blog/php/trabalhando-com-pdo-no-php/