

## SUMÁRIO

<b>1. INTRODUÇÃO CONCEITUAL .....</b>	<b>1</b>
1.1. PDO: PADRONIZAÇÃO E ADAPTABILIDADE DE SISTEMAS.....	1
1.2. SEGURANÇA.....	2
<b>2. PDO NA PRÁTICA .....</b>	<b>2</b>
2.1. INSTANCIAÇÃO DO OBJETO DA CLASSE PDO .....	2
2.2. SEMPRE INSTANCIAR COM TRY CATCH.....	3
2.3. FORMAS DE IMPLEMENTAÇÃO .....	3
2.3.1. PRIMEIRA FORMA: PDO COM POO .....	3
2.3.2. SEGUNDA FORMA: EM UM ARQUIVO SEPARADO .....	4
<b>3. EXECUTANDO INSTRUÇÕES SQL COM PHP.....</b>	<b>4</b>
3.1. UTILIZANDO PREPARE() E EXECUTE().....	5
3.1.1. MÉTODO PREPARE() .....	5
3.1.2. MÉTODO EXECUTE() .....	5
3.2. MÉTODOS BINDPARAM E BINDVALUE .....	6
3.2.1. DIFERENÇA ENTRE BINDPARAM() E BINDVALUE() .....	6
<b>4. RETORNO DA EXECUÇÃO .....</b>	<b>6</b>
4.1. RETORNO PARA INSTRUÇÕES DML E DDL.....	6
4.2. RETORNO COM INSTRUÇÃO DQL OU SELECT .....	7
4.3. FOREACH PARA PERCORRER RETORNOS .....	7
4.3.1. COMO USAR FOREACH .....	7
4.3.2. COMO FOREACH É ÚTIL EM PDO .....	8
<b>5. FONTES.....</b>	<b>9</b>

## 1. INTRODUÇÃO CONCEITUAL

Com a constante necessidade de segurança e abstração de dados surgiu a classe PDO (PHP Data Objects), uma classe, com um conjunto de objetos e métodos, desenvolvida especificamente para trabalhar com procedimentos relacionados a Banco de Dados. O interessante em utilizar este tipo de classe é a abstração de qual banco utilizamos e a segurança extra que esta classe nos oferece.

Quando falamos em “abstração de banco de dados” estamos simplesmente querendo dizer que o PDO nos oferece recurso suficiente para trabalhar implementando toda nossa aplicação sem se preocupar com qual banco estamos utilizando, isso significa que uma mudança posterior na escolha do banco não trará grandes problemas a sua aplicação.

### 1.1. PDO: PADRONIZAÇÃO E ADAPTABILIDADE DE SISTEMAS

A finalidade-mor da criação desta forma de implementação, de comunicação de bancos de dados com PHP, chamada de PDO, é a padronização da tecnologia.

**As funcionalidades da classe do PDO são agregadas por meio do formato de extensão**, podendo ser, a depender da necessidade, desabilitadas ou habilitadas, a partir do arquivo **php.ini**. Por padrão, a exemplo, desde a versão 5 do PHP a extensão PDO\_MySQL é habilitada para uso, ao mesmo tempo que existem outras extensões, para outros tipos de bancos, também habilitáveis.



**O PDO surgiu justamente para resolver as problemáticas referentes aos mais variados SGBD's**, e para que, nas situações em que fossem necessárias mudanças de uns para outros, o processo ocorresse sem a obrigação de haver uma total reformulação do código, mas sim, e apenas, um cumprimento de regras definidas.

Neste sentido, o PDO se apresenta como uma interface que define regras que devem ser seguidas de modo que, se alternado o SGBD, basta modificar o driver de comunicação utilizado. Estes drivers citados, como o pdo\_mysql, seriam as extensões supracitadas, e demonstradas na imagem anterior como sendo o elemento necessário para comunicação efetiva com cada SGBD.

## 1.2. SEGURANÇA

Como qualquer aplicação, não basta sua implementação funcional, mas também são necessárias implementações para atender a problemas possíveis e não imediatamente necessários para a operação do sistema, como questões referentes à segurança – um requisito de software indispensável.

Existem diversos problemas recorrentes, medidas e padrões de projeto conhecidos para fornecer a devida segurança ao sistema, sendo impossível citar todos os problemas, e todas as medidas. Alguns exemplos poderíamos citar, como o ataque de SQL Injection, e a medida de construção de uma aplicação em multilayers, para isolar partes da aplicação, de outras.

## 2. PDO NA PRÁTICA

### 2.1. INSTANCIÇÃO DO OBJETO DA CLASSE PDO

O PDO, na prática, é uma classe, e como uma, quando instanciada, permite o acesso a seus próprios métodos a partir de seu objeto.

Para instanciar um objeto, da classe PDO, é necessário, além da própria instância, a passagem de parâmetros referentes ao acesso ao banco de dados. Sem as definições configuradas, o PDO exige o seguinte parâmetro:

**"mysql:host=localhost;dbname=dbnome", "root", "senha"**

Estes parâmetros são variáveis, e devem ser passados nesta ordem: **\$db\_nome**, **\$host**, **\$usuario** e **\$senha**.

**\$db\_nome:** É o nome do banco de dados a que iremos nos conectar.

**\$host:** Por padrão é 'localhost'. As aspas são necessárias.

**\$usuario:** Por padrão, em usos informais, é igual a 'root'. As aspas são necessárias.

**\$senha:** Por padrão, também em usos informais, não se define uma, sendo portanto igual a '' . As aspas são necessárias.

Assim sendo, para instanciar o objeto da classe PDO, que será utilizado para acessos as funcionalidades da classe, e após criadas e definidas as variáveis:

```
$objeto = new PDO($db_nome, $host, $usuario, $senha);
```

Em outros casos, ao invés de \$db\_nome e \$host, cria-se a variável \$dsn, que é igual a “mysql:host=localhost;dbname=dbnome”. Mas, claro, neste caso o banco acessado não é variável, sendo definido no próprio código o nome do banco acessado.

## 2.2. SEMPRE INSTANCIAR COM TRY CATCH

Por padrão, utiliza-se a funcionalidade Try Catch para **tratamentos de erros** no processo de conexão e instanciação do objeto PDO.

```
try{
    $objeto = new PDO("mysql:host={$host};dbname={$db_nome}", "{$user}", "{$pass}");
}
catch(PDOException $e){
    echo 'Erro com o banco de dados:' . $e->getMessage();
}
catch(Exception $e){
    echo 'Erro:' . $e->getMessage();
}
```

## 2.3. FORMAS DE IMPLEMENTAÇÃO

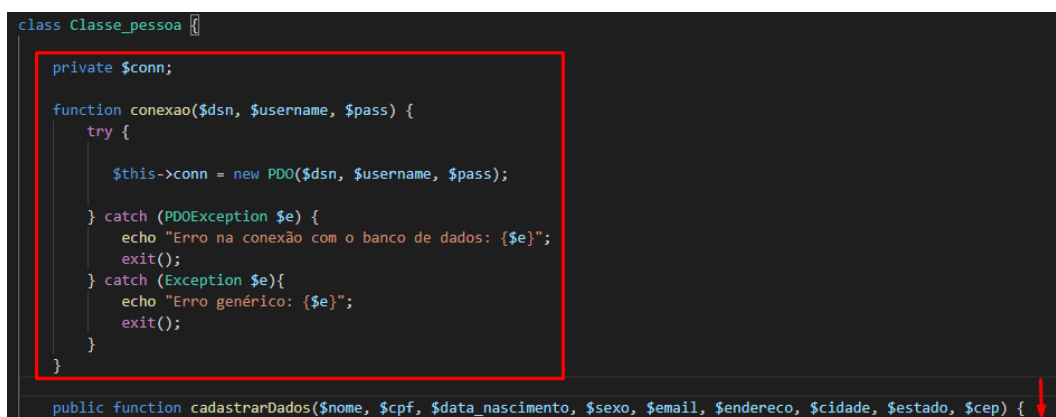
Em termos de “onde implementar”, existem duas formas principais para realizar um código de conexão com o banco de dados. **Ou (i) a conexão pode ser estabelecida como uma classe de conexão criada**, cujo método seria o estabelecimento da conexão a cada vez que chamado, **ou (ii) dentro de um arquivo único** que seria utilizado sempre que necessário por meio de um require\_once.

### 2.3.1. PRIMEIRA FORMA: PDO COM POO

Desta forma, o objeto instanciado com PDO é um atributo de uma classe. Esta classe pode ser, (i) uma classe apenas para conexão, que será utilizada em outras classes, que necessitem de uma conexão com o banco, a partir de herança, ou (ii) pode ser uma classe voltada para uma tarefa específica no banco, mas que irá conter, como atributo e função construtora, também o objeto PDO e a sua instanciação.

**Para a primeira situação, que não é muito utilizada, um breve exemplo** seria uma classe chamada de Cliente, em que herdaria a classe de Conexão, para poder se comunicar com a tabela de Clientes do banco de dados. Assim, essa Classe Clientes teria métodos Getters e Setters, enquanto a classe Conexão apenas, talvez, a função construtora.

**Para a segunda situação, mais comum, e utilizada,** um exemplo seria uma aplicação em que para realizar operações em determinadas tabelas, existiram classes com métodos para estes fins. Todas as classes para operação com tabelas teriam o atributo que seria também o objeto PDO, e a função construtora seria, por padrão, a instanciação do mesmo.



```

class Classe_pessoa {
    private $conn;

    function conexao($dsn, $username, $pass) {
        try {
            $this->conn = new PDO($dsn, $username, $pass);
        } catch (PDOException $e) {
            echo "Erro na conexão com o banco de dados: {$e}";
            exit();
        } catch (Exception $e){
            echo "Erro genérico: {$e}";
            exit();
        }
    }

    public function cadastrarDados($nome, $cpf, $data_nascimento, $sexo, $email, $endereco, $cidade, $estado, $cep) {

```

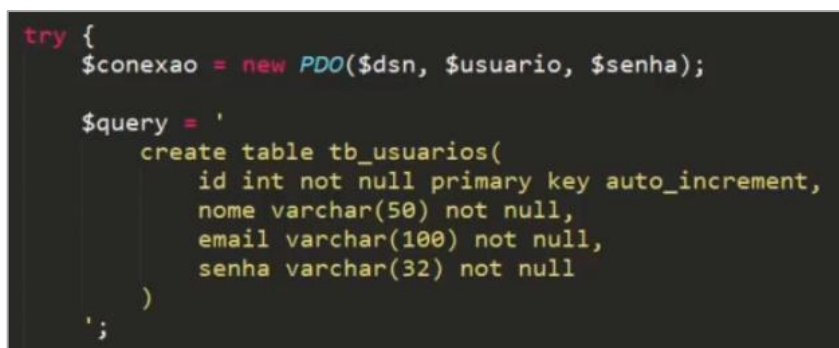
No exemplo da imagem acima, a Classe Pessoa contém diversos métodos – não mostrados- para operação na tabela “Pessoas”, sendo sua função construtora a instanciação do Objeto PDO por meio do atributo \$conn.

### 2.3.2. SEGUNDA FORMA: EM UM ARQUIVO SEPARADO

Uma outra forma muito comum, se não a mais utilizada, é implementar a instanciação do Objeto PDO em um arquivo separado, chamado, por exemplo, de “conexao.php”, e que será utilizado, quando necessário, em outras partes da aplicação, por meio da clausula “require” e/ou “require\_once”.

## 3. EXECUTANDO INSTRUÇÕES SQL COM PHP

Antes de discorrer sobre as formas de executar instruções SQL, deve-se ter em mente que cada uma das formas são métodos cujo parâmetro é a própria instrução SQL. Assim sendo, naturalmente se define a instrução atribuindo- a, em forma de texto, a uma variável criada, como, por exemplo, \$query. Veja um exemplo abaixo:



```

try {
    $conexao = new PDO($dsn, $usuario, $senha);

    $query = '
        create table tb_usuarios(
            id int not null primary key auto_increment,
            nome varchar(50) not null,
            email varchar(100) not null,
            senha varchar(32) not null
        )
    ';

```

Por fim, existem três métodos disponibilizados pela classe PDO para executar queries SQL, isto é, instruções em linguagem SQL, atribuídas a uma variável, em PHP, no banco de dados. Seriam os métodos **exec()**, **query()** e, **o mais comum, e sempre recomendado, prepare()/execute()**.

A diferença entres estes métodos é sutil, embora seja recomendado, por via de regra, o uso de prepare()/execute(). Os dois outros métodos de execução de queries, exec() e query(), seriam alternativas mais eficientes, e menos seguras, para determinadas situações.

### 3.1. UTILIZANDO PREPARE() E EXECUTE()

#### 3.1.1. MÉTODO PREPARE()

Após criada a variável **\$sql**, e atribuída a instrução devida, utiliza-se o método **prepare(\$sql)** para preparar a instrução para execução, retornando um objeto de instrução preparada, da **classe PDOStatement** normalmente chamado de **Statment Object**.

Este novo objeto, normalmente definido como **\$stmt**, naturalmente, terá acesso aos métodos da classe PDOStatement, que são imprescindíveis para a continuação da aplicação. Para que efetivamente a variável \$stmt seja uma instanciación dessa classe, deve-se iguala-la ao chamado do método prepare. Veja abaixo:

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

A variável **\$stmt** recebe o retorno do método da classe PDO, chamada a partir de seu objeto **\$conn**. Este último, lembrando, é a própria instância da classe PDO, criado a partir de `$conn = new PDO(...)`.

#### 3.1.2. MÉTODO EXECUTE()

Facilmente compreensível, o método execute irá executar a instrução SQL, definida em \$sql, e preparada com prepare(\$sql).

```
// $conn = new PDO('mysql:host=localhost;dbname=meuBancoDeDados', $username, $password);
$conn = new PDO('mysql:host=localhost;dbname=dbphp7', $username, $pass);

//PDO :: prepare - Prepara uma instrução SQL para execução no banco, e retorna um objeto de instrução
$stmt = $conn->prepare("SELECT * FROM tb_usuarios ORDER BY des_login");

//Executa a instrução preparada
$stmt->execute();
```

No entanto, e é este o assunto do próximo tópico, a execução do statment (instrução) não se dá, necessariamente, imediatamente após a sua preparação, pois podem existir, e existem, situações em que procedimentos entre ambos são necessários.

### 3.2. MÉTODOS BINDPARAM E BINDVALUE

Ao invés de preparar uma instrução em que os parâmetros são todos definidos, podemos, criar espaços reservados para valores que podem variar. Por exemplo, ao invés de escrever `$sql = "SELECT id FROM pessoas"`, podemos escrever `"SELECT :selecionado FROM :local"`.

Esta instrução, contida na variável `$sql`, e preparada com o método `prepare()` necessita, antes de ser executada, que seus espaços reservados sejam definidos, e neste contexto existem dois métodos para realizar este procedimento, que são `bindParam()` e `bindValue()`.

```
$username = 'root';
$password = '';
$dns = 'mysql:host=localhost;dbname=dbphp7';

$conn = new PDO($dns, $username, $password);

//Poderíamos inserir os valores já, mas vamos usar identificadores
$stmt = $conn->prepare("INSERT INTO tb_usuarios (des_login, des_senha) VALUES(:LOGIN, :PASSWORD)");

$login = "Mario Rogers";
$pass = "cogumelo1010";

$stmt->bindParam(":LOGIN", $login);
$stmt->bindParam(":PASSWORD", $pass);

$stmt->execute();
```

No exemplo da imagem acima, realizado com `bindParam()`, o primeiro parâmetro é igual ao identificador variável, e o segundo é igual ao valor que será considerado para o próprio após a execução da instrução com `execute()`.

#### 3.2.1. DIFERENÇA ENTRE BINDPARAM() E BINDVALUE()

A diferença entre os dois, é que no `bindParam()` o argumento esperado é uma referência (variável ou constante) e não pode ser um tipo primitivo como uma string ou número solto, retorno de função/método. Já `bindValue()` pode receber referências e valores como argumento.

## 4. RETORNO DA EXECUÇÃO

### 4.1. RETORNO PARA INSTRUÇÕES DML E DDL

O retorno da execução, e o prosseguimento da aplicação, deve variar de acordo com a instrução SQL utilizada, isto é, se era de modificação, criação, update, delete, etc. Na maioria dos casos, em que não há uma estrutura esperada, como retorno, que deverá ser trabalhada posteriormente, como no caso do `SELECT`, o retorno será simplesmente `true`, ou `false`, para, respectivamente, uma execução bem sucedida ou falha.

## 4.2. RETORNO COM INSTRUÇÃO DQL OU SELECT

Em instruções de pesquisas e buscas, com SELECT, em que a intenção é trabalhar com valores retornados do banco, existem dois métodos principais, que seriam o **Fetch()** e o **Fetchall()**, pertencente à classe PDOStatement.

**Fetch()**: Retorna uma única row da consulta, ideal para poder utilizar em consultas como login, que retorna somente um resultado.

**FetchAll()**: Retorna um array com todas as linhas da consulta, ideal para uma busca por nome ou por endereço.

Veja abaixo um exemplo, em que é utilizado FetchAll() devido ao fato de que poderá ser retornado mais de um registro com a instrução SQL:

```
$username = "root";
$pass = '';

$conn = new PDO('mysql:host=localhost;dbname=dbphp7', $username, $pass);

$stmt = $conn->prepare("SELECT * FROM tb_usuarios ORDER BY des_login");

$stmt->execute();

$results = $stmt->fetchAll(PDO::FETCH_ASSOC);
```

## 4.3. FOREACH PARA PERCORRER RETORNOS

### 4.3.1. COMO USAR FOREACH

Existem vários modos para percorrer arrays, no entanto, o mais simples deles é utilizando o laço de repetição foreach em PHP. Este comando funciona só com arrays e objetos, e retorna um erro quando utilizado com outros tipos de expressões. Sua sintaxe pode ser dada de duas formas, que são:

**foreach(\$array as \$value)**

**foreach(\$array as \$key => \$value)**

No primeiro exemplo de foreach, acima, o primeiro parâmetro, **\$array**, é o próprio array percorrido, e **\$value** é a variável que irá receber o elemento lido, a cada laço.

No segundo exemplo de foreach, a lógica se mantém, mas existindo, também uma variável que recebe o índice lido, que chamamos de **\$key**.



#### 4.3.2. COMO FOREACH É ÚTIL EM PDO

Este tipo de estrutura para percorrer laços é útil quando em casos de execução de instruções DQL, como de SELECT, em que estruturas serão retornadas e precisam ser percorridas.

Imagine um caso em que são percorridos registros de uma tabela, e retornados diversos deles. Neste caso, a estrutura seria um array de diversas linhas e colunas, ou seja, uma matriz, obtida por um FetchAll(), em que cada linha seria um registro, e cada coluna um atributo ou campo. Assim, poderíamos criar um for, para percorrer as linhas da matriz, e dentro deste um foreach para percorrer cada registro, individualmente.

O caso citado acima é bastante comum quando criadas tabelas dinâmicas com HTML e PHP, em que dentro do for, que percorre as linhas do array, é aberta a linha com a tag table row (tr), e no foreach o \$valor é inserido dentro de uma tag table data (td), seguido, após seu fechamento, do encerramento da table row.

```
for($linha = 0; $linha < $num_registros; $linha ++ ){  
    //Nova linha da tabela HTML  
    echo "<tr>";  
    //No foreach percorremos as colunas  
    foreach($array_registros[$linha] as $coluna => $valor){  
        //O valor da coluna da linha da matriz é lida, (2) um td é criado, e (3) seu valor é o da coluna lida  
        echo "<td scope='row'>$valor</td>";  
    }  
    //Fechamento da linha da tabela HTML  
    echo "</tr>";  
}
```

## 5. FONTES

[https://www.php.net/manual/pt\\_BR/book.pdo.php](https://www.php.net/manual/pt_BR/book.pdo.php)

[https://www.php.net/manual/pt\\_BR/pdo.construct.php](https://www.php.net/manual/pt_BR/pdo.construct.php)

[https://www.php.net/manual/pt\\_BR/class.pdoexception.php](https://www.php.net/manual/pt_BR/class.pdoexception.php)

<https://www.devmedia.com.br/evitando-sql-injection-em-aplicacoes-php/27804>

[https://www.php.net/manual/pt\\_BR/class.pdostatement.php](https://www.php.net/manual/pt_BR/class.pdostatement.php)

[https://www.php.net/manual/pt\\_BR/pdostatement.fetch.php](https://www.php.net/manual/pt_BR/pdostatement.fetch.php)

