SUMÁRIO

1. O QUE É PDO: PHP DATA OBJECT

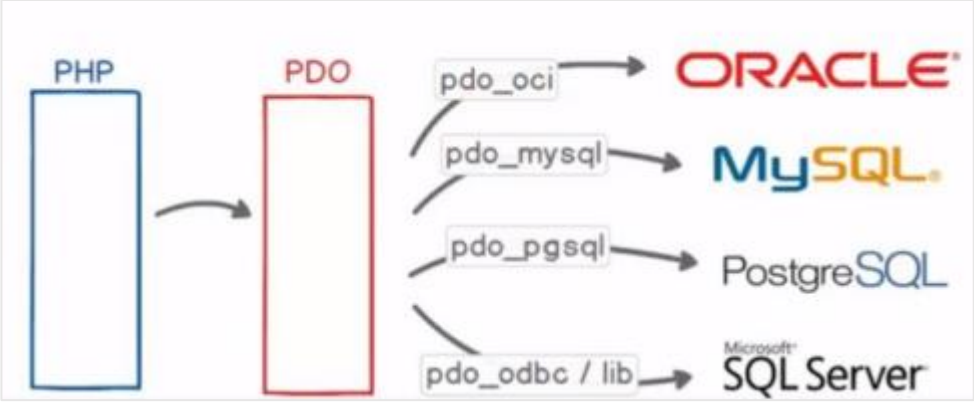
Com a constante necessidade de segurança e abstração de dados surgiu a classe PDO, que com um conjunto de objetos e métodos foi desenvolvida especificamente para trabalhar com procedimentos relacionados a banco de dados.

O ganho em utilizá-la, se em comparação com outras formas de realizar a mesma coisa, isto é, procedimentos com banco de dados a partir do PHP, é a abstração de qual banco utilizamos e a segurança extra oferecida.

1.1. PADRONIZAÇÃO E ADAPTABILIDADE DE SISTEMAS

O PDO serve como uma forma padronizada de trabalhar com banco de dados utilizando PHP, e por isso se pode dizer que "abstrai" as especificidades do banco de dados utilizado. Com essa classe, você pode utilizar MySQL, PostgreSQL, Oracle ou qualquer outro banco relacional sem ter que realizar alterações significativas no fazer do seu sistema.

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, a extensão PDO_MySQL é habilitada para uso, ao mesmo tempo que existem outras extensões, para outros tipos de bancos, também habilitáveis.



Desta forma, o PDO se apresenta como um mecanismo adaptável; se alterado o gerenciador de banco de dados, basta modificar o driver de comunicação utilizado.

2. IMPLEMENTANDO O PDO

Esse tópico é sobre como aplicar a classe PDO. Os subtópicos são os passos da implementação, ou seja, estão dispostos em uma sequência que condiz com a que é necessária na prática.

Além de ser uma "receita" do PDO, estão inclusas aqui as melhores práticas, como uso de transações e de Exceptions, e que normalmente um iniciante não saberia que existem, ou que em algum momento seriam necessárias.

Para mais informações, veja aqui a documentação completa do PDO.

2.1. INSTÂNCIA DA CLASSE: CONEXÃO COM O BANCO DE DADOS

A instância da classe PDO, além da criação de um objeto da classe para o acesso as suas funcionalidades, também configura a própria conexão com o banco de dados. A sintaxe é essa:

$conn = new PDO("[sgbd]:host=[host];dbname=[nome_banco]", "[login]", "[senha]");

O que está entre colchetes são dados variáveis, e os próprios colchetes foram utilizados apenas para indicar isso – não devem existir. **Os dados variáveis são**: "sgbd", é o sistema gerenciador de banco de dados utilizado; "host", que é o nome do host; "nome_banco" que é o nome do banco de dados; "login" que é o login, ou username utilizado no acesso ao gerenciador de banco de dados; e "senha" que é a senha utilizada junto do login do gerenciador. Veja o exemplo abaixo:

$conn = new PDO("mysql:host=localhost;dbname=company", "root", "root");

2.1.1. HABILITANDO AS EXCEPTIONS DO PDO

Por padrão, o modo de tratamento de erro do PDO é "**PDO::ERRMODE_SILENT**", o que significa que quando ocorre um erro de SQL utilizando PDO, nenhum erro ou aviso é emitido e nenhuma exceção lançada.

É possível alterar esse estado padrão, alterando o modo de tratamento de erros do PDO para "**PDO::ERRMODE_EXCEPTION**". Com essa configuração, em caso de erro passará a ser lançada a exceção de nome "**PDOException**".

Para alterar o modo padrão do PDO para tratar erros o que deve ser feito é alterar o atributo "**PDO::ATTR_ERRMODE**". Para alterar atributos do PDO existe o **método setAttribute**, cuja sintaxe é:

setAttribute([atributo a ser alterado], [novo modo do atributo])

Pois bem: no arquivo que houver a instância do objeto PDO, utilizando o objeto PDO, faça:

```
$conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION)
```

Após isso, os métodos do PDO passarão a lançar a Exception "PDOException" em caso de erro, e como com toda Exception, existem métodos para acessar detalhes do erro. Clique aqui para ver os tratamentos dessa Exception.

2.2. MÉTODO BEGINTRANSACTION: USO DE TRANSAÇÕES

O método beginTransaction serve para que possamos utilizar as conhecidas transações do SQL com o PDO.

Antes do método “prepare”, que prepara a query, e que normalmente é o primeiro passo, deve ser chamado esse método, “beginTransaction”, acessível a partir do objeto PDO. Ele inicializa uma transação, e que depois deve ser confirmada com o método “Commit” ou cancelada com o método “rollBack”.

```
$conn->beginTransaction()
```

Se a transação não for confirmada com o método “Commit”, a própria função de execução da query, “execute”, embora escrita antes, será cancelada.

2.3. MÉTODO PREPARE: PREPARAÇÃO DA QUERY

Agora, após a inicialização da transação, deve ser executado o método “prepare” que utiliza prepared statements, ou “instruções preparadas”, que, além de tornarem alguns processos muito mais rápidos, evitam ataques de injeção de SQL.

Esse método recebe uma query, e retorna um objeto PDOStatment, que é uma instância da classe PDOStatement.

```
$statment = $conn->prepare($query)
```

Além disso, e para que o tópico seguinte faça sentido, as queries preparadas pelo método “prepare”, se possuem **especificações**, como valores de um INSERT, ou parâmetros de um SELECT, elas são escritas como **marcas nomeadas**, cuja sintaxe é “:nome”, e que depois são vinculadas aos seus respectivos valores.

```
“SELECT `id`, `nome`, `sexo` FROM Pessoas WHERE `sexo` = :sexo”
```

2.4. MÉTODOS BINDPARAM E BINDVALUE

Esses métodos, da classe PDOStatment, são utilizados, então, para vincular as marcas mapeadas pelo “prepare” a valores propriamente ditos. Eles fazem exatamente a mesma coisa, mas de forma diferente.

O primeiro, “bindParam” vincula uma marca ao valor de uma variável, porque a vinculação se dá por referência a um endereço.

```
$statment->bindParam(“:sexo”, $sexo)
```

O segundo, “bindValue” vincula uma marca a um valor em si, porque a vinculação se dá por uma cópia do valor.

```
$statment->bindParam(“:sexo”, “Masculino”)
```

2.5. MÉTODO EXECUTE: EXECUTANDO A QUERY

O último passo, dentre os principais, é o “execute”, que executa a query preparada e cujas marcas, se existiam, foram valoradas. Ele é um método da classe PDOStatment.

```
$statment->execute()
```

Ele, como os outros, será executado com sucesso, ou poderá falhar e retornar uma Exception.

2.5.1. TRATAMENTO DA EXECUÇÃO DE UM SELECT: ROWCOUNT, FETCH E FETCHALL

Quando a query executada é uma que traz dados do banco, como um SELECT, que de fato é o melhor exemplo, uma validação a mais é necessária, e ainda existem métodos para tratar os dados.

Primeiro, sobre a validação, seria o método “rowCount” da classe PDOStatment. O que acontece é que o método “execute”, por si só não traz dados, mas apenas encontra os registros ou linhas que satisfazem a exigência da query. Esse método, “rowCount”, verifica a quantidade de linhas “afetadas” pela execução da query. No caso de um SELECT, faz sentido haver uma validação adicional para a quantidade de linhas afetadas, porque pode ser que nenhuma seja, e isso poderia ser tratado como um erro.

```
If($statment->rowCount() > 0){ }
```

Agora sobre o tratamento dos dados, que seria o caso em que o "rowCount" é maior do que zero, existem dois métodos comumente utilizados: o **Fetch e o FetchAll**, ambos da classe PDOStatment. Eles fazem a mesma coisa, que é **capturar os registros afetados, e organizá-los em uma estrutura de dados**.

A estrutura de dados criada, por padrão, é indexada. Cada índice é uma linha, e o elemento do índice é um outro array indexado, que representa o registro em si, em que cada índice é uma coluna do registro, e o elemento o valor da coluna.

Mas ambos os métodos aceitam um parâmetro de modo, que dita sobre "como queremos que os dados sejam organizados". Normalmente o modo passado é o "**PDO::FETCH_ASSOC**", que torna essa estrutura de dados um array associativo.

Embora até aqui pareça que são dois métodos idênticos, existe uma diferença entre eles, e ela é muito importante. **O Fetch só pode recuperar um registro**, e por isso é normalmente utilizado na programação de operações de login. **Já o FetchAll pode recuperar um ou vários registros**, o que faz dele útil justamente para as operações em que os dados de vários registros são necessários.

Certo: então devemos utilizar FetchAll para operações que requerem um número indeterminado de registros, e Fetch para as operações que requerem apenas um registro? Depende. **Existe um fator a se considerar** e que normalmente não é considerado, ou até mesmo conhecido, embora seja dedutível, que é: **o consumo de memória do FetchAll**.

O FetchAll captura todos os registros afetados de uma vez só. Essa é uma ação que obviamente requer um uso de memória bastante considerável quanto maior for a quantidade de registros. **Para contornar isso a técnica necessária é essa**: utilizar um Fetch dentro de uma estrutura de repetição de "rowCount" loops, para capturar todos os registros, um por um.

3. A LÓGICA GENÉRICA

Essa é a lógica genérica das descrições do passo a passo realizadas no tópico anterior. Como se dará a implementação, se em arquivos diferentes, classes diferentes, com mais condicionais para cada processo, dependerá dos requisitos do sistema.

```
try{
        $conn = new PDO(...);
        $conn->beginTransaction();
        $statment = $conn->prepare($query);
        // Situacional: BindParam() ou bindValue()
        $statment->execute();
        // Situacional: if($statment->rowCount > 0) { Fetch/FetchAll, Commit  }
        $conn->Commit();
}catch(PDOException $e){
        echo $e->getMessage();
        $conn->rollBack();
}
```