O **PDO (PHP Data Objects)** define uma interface de conexão a banco de dados leve e consistente para PHP. Há a possibilidade de utilização de diversos drivers de conexão que implementam a interface do PDO para vários tipos de bancos de dados.

Como o PDO representa uma camada de abstração de acesso aos dados, as mesmas funções utilizadas para manipular dados ou recuperar informações do banco serão as mesmas, independentemente do banco de dados que esteja sendo usado.

Existem três API's de conexão com o banco de dados em PHP, são elas:

- mysql: Pacote de funções para acesso ao MySQL, foi descontinuado no PHP7.
- mysqli: Extensão da API mysql com suporte a funcionalidades adicionadas a versões posteriores ao MySQL 4.1 - http://www.mysql.com/
- PDO - PHP Data Objects: Interface para acesso a dados do PHP.

O PDO é uma API robusta que pode ser usada, independentemente do driver que você estiver usando.

Muitas pessoas ficam um pouco intimidadas em usar o PDO de início, não por se tratar de uma API difícil de usar, muito pelo contrário, pois é muito fácil de usar, mas sim porque a API mysql é muito fácil de usar e as pessoas acabam se acostumando com ela.

Conexão ao banco de dados

Primeiro vamos ver como se conectar usando o PDO. Nós vamos criar uma nova instância de classe e especificar o driver que vamos usar, no caso o mysql, o nome do banco de dados, nome de usuário e senha.

```php
<?php
$conn = new PDO('mysql:host=localhost;dbname=meuBancoDeDados', $username, $password);
?>
```

Listagem 2. Criando conexão com PDO

Como vimos na **Listagem 2**, também é muito simples se conectar usando o PDO, mas como em toda conexão, é preciso tratar os erros, para se caso aconteça algum erro na conexão, mostrar isso ao usuário.

Para isso vamos usar o try..catch, usado nas linguagens orientada a objetos.

```php
<?php
try {
  $conn = new PDO('mysql:host=localhost;dbname=meuBancoDeDados', $username, $password);
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch(PDOException $e) {
    echo 'ERROR: ' . $e->getMessage();
}
?>
```

Listagem 3. Tratando erros na conexão

O erro padrão do PDO é o PDO::ERRMODE_SILENT, mas no nosso código usamos o PDO::ERRMODE_EXCEPTION e abaixo vou listar as opções que temos:

- PDO::ERRMODE_SILENT
- PDO::ERRMODE_WARNING
- PDO::ERRMODE_EXCEPTION

Fetch

Agora vamos fazer começar a utilizar o select com PDO, buscando determinados resultados. Existem duas formas básicas de se fazer isso: Consultando e Executando e são essas duas que iremos analisar mais pra frente.

```php
<?php
/*
 * Método de conexão sem padrões
 */

$name = 'Ricardo';

try {
    $conn = new PDO('mysql:host=localhost;dbname=meuBancoDeDados', $username, $password);
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $data = $conn->query('SELECT * FROM minhaTabela WHERE nome = ' . $conn->quote($name));

    foreach($data as $row) {
        print_r($row);
    }
} catch(PDOException $e) {
    echo 'ERROR: ' . $e->getMessage();
}

 ?>
```

Listagem 4. Conectando com banco de dados usando PDO

Embora isso funcione, observe que ainda estamos escapando manualmente os dados do usuário com o método PDO::quote. Pense nisso como método, mais ou menos, o equivalente ao mysql_real_escape_string; no PDO.

Em situações, quando você está vinculado aos dados fornecidos pelo usuário para uma consulta SQL, é fortemente aconselhável que você, em vez usar isso, use prepared statements.

Dito isto, se suas consultas SQL não são dependentes de dados do formulário, o método de consulta é uma escolha útil, e torna o processo de loop através dos resultados tão fácil como uma instrução foreach.

```php
<?php
/*
 * Melhor prática usando Prepared Statements
 *
 */

$id = 5;
try {
    $conn = new PDO('mysql:host=localhost;dbname=meuBancoDeDados', $username, $password);
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $stmt = $conn->prepare('SELECT * FROM minhaTabela WHERE id = :id');
    $stmt->execute(array('id' => $id));

    while($row = $stmt->fetch()) {
        print_r($row);
    }
} catch(PDOException $e) {
    echo 'ERROR: ' . $e->getMessage();
}

 ?>
```

Listagem 5. Usando Prepared Statements

Neste exemplo, estamos usando o método de preparo para, literalmente, preparar a consulta, antes de os dados do usuário forem anexados. Com esta técnica, o SQL Injection é praticamente impossível, porque os dados não nunca são inseridos na consulta SQL em si.

Observe que, em vez disso, usamos parâmetros nomeados (:id) para especificar espaços reservados.

Nós executamos a consulta, ao passar uma matriz, que contém os dados que devem ser vinculados a esses espaços reservados.

```php
$stmt->execute(array('id' => $id));
```

Um suplente, mas perfeitamente aceitável abordagem, seria usar o método bindParam, assim:

Uma forma alternativa mas que é perfeitamente aceitável e que pode ser usada sem medo por quem quiser usar, é usar o método bindParam, ficando dessa forma:

```
$stmt->bindParam(':id', $id, PDO::PARAM_INT);
$stmt->execute();
```

Listagem 6. Usando o método bindParam

Depois de chamar o método execute, existem diferentes maneiras de receber os dados: uma matriz (o padrão), um objeto, etc. No exemplo acima, a resposta padrão é: PDO:: FETCH_ASSOC, o que pode ser facilmente substituída caso seja necessário.

```
while($row = $stmt->fetch(PDO::FETCH_OBJ)) {
   print_r($row);
}
```

Listagem 7. Especificando a interação

No código acima, especificamos que nós queremos interagir com o result set da melhor maneira orientada a objetos. Abaixo irei listar algumas das opções disponíveis para essa interação.

- **PDO :: FETCH_ASSOC**: Retorna uma matriz.
- **PDO :: FETCH_BOTH**: Retorna uma matriz, indexada pelo nome da coluna e 0-indexados.
- **PDO :: FETCH_BOUND**: Retorna TRUE e atribui os valores das colunas no seu conjunto de resultados para as variáveis PHP que estavam amarradas.
- **PDO :: FETCH_CLASS**: Retorna uma nova instância da classe especificada.
- **PDO :: FETCH_OBJ**: Retorna um objeto anônimo, com nomes de propriedades que correspondem às colunas.

Mas existe um problema ainda com o que fizemos até agora no código, nós não estamos dando feedback nenhum para o usuário se, por exemplo, quando nenhum resultado for retornado.

No código abaixo iremos consertar essa falta de feedback e dar um retorno ao usuário.

```php
<?php

$stmt->execute(array('id' => $id));

# Pega um array contendo todos os resultados
$result = $stmt->fetchAll();

# Se um ou mais resultados forem retornados...
if ( count($result) ) {
    foreach($result as $row) {
        print_r($row);
    }
} else {
    echo "Nenhum resultado retornado.";
}

 ?>
```

Listagem 8. Retornando feedback ao usuário

Dessa forma, nosso código completo pode ser visto na **Listagem 9**.

```php
<?php


$id = 5;
try {
  $conn = new PDO('mysql:host=localhost;dbname=meuBancoDeDados', $username,
$password);
  $stmt = $conn->prepare('SELECT * FROM minhaTabela WHERE id = :id');
  $stmt->execute(array('id' => $id));


  $result = $stmt->fetchAll();


  if ( count($result) ) {
    foreach($result as $row) {
      print_r($row);
    }
  } else {
    echo "Nennhum resultado retornado.";
  }
} catch(PDOException $e) {
    echo 'ERROR: ' . $e->getMessage();
}


 ?>
```

Listagem 9. Código completo do exemplo