

Alexandre Bezerra Barbosa

Compartilhando conhecimento de TI

[Home](#) [Windows](#) [Suporte técnico](#) [Linux](#) [PHP](#) [Sobre](#)

PHP:: PDO – CRUD COMPLETO

Alexandre Bezerra Barbosa / 4 de setembro de 2016



CRUD COMPLETO COM PDO.

Nível iniciante.

Este artigo descreve detalhadamente e de forma simples em 7 etapas, como você pode criar uma aplicação CRUD. Vamos criar uma agenda de contatos, ou cadastro de contatos como preferir! Serão vistas as quatro operações do banco de dados em ação, utilizando a classe PDO. Além disso, a fim de que você se sinta mais a vontade com o foco do artigo, não vou entrar em detalhe técnico sobre orientação a objetos aqui, antes, meu objetivo é fazer você compreender como criar rapidamente o CRUD sem nenhum auxílio de qualquer Framework. Após concluir as 7 etapas, você terá uma visão bem abrangente da classe PDO e como utilizá-la em seus códigos.

Introdução

Em outros artigos sobre persistência, já abrangei sobre CRUD utilizando inclusive alguns padrões de projetos mas não eram aplicações completas. Entretanto, imagino que para quem está iniciando a programar em PHP e ainda não está acostumado a padrões de projetos, poderá estar com inúmeras dúvidas e precisa de algo simples, porém funcional. Por isso, decidi, criar este artigo que abrange de forma simples o CRUD em PHP fazendo uso do banco de dados MySQL. CRUD é um acrônimo para as quatro operações do Banco: CREATE (Criar), READ (Ler), Update (Atualizar) e DELETE (Excluir). Quando você cria uma aplicação que conte com todas estas operações, está desenvolvendo um CRUD.

Vamos então passar a passo, etapa por etapa entendendo como fazer uma aplicação CRUD (agenda de contatos) desde a conexão com banco de dados. Queri deixar claro que é importante se acostumar a programar conforme o paradigma Orientado a Objetos. Você até pode programar conforme o paradigma procedural, mas estará limitado em alguns aspectos e até irá se deparar com outros problemas que somente terão solução pela Orientação a Objetos. Não entraremos em detalhes sobre isso aqui. Além disso, é importante que você tenha noções da linguagem SQL a fim de ter um melhor aproveitamento.

Primeira etapa: Criando o banco de dados

Precisamos de um banco de dados e elegemos o MySQL como SGBD para gerenciá-lo.

Vamos criar uma agenda de contatos, com três colunas nome, e-mail e telefone.

1- Para criar o banco de dados execute o código:

```
1 | CREATE DATABASE crudsimples DEFAULT CHARACTER SET utf8 DEFAULT COLLATE utf8_general_ci;
```

2- Em seguida selecione o banco:

```
1 | USE crudsimples;
```

3- Então crie a tabela contatos:

```
1 | CREATE TABLE contatos (
2 |     id INT NOT NULL AUTO_INCREMENT,
3 |     nome VARCHAR(45) NOT NULL,
4 |     email VARCHAR(45) NOT NULL,
5 |     celular VARCHAR(15) DEFAULT NULL,
6 |     PRIMARY KEY(id)
7 | );
```

4- Para verificar se está tudo certo:

```
1 | SELECT * FROM contatos;
```

Se você seguir a risca estas etapas acima, não deverá enfrentar nenhum problema.

Segunda etapa: Conexão com banco de dados

Não é novidade que atualmente se adotou a classe PDO, uma interface de conexão, que permite criar a abstração com o sistema de banco de dados. Sendo assim, vamos criando nosso objeto de conexão PDO.

1- Nesta etapa, vamos iniciar o projeto por criar um novo arquivo com extensão .php e, obviamente aqui o nome do arquivo é livre, ou seja, você poderá dar o nome de sua preferência 😊. Além disso, é claro, utilize a IDE de sua preferência ou até um editor de textos se assim preferir. Então como já foi dito, inicie com um arquivo limpo, e digitando apenas as tags de abertura e encerramento do php "<?php" e ">?". Além disso, você pode pular algumas linhas entre elas para facilitar a digitação do código. Após isso, já estará apto para digitar os primeiros códigos entre as tags do php, e inicie esse trabalho por criar um bloco "try e catch" onde, vamos instanciar o objeto de conexão com o banco de dados: PDO.

```
1 | <?php
2 |
3 | try {
4 |     $conexao = new PDO("mysql:host=localhost; dbname=crudsimples", "root", "123456");
5 | } catch (PDOException $erro) {
6 |
7 | }
```

Note como que instanciamos o objeto PDO, sua sintaxe é:

```
<objeto PDO> = new PDO(String <Nome da Fonte de dados>, String <usuário do banco>, String <senha do banco>);
```

Note que o objeto que será associado a variável \$conexao será do tipo PDO, um objeto que representa a conexão com o banco e que será o nosso manipulador.

Como parâmetros, o construtor da classe exige:

- uma DSN (Nome da fonte de dados), onde informamos o driver do banco, o nome do host e o nome do banco de dados e note que são separados por ponte e vírgula. Mas a DSN suporta diferentes métodos de argumentação. Caso tenha interesse dê uma olhada na documentação (http://php.net/manual/pt_BR/pdo.construct.php)
- um nome de usuário para conexão com o banco de dados
- a senha para o usuário informado.

Opcionalmente é possível passar um array associativo com opções de conexão, como um terceiro parâmetro, mas não vou abranger aqui.

Observação: deste momento em diante, não darei tanta ênfase as tags de abertura e fechamento do bloco de códigos php. Serão mencionadas apenas quando for necessário misturar código php com tags html ou adicionar em outro local no arquivo.

Perceba que o parâmetro de catch foi informado como um argumento do tipo PDOException. Esta classe do php estende RuntimeException e é específica para representar os erros gerados pela classe PDO. Para termos uma representação mais específica, a classe PDO nos apresenta um método setAttribute para o objeto PDO, note os blocos acima reescritos com o método, logo abaixo:

```
1 | try {
2 |     $conexao = new PDO("mysql:host=localhost; dbname=crudsimples", "root", "123456");
3 |     $conexao->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
4 | } catch (PDOException $erro) {
5 |     echo "Erro na conexão: " . $erro->getMessage();
6 | }
```

O método setAttribute() nos permite adicionar atributos no objeto de conexão, sua sintaxe:

```
<valor booleano> = setAttribute(inteiro "Atributo", misto "Valor");
```

Note que o método devolve um valor booleano como resposta e assim você pode descobrir se tudo correu bem. Ainda, como atributos, há um do tipo inteiro e outro que pode ser de vários tipos. No caso do argumento inteiro, este método deve receber algum dos atributos válidos para o objeto de conexão. Estes objetos são representados por constantes, semelhantes a esta que estamos

usando aí acima. Se você estiver interessado em saber quais são, veja a documentação:

http://php.net/manual/pt_BR/pdo.setattribute.php. Por hora vamos utilizar `PDO::ATTR_ERRMODE` que indica como relatar o erro.

Entenda que o segundo argumento deve ser compatível com o primeiro. No caso de `PDO::ATTR_ERRMODE`, espera-se receber um dos três modos disponíveis:

- `PDO::ERRMODE_SILENT`: Só define o código de erro
- `PDO::ERRMODE_WARNING`: gerar um `E_WARNING`.
- `PDO::ERRMODE_EXCEPTION`: Lance uma exceção.

Atento a isto, descobrimos que a última opção é a melhor por estarmos trabalhando com uma `PDOException` no nosso bloco `catch`. Sendo assim, agora será possível imprimir uma exceção lançada, por chamar o método `getMessage()` da `PDOException`. É exatamente isso que estamos fazendo dentro do bloco `catch`, como você pode ter notado.

Pois bem, será que já terminamos a parte de conexão? Talvez, mas poderemos melhorar algo aqui: lembre-se que estamos no Brasil, onde é comum utilizar acentos e, por isso, a codificação comum é UTF-8. Como é que informamos isso ao servidor de banco de dados? Note os blocos `try` e `catch` reescritos e terminados:

```
1 try {
2     $conexao = new PDO("mysql:host=localhost; dbname=crudsimples", "root", "123456");
3     $conexao->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
4     $conexao->exec("set names utf8");
5 } catch (PDOException $erro) {
6     echo "Erro na conexão:" . $erro->getMessage();
7 }
```

O que mudou? Foi adicionado uma linha com o método `exec()` da classe `PDO`, que executa uma instrução SQL e retornar o número de linhas afetadas. Neste caso, não esperamos nenhum retorno, mas queremos executar a instrução SQL direto no banco. Note que esta não é uma instrução da classe `PDO` e sim do banco de dados. Caso queira saber mais a respeito desta instrução no MySQL, consulte <http://dev.mysql.com/doc/refman/5.7/en/charset-connection.html>

Muito bem, terminado a segunda etapa de construção da aplicação, a próxima será criar um formulário simples e inserir dados nele, construindo assim a primeira função do CRUD.

Embora este artigo não tenha como meta apresentar a programação POO, pois se concentra na classe PDO, se você tiver interesse em saber como criar uma classe de conexão com o banco de dados, algo bem moderno usado em alguns frameworks modernos, não deixe de ver este meu artigo: [PHP:: Conexão conforme os padrões singleton e Factory Method](#). Nele, é apresentado passo a passo como criar uma classe de conexão capaz de gerar objetos PDO para muitos bancos e ainda, manter uma única instância ativa durante a conexão. Com a classe apresentado neste procedimento, todas as linhas acima, seriam substituídas apenas pela chamada: `$conexao = connection::getInstance('nomeDoArquivo.INI')`.

Terceira Etapa: Criando um formulário

Precisamos de um formulário, então vamos criá-lo por inserir este código HTML bem abaixo do código php que já criamos com a conexão. Lembre-se antes de fechar a tag PHP com um `"?>"` antes de inserir as tags seguintes, se você já não o fez. 😊

```
1 <!DOCTYPE html>
2 <html>
3     <head>
4         <meta charset="UTF-8">
5         <title>Agenda de contatos</title>
6     </head>
7     <body>
8         <form action="?act=save" method="POST" name="form1" >
9             <h1>Agenda de contatos</h1>
10            <hr>
11            <input type="hidden" name="id" />
12            Nome:
13            <input type="text" name="nome" />
14            E-mail:
15            <input type="text" name="email" />
16            Celular:
17            <input type="text" name="celular" />
18            <input type="submit" value="salvar" />
19            <input type="reset" value="Novo" />
20            <hr>
21        </form>
22    </body>
23 </html>
```

Até aqui, se você abrir a página, vai ver o nosso formulário com 3 campos e dois botões, mas ainda não funciona.



Veja que também utilizamos uma tag input do tipo "hidden" e esta será utilizada para o recurso de Update. Visto que, já que mencionei o "Update", vamos já preparar nosso formulário para a função este recurso. Então, dentro de cada tag input, antes do encerramento "/", vamos acrescentar um espaço e adicionar um código php para preenchimento automático. Este código, naturalmente deverá estar delimitado pelas tags do php "<?php" e ">?". Então, leve o cursor até a área informada acima e adicione o código entre as tags do php. Ele deverá ser assim:

```

1  <html>
2  <head>
3  <meta charset="UTF-8">
4  <title>Agenda de contatos</title>
5  </head>
6  <body>
7  <form action="?act=save" method="POST" name="form1" >
8  <h1>Agenda de contatos</h1>
9  <hr>
10 <input type="hidden" name="id" <?php
11 // Preenche o id no campo id com um valor "value"
12 if (isset($id) && $id != null || $id != "") {
13     echo "value=\"{$id}\"";
14 }
15 ?> />
16 Nome:
17 <input type="text" name="nome" <?php
18 // Preenche o nome no campo nome com um valor "value"
19 if (isset($nome) && $nome != null || $nome != ""){
20     echo "value=\"{$nome}\"";
21 }
22 ?> />
23 E-mail:
24 <input type="text" name="email" <?php
25 // Preenche o email no campo email com um valor "value"
26 if (isset($email) && $email != null || $email != ""){
27     echo "value=\"{$email}\"";
28 }
29 ?> />
30 Celular:
31 <input type="text" name="celular" <?php
32 // Preenche o celular no campo celular com um valor "value"
33 if (isset($celular) && $celular != null || $celular != ""){
34     echo "value=\"{$celular}\"";
35 }
36 ?> />
37 <input type="submit" value="salvar" />
38 <input type="reset" value="Novo" />
39 <hr>
40 </form>
41 </body>
42 </html>

```

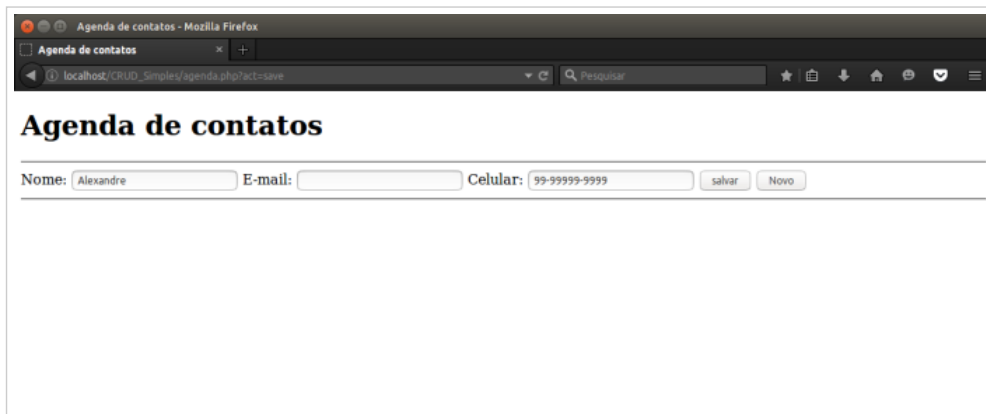
Novamente, até aqui se você abrir a página no browser, ainda não vai acontecer porque estas instruções de decisão apenas estão verificando se uma variável foi definida e se seu valor é diferente de nulo ou diferente de vazio. Todavia, as variáveis ainda não foram definidas e por isso nada acontece. Mas vamos adicionar estas novas instruções antes do bloco "try e catch" na primeira parte do nosso código, ou seja, ficará logo depois da primeira tag de abertura do php "<?php" do início e antes do bloco "try e catch":

```

1  <?php
2  // Verificar se foi enviando dados via POST
3  if ($_SERVER['REQUEST_METHOD'] == 'POST') {
4      $id = (isset($_POST["id"]) && $_POST["id"] != null) ? $_POST["id"] : "";
5      $nome = (isset($_POST["nome"]) && $_POST["nome"] != null) ? $_POST["nome"] : "";
6      $email = (isset($_POST["email"]) && $_POST["email"] != null) ? $_POST["email"] : "";
7      $celular = (isset($_POST["celular"]) && $_POST["celular"] != null) ? $_POST["celular"] : NULL;
8  } else if (!isset($id)) {
9      // Se não se não foi setado nenhum valor para variável $id
10     $id = (isset($_GET["id"]) && $_GET["id"] != null) ? $_GET["id"] : "";
11     $nome = NULL;
12     $email = NULL;
13     $celular = NULL;
14 }

```

Agora, se você executar a página, poderá preencher o formulário e notar uma diferença ao clicar em "Salvar". A diferença é quando clicar no botão "Salvar" e fazer o "submit", pois, os dados permanecerão nos campos como se nada acontecesse.



Artigo sobre CRUD em PHP: Adicionando funcionalidade ao formulário

Como assim? Esta é a nossa intenção neste momento, porque definimos e preenchemos as variáveis via POST, aquelas que são verificadas no código dentro das tags input dentro do formulário. Mesmo com refresh da página, os dados não sumirão, mas continuarão lá. Precisamos agora salvar estes dados no banco de dados e assim alcançar nosso primeiro objetivo. Esta é a nossa próxima etapa: Create! Você deve ter notado também que estamos trabalhando com uma variável \$id, embora esta variável ainda não receba nenhum valor. Não se preocupe logo voltaremos nossa atenção a ela.

Quarta Etapa: Create – salvando novo registro no banco de dados

Nosso formulário contém dados que precisam ser persistidos no banco de dados. Para isso funcionar, vamos adicionar o código após a ultima chave que encerra o “try e catch” criado para a conexão. Ah, há mais um detalhe aqui que reforço: Estamos trabalhando no primeiro bloco de comandos php, então vamos inserir o seguinte código logo abaixo do “try e catch” de conexão. A frase foi repetida para reforçar a atenção! O código é este:

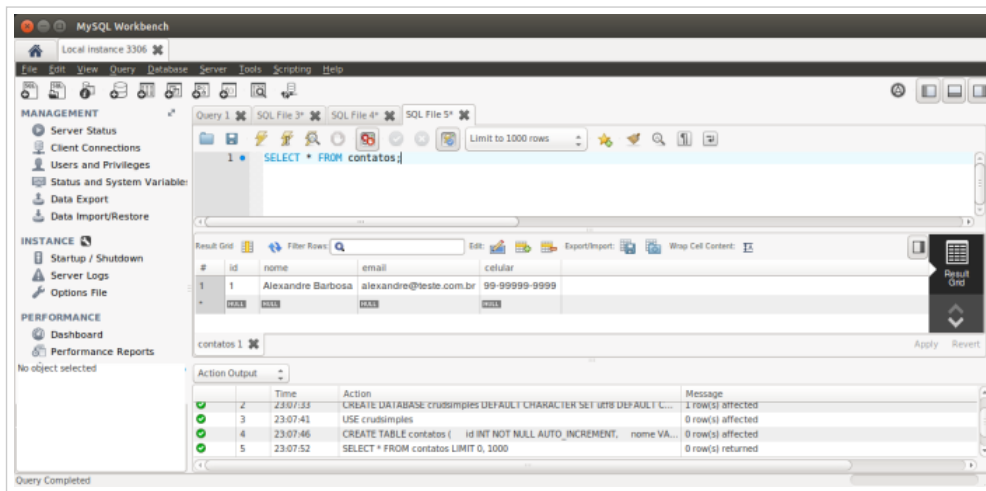
```
1 if (isset($_REQUEST["act"]) && $_REQUEST["act"] == "save" && $nome != "") {
2     try {
3         $stmt = $conexao->prepare("INSERT INTO contatos (nome, email, celular) VALUES (?, ?, ?)");
4         $stmt->bindParam(1, $nome);
5         $stmt->bindParam(2, $email);
6         $stmt->bindParam(3, $celular);
7
8         if ($stmt->execute()) {
9             if ($stmt->rowCount() > 0) {
10                 echo "Dados cadastrados com sucesso!";
11                 $id = null;
12                 $nome = null;
13                 $email = null;
14                 $celular = null;
15             } else {
16                 echo "Erro ao tentar efetivar cadastro";
17             }
18         } else {
19             throw new PDOException("Erro: Não foi possível executar a declaração sql");
20         }
21     } catch (PDOException $erro) {
22         echo "Erro: " . $erro->getMessage();
23     }
24 }
```

Muito bem! se você rodar este código e clicar em salvar, as informações vão ser escritas no banco de dados e o formulário não retornará com mais nenhum valor. Além disso, uma mensagem “Dados cadastrados com sucesso!” será apresentada logo acima do formulário:



Artigo sobre CRUD em PHP: Etapa Create – formulário

Agora note na próxima imagem que os dados podem ser recuperados no Banco de dados:



Artigo sobre CRUD em PHP: Visualizando os dados inseridos no banco de dados

Vamos entender o que foi feito até aqui:

Primeiro, note que iniciamos um bloco "if" para descobrir se há uma ação "act" para salvar "save", se sim, então entra em um novo bloco try e catch. Neste bloco try e catch, utilizamos o objeto de conexão do PDO que está associado a variável `$conexao`. Assim, temos a nossa disposição alguns métodos do objeto PDO para atender as nossas necessidade. O primeiro método é o `prepare()`, note a linha:

```
1 | $stmt = $conexao->prepare("INSERT INTO contatos (nome, email, celular) VALUES (?, ?, ?)");
```

Este método transforma uma declaração SQL na forma de um "objeto declaração" que pode ser manipulado por alguns métodos específicos. Entre estes métodos, o `bindParam()` que vincula uma variável a um espaço demarcado na declaração.

```
1 | $stmt->bindParam(1, $nome);
2 | $stmt->bindParam(2, $email);
3 | $stmt->bindParam(3, $celular);
```

O método `bindParam()` recebe ainda outros argumentos, mas não vamos abrangê-los aqui. Se desejar descobrir mais sobre isso, não deixe de conferir a documentação do php: http://php.net/manual/pt_BR/pdostatement.bindparam.php

A declaração estando pronta, ou seja, o novo objeto PDO Statement, então poderá ser executada. Neste momento que chamamos o método `execute()` do próprio objeto Statement:

```
1 | if ($stmt->execute()) {
```

Você até poderia deixar de usar o método `bindParam` e definir as variáveis dentro de um array passando-os como argumento diretamente ao método `execute()`, todavia isto será valido apenas para tipos String. Além disso, você também poderia colocar a variável diretamente na declaração recebida pelo método `prepare()`, mas estaria vulnerável a problemas de segurança que também não vou destacar neste artigo. Em resumo, procure sempre usar o `bindParam()`.

Ainda, veja que chamamos o método `execute()` dentro de um bloco "if", pois ele retornará um valor booleano true se a instrução for executada com êxito. E por isso, temos um recurso para darmos o início ao tratamento de erros. Visando refinar esse tratamento de erros, aproveitamos para fazer uso de mais um método da PDO Statement que fica disponível após o `execute`, que é o `rowCount()`:

```
1 | if ($stmt->rowCount() > 0) {
```

O `rowCount()` retornará o número de linhas afetadas na tabela por uma última instrução DELETE, INSERT ou UPDATE executada pelo objeto PDOStatement correspondente. Em nosso caso, até agora foi um INSERT. Já que estamos inserindo uma linha, esperamos que como resposta haja um número maior que zero "0". Logo, temos uma segunda etapa para tratar erros que nos permite fazer mais alguma coisa em caso de êxito ou não. E é isso que acontece:

Primeiro, imprimimos uma mensagem informando que tudo correu bem:

```
1 | echo "Dados cadastrados com sucesso!";
```

Em seguida, limpamos as variáveis para que o formulário não seja preenchido novamente nesta etapa:

```
1 | $id = null;
2 | $nome = null;
3 | $email = null;
4 | $celular = null;
```

Concluimos a etapa Create do CRUD. Agora vamos passar para próxima etapa que é o Read.

Quinta Etapa: Read – Lendo os registros no banco

A funcionalidade "Read" pode ser construída de algumas maneiras: por usar um filtro para ler um registro por vez e em seguida o apresentar na tela ou mesmo utilizar algum filtro para ler vários registros e listá-los na tela, ou ainda, chamando uma listagem inteira por trazer todos os registros de uma tabela. Vamos chamar uma listagem inteira, embora isto seja verdade apenas de forma acadêmica. Vamos seguir adiante, criando uma tabela html com a tag table. Você deve posicionar esta tag table logo depois da tag de

fechamento do formulário "</form>" que criamos dentro das tags "<body>" e "</body>" em nosso html e antes da tag de fechamento "</body>". Este é o código:

```
1 <table border="1" width="100%">
2   <tr>
3     <th>Nome</th>
4     <th>E-mail</th>
5     <th>Celular</th>
6     <th>Ações</th>
7   </tr>
8 </table>
```

Se você abrir a página, vai perceber apenas o cabeçalho da tabela, mas nada é impresso abaixo dele e esta é nossa intenção até este momento.



Mas, agora vamos então gerar uma listagem dos registros abaixo deste cabeçalho e para isso, você deverá abrir e fechar as tags do php "<?php" e ">?" logo abaixo da tag de fechamento da linha da tabela "</tr>" e, antes da tag de fechamento da tabela "</table>". Dentro deste bloco de código php, precisamos adicionar um novo bloco "try e catch":

[illegible]

Dentro da tabela entre as tags "<table>" e "</table>", ficará assim:

[illegible]

Abriendo a página, você perceberá que já é possível ver um registro salvo anteriormente. Criamos este registro no teste da quarta etapa.



O que foi feito lá no código inserido?

Notou que preparamos mais uma declaração com o método *prepare()*? Sim, é exatamente para convertermos nossa declaração SQL em um objeto PDO Statement. Depois disto, logo utilizamos o método *execute()* do objeto PDO Statement mas, desta vez, não precisamos informar nenhum parâmetro e por isso não utilizamos o método *bindParam()*. Mas se fôssemos utilizar um filtro, seria ideal utilizar o método *bindParam()*, como já dito anteriormente. Na próxima etapa, vamos precisar construir um “*Read*” desta forma e você irá compreender o porquê.

Depois de executar o método `execute()`, precisamos recuperar o resultado de alguma forma, então iterar sobre cada registro resultante e apresentá-lo na tela. Por isso, criamos uma variável `$rs` de “*result set*” que receberá a cada iteração de loop como sendo um objeto de registro. O método `fetch()` do novo objeto “*result set*”, “busca” o resultado obtido pelo método `execute()` de PDO Statement. Perceba que um argumento é passado ao método `fetch`, o `PDO::FETCH_OBJ`, a fim de informar ao método que queremos obter os registros como objetos. O loop utiliza a função `while`, que talvez você já tenha entendido o motivo, entretanto, apenas para ficar claro, aqui estamos informando que as iterações deverão continuar enquanto um valor seja verdadeiro para `while`. Enquanto for possível iterar pelo “*result set*”, será verdadeiro. Assim, dentro do bloco `while` poderemos nos referir ao objeto em `$rs` por seus atributos. Mas, que atributos são estes? São os nomes das colunas na tabela contatos do banco de dados. Então, isso se torna interessante porque trabalhar com um registro de forma orientado a objetos tornam os nossos projetos bem mais interessantes. É claro que você pode iterar o “*result set*” e trabalhar com array, assim como as antigas técnicas e se desejar saber mais sobre isso leia mais sobre isso, consulte a documentação do php em http://php.net/manual/pt_BR/pdostatement.fetch.php. Entretanto, se estiver criando um sistema totalmente orientado a objetos, o ideal será manipular os resultados como objetos.

Uma vez sabendo como recuperar o valor de cada coluna de um **"result set"**, é fácil imprimi-los na tela e fazemos isso simplesmente com o construtor de linguagem "echo", como foi visto nas linhas seguintes:

```
1 echo "<tr>";
2 echo "<td>".${rs->nome."</td><td>".${rs->email."</td><td>".${rs->celular
3 "</td><td><center><a href=\"\"\"[Alterar]</a>"
4 "&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&";
5 "<a href=\"\"\"[Excluir]</a></center></td>";
6 echo "</tr>";
```

Sexta Etapa: Update – alterando registros salvos no banco.

Até aqui, você deve ter notado que ao trabalhar com a classe PDO seguimos um certo padrão e por isso, adicionar os recursos finais ao nosso aplicativo, será relativamente fácil a você compreender, mesmo agora antes de implementá-los. Note que já deixamos previamente preparados dois links para cada registro que são impressos na tabela html, que apresenta nossos registros. Mas estes links não possuem uma âncora para nenhum lugar. Se você clicar sobre eles, não farão absolutamente nada. Faremos uso do primeiro link para implementar o "Update" mas nada tão complexo assim. Para começar, precisamos informar um filtro logo ali, qual registro será que queremos alterar. Então, logo nos lembramos que para cada registro, temos um campo numérico e que é a chave primária do mesmo. Este campo é o id. Além disso, precisamos nos lembrar que nossa aplicação tendenciosamente decide tudo no "refresh de tela", assim precisaremos indicar uma ação "act" que será interpretada neste momento. Por isso, volte sua atenção para estas linhas:

[illegible]

Dentro da tag "<a>" com rótulo a frente "[Alterar]", preencha entre as duas "aspas escapadas" (\") a seguinte informação:

```
1 | ?act=upd&id=" . $rs->id . "
```

Difícil? Não, não é! Note o que muda:

[illegible]

O que fizemos ali foi indicar na âncora "*href*" do link os parâmetros "*act*" e "*id*" que serão lidos pela mesma página no refresh. Note também que não indicamos uma página alvo para âncora e sim apenas deixamos o sinal de interrogação "?". Se você abrir a página no navegador e levar ponteiro acima do link "Alterar" de cada registro, vai notar na barra de status inferior do browser que agora um link

é informado. Além disso, também já apresenta o id para cada registro. Já que estamos mexendo ali, podemos também já deixar pronto o outro link para “Delete”. Então, no dentro da tag “<a>” com rótulo “[Excluir]” logo a frente, preencha entre as duas “aspas escapadas” (“”) a seguinte informação:

```
1 | ?act=del&id=" . $rs->id . "
```

Veja agora o que muda:

```
1 echo "<td>".$rs->nome."</td><td>".$rs->email."</td><td>".$rs->celular
2      "</td><td><center><a href=\"\"?act=upd&id= \" . $rs->id . "\">[Alterar]</a>\"
3      "<br><br><br><br><br><br><br>\"
4      "<a href=\"\"?act=del&id= \" . $rs->id . "\">[Excluir]</a></center></td>\";
5 echo "</tr>\";
```

O que fizemos alí, também foi indicar na âncora do link os parâmetros “act” e “id” que serão lidos pela mesma página no refresh. Novamente, se você abrir a página no navegador e levar ponteiro até o link “Excluir” de cada registro, vai notar na barra de status inferior do browser que agora também há um link sendo informado. Além disso, já apresenta o id para cada registro.

Certo, até aqui preparamos o nosso aplicativo para receber os novos recursos que precisam ser implementados. Na verdade, apenas a parte de interação com o usuário está pronta, mas não a sua funcionalidade. Então vamos a lógica!

Para alterar um registro, precisamos primeiro recuperar este registro no formulário para dar ao usuário exatamente o registro salvo. E então como faremos isso? Você se lembra das variáveis *\$id*, *\$nome*, *\$email* e *\$celular*? Sim, estas variáveis precisam receber os dados filtrados para preencherem o formulário. Então vamos fazer o seguinte: logo após o final do bloco “if” que foi criado lá na quarta etapa “Create”, ou seja, o bloco que faz a persistência dos dados (Lembra-se?), será necessário adicionar este novo bloco “if” logo abaixo:

```

1 if (isset($_REQUEST["act"]) && $_REQUEST["act"] == "upd" && $id != "") {
2     try {
3         $stmt = $conexao->prepare("SELECT * FROM contatos WHERE id = ?");
4         $stmt->bindParam(1, $id, PDO::PARAM_INT);
5         if ($stmt->execute()) {
6             $rs = $stmt->fetch(PDO::FETCH_OBJ);
7             $id = $rs->id;
8             $nome = $rs->nome;
9             $email = $rs->email;
10            $celular = $rs->celular;
11        } else {
12            throw new PDOException("Erro: Não foi possível executar a declaração sql");
13        }
14    } catch (PDOException $erro) {
15        echo "Erro: ".$erro->getMessage();
16    }
17 }

```

Mais uma vez, quando você abrir a página novamente e clicar no Update, note que os dados daquele registro serão recuperados no formulário, semelhante ao início, quando estávamos enviando via POST para testar o preenchimento do formulário. A diferença desta vez é o processo dos dados serem recuperados não de uma origem via POST, mas agora diretamente do banco de dados.

Percebeu lá no código que inserimos agora que este bloco “if” também é muito semelhante a aquele anterior na quinta etapa onde listamos os registros na tabela? Sim, mas com duas diferenças sutis:

Primeira: Utilizamos um filtro na instrução SQL pelo id e novamente fazemos uso do `bindParam()` para adicioná-lo ao objeto PDO Statement. Além disso, também informamos agora mais um novo parâmetro no método `bindParam()`, que informa qual é o tipo de dado da variável \$id. Este parâmetro é indicado pela constante `PDO::PARAM_INT`.

Segunda: Não utilizamos mais o laço de repetição While! E isso faz sentido aqui, pois se estamos filtrando pela coluna chave primária, esperamos é claro recuperar apenas um único registro.

Na sequência, são recuperados os atributos do **"result set"** para cada variável correspondente, que são lidas logo em seguida durante a construção do formulário. Assim, os campos são preenchidos com os dados das variáveis, "alimentado-os com os valores" em "value". Interessante? 😊

Ok mas, recuperamos os dados no formulário, alteramos e ... e agora para salvar os dados atualizados novamente no banco? Porque se você clicar no botão salvar, o resultado será um novo registro logo abaixo:

Dados cadastrados com sucesso!

Agenda de contatos

Nome: E-mail: Celular:

Nome	E-mail	Celular	Ações
Alexandre Barbosa	alexandre@teste.com.br	99-99999-9999	[Alterar] [Excluir]
Alexandre Bezerra Barbosa	alexandre@teste.com.br	99-99999-9999	[Alterar] [Excluir]

Artigo sobre CRUD em PHP: Está duplicando os dados ao invés de atualizá-los

Falhamos em nosso projeto? Precisamos adicionar mais um botão para "Update"? Claro que não! Mas fica evidente que precisamos fazer uma alteração na lógica para atender a nova funcionalidade. Afinal, os nossos objetivos anteriores estavam sendo alcançados, mas agora estamos em uma nova etapa! 😊

Esta mudança será feita no bloco "if" que criamos na Quarta Etapa: Create. Então, esteja atento as mudanças seguintes:

Primeiro, olhe atentamente a linha abaixo:

```
1 | $stmt = $conexao->prepare("INSERT INTO contatos (nome, email, celular) VALUES (?, ?, ?)");
```

Coloque esta linha dentro do bloco "else" de um novo bloco "if" que criaremos para fazer teste da variável \$id, para checar se a mesma recebeu algum valor. O quê? Sim, desta forma:

```
1 | if ($id != "") {
2 |
3 | } else {
4 |     $stmt = $conexao->prepare("INSERT INTO contatos (nome, email, celular) VALUES (?, ?, ?)");
5 | }
```

Muito bem! então, alí a decisão recai sobre condição em que, se a variável \$id estiver vazia, será feito um **Insert**, como já estava sendo feito antes. Mas e se a variável \$id possuir um valor diferente de vazio?

Neste caso, faremos um "Update"! Aqui está a grande sacada! 😊 Então, o bloco deverá receber uma nova preparação de declaração para "Update", ficando portando, assim:

```
1 | if ($id != "") {
2 |     $stmt = $conexao->prepare("UPDATE contatos SET nome=?, email=?, celular=? WHERE id = ?");
3 |     $stmt->bindParam(4, $id);
4 | } else {
5 |     $stmt = $conexao->prepare("INSERT INTO contatos (nome, email, celular) VALUES (?, ?, ?)");
6 | }
```

Notou que, caso seja recebido um valor em \$id, estaremos executando um "UPDATE" para aquele valor armazenado na variável \$id? 😊

Além disso, como vamos lidar com um quarto parâmetro a ser inserido no objeto PDO Statement, ele também já é informado logo na linha abaixo da instrução contendo o método prepare(), dentro do mesmo bloco "if":

```
1 | $stmt->bindParam(4, $id);
```

Agora a aplicação está funcionando como esperado! Já é possível criar novos registros e atualizá-los! Faça o teste! 😊

Sétima Etapa: Delete – Excluindo registros do banco de dados

Você já deve estar se amarrando em tudo isso até aqui! 😊 😊 😊

Mas ainda falta um recurso para concluirmos nosso projeto do CRUD: *Delete*. Se você foi acompanhando o passo a passo e testando o aplicativo, neste momento já deve contar com um monte de registros que pensa em excluir:



Artigo sobre CRUD em PHP: Avaliando registros criados

Você deve se lembrar de que na etapa anterior, preparamos a interface com usuário, deixando basicamente pronto o Link de exclusão de registro, se lembra? Portanto, falta apenas implementar a funcionalidade para este link! Para isso, logo abaixo do último bloco "if" e antes da tag de encerramento do primeiro bloco php ">" e que se localiza antes das tags html, inclua este novo bloco if:

```

1  if (isset($_REQUEST["act"]) && $_REQUEST["act"] == "del" && $id != "") {
2      try {
3          $stmt = $conexao->prepare("DELETE FROM contatos WHERE id = ?");
4          $stmt->bindParam(1, $id, PDO::PARAM_INT);
5          if ($stmt->execute()) {
6              echo "Registro foi excluído com êxito";
7              $id = null;
8          } else {
9              throw new PDOException("Erro: Não foi possível executar a declaração sql");
10         }
11     } catch (PDOException $erro) {
12         echo "Erro: ".$erro->getMessage();
13     }
14 }

```

Veja que estes códigos já se tornaram muito familiares, mas com certas semelhanças e sutis diferenças:

Primeira observação: nossa declaração é um Delete e recebe um parâmetro id que será associado ao objeto PDO Statement utilizando o método `bindParam()`, novamente com o novo parâmetro `PDO::PARAM_INT`.

Segunda observação: agora não utilizamos mais um método `fetch()` para "juntar" alguma coisa, mas apenas o método `execute()`. Sim, é muito óbvio o motivo disso: porque não precisamos recuperar nenhum valor mas apenas executar o método `execute()` que já o é suficiente para cumprir o objetivo.

Perceba que dentro do bloco "if" do método `execute()` acima, deixamos uma instrução para imprimir na tela uma mensagem, informando que o registro foi excluído com êxito. Além disso, após a exclusão, precisamos anular qualquer valor na variável \$id, para que o nosso formulário retorne livre e pronto para receber novos registros. Se você iniciar a página no browser, verá que o recurso de exclusão agora também funciona perfeitamente!

Conclusão

Pronto! Agora temos um CRUD completo, contendo as quatro operações do banco de dados em funcionamento: Create, Read, Update e Delete! Acredito que por seguir este passo a passo a risca, lendo e relendo se for o caso, você estará apto a criar aplicações com CRUD. Poderá criar até outras aplicações mais complexas do que esta deste artigo!



Artigo sobre CRUD em PHP Agenda de Contatos

Abaixo está o código completo para que você possa conferir:

Dica importante: Nunca recomendo que você seja forte seguidor do uso das teclas CTRL+C e CTRL+V quando está aprendendo, mas o ideal é ir digitando caractere a caractere! Todavia, se preferir fazê-lo a fim de conferir a funcionalidade do código, a decisão está em suas mãos! have a fun! 😊

```

1  <?php
2  /**
3   * Projeto de aplicação CRUD utilizando PDO - Agenda de Contatos
4   *
5   * Alexandre Bezerra Barbosa
6   */
7
8  // Verificar se foi enviando dados via POST
9  if ($_SERVER['REQUEST_METHOD'] == 'POST') {
10     $id = (isset($_POST["id"]) && $_POST["id"] != null) ? $_POST["id"] : "";
11     $nome = (isset($_POST["nome"]) && $_POST["nome"] != null) ? $_POST["nome"] : "";
12     $email = (isset($_POST["email"]) && $_POST["email"] != null) ? $_POST["email"] : "";
13     $celular = (isset($_POST["celular"]) && $_POST["celular"] != null) ? $_POST["celular"] : NULL;
14 } else if (!isset($id)) {
15     // Se não se não foi setado nenhum valor para variável $id
16     $id = (isset($_GET["id"]) && $_GET["id"] != null) ? $_GET["id"] : "";
17     $nome = NULL;
18     $email = NULL;
19     $celular = NULL;
20 }
21
22 // Cria a conexão com o banco de dados
23 try {
24     $conexao = new PDO("mysql:host=localhost;dbname=crudsimples", "root", "123456");
25     $conexao->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
26     $conexao->exec("set names utf8");
27 } catch (PDOException $erro) {
28     echo "Erro na conexão: ".$erro->getMessage();
29 }
30
31 // Bloco If que Salva os dados no Banco - atua como Create e Update
32 if (isset($_REQUEST["act"]) && $_REQUEST["act"] == "save" && $nome != "") {
33     try {
34         if ($id != "") {
35             $stmt = $conexao->prepare("UPDATE contatos SET nome=?, email=?, celular=? WHERE id = ?");
36             $stmt->bindParam(4, $id);
37         } else {
38             $stmt = $conexao->prepare("INSERT INTO contatos (nome, email, celular) VALUES (?, ?, ?)");
39         }
40         $stmt->bindParam(1, $nome);
41         $stmt->bindParam(2, $email);
42         $stmt->bindParam(3, $celular);
43
44         if ($stmt->execute()) {
45             if ($stmt->rowCount() > 0) {
46                 echo "Dados cadastrados com sucesso!";
47                 $id = null;
48                 $nome = null;
49                 $email = null;
50                 $celular = null;
51             } else {
52                 echo "Erro ao tentar efetivar cadastro";
53             }
54         } else {
55             throw new PDOException("Erro: Não foi possível executar a declaração sql");
56         }
57     } catch (PDOException $erro) {
58         echo "Erro: ". $erro->getMessage();
59     }
60 }
61
62 // Bloco if que recupera as informações no formulário, etapa utilizada pelo Update
63 if (isset($_REQUEST["act"]) && $_REQUEST["act"] == "upd" && $id != "") {
64     try {
65         $stmt = $conexao->prepare("SELECT * FROM contatos WHERE id = ?");
66         $stmt->bindParam(1, $id, PDO::PARAM_INT);
67         if ($stmt->execute()) {
68             $rs = $stmt->fetch(PDO::FETCH_OBJ);
69             $id = $rs->id;
70             $nome = $rs->nome;
71             $email = $rs->email;
72             $celular = $rs->celular;
73         } else {
74             throw new PDOException("Erro: Não foi possível executar a declaração sql");
75         }
76     } catch (PDOException $erro) {
77         echo "Erro: ". $erro->getMessage();
78     }
79 }
80
81 // Bloco if utilizado pela etapa Delete
82 if (isset($_REQUEST["act"]) && $_REQUEST["act"] == "del" && $id != "") {
83     try {
84         $stmt = $conexao->prepare("DELETE FROM contatos WHERE id = ?");
85         $stmt->bindParam(1, $id, PDO::PARAM_INT);
86         if ($stmt->execute()) {
87             echo "Registro foi excluído com êxito";
88             $id = null;
89         } else {
90             throw new PDOException("Erro: Não foi possível executar a declaração sql");
91         }
92     } catch (PDOException $erro) {
93         echo "Erro: ". $erro->getMessage();
94     }
95 }
96 }
97 <!DOCTYPE html>

```

◀ ▶

[]'s

Anúncios

**Relatório Forrester Wave**

Faça-o-download do RelatórioQ1

Saiba por que a SAP Hybris foi reconhecida como líder global. Faça-o-download.

hybris.com/forrester-wave/faça-o-download

Anúncio

Compartilhe isso com seus amigos:

[2 blogueiros](#) gostam disto.

4 de setembro de 2016 em PHP, Programação. Tags:Aplicativo, Create, CRUD, Delete, MySQL, PHP, Read, Select, SQL, Update

Posts Relacionados**PHP:: MVC – CRUD COMPLETO****PHP e MySQL :: CRUD – Create, Read, Update, Delete****PHP:: PDO – CONEXÃO CONFORME PADRÕES SINGLETON E FACTORY**[← WINDOWS:: SERVIDORES – BACKUP DO DNS](#)[PHP:: Construindo uma classe para Calendário →](#)**15 comentários sobre “PHP:: PDO – CRUD COMPLETO”**Pingback: [PHP e MySQL :: CRUD – Create, Read, Update, Delete | Alexandre Bezerra Barbosa](#)**Kady Aoun** 22 de abril de 2017 às 18:08

muito bom! gostaria de aprender com o CRUD usado em casses separadas do arquivo index, paara poder usar quando quiser.....

[Responder](#)**Alexandre Bezerra Barbosa** 3 de maio de 2017 às 07:34

Que bom Kady, fico feliz de saber que estou contribuindo em seu aprendizado! Ainda tenho mais dois artigos sobre CRUD que apresentam conceitos através de classes e talvez possa te ajudar no que precisa. Um deles utiliza [um padrão de projeto "Table Data Gateway"](#) e o outro apresenta [uma ideia simplificada com DAO](#). Aproveite este material também! Em breve vou publicar um artigo com um exemplo totalmente POO.

[Responder](#)



Andrei 27 de abril de 2017 às 16:10

Muito bom o artigo, eu estive procurando conteúdos que fossem "limpos" de framework para aprender essa parte do PHP e esse me ajudou bastante, ótima didática, parabéns. Como vi no outro comentário aqui, seria interessante um conteúdo de MVC no PHP também. Abraços.

[Responder](#)



Alexandre Bezerra Barbosa 3 de maio de 2017 às 07:31

Obrigado Andrei, fico muito satisfeito com seu feedback sendo positivo e que de fato tenha alcançado meu objetivo: Ajudar! É verdade que ainda não tenho um artigo com uma aplicação exemplo em MVC ou totalmente orientado a objetos. Mas enquanto este artigo não ficar pronto, tenho mais dois artigos sobre CRUD que apresentam conceitos através de classes. Um deles utiliza [um padrão de projeto "Table Data Gateway"](#) e o outro apresenta [uma ideia simplificada com DAO](#). Espero que possa aproveitar este material também!

[Responder](#)



Altair Julião 1 de outubro de 2017 às 23:33

Ótimo artigo, parabéns! No entanto, fiquei com uma dúvida: pelo que entendi o celular não é um campo obrigatório já que no BD aceita valores NULL, no entanto na verificação de preenchimento do formulário é feita a verificação se o campo está nulo, exigindo seu preenchimento. Estou com um problema aqui para criar um CRUD justamente porque o campo deve aceitar valores nulos, mas pelo que estou lendo o NULL do PHP é diferente do NULL do MySQL.

[Responder](#)



Alexandre Bezerra Barbosa 2 de outubro de 2017 às 08:52

Olá Altair! Obrigado pelo feedback e fico satisfeito de ver que você pode tirar proveito! Você fez esta ótima observação por causa da linha:

```
$celular = (isset($_POST["celular"]) && $_POST["celular"] != null) ? $_POST["celular"] : "";
```

Muito boa sua consideração, porque de valor NULL causa muita confusão para todos iniciantes em SQL, visto que NULL não é a mesma coisa que uma string vazia "". E este é até o caso de estar fazendo uma correção no artigo. Neste caso você pode alterar tranquilamente os "" por NULL e vai funcionar:

```
$celular = (isset($_POST["celular"]) && $_POST["celular"] != null) ? $_POST["celular"] : NULL;
```

Além disso, adicione no começo do script algumas linhas (Abaixo em negrito) para evitar o erro de falta de variável:

```
// Verificar se foi enviando dados via POST
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    $id = (isset($_POST["id"]) && $_POST["id"] != null) ? $_POST["id"] : "";
    $nome = (isset($_POST["nome"]) && $_POST["nome"] != null) ? $_POST["nome"] : "";
    $email = (isset($_POST["email"]) && $_POST["email"] != null) ? $_POST["email"] : "";
    $celular = (isset($_POST["celular"]) && $_POST["celular"] != null) ? $_POST["celular"] : NULL;
} else if (!isset($id)) {
    // Se não se não foi setado nenhum valor para variável $id
    $id = (isset($_GET["id"]) && $_GET["id"] != null) ? $_GET["id"] : "";
    $nome = NULL;
    $email = NULL;
    $celular = NULL;
}
```

Abraço

[Responder](#)

**Rogério Garcia Filho** 17 de outubro de 2017 às 13:02

Alexandre, quero agradecer por este trabalho muito bem explicado e direcionado a um objetivo concreto. Como citou o Andrei estou buscando orientação para criar um framework que me sirva para meus projetos, onde possa entender e resolver de imediato, desta forma uma oportunidade de nos ajudar com um sistema MVC, PHPOO e PDO.

[Responder](#)**Alexandre Bezerra Barbosa** 18 de outubro de 2017 às 00:12

Rogério, muito obrigado pelo seu feedback e fico muito feliz de saber que está ajudando. Então segue pra você em primeira mão o meu novo artigo [PHP::MVC-CRUD COMPLETO](#). Acredito que este vai dar uma grande força no seus estudos. Este artigo foi desenvolvido exatamente na estrutura MVC, POO e PDO! Caso encontre falhas na escrita ou se torne muito complexo de entender, não deixe de dar seu feedback! Se você ainda não estiver me seguindo no blog, lhe incentivo, pois futuramente colocarei mais exemplos. Abraço!

[Responder](#)**Diego Albuquerque** 19 de outubro de 2017 às 14:39

Alexandre, boa tarde! Muito bom o seu artigo! Aprendi muito!! Parabéns!!

No final do meu projeto eu tive um pequeno problema, depois de inserir o contato, quando clico em alterar ele mantém o contato atual e cria um novo.

É isso msm ou era para ele de fato ele substituir o antigo?

PS.: Já comecei a ler seu novo artigo!!!! hehehe

Desde já muito obrigado!!

Um abraço

[Responder](#)**Alexandre Bezerra Barbosa** 19 de outubro de 2017 às 16:53

Olá Diego, obrigado pelo seu feedback! Espero que possa evoluir bastante, há outros artigos sobre Persistência que poderão te dar uma visão bem ampla! Sobre a sua dúvida, entenda que a decisão no projeto está sendo feita em cima do id, então se for passado um id ele decide que é um update, caso contrário será um insert. Verifique estas linhas:

```
1 if ($id != "") {  
2     $stmt = $conexao->prepare("UPDATE contatos SET nome=?, email=?, celular=? WHERE id = ?");  
3     $stmt->bindParam(4, $id);  
4 } else {  
5     $stmt = $conexao->prepare("INSERT INTO contatos (nome, email, celular) VALUES (?, ?, ?)");  
6 }
```

Acredito que corrigindo isso, irá funcionar perfeitamente! Abraço!

[Responder](#)**Fabio Ozuna** 26 de outubro de 2017 às 21:09

Olá Alexandre, parabéns pelo excelente material. Estou em busca de aprendizado e vou testar seu material pois sempre me perco em trabalhar com várias tabelas, ou módulos, preciso criar um app de cadastro de suportes para atendimento e vi que esse tutorial vai me ajudar muito.

[Responder](#)



Diones Diego 28 de outubro de 2017 às 00:59

Olá Alexandre, Primeiramente quero lhe parabenizar pelo trabalho. funcionou direitinho aqui pra mim e acabei aprendendo muito sobre o pdo com o seu post. mas mesmo assim ainda fiquei com algumas duvidas referente alguns pontos. você por acaso teria algum canal no youtube com video aulas explicando os conceitos do PDO? estudei o outro artigo sobre MVC com PDO e POO mas no fim das contas a aplicação não funcionou na minha maquina. Segui todos os passos exatamente como vc explica no tutorial mas não consegui rodar a aplicação. li 3 vezes e escrevi o script de todas as camadas do MVC mas sem sucesso. gostaria muito de ver algum video explicativo para dominar melhor o conteúdo. inclusive estou estudando profundamente POO. aproveito pra dizer que estou ansioso pelo desenvolvimento do framework que prometeu desenvolver em breve. de qualquer forma deixa minha gratidão pelo esforço. forte abraço e sucesso.

[Responder](#)



Alexandre Bezerra Barbosa 28 de outubro de 2017 às 19:13

Olá Diones, fico feliz de saber que você pode tirar proveito do artigo sobre PDO e isso de fato mostra que o artigo alcançou seu objetivo. No meu blog na página "Sobre", têm o meu e-mail. Envie um print de tela com as mensagens de erro, ou dê algum exemplo sobre o que você não conseguiu fazer. Antes, aproveito para dizer e até acredito que vou ter de adicionar essa informação no artigo, é que a versão foi toda desenvolvida para PHP 7 e não funcionará no PHP 5, a menos que, seja feito alterações, por exemplo, referente a imposição do tipo de retorno, a imposição dos tipos aceitos em alguns métodos, etc. De toda forma, não deixe de enviar suas dúvidas pois isso contribui muito para que o artigo se torne melhor, visto que posso re-avaliar a didática utilizada. Ainda não tenho um canal, mas já está em projeto! Abraço!

[Responder](#)

Pingback: [PHP:: MVC – CRUD COMPLETO | Alexandre Bezerra Barbosa](#)

Deixe um comentário

Insira seu comentário aqui...

Pesquisar ...

Tópicos recentes

[PHP:: PDO – CONEXÃO CONFORME PADRÕES SINGLETON E FACTORY](#)

[PHP:: MVC – CRUD COMPLETO](#)

[LINUX::SUPORTE A REDE – SINCRONIZAR HORÁRIO COM NTP – DEBIAN E DERIVADOS](#)

[WINDOWS::SUPORTE A REDES – SINCRONIZAR HORARIO COM NTP](#)

[LINUX:: CONFIGURAR DNS PARA INTERFACE DE REDE PELO TERMINAL](#)

Comentários



[Alexandre Bezerra Ba...](#) em [PHP:: MVC – CRUD COMPLET...](#)



Lauany Reis da Silva em [PHP:: MVC – CRUD COMPLET...](#)



[Alexandre Bezerra Ba...](#) em [PHP:: MVC – CRUD COMPLET...](#)



Harlei Ribeiro Miran... em [PHP:: MVC – CRUD COMPLET...](#)



Carlos Jose em [Tipo de dados String em PHP ...](#)