

### **Observação sobre CRUD anterior**

Criamos na última aula nosso primeiro CRUD, processo de criar, ler, atualizar e deletar. Se vocês quiserem uma lista de contado ou um sistema de gerenciamento de produtos você já tem com base nesse sistema da aula anterior, com a diferença de invés de nome, email, vocês terão nome do produto, preço e quantidade em estoque, enfim coisas desse tipo, o princípio é basicamente igual a esse sistema da Atividade 17. Com tudo o código que digitamos foi essencial para desenvolver e entenderem o processo. Agora vamos passar para um nível a mais, ou seja, nós não usamos orientação a objeto nessa fase anterior nosso sistema. Por exemplo, não criamos uma classe para usuário, pra adicionar o usuário com essa classe, editar, pegar dados com essa classe. Enfim, não fizemos esse procedimento o que vamos fazer a partir de agora, vamos jogar esse sistema para um próximo nível.

### **Conceito Data Access Object (DAO)**

O conceito DAO veio para melhorar, por exemplo, o que fizemos na aula passada. Vamos ver na prática como funciona, qual é a utilidade desse procedimento. Como viram em aula passada, utilizamos classes para eventualmente criar objetos, que vamos utilizar esses objetos no sistema. No sistema da aula passada fizemos um CRUD de usuários. Então poderíamos criar uma classe específica que vai cuidar de todos os aspectos relacionados ao usuário. Então, teríamos a classe usuário que teria as propriedades que o usuário tem, bem como alguns métodos específicos.

#### **CLASSE USUARIO**

propriedades:

- id
- nome
- email

métodos:

- getid
- setid
  
- getnome
- setnome
  
- getemail
- setemail

Então teríamos todas essas propriedades e métodos. Sempre que mexermos com os dados especificamente de usuários, sempre os utilizaria através dessa classe. Dessa forma a gente sempre consegue manipular o que quisermos, responsável no caso de usuário, tudo seria feito por

essa classe do usuário. Tudo passaria por essa classe de usuário, tudo que mexer com dados de usuários passaria por essa classe. Só o ato de fazer isso, não é necessariamente o conceito de Data Access Object (DAO) ela expande um pouco mais, por exemplo, para implementar o DAO a gente vai precisar de criar pelo menos duas classes. O conceito tem, com tudo a aplicação prática ela pode variar de programador para programador, de linguagem para linguagem. Até na própria linguagem tem algumas formas distintas de fazer a aplicação desse conceito que não é uma coisa rígida digamos assim, focada, mas desde que siga o conceito geral do DAO a gente pode considerar uma determinada implementação X, como implementação DAO.

Na prática a gente tem duas classe funcionando. Uma classe que vai ser o próprio item mesmo, ou seja, usuário e vamos ter uma outra classe que podemos chamar classe USUARIODAO. Essa classe vai ter o CRUD, onde vamos ter então a manipulação de banco de dados. E vai fazer com auxílio da classe USUARIO. Por exemplo. na classe USUARIODAO vamos ter método para adicionar, que poderia ser addAdicionar, isso o programador que escolhe, só um exemplo e teria parâmetro uma classe.

## CLASSE USUARIODAO

método para adicionar

- add(Usuario)

Uma maneira incorreta de usar parâmetros é da forma posta na linha a seguir. Por exemplo, quero adicionar o usuário no banco de dados, então vou mandar o nome, email, como parâmetros, tem gente que faz, mas não é a forma correta de utilizar na metodologia DAO.

- add(nome, email)

O que temos que fazer? Sempre que manusear nome, email, id, ou qualquer outra coisa relacionada ao usuário tem que usar quem? Usar a CLASSE USUARIO

Então o que se faria para adicionar um usuario?

## PASSO 1 PARA ADD USUARIO NOVO:

- Cria a classe do novo usuario. Obviamente sem o id, e adicionaria o nome e o email dele por exemplo:

```
$usuario = new Usuario();
```

```
$usuario->setNome('fulano');
```

```
$usuario->setEmail('fulano@gmail.com');
```

Com isso criei um objeto do meu usuario.

Agora mando esse meu objeto Usuario() para dentro do meu USUARIODAO

tipo assim:

```
add(Usuario)
```

Então fariamos algo assim:

```
$usuarioDAO->add($usuario);
```

Então criamos o \$usuario, objeto e jogamos esse objeto para adicionar no banco de dados, \$usuarioDAO->add(\$usuario). Então dentro do USUARIODAO, o método add() adicionaria no banco de dados. Aí podemos fazer o procedimento que quiser no banco de dados.

Agora devem estar se perguntando pra que complicar assim? Por um motivo simples, deixamos a responsabilidade de manipulação de dados relacionado ao usuário concentrada nessa classe USUARIO, sempre que lidar, mexer com usuário, quer seja pegar nome, seta um email, fazer qualquer outro tipo de manipulação com o usuário nessa classe específica dele, para tratar do usuário.

Para concluir o conceito geral do DAO, vocês tem que terem uma classe específica para o item que está manipulando, no caso do exemplo é CLASSE USUARIO e tem que ter outra classe específica para fazer a intermediação entre o banco de dados e a CLASSE USUARIO conforme nosso exemplo.

Então quero adicionar um usuário:

- criamos o objeto do usuario a ser adicionado
- mando o objeto para o intermediário
- o intermediário faz a comunicação com o banco de dados

E esse intermediário pode ser para qualquer intermediário que quiser para um banco de dados como MySQL, ORACLE, PostGreSQL. Dessa forma separamos a comunicação do banco de dados dá aplicação como um todo, o que facilita muito na hora de manutenção do código num sistema.

Por exemplo no sistema da Atividade 17, se quisermos usar outro banco de dados nesse sistema, praticamente temos que refazer todo o sistema novamente, e utilizando a implementação DAO isso não será preciso refazer. Com a implementação DAO podemos mudar totalmente a estrutura de banco de dados que vamos manter o mesmo sistema e vamos alterar um arquivo ou outro.

Deixo a seguir essa explicação feita anteriormente de forma reunida:

## CLASSE USUARIO

propriedades:

- id
- nome
- email

métodos:

- getid

- setid

- getnome

- setnome

- getemail

- setemail

## CLASSE USUARIODAO

- add(Usuario)

## PASSO 1 PARA ADD USUARIO NOVO:

```
$usuario = new Usuario();
```

```
$usuario->setNome('fulano');
```

```
$usuario->setEmail('fulano@gmail.com');
```

```
$usuarioDAO->add($usuario);
```

A partir de agora, vamos para aplicação prática passo a passo da implementação **DAO**. Vamos transformar o sistema da Atividade 17, permanecendo sistema CRUD, mas com funcionamento DAO.

## **Criando um DAO na prática**

Primeira coisa que vamos fazer é criar uma pasta nova chamada models, ou seja, modelos. E dentro dessa pasta vamos criar um arquivo chamado Usuario.php. Por que Usuario com a letra maiúscula? Para ficar no padrão, nesse arquivo vamos criar uma classe Usuario, para isso tenho ela com o mesmo nome. Essa classe Usuario tem que ter a mesma estrutura da tabela usuarios do banco de dados test.

Usuario.php

```
<?php
```

```
class Usuario{
```

```
    //os 3 dados que todo usuario vai ter que ter:
```

```
    private $id;
```

```
    private $nome;
```

```
    private $email;
```

```
    //agora o get e set de cada um deles:
```

```
    //get e set do id:
```

```
    public function getId(){
```

```
        return $this->id; //dá um return em id
```

```
    }
```

```
    public function setId($i){
```

\$this->id = trim(\$i); //recebe e joga no id, trim() caso venha no futuro não só receber números mas textos...

//trim() vai tirar eventuais espaços de um lado ou de outro... é uma boa prática colocar o trim...

```
    }
```

```
    //get e set do nome:
```

```
    public function getNome(){
```

```
        return $this->nome;
```

```
    }
```

```
    public function setNome($n){
```

\$this->nome = ucwords(trim(\$n)); //ucwords() para a primeira letra do nome ficar em maiúscula

```
    }
```

```

//get e set do email:

public function getEmail(){
    return $this->email;
}

public function setEmail($e){
    $this->email = strtolower(trim($e)); //strtolower joga todo email para minúsculo
}

}

?>

```

Acabamos de criar a classe de Usuario, essa classe que vai ser utilizada sempre que tiver manipulando algum usuario. Agora vamos para o segundo passo que é implementar o DAO nesse mesmo arquivo Usuario.php. Que é a segunda classe que vai manipular o banco de dados. Vamos padronizar uma implementação, interface do usuario, ou seja, toda classe que quiser fazer implementação de banco de dados dessa classe Usuario, ou seja, todo DAO de Usuario terá que seguir determinado padrão. Então vamos agora fazer essa interface logo abaixo da classe Usuario no arquivo Usuario.php, que terá todos os métodos que toda implementação precisa ter:

//Essa interface é o CRUD na verdade

```

interface UsuarioDAO{

    public function add(Usuario $u); //add recebe um objeto da class Usuario

    public function findAll();

//findAll(), ou seja, pegue todo mundo, vai retornar vários objetos da classe Usuario, de todos usuarios que achar.

    public function findById($id) // encontrar um usuario só

    public function update(Usuario $u);

    public function delete($id);

}

```

Acabamos de criar a interface DAO para quem quiser usar o DAO de Usuario.

**A seguir coloco o código completo do arquivo Usuario.php:**

```
<?php
```

```
class Usuario{
```

```
    //os 3 dados que todo usuario vai ter que ter:
```

```
    private $id;
```

```
    private $nome;
```

```
    private $email;
```

```
    //agora o get e set de cada um deles:
```

```
    //get e set do id:
```

```
    public function getId(){
```

```
        return $this->id; //dá um return em id
```

```
    }
```

```
    public function setId($i){
```

\$this->id = trim(\$i); //recebe e joga no id, trim() caso venha no futuro não só receber números mas textos...

//trim() vai tirar eventuais espaços de um lado ou de outro... é uma boa prática colocar o trim...

```
    }
```

```
    //get e set do nome:
```

```
    public function getNome(){
```

```
        return $this->nome;
```

```
    }
```

```
    public function setNome($n){
```

```
$this->nome = ucwords(trim($n)); //ucwords() para a primeira letra do nome ficar em maiúscula
```

```
}
```

```
//get e set do email:
```

```
public function getEmail(){
```

```
    return $this->email;
```

```
}
```

```
public function setEmail($e){
```

```
    $this->email = strtolower(trim($e)); //strtolower joga todo email para minúsculo
```

```
}
```

```
}
```

```
//Essa interface é o CRUD na verdade
```

```
interface UsuarioDAO{
```

```
    public function add(Usuario $u); //add recebe um objeto da class Usuario
```

```
    public function findAll(); //findAll(), ou seja, pegue todo mundo, vai retornar vários objetos da classe Usuario, de todos usuarios que achar
```

```
    public function findById($id) // encontrar um usuario só
```

```
    public function update(Usuario $u);
```

```
    public function delete($id);
```

```
}
```

```
?>
```

Agora o próximo passo é criar a implementação do Usuario DAO para MySQL. Para isso não necessariamente vamos criar uma nova pasta chamada **dao**, para ficar mais organizado. Se quiser pode deixar esse arquivo que vamos criar, dentro da pasta models também, sem problemas, mas sempre é bom ter de uma forma separada em termos de organização. Depois de criada a pasta **dao** vamos criar um arquivo dentro dela chamado UsuarioDAOMySQL.php

```
<?php
```



//puxa a classe Usuario e interface

require\_once 'models/Usuario.php'; //para não repetir(puxar) a classe Usuario, ai ele vai dar aviso que já foi puxada

//Implementação, criamos o DAO para MySQL implementando UsuarioDAO

class UsuarioDaoMysql implements UsuarioDAO{

    //Precisamos de um PDO, e vem de fora através de injeção de dependência

    private \$pdo;

    //qdo eu estanciar minha class UsuarioDaoMysql, vou precisar mandar tbem do cara que está fazendo a manipulação

    //que é o próprio PDO então:

    public function \_\_construct(PDO \$driver){ //recebe uma classe do tipo PDO chamada \$driver, tbem podem ser outro nome...

        \$this->pdo = \$driver;

    }

    /\* Observem que agora vamos fazer a implementação de cada um desse a seguir. Copiei eles da interface UsuarioDAO

    para verem que são esses, que vamos trabalhar um por um a seguir na implementação.

    public function add(Usuario \$u);

    public function findAll();

    public function findById(\$id)

    public function update(Usuario \$u);

    public function delete(\$id);

    \*/

    //Implementação:

    public function add(Usuario \$u){

    }

```
public function findAll(){
```

```
}
```

```
public function findById($id){
```

```
}
```

```
public function update(Usuario $u){
```

```
}
```

```
public function delete($id){
```

```
}
```

```
}
```

```
?>
```

Feito esse arquivo UsuarioDaoMysql.php vamos agora começar a implementar lá no sistema do banco de dados test. Começaremos pela página index.php. Aberto a página index.php vamos fazer uma substituição de dados nessa página:

```
index.php
1  <?php
2  require 'config.php';
3
4  $lista = []; //array começa vazio
5  $sql = $pdo->query("SELECT * FROM usuarios");
6  if($sql->rowCount() > 0){
7      $lista = $sql->fetchAll(PDO::FETCH_ASSOC);
8  }
```

Com a página aberta, observem que a primeira linha foi puxado o config.php e abaixo dessa linha, na linha a seguir vamos puxar o UsuarioDAO colocando:

```
require 'dao/UsuarioDaoMysql.php';
```

Feito isso, a seguir vamos instanciar a classe `UsuarioDaoMysql` chamando o PDO:

```
$usuarioDao = new UsuarioDaoMysql($pdo);
```

Feito isso, o próximo passo é pegar a lista de usuario.

```
index.php
1  <?php
2  require 'config.php';
3  require 'dao/UsuarioDaoMysql.php';
4
5  $usuarioDao = new UsuarioDaoMysql($pdo);
6
7  $lista = []; //array começa vazio
8  $sql = $pdo->query("SELECT * FROM usuarios");
9  if($sql->rowCount() > 0){
10 |     $lista = $sql->fetchAll(PDO::FETCH_ASSOC);
11 | }
12
```

Isso quer dizer que não vamos mais precisar da lista de códigos da linha 7 até a linha 11 conforme mostra figura acima. Ou seja, vamos substituir por uma linha de código:

```
$lista = $usuarioDao->findAll();
```

O `findAll()` é public function `findAll(){ }` do arquivo `UsuarioDaoMysql.php`, que é para pegar todos os usuarios.

Vai retornar um array com todos os itens, e todos os itens serão objetos da classe `Usuario`.

```
index.php
1  <?php
2  require 'config.php';
3  require 'dao/UsuarioDaoMysql.php';
4
5  $usuarioDao = new UsuarioDaoMysql($pdo);
6
7  $lista = $usuarioDao->findAll();
8
9
10 ?>
11
12 <html>
13 | <head>
```

Agora precisamos mudar o usuario conforme mostra a figura a seguir do arquivo index.php vai ser um objeto mesmo da classe Usuario, por isso não vamos mais acessar como mostra a figura a seguir:

```
</tr>
<?php foreach($lista as $usuario): ?>
    <tr>
        <!-- <td> <?php echo $usuario['id']; ?> </td> -->
        <td> <?=$usuario['id'];?> </td>
        <td> <?=$usuario['nome'];?> </td>
        <td> <?=$usuario['email'];?> </td>
        <td>
            <a href="editar.php?id=<?=$usuario['id'];?>">[Editar]</a>
            <a href="excluir.php?id=<?=$usuario['id'];?>" onclick="return c
        </td>
    </tr>
<?php endforeach; ?>
```

Então como pego um id nesse objeto?

\$usuario->getId()

ou seja:

<?=\$usuario->getId();?>

Da mesma forma nome, email.

Então vai retornar um array com vários objetos tipo usuario. Veja essa implementação no código:

```
index.php
22     <th>EMAIL</th>
23     <th>AÇÕES</th>
24 </tr>
25 <?php foreach($lista as $usuario): ?>
26     <tr>
27         <td><?=$usuario->getId();?></td>
28         <td><?=$usuario->getNome();?></td>
29         <td><?=$usuario->getEmail();?></td>
30         <td>
31             <a href="editar.php?id=<?=$usuario->getId();?>">[Editar]</a>
32             <a href="excluir.php?id=<?=$usuario->getId();?>" onclick="return
33         </td>
34     </tr>
35 <?php endforeach; ?>
```

Se executarmos esse código vai dar erro, por que não colocamos para retornar nada ainda no `findAll()`.

```
//Implementação:
public function add(Usuario $u){

}

public function findAll(){
}
```

Vamos fazer para retornar um array:

```
public function findAll(){
    $array = [];

    return $array;
}
```

Só fazendo isso já não vai mais dar erro ao executar o `index.php` observem:



Adicionar Usuários

ID	NOME	EMAIL	AÇÕES
----	------	-------	-------

Vamos agora terminar de fazer a implementação no `findAll()`:

Primeiro vamos fazer o próprio PDO para fazer a consulta

Segundo se teve algum item

Terceiro se teve algum item, armazena na variável `$data`

Feito isso, usamos um `foreach()` nos dados que recebemos ( `$data = $sql->fetchAll();` ) para poder criar os nossos objetos.

E dentro do `foreach`, criamos um objeto, preenchemos ele e jogamos no array e assim sucessivamente. Ou seja preenchemos os objetos com os dados que vem do banco de dados. E a seguir pegamos o array e jogamos o objeto dentro dele. Acabamos de fazer a implementação do `findAll()`.

Agora o `findAll()` vai pegar todos os usuários do banco de dados e transformar em objetos do tipo `usuarios` mesmo e retornar esses objetos. Veja como ficou o código:

```

dao > UsuarioDaoMysql.php
22
23 //Implementação:
24 public function add(Usuario $u){
25
26 }
27
28 public function findAll(){
29     $array = [];
30
31     $sql = $this->pdo->query("SELECT * FROM usuarios");
32     if($sql->rowCount() > 0){
33         $data = $sql->fetchAll();
34         foreach($data as $item){
35             $u = new Usuario();
36             $u->setId($item['id']);
37             $u->setNome($item['nome']);
38             $u->setEmail($item['email']);
39         }
40     }
41     return $array;
42 }

```

Então na implementação no arquivo index.php na linha:

```
$lista = $usuarioDao->findAll();
```

estamos dando um findAll(), vai estar retornando um array, vazio ou com os objetos. E nas linhas de código a seguir, está **usando** esses objetos nos códigos conforme mostra a figura a seguir:

```

index.php
22 <th>EMAIL</th>
23 <th>AÇÕES</th>
24 </tr>
25 <?php foreach($lista as $usuario): ?>
26     <tr>
27         <td><?=$usuario->getId();?></td>
28         <td><?=$usuario->getNome();?></td>
29         <td><?=$usuario->getEmail();?></td>
30         <td>
31             <a href="editar.php?id=<?=$usuario->getId();?>">[Editar]</a>
32             <a href="excluir.php?id=<?=$usuario->getId();?>" onclick="return
33         </td>
34     </tr>
35 <?php endforeach; ?>

```

Percebam que no arquivo index.php estamos puxando os dados do banco de dados e apresentando esses dados, isso somente que é feito no arquivo. Tudo sempre com a classe Usuario. Fizemos o nosso **read** dos dados do nosso CRUD.

Agora vamos partir para a implementação do nosso botão adicionar usuário:



The screenshot shows a web browser at localhost/php/index.php. Below the address bar is a link "Adicionar Usuários". Below that is a table with the following data:

ID	NOME	EMAIL	AÇÕES
1	Sandra Hedler De Amorim	profisandraprotasio@gmail.com	<a href="#">[Editar]</a> <a href="#">[Excluir]</a>

Vamos abrir o arquivo adicionar\_action.php que é onde vamos fazer manipulações específicas.

Primeira coisa que temos é puxando o config.php e puxar o dao.

```
adicionar_action.php
1  <?php
2  //conexão com o banco de dados
3  require 'config.php';
4  require 'dao/UsuarioDaoMysql.php';
5
6  $usuarioDao = new UsuarioDaoMysql($pdo);
7
```

Essas três linhas específicas que temos ter no início do nosso código do arquivo. Ou seja, puxo o config.php, puxo o dao e instancio a classe do dao.

Agora observem no trecho de código a seguir, que está buscando um usuário a partir de um e-mail.

```
if($name && $email){
    //validação do e-mail:
    $sql = $pdo->prepare("SELECT * FROM usuarios WHERE email = :email");
    $sql->bindValue(':email', $email);
    $sql->execute();
}
```

Isso significa que precisamos de um findByEmail(\$email). Então vamos no arquivo Usuario.php e então vamos na nossa classe Usuario mais especificamente no dao e acrescentamos:

```
public function findByEmail($email);
```

Vai retornar o email do objeto achado. Ficando assim o código:

```

    }
    //Essa interface é o CRUD na verdade
    interface UsuarioDAO{
        public function add(Usuario $u); //add recebe um objeto
        public function findAll(); //findAll(), ou seja, pegue
        public function findByEmail($email);
        public function findById($id); // encontrar um usuario
        public function update(Usuario $u);
        public function delete($id);
    }

```

O próximo passo vamos no arquivo UsuarioDaoMysql.php e fizemos a implementação da busca pelo email.

```

        return $array;
    }
    //implementação email
    public function findByEmail($email){
    }

    public function findById($id){
    }
}

```

Próximo passo vamos no arquivo adicionar\_action.php e adicionar o findByEmail(\$email). Achou insere não achou insere. Quer dizer que todo esse bloco de código que vamos implementar.



```

adicionar_action.php
12
13 if($name && $email){
14     //implementação do findByEmail($email)
15     |
16     //validação do e-mail:
17     $sql = $pdo->prepare("SELECT * FROM usuarios WHERE email = :email"); //sempre qdo
18     $sql->bindValue(':email', $email);
19     $sql->execute();
20
21     if($sql->rowCount() === 0){
22         //query de inserção
23         $sql = $pdo->prepare("INSERT INTO usuarios (nome, email) VALUE (:name, :email)");
24         $sql->bindValue(':name', $name);
25         $sql->bindValue(':email', $email);
26         $sql->execute();
27         header("Location: index.php");
28         exit;
29
30     } else {
31         header("Location: adicionar.php");
32         exit;
33     }
}

```

A seguir como ficou com a implementação do findByEmail(\$email):

```

if($name && $email){
    //implementação do findByEmail($email)
    if($usuarioDao->findByEmail($email) === false){
        //Se nao achou adiciona
        $novoUsuario = new Usuario();
        $novoUsuario->setNome($name);
        $novoUsuario->setEmail($email);

        //Mando o Dao, adiciona
        $usuarioDao->add($novoUsuario);
        //Rediciona
        header("Location: index.php");
        exit;
    }else {
        header("Location: adicionar.php");
        exit;
    }
} else {
    header("Location: adicionar.php");
    exit;
}
}

```

O próximo passo agora vamos no arquivo UsuarioDaoMysql.php e vamos adicionar, no findByEmail(\$email) e no add(Usuario \$u) que até aqui não tem nada neles:

```
UsuarioDaoMysql.php
}
//implementação email
public function findByEmail($email){
    $sql = $this->pdo->prepare("SELECT * FROM usuarios WHERE email = :email");
    $sql->bindValue(':email', $email);
    $sql->execute();
    //Verifica se tem usuario c esse email, vai montar esse objeto agora:
    if($sql->rowCount() > 0){
        $data = $sql->fetch();
        $u = new Usuario();
        $u->setId($data['id']);
        $u->setNome($data['nome']);
        $u->setEmail($data['email']);
        return $u; //retorna o próprio objeto
    } else {
        return false;
    }
}
```

Feita a implementação do findByEmail(\$email) vamos agora fazer a add().

```
public function add(Usuario $u){ }
```

O add() recebe o usuario e faz uma adição no banco de dados.

```
//Implementação:
public function add(Usuario $u){
    $sql = $this->pdo->prepare("INSERT INTO usuarios(nome, email) value (:nome, :email)");
    $sql->bindValue(':nome', $u->getNome());
    $sql->bindValue(':email', $u->getEmail());
    $sql->execute();
    // já adicionamos o objeto, e agora pegamos o id e jogamos dentro desse objeto:
    $u->setId( $this->pdo->lastInsertId()); //método do pdo, pega o último id adicionado
    return $u;
}
```

Feita a implementação na nossa public function add(Usuario \$u){ } agora é só testar. Acabamos de fazer o processo de CREATE do nosso CRUD.

Vamos agora para o processo de **edição** do usuário no banco de dados, nosso UPDATE do CRUD.

Nesse processo vamos precisar de duas implementações diferentes. Vamos abrir o arquivo editar.php, e o procedimento além de pegar o id do usuario está usando a query para pegar as informações e etc. Esses procedimentos vamos tirar e usar o findById(\$id). Por que, nos temos o

id do usuario e queremos achar o usuario baseado no id desse usuario, por isso o findById(\$id) é perfeito para implementação nesse arquivo editar.php.

Com a página aberta, observem que a primeira linha foi puxado o config.php e abaixo dessa linha, na linha a seguir vamos puxar o UsuarioDAO colocando:

```
require 'config.php';
```

```
require 'dao/UsuarioDaoMysql.php';
```

Feito isso, a seguir vamos instanciar a classe UsuarioDaoMysql chamando o PDO:

```
$usuarioDao = new UsuarioDaoMysql($pdo);
```

Feito isso, o próximo passo é pegar o id e verifica se o id foi enviado. O próximo passo é pegar o info, que usamos para pegar os procedimentos \$info['id']..., que vai receber um objeto da classe Usuario, ou caso não ache esse objeto vai ser um false.

Se caso usuário igual a false e não achar o id, não entra no if e manda pro index.php. Se mandar um id que não existe, vai entrar no if, vai procurar e não vai achar, vai retornar um false, ou seja, o usuario vai ficar false. Vai no próximo if e usuario false, manda pro index.php.



```
editar.php
1  <?php
2      require 'config.php';
3      require 'dao/UsuarioDaoMysql.php';
4      $usuarioDao = new UsuarioDaoMysql($pdo);
5
6
7      $usuario = false; //inicialmente esse usuario vai ser false
8      $id = filter_input(INPUT_GET, 'id');
9      if($id){
10         //1ª implementação
11         $usuario = $usuarioDao->findById($id);
12     }
13     //2ª verificação
14     if(usuario ===false){ //se usuario não é false
15         header("Location: index.php");
16         exit;
17     }
18
19  ?>
```

Agora a segunda implementação da classe Usuario no arquivo editar.php:

```

21 <html>
22     <head>
23         <meta charset="UTF-8">
24     </head>
25
26 <h2> Editar Usuário </h2>
27
28 <form method="POST" action="editar_action.php">
29
30 <!-- 2º Passo da implementação da classe Usuario: adicionar um value... -->
31     <input type="hidden" name="id" value="<?=$usuario->getId();?>" />
32     <label>
33         Nome: <br/>
34
35         <input type="text" name="name" value="<?=$usuario->getNome();?>" />
36     </label><br/><br/>
37
38     <label>
39         E-mail:<br/>
40         <input type="email" name="email" value="<?=$usuario->getEmail();?>" />
41     </label><br/><br/>
42
43     <input type="submit" value="Editar" />
44 </form>

```

Vamos agora fazer a implementação do `findById($id)` no arquivo `UsuarioDaoMysql.php` que é igual ao `findByEmail($email)`.

```

69 public function findById($id){
70     $sql = $this->pdo->prepare("SELECT * FROM usuarios WHERE id = :id");
71     $sql->bindValue(':id', $id);
72     $sql->execute();
73     //Verifica se tem usuario c esse email, vai montar esse objeto agora:
74     if($sql->rowCount() > 0){
75         $data = $sql->fetch();
76         $u = new Usuario();
77         $u->setId($data['id']);
78         $u->setNome($data['nome']);
79         $u->setEmail($data['email']);
80         return $u; //retorna o próprio objeto
81     } else {
82         return false;
83     }
84 }
85
86

```

Vamos agora abrir o arquivo **editar\_action.php**.

Com a página aberta, observem que a primeira linha foi puxado o config.php e abaixo dessa linha, na linha a seguir vamos puxar o UsuarioDao colocando:

```
require 'config.php';
```

```
require 'dao/UsuarioDaoMysql.php';
```

Feito isso, a seguir vamos instanciar a classe UsuarioDaoMysql chamando o PDO:

```
$usuarioDao = new UsuarioDaoMysql($pdo);
```

Feito isso, o próximo passo é pegar o id, nome e email.

```
editar_action.php
1  <?php
2  //conexão com o banco de dados
3  require 'config.php';
4  require 'dao/UsuarioDaoMysql.php';
5
6  $usuarioDao = new UsuarioDaoMysql($pdo);
7
8  $id = filter_input(INPUT_POST, 'id');
9  $name = filter_input(INPUT_POST, 'name');
10 $email = filter_input(INPUT_POST, 'email', FILTER_VALIDATE_EMAIL);
11
12 if($id && $name && $email){
13
14     //implementação:
15     $usuario = $usuarioDao->findById($id); //pega do bco
16     $usuario->setNome($name); //altera o nome
17     $usuario->setEmail($email); //altera o email
18
19     $usuarioDao->update($usuario); //e mando para update
20
21     header("Location: index.php");
22     exit;
23
24 } else {
25     header("Location: editar.php?id=".$id); //qlquer problema volta p/ editar c id do usuário
26     exit;
27 }
```

Feito as alterações de implementação nesse arquivo, vamos agora no arquivo UsuarioDaoMysql.php e vamos fazer a implementação no update(Usuario \$u). Observem que esse update está recebendo o próprio usuario e agora temos que fazer a query.

```
dao > UsuarioDaoMysql.php
88 public function update(Usuario $u){
89     $sql = $this->pdo->prepare("UPDATE usuarios SET nome = :nome, email = :email WHERE id = :id");
90     $sql->bindValue(':nome', $u->getNome());
91     $sql->bindValue(':email', $u->getEmail());
92     $sql->bindValue(':id', $u->getId());
93     $sql->execute();
94
95     return true;
96
97 }
98
```

Feita a implementação do processo de edição do usuário no banco de dados.

Vamos agora implementar o Dao para o último item que é o DELETE, o ato de excluir. Vamos abrir o arquivo excluir.php.

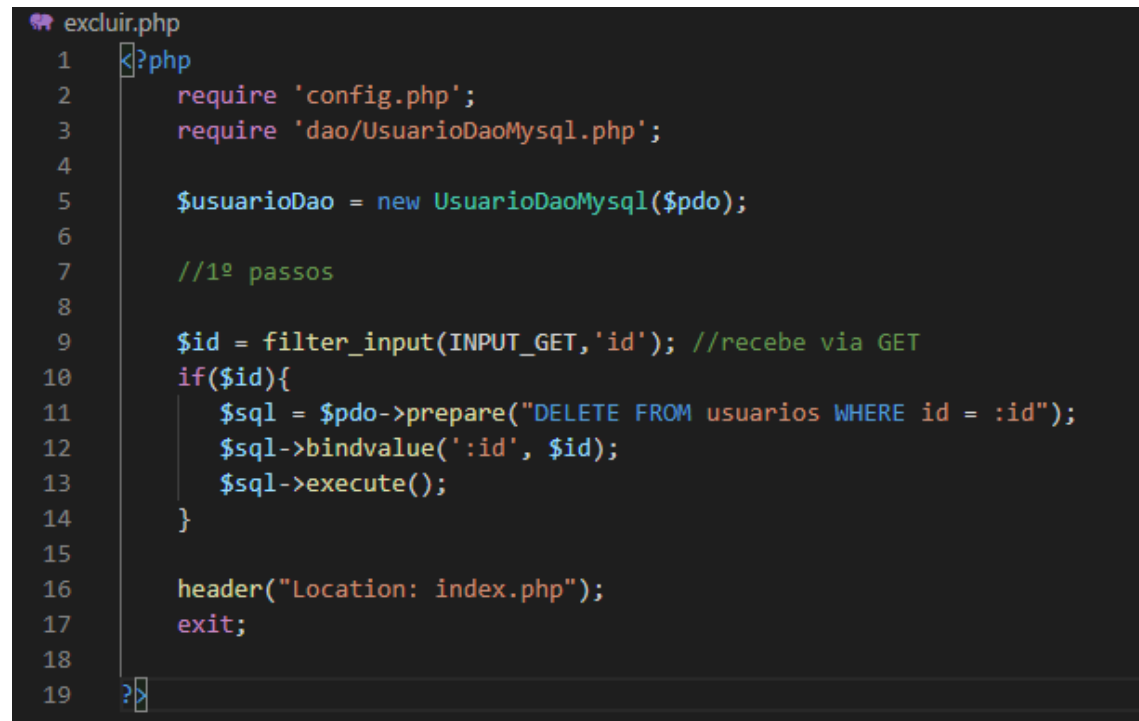
Com a página aberta, observem que a primeira linha foi puxado o config.php e abaixo dessa linha, na linha a seguir vamos puxar o UsuarioDAO colocando:

```
require 'config.php';
```

```
require 'dao/UsuarioDaoMysql.php';
```

Feito isso, a seguir vamos instanciar a classe UsuarioDaoMysql chamando o PDO:

```
$usuarioDao = new UsuarioDaoMysql($pdo);
```



```
excluir.php
1  <?php
2      require 'config.php';
3      require 'dao/UsuarioDaoMysql.php';
4
5      $usuarioDao = new UsuarioDaoMysql($pdo);
6
7      //1º passos
8
9      $id = filter_input(INPUT_GET, 'id'); //recebe via GET
10     if($id){
11         $sql = $pdo->prepare("DELETE FROM usuarios WHERE id = :id");
12         $sql->bindValue(':id', $id);
13         $sql->execute();
14     }
15
16     header("Location: index.php");
17     exit;
18
19  ?>
```

Agora vamos para a implementação:

```

excluir.php
1  <?php
2      require 'config.php';
3      require 'dao/UsuarioDaoMysql.php';
4
5      $usuarioDao = new UsuarioDaoMysql($pdo);
6
7      $id = filter_input(INPUT_GET, 'id'); //recebe via GET
8      if($id){
9          //Implementação:
10         $usuarioDao->delete($id);
11     }
12
13     header("Location: index.php");
14     exit;
15

```

Feito isso, vamos abrir o arquivo UsuarioDaoMysql.php, para a implementação no delete(\$id).

```

99     public function delete($id){
100         $sql = $this->pdo->prepare("DELETE FROM usuarios WHERE id = :id");
101         $sql->bindValue(':id', $id);
102         $sql->execute();
103     }
104

```

Acabamos de fazer a implementação no arquivo UsuarioDaoMysql.php de todos os itens, como add, findAll(), findByEmail(\$email), findById(\$id), update(), delete(\$id). As implementações direto no banco de dados estão nesse arquivo.

No arquivo Usuario.php está a própria classe, a classe principal que trata dos usuários e a interface dos usuarios, a interface UsuarioDAO.

Então que quiser implementar um DAO para a classe Usuario tem que ter os:

add(Usuario \$u)

findAll()

findByEmail(\$email)

findById(\$id)

update(Usuario \$u)

delete(\$id)

Acabamos de fazer a implementação com o DAO como podem ver, está tudo organizada e funcionando.

Se futuramente quiserem mudar o sistema de banco de dados, por exemplo, deixar de usar MySQL e passar a usar Oracle ou PostgreSQL, só vai mudar no arquivo UsuarioDaoMysql.php a implementação da forma que adiciona, a forma como altera e etc. Por exemplo, na classe, na linha `class UsuarioDaoMysql implements UsuarioDAO{ ... }` usando outro banco de dados, para continuarmos usando nosso pdo, podemos fazer assim nessa linha de comando:

`class UsuarioDaopdo implements UsuarioDAO{ ... }` que vai continuar usando o pdo.