TiCs

D

Linguagem PHP com acesso a Banco de Dados

Unidade D
Linguagem de Programação Web

UNIDADE D

LINGUAGEM PHP COM ACESSO A BANCO DE DADOS

PHP com Banco de Dados

Nesta unidade vamos abordar em detalhes o acesso a banco de dados com a linguagem PHP. Hoje é fundamental criarmos sites dinâmicos que facilitem a atualização de conteúdo. Isso só é possível com o armazenamento dos dados em banco de dados. Além disso, a tendência é que as aplicações, como por exemplo, sistemas de gestão empresarial, gestão de pessoas, entre outros, migrem para o ambiente web, ou seja, sejam desenvolvidos para acesso no navegador.

Como já comentamos anteriormente, o PHP possibilita acesso a diversos bancos de dados. Nesta unidade vamos apresentar a integração com o banco de dados MySql, mas de uma forma que fica fácil a utilização de outros bancos de dados.

Inicialmente veremos quais as funções principais para acesso ao banco de dados por meio da linguagem PHP, para depois criarmos um exemplo completo de manutenção de dados e posterior apresentação no site. Quando nos referimos a uma manutenção completa quer dizer ter as funções de incluir, alterar e excluir dados de uma tabela do banco de dados. Então, mãos a obra...

Sites Dinâmicos

Antes de iniciarmos, vamos ver uma situação de uso de banco de dados para que você compreenda como funcionam os sites dinâmicos. Imagine um que site precisa mostrar quais as cidades que o IFSUL tem campus. Sabemos que os Institutos Federais estão em expansão e por isso novos campi podem surgir. Por isso, é interessante termos essa informação em uma base de dados e fornecer uma forma de o administrador do site manter estes dados. Um site dinâmico (com banco de dados) possui o que chamamos de **Área Administrativa**, onde apenas pessoas com permissão podem ter acesso e manter as informações do site. Para acessar esta área o usuário precisa se autenticar informando login e senha. As figuras D.1 e D.2 apresentam um exemplo de área administrativa para manter os dados de cidades (cidades que possuem campi do IFSUL).

A figura D.1 apresenta uma página que lista as cidades (armazenadas em uma tabela do banco de dados). Observe que possui links para incluir uma nova cidade, para alterar os dados de cada cidade e para excluir cada cidade. Quando os links incluir ou alterar forem clicados será aberta uma outra página com um formulário para entrada de dados. A figura D.2 apresenta a página do formulário para incluir dados de uma nova cidade. Este é um exemplo de Área Administrativa para manter dados do banco de dados.

Por outro lado, o site público aos usuários busca os dados deste banco para apresentar as informações aos internautas. A figura D.3 apresenta o que chamamos de **Área Pública** do site, onde qualquer usuário pode visualizar as informações.

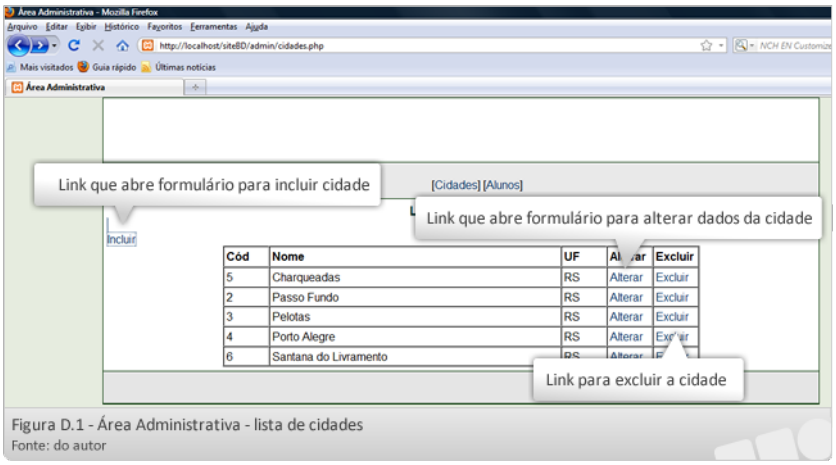

Figura D.1 - Área Administrativa - lista de cidades
Fonte: do autor

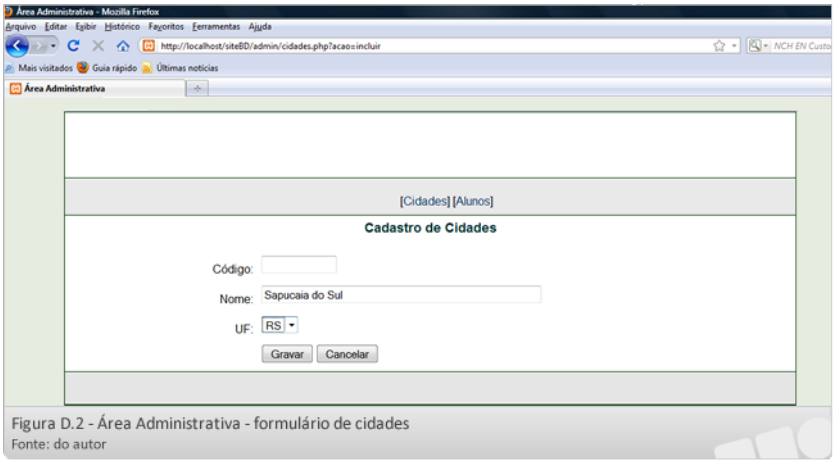

Figura D.2 - Área Administrativa - formulário de cidades
Fonte: do autor

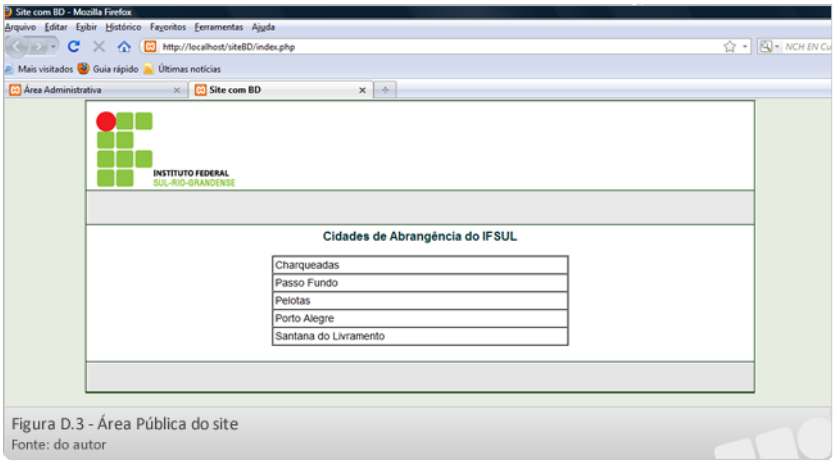

Figura D.3 - Área Pública do site
Fonte: do autor

Configurações iniciais

Antes de começarmos a trabalhar, vamos ver o que vamos precisar para realizar nossos testes.

Biblioteca ADODB para acesso a banco de dados com PHP

Para acesso ao banco de dados MySQL vamos utilizar uma biblioteca chamada ADODB (*Active Data Objects Data Base*). ADODB é conjunto de bibliotecas/classes que padronizam as funções de banco de dados do PHP. O objetivo é ocultar as diferenças existentes entre os diferentes bancos de dados e prover um método simples de fazer consultas e manipulações nas bases de dados com o mínimo de alteração de código. Desta forma, podemos com uma simples alteração mudar o banco de dados que estamos utilizando para outro (por exemplo de MySQL para PostgreSQL, se o esquema do banco de dados for o

mesmo).

Você pode fazer download da biblioteca em

<http://sourceforge.net/projects/adodb/files/>

Com a ADODB podemos acessar os bancos de dados: MySql, Oracle, SQL Server, Sybase, Informix, PostgreSQL, FrontBase, SQLite, Interbase/Firebird, FoxPro, Access, ADO, DB2, SAP DB, ODBC.

Para dar sequência vamos organizar uma estrutura de site para realizarmos nossos testes. Você poderá fazer o download do material disponibilizado que se chama ***siteBD.zip***.

O arquivo deve ser descompactado em C:\xampp\htdocs. A estrutura ficará como da figura D.4.

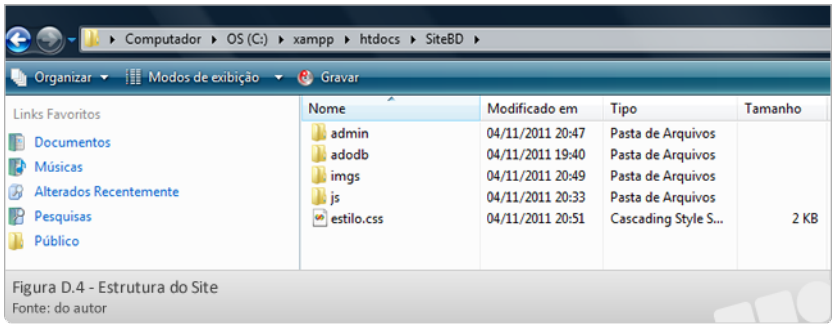

Figura D.4 - Estrutura do Site
Fonte: do autor

Alguns apontamentos sobre a estrutura:

- pasta admin conterá todos os arquivos da Área Administrativa
- pasta adodb é da biblioteca ADODB que vamos usar
- pasta imgs para imagens do site
- pasta js para arquivos de JavaScript do site
- arquivo estilo.css contém as folhas de estilo usadas no site

Atenção:

Esta disciplina não aborda CSS (folha de estilo). Porém você já deve ter visto em outra disciplina que CSS é uma linguagem para estilos que define o layout de documentos (X)HTML. Por exemplo, CSS controla fontes, cores, margens, linhas, alturas, larguras, imagens de fundo, posicionamentos e muito mais.

Banco de Dados MySQL

Agora vamos criar nosso banco de dados para que possamos depois através do PHP acessar o MySQL. Com o servidor web Apache e o banco de dados MySQL rodando, acesse no navegador o endereço:

<http://localhost/phpmyadmin/>

Você estará acessando uma ferramenta para administrar o MySQL, é o phpMyAdmin. Esta ferramenta já foi instalada junto com o Xampp e facilita a criação do banco de dados.

Agora crie o banco de dados chamado "curso".

Neste banco vamos criar, inicialmente, duas tabelas: cidades e alunos, conforme script abaixo:

```
CREATE TABLE IF NOT EXISTS cidades (
  id BIGINT NOT NULL auto_increment ,
```

```
  nome VARCHAR( 50 ) NOT NULL ,
  uf CHAR( 2 ) NOT NULL ,
  PRIMARY KEY ( id )
);


CREATE TABLE IF NOT EXISTS alunos (
  id bigint NOT NULL auto_increment,
  nome varchar(100) NOT NULL,
  email varchar(100) NOT NULL,
  fone varchar(15) NOT NULL,
  nascimento date NOT NULL,
  cidade     int(11) NOT NULL,
  PRIMARY KEY  (id)
);
```

Parada Obrigatória:

Antes de continuar assista o vídeo **Instalação e configuração de ferramentas**, disponível no Ambiente Virtual de Aprendizagem.

Conexão com o banco de dados

Para conectar com o banco de dados vamos usar um método da biblioteca ADODB. Crie um arquivo na pasta admin nomeado de conexao.php . Vejamos o código do conexao.php da figura D.5 e vamos analisá-lo:

Linha 3 – Define uma variável para especificar o caminho da biblioteca ADODB (lembre-se que o *conexao.php* está na pasta *admin*, por isso estamos saindo desta pasta para então entrar em *adodb*).

Linha 4 – Faz include do arquivo *adodb.inc.php* da pasta da biblioteca ADODB. Por convenção os arquivos que são *inc.php* indicam que serão arquivos para include.

Linha 6 – Cria a instância *$con* da conexão com o tipo de banco especificado, neste caso MYSQL. Por exemplo, se o banco para acesso fosse o PostgreSQL, seria passado por parâmetro 'postgres'.

Linha 8 – Desabilita a opção de debug. Esta opção é interessante para o desenvolvedor identificar erros durando a programação. Quando precisar verificar como os comandos estão sendo executados no banco de dados, esta opção pode ficar ligada, atribuindo *true*. Por hora vamos deixar desabilitada (*false*).

Linha 10 - Executa o método *Connect* com os parâmetros de conexão (servidor, usuário, senha, nome_banco). Neste caso estamos usando o usuário padrão "*root*", que inicialmente não possui senha. É interessante criar um usuário com senha específico para cada banco de dados. Após a conexão vem "*or die(....)*", esta função mostra uma mensagem caso ocorra um erro na execução do comando e interrompe o script.

```php
<?php

$caminho_adodb = "../adodb";
include("$caminho_adodb/adodb.inc.php");

$con = ADONewConnection('mysql');

$con->debug = false;/// Debug do SQL

$con->Connect("localhost", "root", "", "curso") or die
("Erro ao conectar o banco de dados");

echo "conectou com o banco de dados curso";

?>
```

Figura D.5 - Código para conexão com o banco de dados
Fonte: do autor

Dica:

A função ***die()*** aborta imediatamente a execução da aplicação. Essa é uma função simples para o tratamento de erros, pois ela encerra a execução do script.

Crie o arquivo *conexao.php* e faça o teste no navegador. Este arquivo de conexão será usado toda vez que precisarmos interagir com o banco de dados, por isso, após fazer os testes e verificar que está conectando, apague a linha 13 do arquivo.

Executar SQL

Toda e qualquer manipulação de dados do banco de dados será feita usando a linguagem SQL. Vamos ver como é o script SQL para incluir um registro na tabela '**cidades**' do nosso banco '**curso**':

```
insert into cidades (nome, uf) values ('Pelotas','RS')
```

Você observou que o campo **id** não aparece no SQL? Isso porque ele foi definido como *auto_increment* quando a tabela foi criada. Com isso, não devemos passar um valor para campo id, ele terá seu valor gerado automaticamente. Este comando SQL cria um novo registro na tabela. Agora vamos ver como fica a execução no PHP com a biblioteca ADODB. A figura D.6 mostra o código.

```php
<?php

    include('conexao.php');

    $sql = "insert into cidades (nome, uf) values ('Pelotas','RS')";

    $rs = $con->Execute($sql);

    if (!$rs) {
        echo  $con->ErrorMsg();
    }else
        echo "Dados incluídos";

?>
```

Figura D.6 - Inserir dados na tabela
Fonte: do autor

Para este teste crie o arquivo *inserir.php* na pasta *c:\xampp\htdocs\siteBD\admin*, como o código acima. Vejamos alguns apontamentos:

Linha 3 – Faz include do arquivo *conexao.php* que possui a conexão com o banco de dados. Veja como é interessante ter este arquivo, pois se precisar, por exemplo, trocar a senha, será necessário alterar

apenas em um único arquivo;

Linha 5 – Cria a variável *$sql* com o comando SQL;

Linha 7 – Executa o SQL (que está na variável *$sql*). O resultado é atribuído para variável $rs (recebe *fal. se* se ocorrer erro ou o retorno do SQL se realizado com sucesso). Veja que *$con* é a variável da conexão que foi criada no *conexao.php*. Mai uma coisinha: *$rs* vem de *result set* (conjunto de resultados).

Linhas 9 e 10 – Testa se ocorreu erro, se *$rs* for *false* (com ! vai se tornar *true*), ou seja, tem erro, mostra a mensagem retornada do banco de dados.

Atenção

Se o código não possuir erros, a execução do arquivo gerará um novo registro na tabela *cidades*, porém se executares o código mais de uma vez, será criada mais uma cidade com o mesmo nome. Por isso, para testar várias vezes, troque o nome da cidade para incluir cidades diferentes. Visualize as cidades usando o phpMyAdmin *<http://localhost/phpmyadmin>.*

Bem, agora precisamos alterar um registro de cidade, como faremos? A lógica é a mesma, o que vai mudar é o comando SQL. O comando abaixo altera o nome da cidade cujo id (identificador) da cidade é 2:

```
update cidades set nome = 'PELOTAS' where id = 2
```

Observe que o nome da cidade ficou entre aspas simples. Isso porque o campo nome no banco é um *varchar*. O mesmo vale para os tipos *char* e *date*. Já campos do tipo numérico, como *int*, *bigint*, *decimal*, não deve-se usar aspas. Outra dica aqui é sempre atribuir o SQL dentro de aspas duplas:

```
$sql = "update cidades set nome = 'PELOTAS' where id = 2 ";
```

Para o processo de alteração dos dados crie o arquivo *alterar.php* (também na pasta *admin*). A figura D.7 mostra o código do *alterar.php*. Veja que a estrutura é a mesma, o que muda é o SQL.

```
1  <?php
2
3      include('conexao.php');
4
5      $sql = "update cidades set nome = 'PELOTAS' where id = 2 ";
6
7      $rs = $con->Execute($sql);
8
9      if (!$rs) {
10         echo  $con->ErrorMsg();
11     }else
12         echo "Dados alterados";
13
14 ?>
```

Figura D.7 - alterar um registro da tabela
Fonte: do autor

Para a exclusão de uma cidade seguimos a mesma lógica. Crie o arquivo *excluir.php* (na pasta admin) com o código apresentado na figura D.8.

```php
<?php

    include('conexao.php');

    $sql = "delete from cidades where id = 1";

    $rs = $con->Execute($sql);

    if (!$rs) {
        echo  $con->ErrorMsg();
    }else
        echo "Cidade excluída";

?>
```

Figura D.8 - excluir um registro da tabela
Fonte: do autor

Listar dados

Já vimos como incluir, alterar e excluir dados. Só falta a listagem de dados. Da mesma forma vamos executar um SQL (select), mas precisaremos tratar os dados retornados, que geralmente são vários registros. Se precisarmos listar todos registros da tabela cidades (todas as cidades), o comando SQL será:

```
select * from cidades order by nome
```

Vamos dar uma olhada no código da figura D.9 (arquivo *listar.php*).

```php
<?php

    include('conexao.php');

    $sql = "select * from cidades order by nome";

    $rs = $con->Execute($sql);

    if (!$rs) {

        print $con->ErrorMsg();

    }else

    while(!$rs->EOF){

        $nome = $rs->fields['nome'];
        $uf   = $rs->fields['uf'];

        echo "$nome - $uf <br/> ";

        $rs->MoveNext();
    }
?>
```

Figura D.9 - código para listar dados
Fonte: do autor

Para este teste crie o arquivo *listar.php* na pasta *c:\xampp\htdocs\siteBD\admin*, com o código acima. Vejamos alguns apontamentos:

Linha 3 – Faz include do arquivo *conexao.php* que possui a conexão com o banco de dados;

Linha 5 – Cria a variável *$sql* com o comando SQL;

Linha 7 – Executa o SQL (que está na variável *$sql*). O resultado é atribuído para variável *$rs* (recebe *false* se ocorrer erro ou o retorno do SQL se realizado com sucesso). *$rs* faz referência a conjunto de resultados, que são vários registros de cidades;

Linhas 9 e 11 – Testa se ocorreu erro, se *$rs* for *false* (com ! vai se tornar *true*), ou seja, tem erro, mostra a mensagem retornada do banco de dados;

Linha 13 – Se não ocorreu erro então vai executar o código para mostrar os dados;

Linha 15 – Para listar os dados vamos precisar de um laço, cada vez que passar pelo laço lista uma cidade. Este laço *while* verifica se possui resultado para listar: *EOF* sigla de *END OF FILE* que significa "fim de arquivo". `$rs->EOF` retorna *true* se for fim do arquivo e *false* em caso contrário. O laço pára a execução quando não possuir mais resultados (chega ao fim dos resultados);

Linhas 17 e 18 – Recupera os dados que deseja mostrar (atribui para as variáveis *$nome* e *$uf*). Para pegar os campos usa `$rs->fields['nome_campo']` onde *nome_campo* é exatamente o nome do campo no banco de dados. No nosso caso queremos mostrar o nome da cidade (*nome*) e a unidade federativa (*uf*).

Linha 20 – Mostra os dados da cidade e quebra linha;

Linha 22 – Para passar ao próximo registro executa `$rs->MoveNext();`

A figura D.10 mostra o resultado do código listar.php no navegador, listando uma cidade por linha.

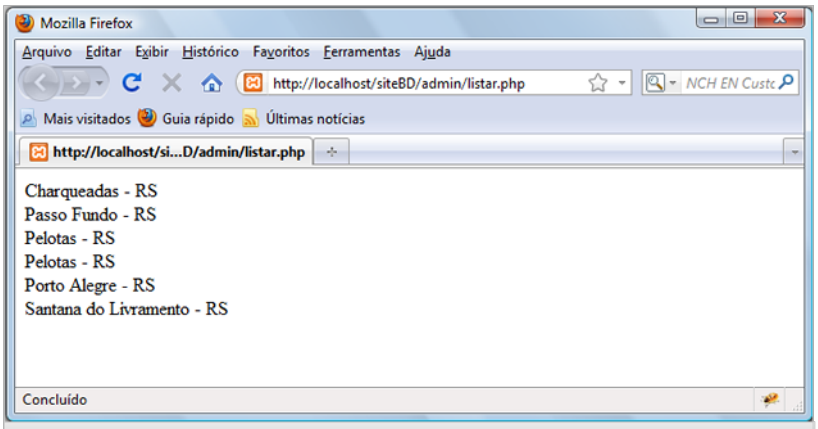

Figura D.10 - Dados de cidades no navegador
Fonte: do autor

Síntese

Bem, para acessar banco de dados com a linguagem PHP utilizamos a biblioteca ADODB. Seu uso é interessante por abstrair aspectos de acesso ao banco, tornando mais fácil a migração de um banco para outro. Vimos como fazer a conexão com o banco e usamos linguagem SQL para manipular os dados com o banco de dados. Principalmente, vimos como tratar dados retornados do banco de dados. Agora precisamos integrar melhor isso com o (X)HTML, assim como mostramos no início desta Unidade nas figura D.1, D.2 e D.3. Antes de seguirmos, é importante que esta parte tenha sido bem assimilada, por isso faça a atividade proposta a seguir.

Atividades - Parte 1

1. Agora é com você. Dado o script abaixo de criação da tabela cursos crie os arquivos:

incluir_cursos.php – para incluir um registro em cursos
alterar_cursos.php – para alterar um registro de cursos
excluir_cursos.php – para excluir um registro de cursos
listar_cursos.php – para listar todos os cursos

```
CREATE TABLE IF NOT EXISTS cursos (
        id BIGINT NOT NULL auto_increment ,
```

```
        nome VARCHAR( 50 ) NOT NULL ,
        horas integer NOT NULL ,
        PRIMARY KEY ( id )
    );
```

Obs.: crie a tabela cursos no mesmo banco de dados já criado (curso)

Após a realização das atividades participe do chat em horário marcado pelo professor formador para discutir questões relativas aos exercícios propostos.

Área Administrativa

Nesta segunda parte da unidade D vamos desenvolver passo a passo uma manutenção completa, incluindo listagem de dados, inclusão, alteração e exclusão (como foi mostrado nas figuras D.1, D.2 e D.3 da primeira parte da Unidade D). Utilizaremos a estrutura já passada do site *siteBD* e o banco de dados *curso*. Este material foi preparado de forma que você acompanhe fazendo cada passo do processo de construção da solução.

Estrutura de arquivos da área administrativa

A pasta *admin* do projeto já possui alguns arquivos. Veja na tabela abaixo uma breve descrição sobre cada arquivo.

Arquivo	Descrição
`acessonegado.php`	Arquivo que mostra mensagem de acesso negado à área administrativa
`conexao.php`	Arquivo para conexão com o banco de dados
`funcoes.php`	Arquivo de funções
`index.php`	Arquivo inicial da área administrativa. Mostra apenas o menu
`login.php`	Arquivo com formulário para login
`logininvalido.php`	Arquivo que mostra mensagem de login inválido para o usuário que está tentando se logar na área administrativa
`menusup.php`	Arquivo com as opções de menu

Atenção:

Para o usuário utilizar a área administrativa é necessário a autenticação informando login e senha. Veremos este processo na última parte desta unidade.

O arquivo *index.php* é o primeiro a ser apresentado (por default) para o usuário na área administrativa. Vamos ver a sua estrutura:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Área Administrativa</title>
<link href="../estilo.css" rel="stylesheet" type="text/css" />
</head>
<body>
<div id="geral">
  <div id="topo"></div>
  <div id="menu">
    <?php include('menusup.php');?>
  </div>
  <div id="conteudo">
  </div>
  <div id="rodape"></div>
</div>
</body>
</html>
```

Figura D.11 - Estrutura do arquivo index.php
Fonte: do autor

No corpo do documento existe uma estrutura de *divs* para organizar o layout e que são formatadas por folha de estilo (arquivo *estilo.css*). Na *div menu* faz o include do arquivo *menusup.php* com as opções de menu (figura D.12). As opções de menu por enquanto são duas, cidades e alunos. Caso seja necessário incluir mais opções basta acrescentar neste arquivo.

```
<ul>
    <li>[<a href="cidades.php">Cidades</a>]</li>
    <li>[<a href="alunos.php">Alunos</a>]</li>
</ul>
```

Figura D.12 - Código do arquivo menusup.php
Fonte: do autor

A figura D.13 mostra a visualização do *index.php* no navegador.

Figura D.13 - Visualização do index.php no navegador
Fonte: do autor

Manutenção completa

Nosso objetivo é mostrar a manutenção completa da tabela de cidades (listar, incluir, alterar e excluir) na área administrativa do site (*siteBD*). Para tanto, trabalharemos com uma estrutura de manutenção com os seguintes arquivos:

Arquivo	Descrição
`cidades_list.php`	É o arquivo para listar os dados de cidades. Contém código (X)HTML e PHP
`cidades_form.php`	É o arquivo que contém o formulário para incluir e alterar dados. Contém código (X) HTML e PHP.
`cidades.php`	É o **gerenciador** da manutenção. Este arquivo contém apenas código PHP e é responsável por receber as requisições do usuário e proceder a execução do código correspondente.

Os arquivos *cidades_list.php* e *cidades_form.php* inicialmente possuem a mesma estrutura do *index.php*. A partir desta estrutura básica, primeiro vamos incluir a parte (X)HTML para depois fazer a programação PHP.

- Para começar, vamos ver como é a estrutura básica do arquivo *cidades_list.php* (figura D.14). Observe que a figura está mostrando apenas a parte de código que está na *div conteudo*, pois já vimos na figura D.11 a estrutura básica. Por hora o código não possui programação PHP, apenas a estrutura (X)HTML. Na figura D.15 você pode conferir como fica a apresentação do *cidades_list.php* no navegador. Alguns apontamentos sobre o código:
- Veja que existem alguns links, na linha 17, 30 e 33. Todos são direcionados para *cidades.php*, este arquivo é o gerenciador da manutenção, por isso, sempre ele será requisitado.Veja que é uma requisição GET passando por parâmetro um valor para *acao*. Este parâmetro *acao* (ação) indica qual a intenção do usuário, neste arquivo pode ser: incluir, alterar e excluir.
- Entre as linhas 19 e 25 temos a primeira linha da tabela que é o cabeçalho;
- Entre as linhas 26 e 35 temos a segunda linha da tabela. Esta linha posteriormente será repetida para cada registro da tabela *cidades* mostrando seus dados. Na penúltima coluna tem o link para alterar e na última coluna há um link para excluir que faz uma chamada da JavaScrip para confirmar a exclusão.

```
15 <div id="conteudo">
16   <div id="titulo">Lista de Cidades</div>
17     <a href="cidades.php?acao=incluir"><br />Incluir</a><br />
18     <table width="600" border="1" align="center" cellpadding="3" cellspacing="0">
19       <tr>
20         <th width="50" scope="col">Cód</th>
21         <th scope="col">Nome</th>
22         <th width="50" scope="col">UF</th>
23         <th width="50" scope="col">Alterar</th>
24         <th width="50" scope="col">Excluir</th>
25       </tr>
26       <tr>
27         <td>id</td>
28         <td>nome</td>
29         <td>uf</td>
30         <td><a href="cidades.php?acao=alterar">Alterar</a></td>
31         <td><a href="javascript:
32         if (confirm('Confirma exclusão?')){
33           location='cidades.php?acao=excluir'
34         }">Excluir</a></td>
35       </tr>
36     </table>
37 </div>
```

Figura D.14 - Código do cidades_list.php
Fonte: do autor

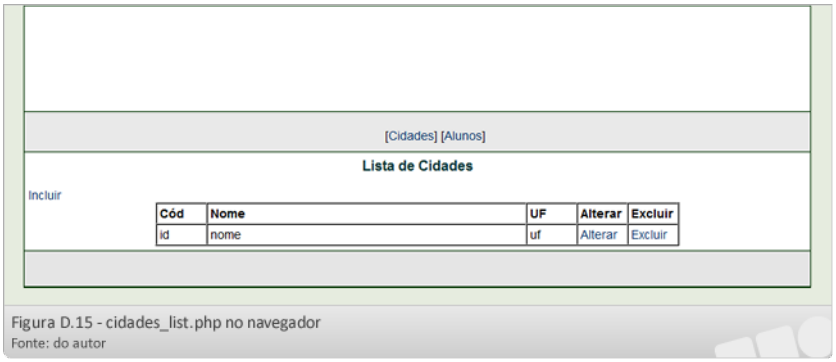

Figura D.15 - cidades_list.php no navegador
Fonte: do autor

Na figura D.16 pode ser analisado o código (X)HTML do formulário (cidades_form.php). Alguns apontamentos sobre o (X)HTML:

Linha 15 – O *action* do *form* faz referência ao cidades.php, ou seja, quando submeter o form (clicar em gravar) chamará o *cidades.php*;

Linha 20 – *Readonly* indica que a caixa de texto é apenas leitura. O usuário não pode editar, isso porque o campo *id* de cidades é gerado automaticamente;

Linha 29 – Possui um elemento *select* chamado *uf* sem *options*. Depois acrescentaremos as opções.

Linha 34 e 35- Quando clicar no botão cancelar altera a *location* para *cidades.php* (chama o *cidades.php*) ;

Linha 36 – Tem um campo oculto chamado *acao*. Logo veremos porque este campo é necessário.

É muito importante saber o nome dos objetos criados no *form*, principalmente: *id*, *nome*, *uf*, *acao*. Vamos precisar depois.

```
14 <div id="conteudo">
15   <form id="form1" name="form1" method="post" action="cidades.php">
16   <div id="titulo">Cadastro de Cidades</div>  <br/>
17   <table width="600" border="0" align="center" cellpadding="5" cellspacing="0">
18   <tr>
19     <td align="right"><label for="id">Código:</label></td>
20     <td><input name="id" type="text" id="id" value="" size="10" readonly="readonly" />
21       </td>
22   </tr>
23   <tr>
24     <td align="right"><label for="nome">Nome:</label></td>
25     <td><input name="nome" type="text" id="nome" value="" size="50" maxlength="50" /></td>
26   </tr>
27   <tr>
28     <td align="right"><label for="uf">UF:</label></td>
29     <td><select  name="uf" id="uf">  </select></td>
30   </tr>
31   <tr>
32     <td align="right"> </td>
33     <td><input type="submit" name="botao1" id="botao1" value="Gravar" />
34         <input type="button" name="botao2" id="botao2" value="Cancelar"
35           onclick="location='cidades.php' " />
36         <input name="acao" type="hidden" id="acao" value="" /></td>
37   </tr>
38   </table>
39   </form>
40 </div>
```

Figura D.16 - Código do cidades_form.php
Fonte: do autor

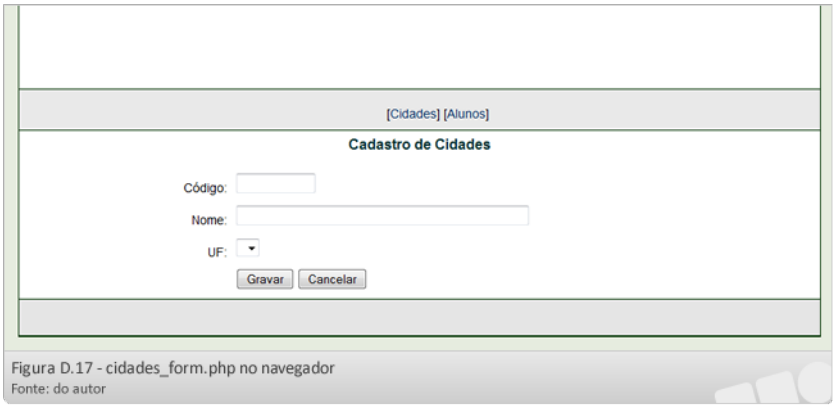

Figura D.17 - cidades_form.php no navegador
Fonte: do autor

Agora vamos para a programação PHP que vai integrar os arquivos da listagem e formulário. O código referente ao gerenciador da manutenção, arquivo *cidades.php* é mostrados na figura D.18.

```php
<?php
  include('conexao.php');
  include('funcoes.php');

  $acao = $_REQUEST['acao'];
  $redireciona = false;

  if ($acao == ""){
      $sql = "select * from cidades order by nome";
      $rs  = $con->Execute($sql);
      include('cidades_list.php');
  }

  if ($acao == "incluir"){
      include('cidades_form.php');
  }

  if ($acao == "alterar"){
      $id = $_REQUEST['id'];
      $sql = "select * from cidades where id = $id";
      $rs  = $con->Execute($sql);
      include('cidades_form.php');
  }
  if ($acao == 'excluir'){
      $id = $_REQUEST['id'];
      $sql = "delete from cidades where id = $id";
      $rs = $con->Execute($sql);
      $redireciona = true;
  }
  if ($acao == "gravar_incluir"){
     $nome = $_POST['nome'];
     $uf   = $_POST['uf'];
     $sql = "insert into cidades (nome, uf) values ('$nome', '$uf')  ";
     $rs = $con->Execute($sql);
     if (!$rs) // se ocorreu erro no sql...
        $mensagem = "Erro ao incluir dados";
     $redireciona = true;
  }
  if ($acao == "gravar_alterar"){
      $id   = $_POST['id'];
      $nome = $_POST['nome'];
      $uf   = $_POST['uf'];
      $sql = "update cidades set nome = '$nome', uf = '$uf'  where id = $id ";
      $rs = $con->Execute($sql);
      if (!$rs)
        $mensagem = "Erro ao alterar dados";
      $redireciona = true;
  }

  if ($mensagem > ''){
     echo "<script> alert('$mensagem');</script>";
  }

  if ($redireciona == true){
     echo "<script> location='cidades.php' </script> ";
  }

?>
```

Figura D.18 - arquivo cidades.php
Fonte: do autor

Vamos analisar todo código:

Linha 5 – O `$_REQUEST[]` é um *array* super global do PHP que pode ser usado para pegar dados que sejam recebidos tanto pelo método GET quanto pelo POST. *$acao* será o parâmetro utilizado para verificar a ação que o usuário selecionou, que pode ser incluir, alterar, excluir, entre outras. O valor para *acao* pode ser passado tanto por GET como POST, por isso estamos usando o `$_REQUEST`;

Linha 6 – A variável *$redireciona* quando possuir valor *true* vai redirecionar para a listagem das cidades. Inicialmente possui valor *false*;

Linha 8 – Quando a *$acao* não possuir um valor, vamos sempre mostrar a listagem de cidades. Isso vai ocorre quando for chamado diretamente *cidades.php* sem passar nenhum parâmetro. Neste caso, executa o SQL para selecionar os registros de cidades em ordem de nome e faz o include do *cidades_list.* php que vai listar os dados;

Linhas 14 e 15 – Quando a *$acao* for incluir faz o include do *cidades_form.php* que vem em branco.

Linhas 18 até 23 – Quando a *$acao* for alterar, vai fazer a alteração de uma cidade cujo *id* (identificador) será recebido por parâmetro. A linha 19 pega o valor de *id*, para então na linha 20 montar o SQL que busca o registro da cidade e executa na linha 21. Após faz o include de *cidades_form.php* que mostrará o formulário com os dados da cidade.

Linhas 24 até 29 - Quando a *$acao* for *excluir*, vai fazer a exclusão de uma cidade cujo *id* (identificador) será recebido por parâmetro. A linha 25 pega o valor de *id*, para então na linha 26 montar o SQL que busca o registro da cidade e executa na linha 27. Na linha 28 altera o valor da variável *$redireciona* para *true* (isto vai fazer voltar para a listagem atualizada).

Linhas 30 até 38 - Quando a *$acao* for *gravar_incluir*, vai fazer a inclusão de uma cidade cujos dados vieram do formulário. Primeiro uma questão, de onde vem o valor *gravar_incluir*? Deverá vir do formulário quando for uma inclusão. Veremos como fazer isso quando incluirmos programação PHP no *cidades_form.php*. Lembra que destacamos a importância de saber o nome dos objetos do formulário? Agora vamos precisar deles. Nas linhas 31 e 32 estamos recuperando a informação do nome e da UF da cidade que foram informados no formulário (usa método POST). Após montamos um SQL de inserção usando estes valores. Outra forma seria fazer diretamente o uso do $_POST no SQL, desta forma:

```
$sql = "insert into cidades (nome, uf) values

        ('$_POST[nome]', '$_POST[uf]')  ";
```

Fique a vontade para usar a forma que achar melhor. Na linha 34 o SQL é executado atribuindo o resultado para *$rs*. Na linha 35 testa se ocorreu erro e na linha 36 atribui uma mensagem de erro para variável *$mensagem* (se for o caso). Na linha 37 altera o valor da variável *$redireciona* para *true* (isto vai fazer voltar para a listagem atualizada).

Linhas 39 até 48 - Quando a *$acao* for *gravar_alterar*, vai fazer a alteração de uma cidade cujos dados vieram do formulário. De onde vem o valor *gravar_alterar*? Deverá vir do formulário quando for uma alteração. Veremos como fazer isso quando incluirmos programação PHP no *cidades_form.php*. Nas linhas 40, 41 e 42 estamos recuperando a informação *id*, *nome* e *UF* da cidade do formulário (usa método POST). Após montamos um SQL de alteração usando estes valores. Na linha 44 o SQL é executado atribuindo o resultado para *$rs*. Na linha 45 testa se ocorreu erro e na linha 46 atribui uma mensagem de erro para variável *$mensagem* (se for o caso). Na linha 47 altera o valor da variável *$redireciona* para *true* (isto vai fazer voltar para a listagem atualizada).

Linhas 50 até 52 – Se a variável *$mensagem* possuir algum valor, faz mostrar a mensagem usando JavaScript.

Linhas 54 até 56 – Se a variável *$redireciona* possuir valor *true*, usa JavaScript para redirecionar para *cidades.php*(vai mostrar a listagem).

Nosso *cidades.php* está pronto. Agora precisamos retornar ao *cidades_list.php* e ao *cidades_form.php* para incluir a programação PHP. Com esta estrutura que trabalhamos, teremos pouca codificação PHP neste dois arquivos. A maior parte da codificação é no *cidades.php*. Isso é interessante, pois facilita a manutenção posterior do código. Vamos ver agora o código completo dos arquivos *cidades_list.php* e *cidades_form.php* e alguns comentários.

cidades_list.php

A figura D.19 apresenta o *cidades_list.php* com a inclusão da programação PHP. Vamos comentar agora o que programamos neste arquivo:

Para mostrar todas as cidades retornadas a partir do SQL executado na linha 10 do *cidades.php,* precisamos de um laço (*while*). Entre as linhas 26 de 31 foi incluído um código PHP com o início de laço

while para percorrer todos resultado. Nas linhas 28, 29 e 30 recuperamos os dados que vamos mostrar, atribuindo para variáveis. É importante observar que a segunda linha da tabela fica dentro do laço, ou seja, será repetida para cada cidade. Na linha 41 tem o código para avançar para o próximo registro e o laço é fechado na linha 42;

Para mostrar os dados nas colunas da tabela, foram mostradas as variáveis nas linhas 33, 34 e 35;

Para o processo de alteração e exclusão precisamos saber o *id* (identificador/chave) do registro. Por isso, nos links de alterar e excluir precisamos passar esta informação. A adição de código nas linhas 36 e 38 tem este objetivo. Acrescentamos no link mais um parâmetro (& para indicar que vem outro parâmetro) cujo nome é *id* e usando PHP mostramos o valor da variável *$id*.

```
15 <div id="conteudo">
16   <div id="titulo">Lista de Cidades</div>
17   <a href="cidades.php?acao=incluir">Incluir</a><br />
18   <table width="600" border="1" align="center" cellpadding="3" cellspacing="0">
19   <tr>
20      <th width="50" scope="col">Cód</th>
21      <th scope="col">Nome</th>
22      <th width="50" scope="col">UF</th>
23      <th width="50" scope="col">Alterar</th>
24      <th width="50" scope="col">Excluir</th>
25   </tr>
26   <?php
27      while(!$rs->EOF){ //enquanto não for o fim do resultado fica no laço
28         $id    = $rs->fields['id'];
29         $nome  = $rs->fields['nome'];
30         $uf    = $rs->fields['uf'];
31   ?>
32   <tr>
33      <td><?php echo $id;?></td>
34      <td><?php echo $nome;?></td>
35      <td><?php echo $uf;?></td>
36      <td><a href="cidades.php?acao=alterar&id=<?php echo $id;?>">Alterar</a></td>
37      <td><a href="javascript:  if (confirm('Confirma exclusão?')){
38            location='cidades.php?acao=excluir&id=<?php echo $id;?>'}">Excluir</a></td>
39   </tr>
40   <?php
41         $rs->MoveNext();//mover para o próxima cidade
42     } //fim do laço
43   ?>
44   </table>
45 </div>
```

Figura D.19 - código do cidades_list.php com PHP
Fonte: do autor

cidades_form.php

A figura D.20 apresenta o *cidades_form.php* com a inclusão da programação PHP. O que basicamente foi alterado:

Inserção de código PHP no atributo *value* para mostrar os dados que vem do banco de dados (vai mostrar quando o formulário é chamado para uma alteração de cidade). Nas linhas 21 e 27 utiliza a variável *$rs* do resultado do SQL (criado na linha 21 do *cidades.php*) e pega o valor do campo usando *fields*;

Para mostrar a lista de estados no menu de lista, estamos incluindo na linha 33 a chamada de uma função denominada *lista_uf()* que está definida no arquivo *funcoes.php* (foi incluído no *cidades.php*). Esta função recebe por parâmetro a UF da cidade. Este parâmetro serve para trazer selecionada a UF da cidade quando for um processo de alteração. A seguir vamos ver em detalhes a função *lista_uf()*;

Lembra que comentamos sobre os valores *gravar_incluir* e *gravar_alterar* do *$acao*? Quando o formulário for submetido enviará um valor para o campo oculto acao que está na linha 41. Atribuímos para o valeu dele o seguinte:

```
gravar_<?php echo $acao;?>
```

Isto significa que quando o formulário for chamado a partir de um valor de *$acao* igual a *incluir*, o campo oculto ficará valendo `gravar_incluir`. Se for chamado a partir de um valor de *$acao* igual a *alterar*, o campo oculto ficará valendo `gravar_alterar`. Com isso, podemos saber no *cidades.php* se é necessário fazer um processo de inclusão ou alteração.

```
15 <div id="conteudo">
16     <form id="form1" name="form1" method="post" action="cidades.php">
17     <div id="titulo">Cadastro de Cidades</div>     <br/>
18     <table width="600" border="0" align="center" cellpadding="5" cellspacing="0">
19     <tr>
20     <td align="right"><label for="id">Código:</label></td>
21     <td><input name="id" type="text" id="id" value="<?php echo $rs->fields['id'];?>"
22         size="10" readonly="readonly" />
23     </td>
24     </tr>
25     <tr>
26     <td align="right"><label for="nome">Nome:</label></td>
27     <td><input name="nome" type="text" id="nome" value="<?php echo $rs->fields['nome'];?>"
28      size="50" maxlength="50" /></td>
29     </tr>
30     <tr>
31     <td align="right"><label for="uf">UF:</label></td>
32     <td><select  name="uf" id="uf">
33        <?php  echo lista_uf($rs->fields['uf']);  ?>
34        </select></td>
35     </tr>
36     <tr>
37     <td align="right"> </td>
38     <td><input type="submit" name="botao1" id="botao1" value="Gravar" />
39         <input type="button" name="botao2" id="botao2"
40         value="Cancelar"  onclick="location='cidades.php' " />
41        <input name="acao" type="hidden" id="acao" value="gravar_<?php echo $acao;?>" /></td>
42     </tr>
43     </table>
44     </form>
45 </div>
```

Figura D.20 - código do cidades_form.php com PHP
Fonte: do autor

Funcão lista_uf()

Como visto acima, usamos a função *lista_uf()* para mostrar a lista de unidades federativas no *cidades_form.php*. Agora vamos dar uma olhada com mais detalhes nesta função que está definida no *funcoes.php.* A figura D.21 mostra o código da função. Uma primeira observação importante a fazer é sobre como é montada uma lista de opções para um menu de lista. O código de exemplo abaixo apresenta cinco opções, sendo que a primeira não possui valor, fica em branco. Cada opção está em um elemento *option* que por sua vez possui um atributo *value* para o valor correspondente ao que está sendo mostrado, no caso da UF tem o mesmo valor. Mas observe que uma *option* possui atributo *selected*. Isto significa que o elemento que possuir este atributo virá com este valor selecionado para o menu de lista. Neste exemplo o menu de lista viria com a opção AM selecionada. No caso do nosso exemplo este atributo será importante para mostrar a UF correspondente à cidade que está sendo alterada.

```
<option></option>

<option value='AC'>AC</option>

<option value='AL'>AL</option>

<option value='AM' selected>AM</option>

<option value='AP'>AP</option>
```

Alguns apontamentos:

Linha 4 – Define um *array* com todas as unidades federativas;

linha 7 – Ordena o vetor em ordem alfabética;

linha 8 – Inicializa a variável *$lista* com a primeira opção em branco;

linha 9 - Faz um *foreach* para percorrer todo o *array*. *$sigla* recebe cada vez um dos valores do vetor;

linha 10 – Cria a variável *$s* sendo uma string sem valor;

linhas 11 e 12 – Verifica se o valor do vetor (*$sigla*) é igual a *$uf* recebida por parâmetro (UF da cidade a ser alterada). Se for igual atribui "*selected*" para variável *$s*. Isso fará com que esta UF seja a selecionada.

Linha 13 – Monta a opção e concatena na variável *$lista*;

Linha 14- Fim do *foreach*;

Linha 15- Retorna as opções montadas (variável *$lista*).

```php
function lista_uf($uf){
    $ufs = array('RS','SC','PR','SP','RJ',
                 'AM', 'AC','AL','AP','BA','CE','DF','ES','GO','MA',
                 'MG','MS','MT','PA','PB','PE','PI','RN','RO','RR','SE','TO');
    sort($ufs);//ordena o array
    $lista = "<option></option>";
    foreach($ufs as $sigla){
       $s = "";
       if($sigla == $uf)
          $s = "selected";
       $lista .= "<option value='$sigla' $s >$sigla</option>";
    }
    return $lista;
}
```

Figura D.21 - código da função lista_uf()
Fonte: do autor

Mostrar dados na Área Pública

Só falta vermos como mostrar os dados de cidades na Área Pública do site. Vamos criar um arquivo chamado *cidades.php* na pasta *c:\xampp\htdocs\siteBD*. A figura D.22 apresenta o código do arquivo *cidades.php*. Não temos nada de novo aqui, estamos usando tudo o que já vimos anteriormente.

Dica:

Na linha 21 do cidades.php faz o include do *conexao.php*. Observe que o caminho muda, pois está buscando na pasta *admin*.

```php
<div id="conteudo">
  <div id="titulo">Lista de Cidades</div>  <br />
  <table width="600" border="1" align="center" cellpadding="3" cellspacing="0">
  <tr>
     <th scope="col">Nome</th>
     <th width="50" scope="col">UF</th>
     </tr>
  <?php
      include("admin/conexao.php");
      $sql = "select * from cidades order by nome";
      $rs  = $con->Execute($sql);
      while(!$rs->EOF){ //enquanto não for o fim do resultado fica no laço
        $nome = $rs->fields['nome'];
        $uf   = $rs->fields['uf'];
  ?>
  <tr>
     <td><?php echo $nome;?></td>
     <td><?php echo $uf;?></td>
  </tr>
  <?php
         $rs->MoveNext();//mover para o próxima cidade
       } //fim do laço
  ?>
  </table>
</div>
```

Figura D.22 - Código do arquivo cidades.php da área pública
Fonte: do autor

Atenção:

A estrutura do arquivo, na parte (X)HTML, é a mesma do *index.php* da área administrativa, apenas retiramos o menu. Neste momento não nos preocupamos tanto com o aspecto visual, haja vista que nosso propósito é a programação PHP.

A figura D.23 mostra a visualização do arquivo *cidades.php* da Área Pública no navegador.

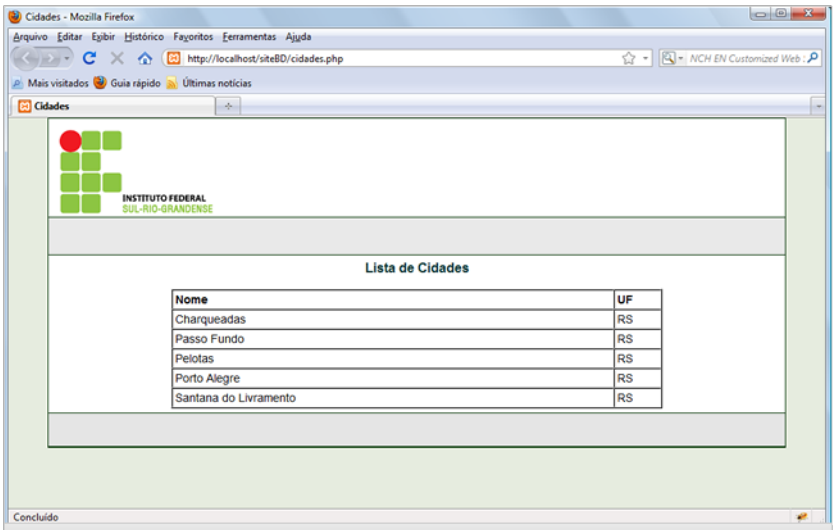

Figura D.23 - Mostrando cidades na área pública
Fonte: do autor

Tratamento chave estrangeira

Outro recurso importante é o tratamento de chave estrangeira. Para explicitar vamos ver a estrutura da tabela *alunos* que criamos no início da Unidade D. Veja abaixo que a tabela possui um campo chamado *cidade* que é do tipo *int(11)*. O valor do campo cidade faz referência a chave primária da tabela *cidades* (campo *id*).

```
CREATE TABLE IF NOT EXISTS alunos (
  id bigint NOT NULL auto_increment,
  nome varchar(100) NOT NULL,
  email varchar(100) NOT NULL,
  fone varchar(15) NOT NULL,
  nascimento date NOT NULL,
  cidade     int(11) NOT NULL,
  PRIMARY KEY  (id)
);
```

Pensando no formulário de alunos, podemos fazer a seleção da cidade conforme mostra a figura D.24, buscando os dados da tabela *cidades* e mostrando em um menu de lista.

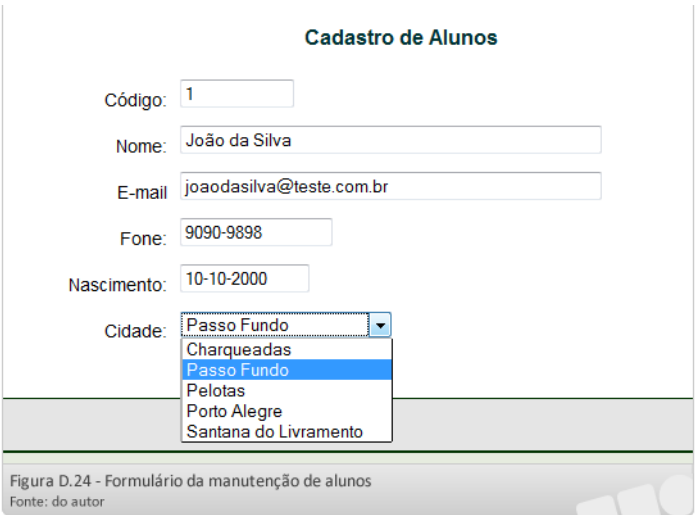

Figura D.24 - Formulário da manutenção de alunos
Fonte: do autor

O menu de lista seria montado com as opções da forma apresentada abaixo, onde mostra o nome da cidade, mas o valor correspondente ao nome da cidade é o identificador (*id*):

```
<select name='cidade'>
<option></option>
<option value='5'>Charquedas</option>
<option value='2' selected >Passo Fundo</option>
<option value='3' >Pelotas</option>
<option value='4' >Porto Alegre</option>
<option value='6'>Santana do Livramento </option>
</select>
```

Para montar este menu de lista, a biblioteca ADODB possui um método chamado *GetMenu*. Por isso, no arquivo *funcoes.php* criamos a função *lista_cidades()* como exemplo para tratar chave estrangeira com o *GetMenu*. Vamos dar uma olhada no código conforme figura D.25:

Linha 19 – Definição da função *lista_cidades()* que recebe dois parâmetros: a instância da conexão e o nome da cidade que deve ser mostrado no menu de lista por padrão;

Linha 21 – Monta o SQL para buscar os dados da cidade. Importante a ordem dos campos no *select*, deve sempre primeiro vir a descrição e depois o campo que é chave primária;

Linha 22 – Executa o SQL e atribui o resultado para *$rs*;

Linha 23 – Testa se chegou um valor no parâmetro *$nome*. Se chegou valor significa que é uma alteração, senão uma inclusão. O menu de lista é montado de forma diferente para cada situação.

Linha 24 – Retorna o menu montado. O menu se chamará cidade, virá selecionado por padrão a cidade cujo nome é igual ao valor de *$nome*, e não possui um primeiro valor do menu em branco;

Linha 26 – Retorna o menu montado. O menu se chamará cidade, virá selecionado com a primeira opção em branco;

```
19 function lista_cidades($con, $nome){
20     //no sql deve sempre virprimeiro o nome e depois o código (id)
21     $sql = "select nome, id from cidades order by nome";
22     $rs = $con->Execute ($sql);
23     if ($nome > "")
24       return $rs->GetMenu ('cidade',$nome,false);//alterar
25         else
26       return $rs->GetMenu('cidade',null,true);//incluir
27 /*
28     GetMenu:
29     1° parametro: nome do menu de lista
30     2° parametro: valor para mostrar na caixa
31     3° parametro: true para mostrar a primeira opçao vazia e false
32     para mostrar um valor (do 2° parametro)
33 */
34 }
```

Figura D.25 - Código da função lista_cidades()
Fonte: do autor

Atenção:

Note que o método *GetMenu* monta toda a estrutura do menu de lista, o *select* e as *option*

Abaixo segue um exemplo de chamada da função no formulário dentro da coluna que se deseja mostrar o menu de lista:

```
<td><?php echo lista_cidades($con, $rs->fields['cidade_nome']);  ?></td>
```

Tratamento de datas e valores

Para campos do tipo *float*, *decimal* e *double*, o banco de dados Mysql vai retornar o ponto (.) como separador decimal. Para formatar um valor podemos utilizar a função *number_format* do PHP. Por exemplo, considere que a variável *$valor* possui um valor retornado do banco de dados:

```
$fvalor = number_format($valor, 2, ',', '.');
```

Os parâmetros na sequência são:

- valor a ser formatado
- número de casas decimais
- caractere para separador decimal
- caractere para separador de milhar

Também precisamos nos preocupar no formato para gravar um valor no banco. Considere que usuário tenha informado o valor 4.560,80, para gravar no banco o formato correto é 4560.80. Para fazer esta conversão podemos usar a função abaixo que pode ser definida no *funcoes.php* e ser chamada sempre que necessário:

```
function fvalor_banco($val){

        $val = str_replace(".","",$val);
    $val = str_replace(",",".",$val);
    return $val;
}
```

Para chamar a função:

```
$valor = fvalor_banco($valor);
```

Atenção:

O primeiro *str_replace* da função retira o ponto. O segundo *str_replace* troca vírgula por ponto.

O MySQL trabalha com datas no formata ano/mês/dia. Já o nosso formato é dia/mês/ano. Por isso, sempre que pegarmos uma data do banco precisamos converter para nosso formato e quando enviarmos uma data para o banco devemos tratar para ser no formato aceito pelo MySQL.

A função abaixo recebe uma data no formato a/m/d e transforma para o formato d/m/a.

```
function fdata($dt){
  $data = explode('-', $dt);
  return "$data[2]/$data[1]/$data[0]";
}
```

A função abaixo recebe uma data no formato d/m/a e transforma para o formato a/m/d.

```
function fdata_banco($dt){
  $data = explode('/', $dt);
  return "$data[2]/$data[1]/$data[0]";
}
```

Dica:

Inclua estas duas funções no arquivo *funcoes.php* .

Síntese

Bem, esta parte da Unidade D mostrou em detalhes a construção de uma manutenção completa usando a linguagem PHP com acesso a banco de dados. Agora é importante que você pratique, por isso faça a atividade proposta e anote todas as suas dúvidas para posteriormente discutir tais questões no fórum. Bom trabalho!

Atividades - Parte 2

1. Considerando a tabela *alunos* criado no banco de dados *curso* faça a manutenção completa para a tabela (listar, incluir, alterar e excluir).

a) utilize a estrutura do site *siteBD*
b) deve usar o padrão de manutenção apresentado na disciplina
c) observar que há um campo de chave estrangeira (usar o modelo da função *lista_cidades()*)
d) fazer a validação do formulário (campos obrigatórios e validação de data) usando JavaScript

Após a realização das atividades participe do fórum de discussão proposto pelo professor formador.

Autenticação de usuário para área administrativa

Nesta segunda parte da unidade D vamos mostrar como construir um mecanismo de autenticação de usuário para acesso a área administrativa.

Controle de acesso

O processo de autenticação de usuário para acesso a área administrativa é extremamente importante, uma vez que não podemos deixar esta área do site vulnerável para acesso de usuários sem permissão. Imagine qualquer usuário tendo acesso aos recursos de incluir, alterar e excluir dados da base de dados do site. Por isso, é importante que somente usuários registrados tenham acesso. Inicialmente o usuário deve informar seu *login* e senha em uma tela com mostra a figura D.26.

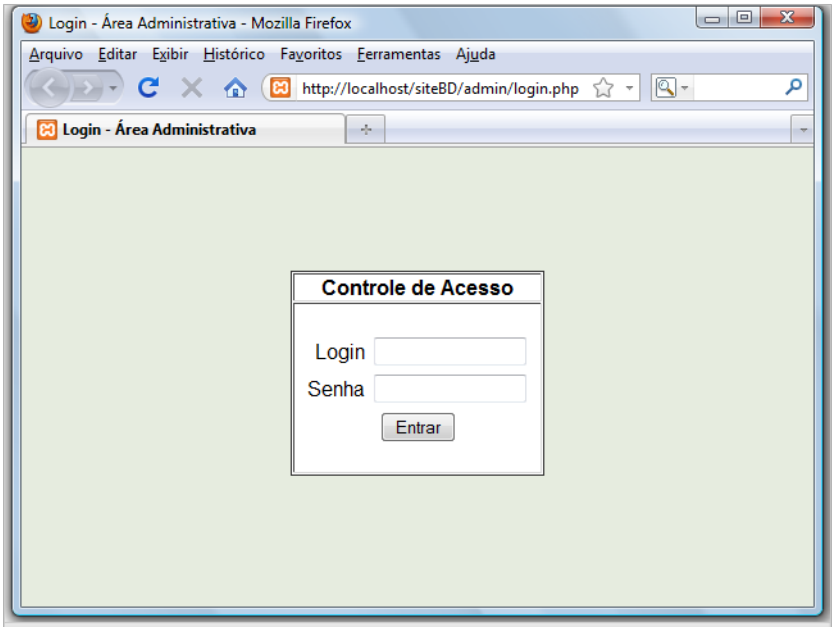

Figura D.26 - Tela de login
Fonte: do autor

Após informar *login* e senha válidos, o usuário tem acesso a área administrativa (como a figura D.27). Em cada acesso de página da área administrativa é verificado se o usuário se autenticou. Veja na figura que foi mostrada uma saudação ao usuário: "Olá Daniel!". Isso porque o *login* do usuário que está logado fica na sessão.

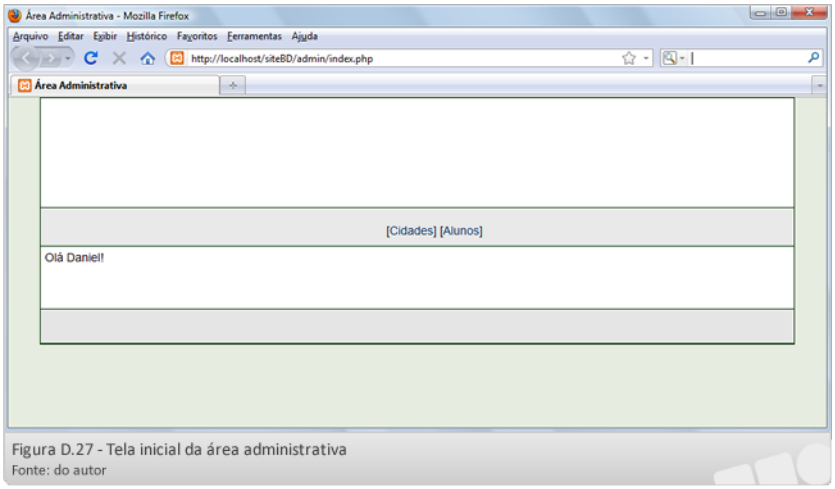

Figura D.27 - Tela inicial da área administrativa
Fonte: do autor

Agora vamos ver passo a passo o que precisamos para fazer a autenticação do usuário e controlar a sessão para a área administrativa.

Tabela usuários

Primeiro precisamos ter no nosso banco de dados (*curso*) uma tabela para registrar os usuários que tem acesso a área administrativa. O script abaixo deve ser usado para criar a tabela *usuarios*.

```
CREATE TABLE  usuarios (
  login VARCHAR( 20 ) NOT NULL ,
  nome VARCHAR( 50 ) NOT NULL ,
  senha VARCHAR( 35 ) NOT NULL ,
PRIMARY KEY ( login)  );
```

A tabela contém apenas três campos, sendo o *login* a chave primária. Além disso, temos o nome do usuário e a senha. Note que o campo senha foi criado com tamanho 35, isso porque a senha será criptografada.

No exemplo que vamos usar, todos os usuário vão possuir o mesmo tipo de acesso. Mas existem sistemas que podem necessitar de níveis diferentes de acesso. Neste caso deverá ser projetada uma estrutura diferente, de forma a identificar qual o nível de acesso do usuário.

Não vamos criar a manutenção de usuários neste material, pois nosso objetivo agora é fazer a autenticação do usuário. Por isso, para incluir dados na tabela crie um arquivo nomeado de *inserir_usuario.php*. A figura D.28 mostra o código do *inserir_usuario.php*.

Só precisamos chamar a atenção de uma função usada neste código. É a função ***md5*** da linha 5. A função ***md5*** faz a criptografia do valor passado por parâmetro usando um algoritmo unidirecional, isto significa que não existe o processo de descriptografar. Veremos depois como vamos testar a senha. A função gera uma string de 32 caracteres.

Ao executar o *inserir_usuario.php* será criado um usuário com os seguintes dados:

- *Login* = root
- Nome = Adminstrador
- Senha = e8d95a51f3af4a3b134bf6bb680a213a

Atenção:

Lembre-se que e8d95a51f3af4a3b134bf6bb680a213a é a criptografia de “senha”.

```php
<?php

    include('conexao.php');

    $senha = md5('senha'); //e8d95a51f3af4a3b134bf6bb680a213a

    $sql = "insert into usuarios (login, nome, senha) values
    ('root','Administrador', '$senha')";

    $rs = $con->Execute($sql);

    if (!$rs) {
        echo  $con->ErrorMsg();
    }else
        echo "Dados inseridos";

?>
```

Figura D.28 - Código do inserir_usuario.php
Fonte: do autor

Dica:

Lembre-se dos dados do usuário para posteriormente fazer o *login*. Você também pode visualizar estes dados pelo *phpMyAdmin* *<http://localhost/phpMyAdmin)>.*

Tela Login

Para tela de *login* crie o arquivo *login.php* na pasta *admin* (c\xampp\htdocs\siteBD\admin). O código da tela de *login* é mostrado na figura D.29. A parte de código apresentada está no elemento *body*. Destaque para:

- No *action* do *form* (linha 18) chama *index.php*, que a página inicial da área administrativa. Como o botão Entrar é

do tipo *submit*, quando for acionado chamará o *index.php*;

- Possui os campos *login* e senha;
- O campo senha é do tipo *password*.

```
11  <table width="190" border="1" align="center" cellpadding="0" cellspacing="1"
12  bordercolor="#333333" bgcolor="#FFFFFF" >
13  <tr>
14    <td height="19" align="center"><strong>Controle de Acesso</strong></td>
15  </tr>
16  <tr>
17    <td height="109">
18    <form action="index.php" method="POST" name="frmlogin" id="frmlogin"> <br>
19    <table width="192" border="0" align="center" cellpadding="0" cellspacing="7">
20    <tr>
21       <td width="28%" height="21" align="right"> Login </td>
22       <td width="72%"> <input name="login" type="text" id="login" size="15" maxlength="30"> </td>
23    </tr>
24    <tr>
25       <td align="right"> Senha</td>
26       <td> <input name="senha" type="password"  id="senha"  size="15" maxlength="30"> </td>
27    </tr>
28    <tr align="center" >
29       <td colspan="2"> <input name="btnEntrar" type="submit" id="btnEntrar" value="Entrar" > </td>
30    </tr>
31    </table>
32    </form>
33    </td>
34  </tr>
35  </table>
```

Figura D.29 - código do login.php
Fonte: do autor

Controle de acesso

Agora que já temos uma tabela *usuarios* e a tela de *login*, vamos fazer o controle de acesso a área administrativa. Este controle será feito pelo arquivo *controle_acesso.php*. Importante: para controlar se o usuário está logado vamos criar uma variável na sessão chamada *ses_usu_login* que conterá o *login* do usuário. A figura D.30 mostra o código completo do *controle_acesso.php*. Vamos analisá-lo:

Linha 2 – Inicia a sessão;

Linha 3 – Verifica se tem a variável *ses_usu_login* na sessão, se não possuir entra neste if ;

Linha 5 – Para verificar se veio da tela de *login*, testa se existe e se há um valor para *login* no array *$_REQUEST[]*, se existir entra neste IF;

Linha 6 – Atribui a senha que recebeu da tela de *login* para a variável *$senha*, já criptografando. Por que a criptografia? Porque a senha do banco está criptografada e não há mecanismo para reverter. Por isso, vamos testar as duas criptografadas;

Linhas 7 e 8 – Monta o SQL para consultar no banco se existe usuário com o login e senha informados;

Linha 9 – Executa o SQL e atribui o resultado para $rs;

Linhas 11 e 12 – Verifica novamente (pode não ter retornado registro) se o registro retornado possui o login igual ao informado. Se *true* cria a variável *ses_usu_login* na sessão atribuindo o *login*.

Linhas 14 até 16 – Se não encontrar usuário no banco, faz include do arquivo que mostra mensagem de *login* inválido (*logininvalido.php*) e executa a função *exit* (pára a execução);

Linhas 19 até 21 – Se não veio da tela de *login*, faz include do arquivo que mostra mensagem de acesso negado (*acessonegado.php*) e executa a função *exit* (pára a execução);

Linha 24 – Entra no *else* se possuir na sessão a variável *ses_usu_login*

Linhas 25 até 29 – Se a variável da sessão não possuir valor, ou seja não tem um *login*, retira a variável da sessão, faz include do arquivo que mostra mensagem de acesso negado (*acessonegado.php*) e executa a função *exit* (pára a execução);

```php
<?php
  session_start();
  if(session_is_registered("ses_usu_login") == false ){

    if (isset($_REQUEST["login"]) and $_REQUEST['login'] != ""){
          $senha = md5($_REQUEST['senha']);//md5
          $sql = "select * from usuarios where
                  login = '$_REQUEST[login]' and senha = '$senha'";
          $rs = $con->Execute($sql);

          if ($rs->fields['login'] == $_REQUEST['login']) {
              $_SESSION['ses_usu_login'] = $_REQUEST['login'];
          }
          else{
              include("logininvalido.php");
              exit;
          }
      }
      else{
          include("acessonegado.php");
          exit;
      }
  }
  else {
      if ($_SESSION['ses_usu_login'] <= ""){
          session_unregister("ses_usu_login");
          include("acessonegado.php");
          exit;
      }
  }
?>
```

Figura D.30 - código do controle_acesso.php
Fonte: do autor

Agora resta saber como vamos usar o *controle_acesso.php*. Ele servirá para controlar o acesso a todas as páginas da área administrativa. Por isso, no início de cada arquivo, que é chamado diretamente, faremos um include do *controle_acesso.php* (*index.php, cidades.php, alunos.php*). O include do controle_acesso.php será sempre após o include do *conexao.php*, pois ele precisa da conexão com o banco de dados.

Veja como fica no *index.php* (figura D.31). Na linha 19 mostra uma saudação ao usuário logado.

Dica:

Lembre-se que o session_start deve vir antes de qualquer escrita do documento (X(HTML)). Por isso o include fica no início do arquivo.>

```php
<?php
   include("conexao.php");
   include("controle_acesso.php");
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Área Administrativa</title>
<link href="../estilo.css" rel="stylesheet" type="text/css" />
</head>
<body>
<div id="geral">
  <div id="topo"> </div>
  <div id="menu">
    <?php include('menusup.php');?>
  </div>
  <div id="conteudo">
  <div>Olá <?php echo $_SESSION['ses_usu_login']."!";?></div>
  <br/>
  <br/>
  </div>
  <div id="rodape"></div>
</div>
</body>
</html>
```

Figura D.31 - index.php com o uso do controle_acesso.php
Fonte: do autor

A figura D.32 mostra como fica o arquivo *cidades.php* com o include do *controle_acesso.php*. O include foi feito na linha 3.

```
1  <?php
2    include('conexao.php'); // incluir arquivo de conexão
3    include('controle_acesso.php');
4    include('funcoes.php'); // incluir arquivo de funções
5
6    $acao = $_REQUEST['acao'];
7    $redireciona = false;
```

Figura D.32 - início do cidades.php com o include do controle_acesso.php
Fonte: do autor

Após a inclusão do *controle_acesso.php* nos arquivos, vamos fazer um teste. Antes de tentar fazer o *login*, tente acessar diretamente o *index.php*:

```
http://localhost/admin/index.php
```

Se tudo estiver certo, deverá mostrar a mensagem "Acesso Negado!". Isso ocorre porque o *controle_acesso.php* verificou que não tem usuário na logado na sessão. Faça outros testes agora e veja o que acontece:

Acesse também diretamente *http://localhost/admin/cidades.php*

Na tela de *login* informe usuário e senha inválidos

Faça *login* e agora tente acessar *http://localhost/admin/cidades.php*

Logout

Para sair da área administrativa pode ser criado o arquivo *logout.php* com o código para destruir a sessão. O código do *logout.php* é mostrado na figura D.33.

```
1  <?php
2
3    session_start();
4    session_destroy();
5
6  ?>
7  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
8  <html xmlns="http://www.w3.org/1999/xhtml">
9  <head>
10     <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
11     <title>Área Administrativa</title>
12     <link href="../estilo.css" rel="stylesheet" type="text/css" />
13 </head>
14 <body>
15     <div align="center">Sessão finalizada<br />
16         <a href="login.php">Voltar para login </a><br />
17     </div>
18 </body>
19 </html>
```

Figura D.33 - Código do logout.php
Fonte: do autor

Síntese

Esta parte da Unidade D mostrou como fazer a autenticação de usuário para acesso à área administrativa e como usar sessão para este controle. Agora é com você! Faça a atividade proposta e anote todas as suas dúvidas para posteriormente discutir tais questões no fórum. Bom trabalho!

Atividades - Parte 3

1. Considerando a tabela *usuarios* do *script* abaixo, faça a manutenção completa para a tabela (listar, incluir,

alterar e excluir).

a) utilize a estrutura do site *siteBD*
b) deve usar o padrão de manutenção apresentado na disciplina
c) usar mecanismo de criptografia da senha
d) no formulário de entrada de dados insira uma caixa de texto para confirmação da senha (terá os campos senha e confirma senha)
e) fazer a validação do formulário (campos obrigatórios e confirmação de senha) usando JavaScript
f) use o *controle_acesso.php* para verificar se o usuário está logado.

```
CREATE TABLE  usuarios (
 login VARCHAR( 20 ) NOT NULL ,
 nome VARCHAR( 50 ) NOT NULL ,
 senha VARCHAR( 35 ) NOT NULL ,
PRIMARY KEY ( login)  );
```

Após a realização das atividades participe do fórum de discussão proposto pelo professor formador.