TUPY VIRTUAL

SOCIESC
Educação e Tecnologia

Programação de Página WEB - PHP

Edição nº1 - 2007

ROSEMARY FRANCISCO

Apoio

Ministério da Ciência e Tecnologia
BRASIL UM PAÍS DE TODOS
GOVERNO FEDERAL

Gestão e Execução

BRASSCOM
Associação Brasileira de Empresas de Tecnologia de Informação e Comunicação
SOFTEX
EXCELLENCE IN SOFTWARE

Conteúdo e Tecnologia

DATASUL
SOCIESC
Educação e Tecnologia

Apresentação 

Este livro didático contém a disciplina de Programação Página WEB - PHP.

O conteúdo desse material irá disponibilizar aos alunos do EAD os conceitos fundamentais do Sistema Gerenciador de Banco de Dados, bem como as principais técnicas que são utilizadas para modelagem, criação e manutenção de um banco de dados.

Para sua melhor compreensão, o livro está estruturado em duas partes. Na primeira, são apresentados os conceitos fundamentais do banco de dados, que proporcionará um primeiro contato com essa tecnologia. Também na primeira parte, é demonstrado o funcionamento do Modelo Relacional, que é a base para a tecnologia de banco de dados. A segunda parte engloba as técnicas para a definição de um projeto de banco de dados: normalização, modelagem e as dependências funcionais e, ainda, uma análise mais ampla sobre o funcionamento e os recursos disponibilizados pelos Sistemas Gerenciadores de Banco de Dados.

Lembre-se de que a sua passagem por esta disciplina será também acompanhada pelo Sistema de Ensino **Tupy Virtual**, seja por correio postal, fax, telefone, e-mail ou Ambiente Virtual de Aprendizagem. Sempre entre em contato conosco quando surgir alguma dúvida ou dificuldade.

Toda a equipe terá a maior alegria em atendê-lo, pois a sua aquisição de conhecimento nessa jornada é o nosso maior objetivo.

Acredite no seu sucesso e bons momentos de estudo!

Equipe Tupy Virtual.

SUMÁRIO

Carta da Professora

O conhecimento é como um jardim: se não for cultivado, não pode ser colhido. (Provérbio Africano)

Caro(a) aluno(a),

Nos próximos capítulos, terá início a caminhada rumo ao conhecimento da programação para Internet utilizando a linguagem de programação PHP. Linguagem que foi concebida apenas para uso particular, hoje em dia expande-se tanto no meio acadêmico quanto em aplicações comerciais que estão espalhadas pelo mundo.

Com o avanço dos meios de comunicação e a rápida expansão dos serviços disponíveis pela Internet, tornou-se imprescindível aos profissionais de informática, principalmente aos desenvolvedores de aplicações, o conhecimento de uma linguagem que permita construir e disponibilizar diversas aplicações para a Internet. A Internet tem mudado muitos conceitos e o seu foco principal, hoje em dia, é disponibilizar serviços de maneira rápida a qualquer momento e para qualquer usuário.

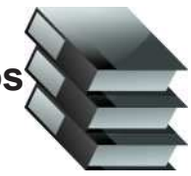
Assim, nosso objetivo maior, depois de percorrida esta caminhada, é estarmos inseridos no contexto do desenvolvimento das aplicações para a Internet.

Durante esta caminhada, conheceremos a origem da linguagem PHP, seus principais conceitos, sua sintaxe básica e, ao final, desenvolvendo um estudo de caso com o intuito de praticarmos melhor o conhecimento adquirido.

Sendo assim, convido você para, juntos, agora virtualmente, vencer este novo desafio!

Professora Rosemary Francisco

Cronograma de Estudos



Acompanhe no cronograma abaixo os conteúdos das aulas, e atualize as possíveis datas de realização de aprendizagem e avaliações.

Semanas	Carga Horária	Unidade	Data/Avaliação
1	2	Visão geral do gerenciamento de bancos de dados	_/_ a _/_
	2	Arquitetura de sistemas de bancos de dados	_/_ a _/_
	2	Introdução aos bancos de dados relacionais	_/_ a _/_
	2	Uma introdução à SQL	_/_ a _/_
	2	Introdução ao Modelo Relacional	_/_ a _/_
	3	Álgebra Relacional	_/_ a _/_
	3	Cálculo relacional	_/_ a _/_
	2	Integridade	_/_ a _/_
2	2	Dependências funcionais	_/_ a _/_
	4	Normalização	_/_ a _/_
	4	Modelagem semântica	_/_ a _/_
	2	Tabelas	_/_ a _/_
	2	Índices	_/_ a _/_
	1	Consultas de Seleção	_/_ a _/_
	2	Consultas de Ação	_/_ a _/_
	1	Visões – *Views*	_/_ a _/_
	2	Procedimentos Armazenados – *Store Procedures*	_/_ a _/_
	2	Disparadores – *Triggers*	_/_ a _/_

PLANO DE ESTUDOS

Ementa

Conceito de Sistemas Gerenciadores de Banco de Dados. Componentes. Modelos de SGBD. Criação de um banco de dados. Projetando e criando tabelas. Gerenciamento do banco de dados. Consultando o banco de dados. Criando relatórios. Otimização do banco de dados e segurança do banco de dados. SQL. Banco de dados distribuídos. Principais produtos SGBD do mercado. Banco de dados e o paradigma de objetos.

Objetivos da Disciplina

• Geral

Compreender os sistemas de bancos de dados.

• Específicos

- Introduzir os conceitos fundamentais dos sistemas de bancos de dados.
- Relacionar o funcionamento do Modelo Relacional e os objetos que compõem o Modelo Relacional.
- Demonstrar as melhores práticas para normalização e modelagem dos dados em um Projeto de Banco de Dados.
- Ampliar os conhecimentos e a aplicação da linguagem de banco de dados SQL
- Ampliar os conhecimentos sobre o funcionamento e os recursos disponíveis pelos Sistemas Gerenciadores de Banco de Dados.

Carga Horária: 40 horas/aula.

Aula 1

VISÃO GERAL SOBRE A LINGUAGEM PHP

1

Tutu Virtual

Nesta próxima aula você irá conhecer como funcionam as linguagens de programação para a Internet, tendo como foco principal a linguagem PHP. Veremos o que é PHP, como e quando surgiu essa linguagem, quais as principais vantagens em utilizar a linguagem PHP para desenvolver aplicações para Internet e como deve ser o ambiente de desenvolvimento para trabalhar com a linguagem.

Bom estudo!

Objetivos da Aula 

Ao final desta aula você deverá ser capaz de:

• Identificar o que é uma arquitetura cliente-servidor;

• Analisar as vantagens da utilização da linguagem PHP para programação para Internet;

• Identificar qual é o ambiente que deve ser utilizado para programar com PHP.

Conteúdos da Aula 

Acompanhe os conteúdos desta aula. Se você preferir, assinale-os à medida em que for estudando.

• Introdução às Linguagens de programação para Internet;

• O que é PHP?;

• Como surgiu a linguagem PHP?;

• Vantagens da linguagem PHP;

• Como trabalhar com PHP;

• Exercícios propostos.

1 INTRODUÇÃO AS LINGUAGENS DE PROGRAMAÇÃO PARA INTERNET

Como já sabemos, a Internet é uma rede que surgiu com o intuito principal de compartilhar informações entre os usuários interligados por meio dela. No início, as informações que eram compartilhadas tinham um foco de estudo e pesquisa e os usuários que tinham acesso a essa rede eram poucos. Com o avanço das tecnologias e a expansão da rede para outros usuários, o acesso às informações e aos serviços disponíveis pela Internet aumentou consideravelmente.

De acordo com as pesquisas realizadas em Junho/2007, pelo Internet World Stats, estima-se que a Internet é usada por 18% da população mundial (em torno de 1,1 bilhão de pessoas). Este acesso vem aumentando gradativamente, em Março/2007 essa mesma pesquisa foi realizada e as estatísticas apontaram que 16,9% da população tinham acesso à Internet. Isso significa que apenas em um mês tivemos um aumento de 1,1% de acessos, um aumento realmente considerável analisando as devidas proporções (bilhões de pessoas).

Não são somente os acessos à rede que vêm aumentando, os tipos de serviços disponibilizados também estão em larga expansão. Os serviços pioneiros como WWW – World Wide WEB – que permitiam apenas a publicação de páginas institucionais, com informações estáticas, estão sendo substituídas por páginas com conteúdo cada vez mais dinâmico, possibilitando aos próprios usuários da rede a interação e publicação dos conteúdos dessas páginas.

Para permitir que os usuários da rede interajam dessa forma, é necessário que as aplicações disponíveis na Internet sejam desenvolvidas com uma linguagem que permita esses tipos de recursos. Nesse cenário, surgiram as várias linguagens de programação para a Internet, que permitem aos desenvolvedores criar aplicações e serviços com os mais variados recursos para disponibilizar na Internet.

As aplicações desenvolvidas para a Internet, utilizam a arquitetura cliente-servidor. O que é uma arquitetura cliente-servidor? A figura 1 demonstra o funcionamento da arquitetura cliente-servidor.

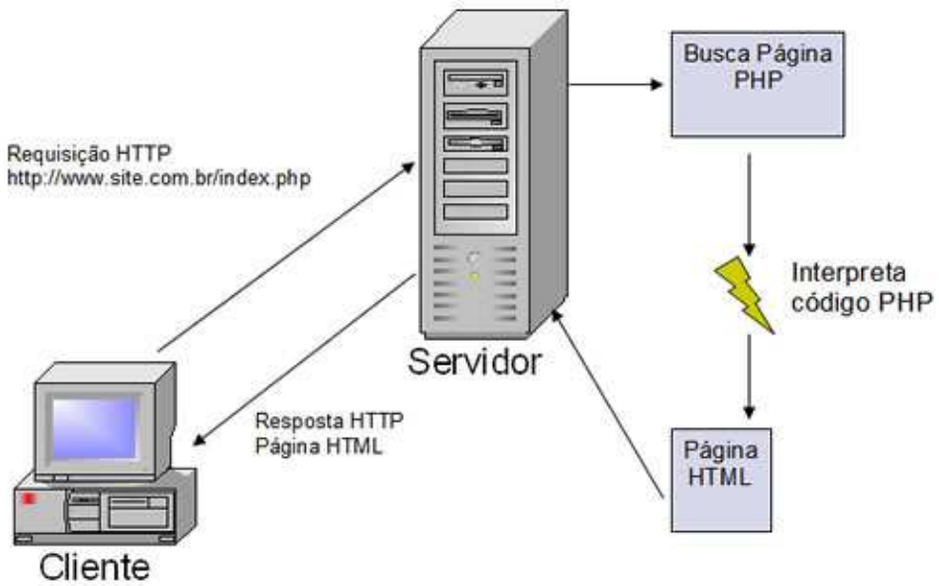


Figura 1 – Representação simplificada da arquitetura cliente-servidor

Analisando a representação da figura 1, vemos um computador representando a máquina cliente, que pode ser entendida como o computador dos usuários acessando a Internet. A máquina cliente faz uma requisição de serviço por meio do protocolo http – protocolo que nos permite visualizar os serviços disponíveis na Internet - para um servidor. Devemos lembrar que a conexão entre a máquina cliente e o servidor pode ser feita por meio da Internet (quando se trata de um serviço disponível na Internet), ou então por meio de uma rede interna (no caso das empresas que disponibilizam aplicações cliente-servidor dentro da sua estrutura organizacional).

O servidor, por sua vez, ao receber a requisição, independente se é um serviço interno ou externo, estará analisando a requisição feita e disponibilizando o serviço. Na figura 1, o serviço requisitado foi desenvolvido utilizando a linguagem PHP, portanto, o servidor irá localizar a rotina PHP, interpretar o código PHP da rotina utilizando o interpretador do PHP instalado no servidor e retornar ao cliente o resultado da rotina em formato de página HTML.

Veremos mais adiante, neste mesmo capítulo, como deve ser o ambiente para trabalharmos com PHP. Porém por meio da figura 1, já podemos verificar que será necessário um interpretador da linguagem PHP para retornar ao usuário da aplicação o resultado final esperado.

O surgimento das linguagens de programação para Internet foi justamente

durante a fase de repercussão das redes de computadores e da arquitetura cliente-servidor. As empresas fornecedoras de linguagens de programação e também os especialistas em serviços para Internet identificaram a demanda e assim começaram a nascer as linguagens voltadas para o desenvolvimento de aplicações cada vez mais dinâmicas para a Internet.

1.1 O QUE É PHP?

PHP pode ser traduzido para: “PHP Hipertext Preprocessor” e é uma linguagem de programação voltada para o desenvolvimento de aplicações para a Internet. Por ser uma linguagem interpretada, característica comum entre as linguagens para a Internet, é necessária a instalação de um interpretador no servidor da aplicação que irá interpretar as instruções da linguagem e retornar o resultado no formato de uma página HTML para o navegador do usuário.

A linguagem PHP é uma linguagem “Open Source”, ou seja, pode ser utilizada por qualquer usuário sem a necessidade de compra de licença para a utilização e desenvolvimento de aplicações. Também pode ser considerada como uma linguagem multiplataforma, pois o interpretador da linguagem tem compatibilidade com mais de um sistema operacional.

Essas características são as principais motivações que fazem com que a linguagem seja cada vez mais popular entre os desenvolvedores de aplicações para a Internet.

1.2 COMO SURGIU?

A linguagem PHP foi criada em 1994 por Ramus Lerdorf. Nessa época, a idéia de Ramus era apenas facilitar a publicação de informações no seu site pessoal.

Em 1995, o interpretador foi publicado e divulgado em uma comunidade “Open Source” e começou a ser utilizada por outros usuários.

Fique sabendo

Uma comunidade “Open Source” é uma comunidade de desenvolvedores que desenvolvem e disponibilizam aplicações e outros recursos que auxiliam no processo de desenvolvimento. Todos os recursos e aplicações disponibilizados por esse tipo de comunidade são livres de licença de uso.

Entre os anos de 1996 e 1997, a linguagem já estava bem popularizada entre as diversas comunidades livres e seus usuários e, nessa época, a origem do interpretador e suas atualizações ficaram sob a responsabilidade da comunidade livre da linguagem PHP, que até hoje é responsável pelas correções e atualizações da linguagem.

Hoje em dia, a linguagem é reconhecida mundialmente e muitas aplicações já foram desenvolvidas com ela. É a linguagem favorita dos acadêmicos e desenvolvedores que utilizam a plataforma Linux.

A versão mais recente e estável da linguagem é a versão 5, porém espera-se para o próximo ano, em 2008, o lançamento da versão 6, que já possui algumas distribuições na fase beta – fase de testes.

1.3 VANTAGENS DA LINGUAGEM PHP

As principais vantagens da linguagem são:

• É livre de licença – utilização gratuita;

• É multiplataforma, podendo ser utilizada na maioria dos sistemas operacionais – sistemas operacionais mais populares;

• A grande maioria dos provedores externos – empresas que vendem espaço para armazenamento de sites e aplicações na Internet – possuem a instalação do interpretador do PHP em seus servidores;

• Possui muitos recursos distribuídos em bibliotecas de fontes nas comunidades livres – recursos que auxiliam no desenvolvimento das aplicações;

• A comunidade livre de PHP está sempre desenvolvendo e distribuindo novos recursos nas bibliotecas de fontes para os desenvolvedores;

• A linguagem tem suporte e integração com os bancos de dados mais populares como MySQL, SQL Server, Oracle, DB2, entre outros;
• Permite o desenvolvimento de aplicações dinâmicas com recursos como: envio de e-mails, autenticação de usuários, conteúdo dinâmico entre outros.

Além das vantagens citadas acima, a linguagem também tem um diferencial importante em relação às demais: é uma linguagem de fácil aprendizagem.

1.4 COMO TRABALHAR COM PHP

Os primeiros passos para iniciar o desenvolvimento de aplicações com a linguagem PHP é a preparação do ambiente de desenvolvimento. Para isso, certifique-se de que você tem os requisitos mínimos e o ambiente do PHP instalado em seu computador.

O ambiente de desenvolvimento do PHP é composto de um servidor WEB, responsável pelo armazenamento e publicação das páginas, e o interpretador da linguagem PHP. O servidor WEB mais utilizado pelos desenvolvedores PHP é o Apache, que é um servidor gratuito, mas o PHP também pode ser configurado em outros servidores WEB como o famoso IIS da Microsoft. Além disso, o ambiente de desenvolvimento pode também ser composto de um gerenciador de banco de dados para o armazenamento dos dados da aplicação dinâmica. O banco de dados MySQL é o favorito para o desenvolvimento de aplicações para Internet com PHP.

Para facilitar a configuração do ambiente de desenvolvimento, existe atualmente uma ferramenta chamada Vertrigo que permite a instalação e configuração de todo o ambiente de desenvolvimento PHP. Essa ferramenta instala inclusive o banco de dados MySQL e permite que aplicações desenvolvidas com PHP sejam integradas com o MySQL.

Para instalar o Vertrigo, você pode fazer o download através do site **http://vertrigo.sourceforge.net/?lang=br** e seguir as instruções de instalação e configuração disponíveis no site. A instalação do Vertrigo é bem simples, possui a característica: "**Next, Next, Finish**" ou, em português: **Próximo Passo, Próximo Passo, Concluir**.

As informações do instalador também são bem simples e intuitivas, basta selecionar o idioma Português para instalação e seguir as instruções seguintes.

Após a instalação do Vertrigo você já possuirá o ambiente de desenvolvimento configurado para desenvolver suas aplicações PHP.

Para iniciarmos o desenvolvimento de aplicações em PHP, inicialize o servidor do Vertrigo por meio do menu **Iniciar >> Programas >> VertrigoServ**. Será aberta a janela do Vertrigo com as informações das ferramentas que foram instaladas e configuradas em seu computador. Para inicializar o servidor, basta clicar no botão: "**Hide this window and start server**" que significa: Esconda essa janela e inicie o servidor. A figura 2 ilustra a janela do VertrigoServ.

Figura 2 – Janela do VertrigoServ

Após iniciar o VertrigoServ, será mostrado um ícone do servidor na barra de status do Windows, ao lado do relógio. Esse ícone estará sempre visível enquanto o serviço do Vertrigo estiver ativo.

Finalmente, para validarmos que está tudo certo com o Apache – servidor WEB instalado para armazenar e publicar as páginas em PHP – abra um navegador e digite o seguinte comando na barra de endereços:

http://localhost

Após digitar o comando, será mostrada a tela do VertrigoServ. A figura 3 mostra a tela do VertrigoServ.

Figura 3 – Tela do VertrigoServ

Após digitar o comando, será mostrada a tela do VertrigoServ. A figura 3 mostra a tela do VertrigoServ.

Para visualizar a versão do interpretador do PHP instalado por meio do Vertrigo, clique no link: "**View phpinfo screen**", disponível no item: "**Tools and Links**". Após clicar no link citado, será mostrada uma tela com as informações do interpretador do PHP: versão e bibliotecas instaladas. A figura 4 ilustra a tela com as informações do PHP.

Figura 4 – Tela com as Informações do PHP

Após validar a configuração e o funcionamento do ambiente de desenvolvimento, precisamos escolher um editor de comandos para podermos dar início ao conhecimento e aprendizagem da linguagem PHP.

Atualmente, no mercado, há muitos editores que permitem aos desenvolvedores de PHP uma boa produtividade, porém muitos desses editores não possuem uma versão gratuita para uso. É importante ressaltar aqui, que você pode trabalhar com a linguagem PHP utilizando até um bloco de notas, contudo, como já informamos anteriormente, um editor PHP irá agilizar e melhorar a produtividade durante o desenvolvimento.

No decorrer do nosso curso, utilizaremos o editor: PHP Editor, desenvolvido em português e é uma ferramenta gratuita. Para obter a versão do PHP Editor, entre no site: **http://paginas.terra.com.br/informatica/php_editor/download.html** e clique no link: "**Mirror 2**" para fazer o download do arquivo. Após o download do arquivo, clique duas vezes em cima do arquivo executável para fazer a instalação que é bem simples, basta seguir as instruções da instalação. As figuras 5 a 8 ilustram o processo de instalação do PHP Editor.

Figura 5 – Tela 1 da Instalação do PHP Editor

Após clicar no arquivo de instalação do PHP Editor, será exibida a primeira tela com instruções para a instalação. Para prosseguir com a instalação, clique no botão avançar.

Em seguida, o instalador irá solicitar a confirmação do local para instalação do PHP Editor, geralmente o diretório: **C:\Arquivos de Programas\PHP Editor**. Selecione o diretório onde deverá ser feita a instalação por meio do botão: **Procurar**, ou deixe o caminho sugerido pelo instalador e clique no botão: **Instalar** para prosseguir a instalação.

Figura 6 – Tela 2 da Instalação do PHP Editor

Aguarde até o processo de instalação finalizar (figura 7). Quando o processo de instalação finalizar, será mostrada a tela com as informações da instalação. Para prosseguir, clique no botão: **Terminar.** Caso a opção: "**Executar o PHP Editor**" esteja selecionada – esta opção vem selecionada por padrão - o PHP Editor será aberto.

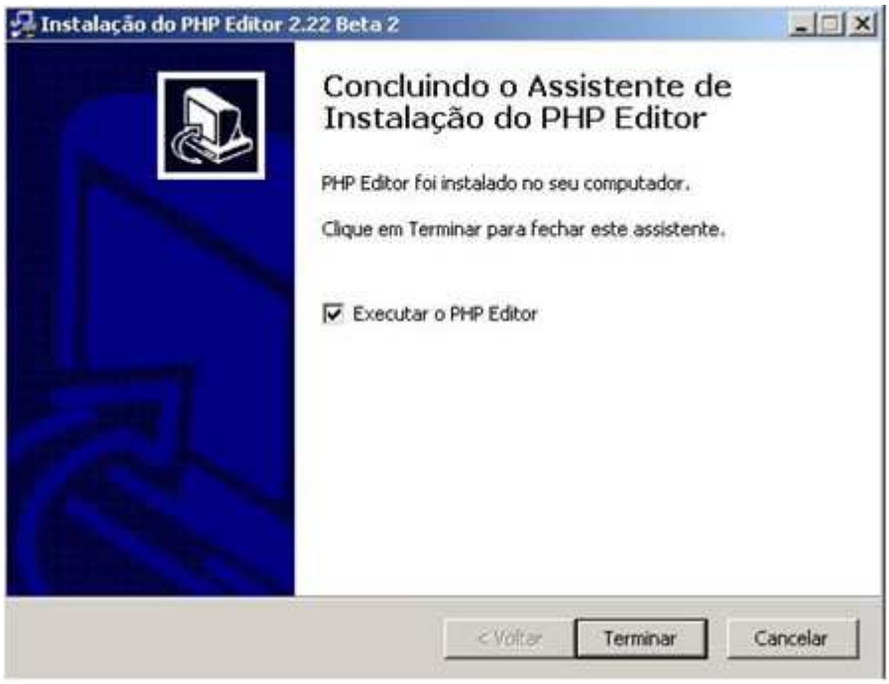


Figura 7 – Tela 3 da Instalação do PHP Editor

Ao clicar no botão: **Terminar**, o PHP Editor irá questionar se você **deseja relacionar todos os arquivos com extensão: .php com a ferramenta**. Ao confirmar esta opção, todos os arquivos criados com a extensão: .php sempre serão abertos com o PHP Editor. Esse é o último passo para a instalação da ferramenta. Concluído esse passo, já será possível trabalhar com a ferramenta. A figura 8 ilustra a ferramenta PHP Editor que utilizaremos durante o curso.

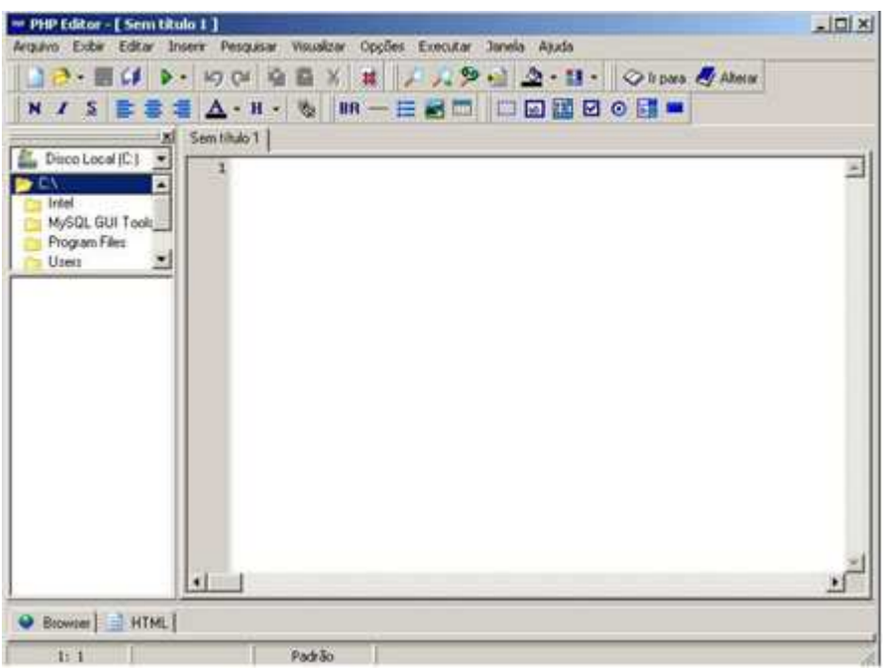


Figura 8 – Ferramenta PHP Editor

No próximo capítulo, vamos conhecer a sintaxe da linguagem PHP.

Síntese 

Nesta aula vimos:

• Uma introdução às linguagens para desenvolvimento de aplicações para Internet;

• A definição de PHP;

• Como surgiu a linguagem PHP;

• Quais as vantagens da linguagem PHP em relação às demais linguagens;

• Quais são os pré-requisitos para trabalhar com PHP.

Exercícios propostos 

1) Complete a frase abaixo com uma das alternativas, a ordem das palavras deve corresponder à lógica da frase.

A linguagem PHP possui um .... responsável pela transformação das páginas .... em páginas .... . Essas páginas .... são as páginas que o .... irá disponibilizar ao ....

a. Servidor Web, HTML, PHP, PHP, Usuário, Interpretador

b. Servidor Web, PHP, HTML, PHP, Interpretador, Usuário

c. Interpretador, PHP, HTML, HTML, Servidor WEB, Usuário

d. Interpretador, HTML, PHP, PHP, Servidor WEB, Usuário

2) Selecione a alternativa incorreta. O PHP pode ser definido como:

a. Uma linguagem de programação para Internet

b. Uma linguagem “Open Source”

c. Uma linguagem interpretada

d. Uma linguagem compilada

3) Selecione a alternativa correta. O PHP atualmente é mantido:

a. Pelo criador da linguagem

b. Por uma comunidade de softwares que necessita licença

c. Por uma comunidade de softwares livres

d. Por uma empresa privada

4) Qual das alternativas abaixo não é uma vantagem na utilização do PHP?

a. O PHP é multiplataforma

b. O PHP é “Open Source”

c. O PHP possui integração com diversos banco de dados

d. O PHP possibilita o desenvolvimento de aplicações estáticas

5) Para desenvolver aplicações com PHP, é necessário ter um ambiente de desenvolvimento. Quais dos itens abaixo são obrigatórios nesse ambiente de desenvolvimento (é permitida múltipla escolha):

a. Servidor WEB

b. Banco de Dados

c. Interpretador PHP

d. Editor PHP

Aula 2

INTRODUÇÃO À LINGUAGEM PHP

2

Tutu Virtual

Caro aluno(a)!

Nesta segunda aula você irá aprender a sintaxe básica da linguagem PHP e também vai desenvolver o primeiro exemplo com a linguagem. Após o desenvolvimento desse exemplo, será mostrado como testar o exemplo no ambiente de desenvolvimento configurado.

Bom estudo!

Objetivos da Aula 

Ao final desta aula, você deverá ser capaz de:

- Escrever uma instrução básica em PHP;
- Testar o funcionamento de uma página PHP.

Conteúdos da Aula 

Acompanhe os conteúdos desta aula. Se você preferir, assinale-os à medida em que for estudando.

- Conhecendo a sintaxe básica da linguagem;
- Desenvolvendo o primeiro exemplo com PHP;
- Exercícios propostos

1 CONHECENDO A SINTAXE BÁSICA DA LINGUAGEM 

Grande parte das aplicações desenvolvidas em PHP possui a seguinte estrutura:

• Interface com o usuário – aparência da aplicação representada pela parte HTML com imagens e textos estáticos e/ou dinâmicos;

• Scripts com instruções em PHP – representando a lógica de negócio da aplicação;

• Integração com banco de dados – representando os dados dinâmicos que serão disponibilizados na aplicação.

Essa estrutura também é utilizada em aplicações desenvolvidas com outras linguagens, pois esse é um padrão seguido por grande parte de desenvolvedores de aplicações para a Internet.

Portanto, é importante lembrar que, além de utilizarmos a linguagem PHP para desenvolvermos aplicações dinâmicas, também devemos conhecer os componentes básicos da linguagem HTML e os conceitos básicos de banco de dados.

1.1 Delimitando o código PHP

Para fazer a interpretação de um código PHP e disponibilizar no formato HTML, o interpretador procura um identificador que sinaliza o início do código PHP. Esse identificador é chamado de **delimitador**. Por meio do delimitador, o interpretador saberá onde é o início e o fim do código PHP.

O delimitador PHP é representado pelos símbolos:

• <?php ou somente <? = início do bloco de comandos;

• ?> = fim do bloco de comandos.

O exemplo abaixo demonstra como utilizar o delimitador PHP:

```
<?php
//instruções PHP;
?>
```

ou

```
<?
//instruções PHP;
?>
```

Esse é o primeiro conceito que devemos utilizar quando estamos desenvolvendo com PHP. Sempre que incluirmos alguma instrução PHP, é necessário incluir a instrução dentro do bloco de delimitadores.

Como vimos, podemos utilizar arquivos / páginas em PHP para montar a interface / leiaute da aplicação. Esse leiaute, muitas vezes, é criado com o auxílio da linguagem HTML que permite a inserção de estilos de fontes diferentes, imagens e outros componentes interativos.

Por esse motivo, é comum, quando desenvolvemos utilizando PHP, a interação entre instruções PHP e instruções HTML no mesmo arquivo. A função da linguagem HTML será formatar o leaiute da aplicação dinâmica, de acordo com a disposição das instruções PHP e HTML.

Vamos analisar o código da página PHP abaixo:

Nome da página: exemplo1.php

Conteúdo da página:

```
<html>
    <body>
        <b><? print "Olá Seja Bem Vindo ao PHP" ?> </b>
    </body>
</html>
```

Analisando o conteúdo da página, as instruções que estão em negrito são específicas da linguagem HTML e a instrução que não está em negrito é uma instrução PHP. O interpretador irá identificar a existência de uma instrução PHP por meio dos delimitadores: `<? ?>` e interpretará a instrução que foi definida. As instruções HTML foram incluídas na página apenas para facilitar a formatação do texto que será mostrado, nesse caso, um texto em negrito, portanto, não terá nenhuma interferência

1.2 Separador de instruções

Outro item importante que devemos utilizar no desenvolvimento com PHP é o separador das instruções. Cada instrução definida em uma página em PHP deve ser finalizada com um ponto e vírgula “;”.

O interpretador do PHP lê as instruções de uma página linha a linha. Quando ele identifica o final de uma instrução, interpreta a instrução e, caso tenha sucesso, irá disponibilizando o resultado parcialmente. Isso significa que a página HTML, resultante do processo de interpretação, será montada aos poucos, conforme a finalização da interpretação das instruções. Esse processo, muitas vezes, é transparente aos olhos do usuário, sendo possível sua identificação somente quando há lentidão na montagem da página HTML. A lentidão pode ser causada por uma série de fatores como: problema na rede, problema no computador ou ainda problema na instrução PHP definida.

1.3 Nomes de variáveis

Diferente da maioria das linguagens de desenvolvimento, tanto para a Internet quanto para aplicações diversas, a linguagem PHP possui uma característica específica para declaração de nomes de variáveis. Ao declarar um nome para uma variável, é necessário incluir o símbolo: “$” antes da variável. Esse símbolo irá identificar que o nome a seguir é uma variável. O interpretador PHP somente reconhecerá uma variável se a mesma iniciar com o símbolo “$”.

Outra característica importante a ser ressaltada é que o PHP é “case-sensitive”, ou seja, ele faz distinção entre maiúsculas e minúsculas. Veja no exemplo abaixo a utilização de variáveis:

```
<?
$nome = “José”;
print “Olá ” . $nome; // vai mostrar José
print “<br>”;
print “Olá ” . $NOME; // não vai mostrar nada
?>
```

1.4 Comentários

Como já sabemos, os comentários são muito importantes durante o desenvolvimento das aplicações, por meio deles teremos como compreender o que faz uma determinada instrução ou bloco de instruções. Assim, quando for necessário executar alguma manutenção na instrução, será possível fazê-la de forma mais rápida.

Em PHP há dois tipos de comentários:

• Comentários de linha = //

• Comentários de bloco = /* o comentário fica aqui */

Os comentários de linha são utilizados quando precisamos fazer um comentário rápido, que ocupe apenas uma linha. Já os comentários de bloco são utilizados quando o comentário a ser feito possui mais de uma linha de texto. Veja o exemplo:

EX.

```
<?
//esse é um comentário de linha, fica apenas no código
print "Essa frase está abaixo do comentário de linha";
//mais um comentário de linha, a linha abaixo utilizada o
comando <BR> do //HTML, para pular duas linhas entre as
frases que serão mostradas na tela
print "<br><br>";
/*
Esse é um comentário de bloco.
Pode ter mais de uma linha de comentário, porém assim como
o comentário de linha, também somente fica no código e nun-
ca será interpretado e mostrado na página final em HTML
para o usuário.
*/
print "Essa frase será mostrada duas linhas abaixo da pri-
meira frase e abaixo do comentário de bloco";
?>
```

1.5 Imprimindo código na tela

Como já vimos nos exemplos anteriores, utilizamos o comando: print para imprimir o resultado das instruções na tela do navegador. Em PHP existem dois comandos que permitem imprimir textos e códigos na tela:

- print
- echo

Ambos possuem a mesma função: imprimir na tela, e não existe nenhum tipo de distinção entre eles, portanto, em seus códigos, você poderá utilizar o comando que lhe parece mais agradável. No próximo tópico criaremos e testaremos o primeiro exemplo e ficará mais clara a função do comando de impressão na tela.

2 DESENVOLVENDO O PRIMEIRO EXEMPLO COM PHP 

Agora que já conhecemos a sintaxe básica do PHP, vamos desenvolver nosso primeiro exemplo utilizando o PHP Editor para criar uma página em PHP e o ambiente de desenvolvimento do Vertrigo para testar o funcionamento da página criada.

Primeiro devemos iniciar o servidor do Vertrigo.

Vá até o menu **Iniciar >> Programas >> VertrigoServ** e selecione o item: **VertrigoServ**. Ao clicar nesse item, será mostrada a tela de inicialização do serviço do Vertrigo, veja a figura 2 ilustrada no capítulo 1 desta apostila. Clique no botão: "**Hide this window and Start Server**" para iniciar o serviço. Em seguida, abra o navegador da Internet e informe o comando abaixo na barra de endereços:

http://localhost

Será mostrada a página do VertrigoServ (figura 3 do capítulo 1 desta apostila). Seguidos esses passos, já podemos iniciar o desenvolvimento do nosso exemplo.

Para conseguirmos testar os nossos exemplos, será necessário disponibilizar os arquivos dentro do servidor WEB Apache**,** instalado pelo VertrigoServ. Dessa forma, todos os arquivos de exemplo deverão estar dentro de um diretório chamado: "www" que está dentro do diretório de Instalação do VertrigoServ. Se você optou pelo

caminho padrão de instalação para VertrigoServ no momento da instalação, o diretório “www” estará disponível em **C:\Arquivos de Programas\VertrigoServ\www**. Se você optou por outro caminho de instalação, procure o caminho que você informou no momento da instalação e procure pelo diretório: “www”.

Após encontrar o diretório: “www”, crie um novo diretório dentro dele chamado: “apostila_php”. Todos os arquivos desenvolvidos em PHP durante o curso deverão ser disponibilizados nesse diretório para realizar os testes.

Com o diretório criado, vamos agora abrir o PHP Editor para criar nosso primeiro exemplo. Os outros exemplos mostrados na apostila poderão ser testados também da mesma forma que faremos com este exemplo. Abra o PHP Editor e, na barra de endereços mostrada, procure pelo diretório: “apostila_php” criado dentro do diretório: “www” do VertrigoServ. A figura 9 mostra onde deve ser selecionado o diretório: “apostila_php”.

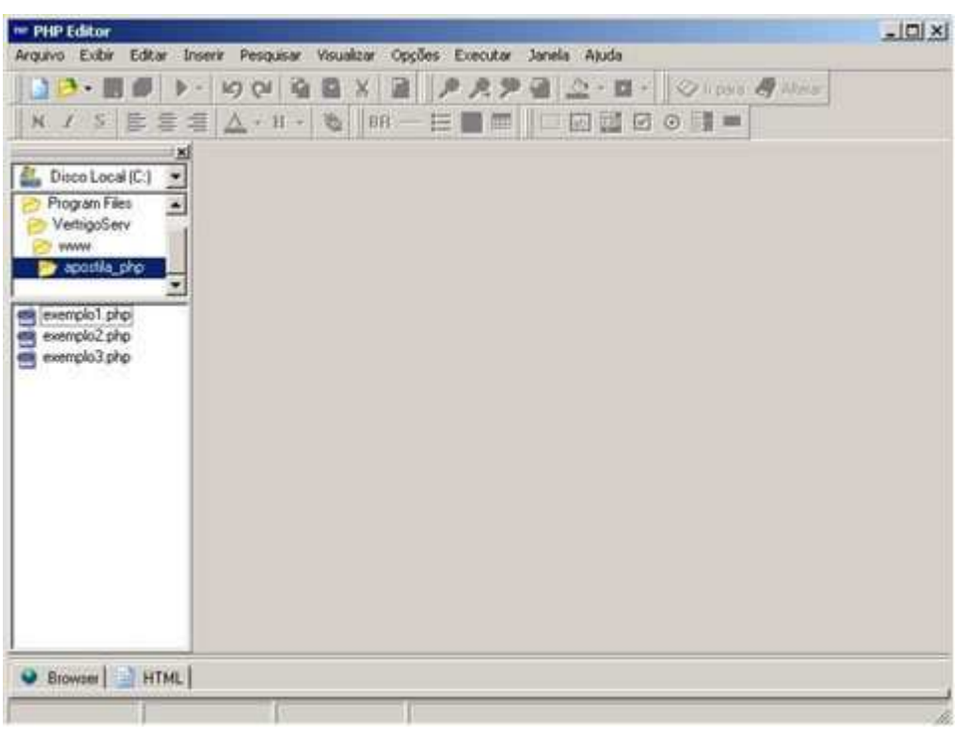


Figura 9 – Selecionando diretório: apostila_php no PHP Editor

Com o diretório: “apostila_php” selecionado, já é possível criar o arquivo PHP que possuirá a instrução do primeiro exemplo. Crie um novo arquivo clicando no ícone de criação de arquivo ou acessando o menu: **Arquivo >> Novo**. O PHP Editor irá questionar qual o tipo de arquivo que você deseja criar, selecione o tipo de arquivo:

/PHP e clique no botão: **OK**. A figura 10 ilustra a janela com as opções de arquivos que podem ser criados.

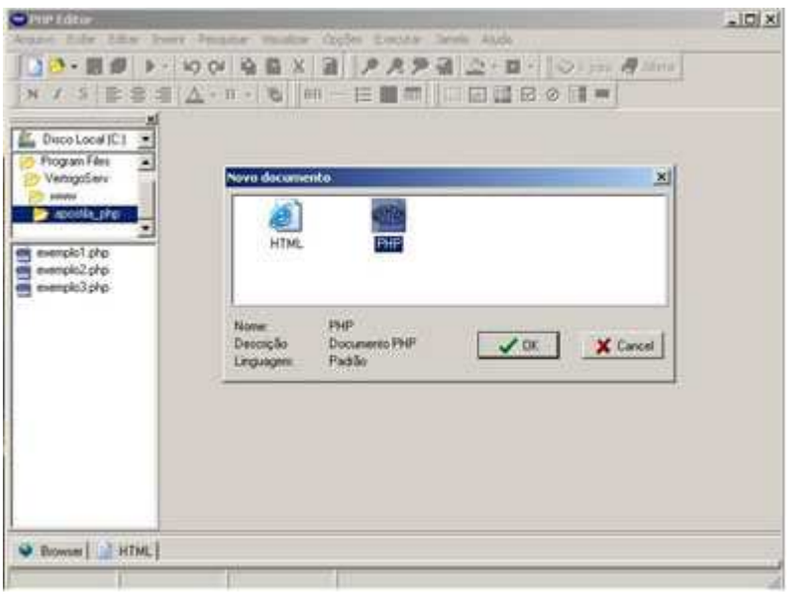


Figura 10 – Criando arquivos no PHP Editor

A seguir será mostrada a estrutura de um arquivo em PHP. A figura 11 ilustra o formato de um novo arquivo em PHP criado pelo PHP Editor.

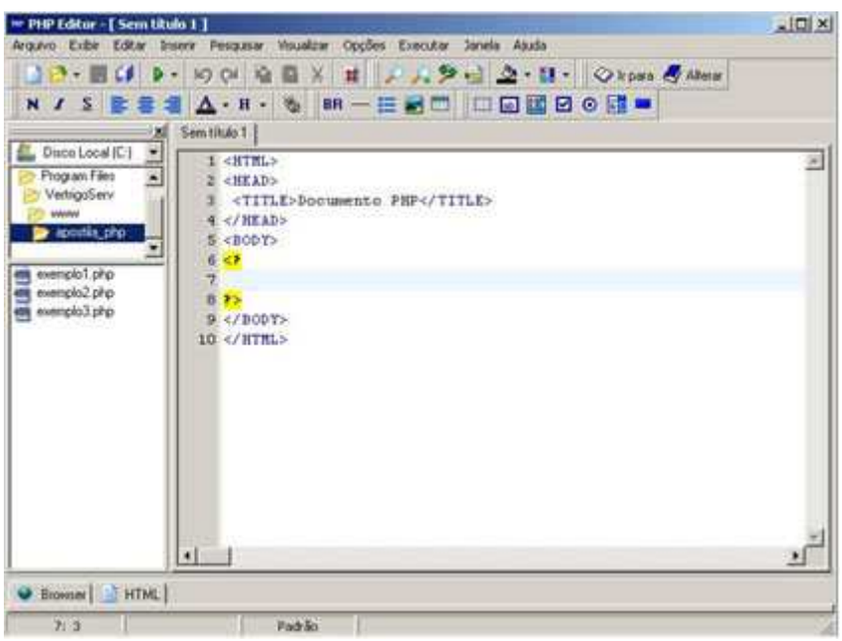


Figura 11 – Novo arquivo PHP

As instruções em PHP do exemplo deverão ficar dentro dos delimitadores do PHP: <? e ?>. Os demais comandos que aparecem no arquivo são comandos HTML

utilizados para a formatação da página HTML final, criada pelo interpretador PHP, ao final da interpretação de todas as instruções PHP contidas no arquivo.

O comando: <TITLE></TITLE> do HTML é o responsável por alterar o título da página PHP que aparece na barra azul do navegador de Internet. Assim, vamos alterar o texto: “Documento PHP”, que está dentro do comando HTML, para “Primeiro Exemplo em PHP”. Isso fará com que o texto informado seja mostrado como título da página no navegador.

Feita a alteração, vamos colocar as instruções PHP do exemplo dentro dos delimitadores PHP. Digite a seguinte instrução:

EX.

```
<?
//Esse é o primeiro exemplo em PHP que vamos testar
//Iremos imprimir o texto: “O teste funcionou ok!”

//Primeiro vamos criar uma variável que vai receber o tex-
to:
$texto = “O teste funcionou ok!”;

//Agora vamos imprimir esse texto na tela utilizando o co-
mando print:
print $texto;
?>
```

Depois salve o arquivo clicando no ícone do **disquete** ou selecionando o menu: **Arquivo >> Salvar**. Salve o arquivo com o nome: **exemplo4.php**. A figura 12 ilustra como ficará o código arquivo **exemplo4.php**.

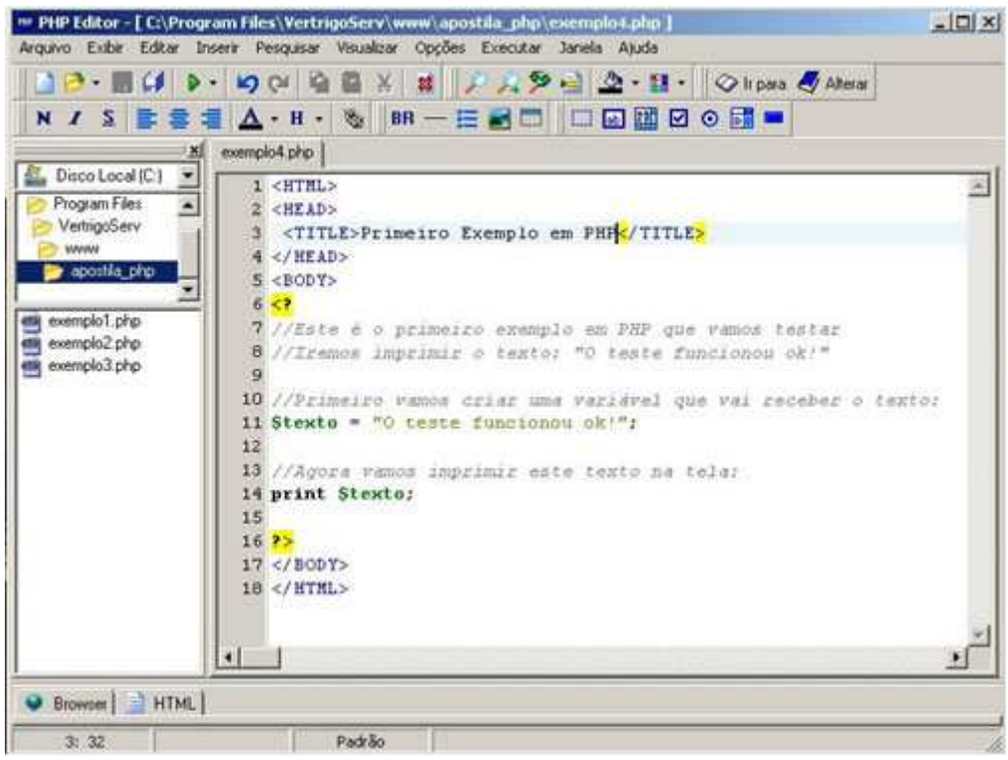


Figura 12 – Arquivo exemplo4.php

Para testarmos a execução do arquivo de exemplo criado, devemos acessar o navegador de Internet e informar o seguinte comando na barra de endereços:

http://localhost/apostila_php/exemplo4.php

Ao executar esse comando no navegador, o arquivo: exemplo4.php será interpretado pelo interpretador PHP, em seguida, transformado em uma página HTML e, ao final, o conteúdo da página HTML gerado é disponibilizado pelo servidor WEB Apache do ambiente VertrigoServ. A figura 13 ilustra o resultado do teste.

Figura 13 – Testando o exemplo4.php

Todos os arquivos desenvolvidos em PHP deverão ser testados dessa forma. Para os nossos exemplos, sempre criaremos os arquivos PHP dentro do diretório: “apostila_php”, porém para suas próximas aplicações, crie um novo diretório e disponibilize os arquivos dentro desse diretório para testes. Para testar no navegador, sempre coloque o nome do diretório criado após o comando: http://localhost/ e informe o nome do arquivo que deseja testar. Seguem alguns exemplos de chamadas de arquivos para testar no navegador.

1. Testar o arquivo: principal.php disponível no diretório: aplicacao1:

http://localhost/aplicacao1/principal.php

2. Testar o arquivo: teste.php disponível no diretório: aplicacao2:

http://localhost/aplicacao2/teste.php

3. Testar o arquivo: exemplo1.php disponível no diretório: apostila_php:

http://localhost/apostila_php/exemplo1.php

Síntese 

Nesta aula vimos:

• A sintaxe básica da linguagem PHP;

• Como desenvolver e testar um exemplo básico em PHP.

Exercícios propostos 

1) Qual das alternativas abaixo não faz parte da estrutura básica de uma aplicação desenvolvida com PHP:

a. Interface com usuário

b. Página com instruções PHP

c. Interface com o banco de dados

d. Integração com banco de dados

2) Para utilizar uma instrução PHP em uma página, é necessário (apenas uma alternativa está correta):

a. Utilizar os delimitadores PHP representados pelos caracteres: <? E ?>

b. Declarar uma variável

c. Utilizar comandos HTML

d. Montar a instrução PHP no corpo da página HTML

3) Todas as instruções PHP devem ser finalizadas com (apenas uma alternativa está correta):

a. Ponto (.)

b. Ponto e vírgula (;)

c. Vírgula (,)

d. Todas as alternativas estão incorretas

4) Ao declarar uma variável é importante observar (é permitida múltipla escolha):

a. Se o símbolo “$” foi utilizado no início do nome da variável

b. Se foi declarado o tipo de dados da variável

c. Se a variável está com letras maiúsculas e minúsculas por causa do case-sensitive

d. O tipo de valor que a variável está recebendo

5) O comando PHP que imprime instruções na tela é o (é permitida múltipla escolha):

a. printf

b. print

c. printout

d. echo

Aula 3

TIPOS DE DADOS SUPORTADOS PELA LINGUAGEM PHP

3

Caro aluno(a)!

Nesta aula você estudará sobre os tipos de dados suportados pela linguagem PHP. Mostraremos como utilizar os tipos de dados, como transformar uma variável que possui um tipo de dado para outro tipo de dados e vamos também testar alguns exemplos utilizando os tipos de dados vistos.

Boa aula!

Objetivos da Aula 

Ao final desta aula, você deverá ser capaz de:

- Identificar quais são os tipos de dados utilizados pelo PHP;
- Utilizar corretamente os tipos de dados nas variáveis;
- Transformar um tipo de dado em outro tipo de dado;
- Testar o funcionamento dos tipos de dados.

Conteúdos da Aula 

Acompanhe os conteúdos desta aula. Se você preferir, assinale-os à medida em que for estudando.

- Introdução aos Tipos de Dados do PHP;
- Transformando Tipos de Dados;
- Exercícios propostos.

1 INTRODUÇÃO AOS TIPOS DE DADOS DO PHP 

A linguagem PHP, assim como em outras linguagens para desenvolvimento de aplicações para Internet, não exige a declaração do tipo de dados no momento da criação de uma variável. Mas se não informamos qual é o tipo de dado de uma variável, como o PHP consegue trabalhar com os tipos de dados e executar cálculos matemáticos, por exemplo?

A linguagem PHP consegue fazer a distinção entre os tipos de dados, por meio do conceito: tipagem automática, utilizada pelo interpretador PHP. Mas o que é a tipagem automática?

A tipagem automática é o processo que o interpretador PHP utiliza para analisar qual é o valor que está sendo atribuído a uma variável e setar o tipo de dados desta variável de acordo com o tipo de dados do valor atribuído à variável. Veja o exemplo abaixo:

```
<?
//A variável está recebendo um valor inteiro:
$numero1 = 10;

//A segunda variável está recebendo um valor com casas de-
cimais:
$numero2 = 10.5;

//Vamos imprimir o valor das duas variáveis e testar:
print $numero1;
print " - ";
print $numero2;
?>
```

Se você testar o exemplo acima, irá verificar que os valores atribuídos às duas variáveis foram impressos corretamente na tela. Até aqui tudo certo, mas como podemos validar que a variável: `$numero1` está atribuída com o tipo de dados: inteiro e a variável: `$numero2` está atribuída com o tipo de dado: número com casas decimais? Veremos nos próximos capítulos da apostila que a linguagem PHP fornece uma série de funções que permitem verificar e executar ações diversas. Nesse exemplo, portanto, para verificarmos o tipo de dado de uma variável, poderemos utilizar uma função que o PHP disponibiliza e que mostra qual é o tipo de dado de uma variável. A função que utilizaremos para isso é a função: `gettype(nome_variavel)`. Essa função irá retornar o tipo de dado de uma variável.

Vamos então incluir as linhas abaixo no nosso exemplo:

```
<?
//vamos utilizar o comando <BR> do HTML para pular duas
linhas
print "<BR><BR>";
//em seguida vamos imprimir o tipo de dado da variável
$numero1
print "Tipo de dado da variável numero1 : " .
gettype($numero1);
//vamos utilizar o comando <BR> do HTML para pular mais
duas linhas
print "<BR><BR>";
//depois vamos imprimir o tipo de dado da variável $nume-
ro2
print "Tipo de dado da variável numero2 : " .
gettype($numero2);
?>
```

O resultado do teste é mostrado na figura 14:

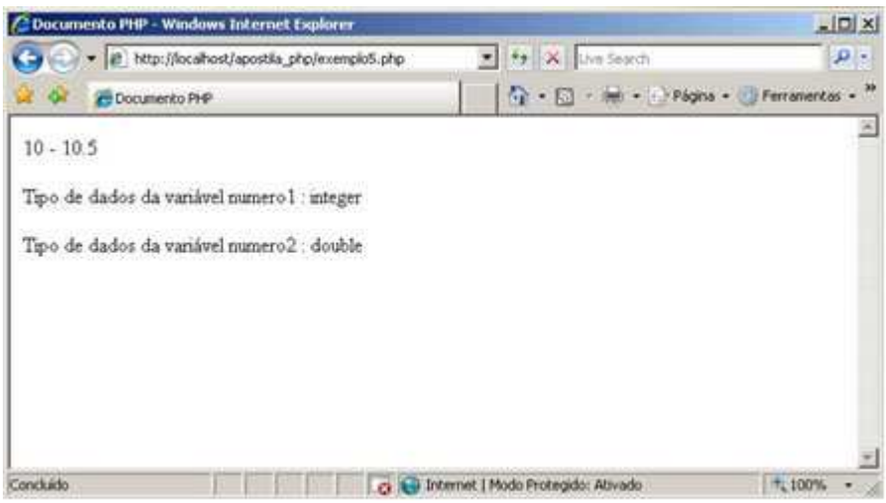


Figura 14 – Testando o exemplo5.php

De acordo com o resultado, podemos verificar que a variável `$numero1` assumiu o tipo de dado: **integer** que suporta números inteiros e, a variável `$numero2` assumiu o tipo de dado: **double** que suporta números com casas decimais.

Assim, mesmo não informando na instrução de declaração de uma variável qual é o tipo de dado que ela deverá assumir, o interpretador do PHP irá selecioná-lo automaticamente, de acordo com o tipo de dado do valor da variável. Vamos verificar agora quais são os tipos de dados que a linguagem PHP fornece para o desenvolvimento de aplicações.

A linguagem PHP tem suporte para os seguintes tipos de dados:

- Números inteiros
- Números com casas decimais
- Textos
- Booleanos
- Array
- Objeto

Vamos analisar, um a um, cada tipo de dado.

1.1 Números Inteiros

É considerado inteiro um número do conjunto de inteiros da matemática: Z = {..., -2, -1, 0, 1, 2, ...}. Os números inteiros podem ser utilizados em decimal (base 10), hexadecimal (base 16) ou octal (base 8), ou ainda como um número inteiro positivo ou negativo. Veja no exemplo abaixo a forma de aplicação dos inteiros:

```
<?
$inteiro1 = 1234; // número decimal
$inteiro2 = -123; // um número negativo
$inteiro3 = 0123; // número octal (equivalente a 83 em de-
cimal)
$inteiro4 = 0x1A; // número hexadecimal (equivalente a 26
em decimal)
?>
```

Os números inteiros hexadecimais devem sempre iniciar com o valor: “0x” e os números inteiros octal devem sempre iniciar com o valor: “0”. Esses valores, no início, são os responsáveis pela interpretação correta do formato da variável inteiro pelo interpretador PHP.

O tipo de dados que suporta números inteiros em PHP é chamado de: **integer**.

1.2 Números com casas decimais

O tipo de dado que suporta números com casas decimais em PHP é chamado de: **double**. Esse tipo de dado permite uma precisão de até 14 decimais. Os números com casas decimais podem ser especificados da seguinte forma:

```
<?
$numero1 = 1.234;
$numero2 = 23e4; // representa o número 230.000
?>
```

1.3 Textos

O tipo de dado que suporta textos em PHP é chamado de: **string**. Para atribuir valores do tipo texto em uma variável PHP pode-se utilizar tanto aspas duplas quanto aspas simples. Existe, porém, uma diferença na interpretação dos valores das variáveis dependendo do tipo de aspas utilizado. As diferenças são:

- **Atribuindo valores às variáveis do tipo: string com aspas simples:**
  - O valor da variável será exatamente o texto que está entre as aspas simples.
- **Atribuindo valores às variáveis do tipo: string com aspas duplas:**
  - O valor da variável poderá conter outra variável entre as aspas duplas ou ainda, caracteres de escape que permitem a formatação dos dados.

Para compreender melhor a diferença entre a atribuição de valores com aspas simples e com aspas duplas, vamos verificar o exemplo:

EX.

```
<?
/* atribuindo texto com aspas simples, vai ignorar o caracter de escape \n e mostrar como parte do texto */
$texto1 = ' Esse é o texto atribuído com aspas simples, vai ignorar o \n  informado ';
print $texto1;

//vamos utilizar o comando <br> do HTML para pular uma linha
print "<br>";

/* atribuindo texto com aspas duplas, vai interpretar o caracter de escape \n  */
$texto2 = " Esse é o texto atribuído com aspas simples, vai interpretar o \n informado ";
print $texto2;
?>
```

Analisando o resultado do exemplo, vamos verificar que a atribuição feita com aspas simples mostra o texto como foi escrito. O caracter de escape: \n é completamente ignorado. Já utilizando a atribuição com aspas duplas, o caracter de escape é interpretado, portanto, não é mostrado como um texto na tela.
Os caracteres de escape mais utilizados em PHP são:

Sintaxe	Significado
\n	Nova linha
\r	Retorno de carro (semelhante a \n)
\t	Tabulação horizontal
\\	A própria barra (\ )
\$	O símbolo $
\’	Aspa simples
\”	Aspa dupla

Verifique que o caracter de escape: \n, utilizado no exemplo, executa a ação de quebra de linha. Analisando novamente o resultado do nosso exemplo, executando o exemplo por meio do navegador, vamos identificar que a quebra de linha não foi feita. Por que não foi feita a quebra de linha?

A resposta correta para essa questão é: por causa do HTML. A linguagem HTML sozinha não entende a ação dos caracteres de escape. É necessário um comando HTML para formatar o texto de acordo com os caracteres de escape informados. O comando em HTML que faz a formatação do texto com caracteres de escape é o <pre></pre>. Vamos então melhorar o nosso exemplo incluindo uma nova impressão da variável utilizando o comando HTML que interpreta os caracteres de escape:

EX.

```
<?
//verificar texto2 com a formatação de quebra de linha do \n
print “<pre>$texto2</pre>”;
?>
```

Executando novamente o exemplo completo no navegador, podemos verificar que a quebra de linha foi feita corretamente. Inclusive, se analisarmos a última linha do exemplo, a variável `$texto2` foi incluída dentro do comando `<pre></pre>`

e também dentro das aspas duplas. O texto que estava dentro da variável `$texto2` foi mostrado corretamente. Isso acontece porque o interpretador PHP consegue interpretar uma variável mesmo que esteja dentro de um texto com aspas duplas. Já com aspas simples isso não seria possível. Vamos validar essa informação tentando executar a última instrução de comando do nosso exemplo trocando as aspas duplas por aspas simples:

ex.

```
<?
//verificar a impressão na tela com aspas simples
print '<pre>$texto2</pre>';
?>
```

Executando o exemplo novamente, o resultado será a ilustração da figura 15:

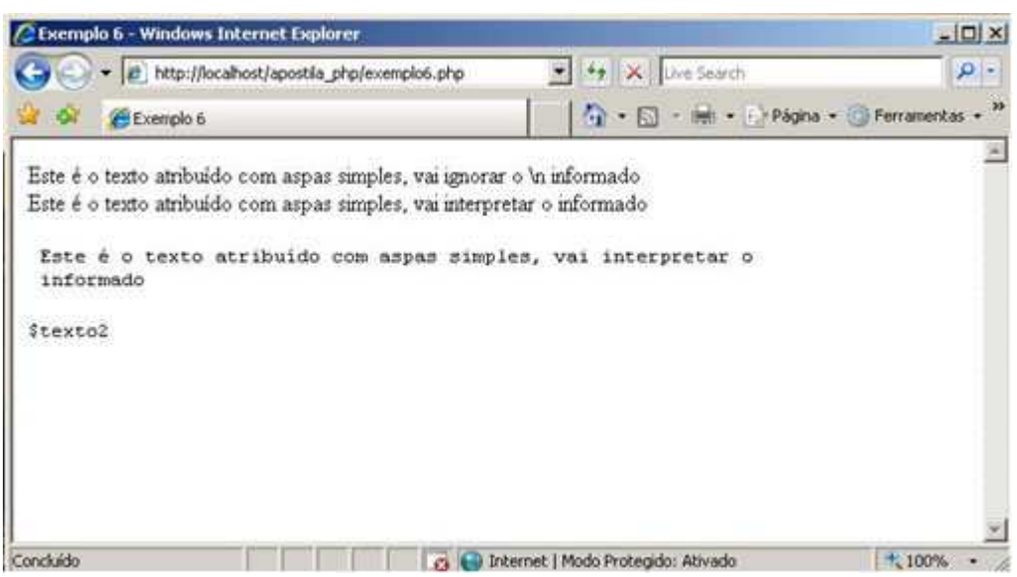


Figura 15 – Testando o exemplo6.php

Na última linha impressa, o interpretador PHP não conseguiu interpretar a variável `$texto2`, pois a variável foi colocada entre **aspas simples**.

1.4 Booleanos

O dado boolean é o tipo mais simples que existe no PHP. É um tipo de dado comum na maioria das linguagens de programação, pois recebe somente dois valores:

1. Verdadeiro = True ou 1
2. Falso = False ou 0

Esse tipo de dado é muito utilizado para validações do resultado de outras instruções. Utilizá-lo-emos bastante nos exemplos dos próximos capítulos.

1.5 Array

O tipo de dado array é utilizado como um vetor na linguagem PHP. Vetores são mapeamentos de valores que ficam armazenados no disco – HD – do computador. Podemos criar uma variável do tipo array e utilizar as posições do mapeamento do disco para armazenar mais de um valor. A quantidade das posições é informada na criação do array.

Existem duas maneiras de criar uma variável do tipo array, veja o exemplo.

```
<?
//forma 1 - criando um array e setando o valor do array
$array1[1] = "Posição 1 do array1";
$array1[2] = "Posição 2 do array1";
$array1[3] = "Posição 3 do array1";
//imprimindo os dados do array1
print " $array1[1] -  $array1[2] -  $array1[3] <BR>";
//forma 2 - criando um array e setando o valor do array
$array2 = array(1=>"Posição 1 do array2", 2=> "Posição 2 do
array2", 3=>"Posição 3 do array2");
//imprimindo os dados do array1
print " $array2[1] -  $array2[2] -  $array2[3] <BR>";
?>
```

No exemplo, criamos dois arrays utilizando as duas formas possíveis, mas utilizamos como chave da posição números inteiros. Em PHP podemos também criar arrays utilizando textos como chave da posição do mapeamento dos dados.

Veja o exemplo.

```
<?
//criando array com chave de posição do tipo texto
$arrayTexto[“Nome”] = “José Aparecido”;
$arrayTexto[“Idade”] = 22;
//imprimindo os dados do arrayTexto
print $arrayTexto[“Nome”];
print “<br>”;
print $arrayTexto[“Idade”];
?>
```

Além de podermos utilizar chave de posição do tipo texto, podemos também atribuir qualquer tipo de dado suportado pelo PHP para a posição de uma variável do tipo: array. Trabalharemos mais com arrays nos próximos capítulos do nosso livro-texto.

1.6 Objeto

Desde a versão 4 a linguagem PHP tem suporte à programação orientada a Objetos. A partir dessa versão todos os conceitos utilizados pela Orientação a Objetos podem ser utilizados em uma aplicação desenvolvida em PHP. Na versão 5 porém, esses conceitos foram reforçados e melhorados na linguagem e ficou mais fácil desenvolver aplicações utilizando os paradigmas da Orientação a Objetos.

No capítulo 7 da apostila analisaremos com mais detalhes a programação orientada a objetos em PHP. O estudo de caso que desenvolveremos também utilizará os conceitos da Orientação a Objetos.

2 TRANFORMANDO OS TIPOS DE DADOS 

Como o interpretador PHP utiliza a tipagem automática para indicar o tipo de dado de uma variável, pode acontecer que o tipo de dado indicado não seja o mais

adequado ou o esperado por algum procedimento da aplicação. Por esse motivo, há possibilidade de alterarmos o tipo de dado de uma variável, para que os procedimentos possam ser executados corretamente.

A linguagem PHP disponibiliza três formas para transformação dos tipos de dados de uma variável.

2.1 Coerção

A Coerção é uma forma utilizada automaticamente pelo interpretador PHP. Um exemplo de coerção é utilizado quando uma variável recebe o resultado de uma operação matemática entre dois números, sendo um número inteiro e outro número com casas decimais. A variável que receberá o valor terá seu tipo de dados alterado para números com casas decimais ou o tipo de dado: double. Veja o exemplo.

```
<?
// exemplo de coerção
$inteiro = 1;
$real = 2.5;

print " tipo de dados da variável $inteiro = " .
gettype($inteiro) . "<br>";
print " tipo de dados da variável $real = " . gettype($real)
. "<br>";

$resultado = $inteiro;
print " valor da variável resultado = $resultado <br>";
print " tipo de dados da variável resultado = " .
gettype($resultado) . "<br>";

// gerando a coerção do tipo de dado da variável resultado
com a operação //matemática de soma
$resultado = $inteiro+$real;
```

```
print " valor da variável resultado = $resultado <br>";
print " tipo de dado da variável resultado = " .
gettype($resultado) . "<br>";
?>
```

Ao analisarmos o resultado do exemplo, podemos verificar que o tipo de dado da variável `$resultado` é indicado como: **integer**, quando a variável `$resultado` recebe a variável `$inteiro`. A variável resultado, porém, passou a assumir o tipo de dados: **double**, quando foi atribuído o valor da operação matemática entre a variável `$inteiro` e a variável `$real` para a variável `$resultado`. Isso ocorre porque a operação matemática de somar dois números: um número inteiro e o outro número real – com casas decimais – resulta opera em um número real.

Então a operação é utilizada automaticamente pelo interpretador PHP e não temos como utilizar manualmente a operação de coerção.

2.2 Transformação explícita de tipos de dados

A transformação explicíta pode ser utilizada pelos desenvolvedores e é considerada a mais básica de transformação de tipos de dado das variáveis. Para utilizá-lo precisamos dos "cast", conversores de um tipo de dado para outro tipo de dado.

Os tipos de cast permitidos são:

- (int), (integer) = altera o tipo de dado para integer;
- (real), (double), (float) = altera o tipo de dado para float;
- (string) = altera o tipo de dado para string;
- (array) = altera o tipo de dado para array;
- (object) = altera o tipo de dado para objeto.

Veja o exemplo.

```
<?
// exemplo utilizando cast
$numero = 1;
```

```
//imprimir o valor e o tipo de dado da variável $numero
print " valor da variável = $numero <br>";
print " tipo de dado da variável = " . gettype($numero) .
"<br>";

// fazer a conversão do tipo de dado da variável $numero
utilizando cast
$numero = (double) 1;

print " valor da variável depois do cast = $numero <br>";
print " tipo de dado da variável depois do cast = " .
gettype($numero) . "<br>";
?>
```

Executando o exemplo acima, podemos verificar que a variável `$numero` passou a assumir o tipo de dado: **double**.

2.3 Transformando tipos de dados com a função settype

Essa forma também pode ser utilizada pelos desenvolvedores, basta que a função **settype** da linguagem PHP seja informada na instrução. O exemplo a seguir ilustra a utilização da função **settype(nome_variavel)**:

EX.

```
<?
// exemplo utilizando a função settype
$texto = "10";

//imprimir o valor e o tipo de dado da variável $texto
print " valor da variável = $texto <br>";
print " tipo de dado da variável = " . gettype($texto) .
"<br>";
```

```
// fazer a conversão do tipo de dado da variável $texto
utilizando settype
settype($texto,integer); // convertendo o tipo de dado para
integer

print " valor da variável depois da transformação = $texto
<br>";
print " tipo de dado da variável depois da transformação =
" . gettype($texto) . "<br>";
?>
```

No exemplo, foi feita a transformação de um tipo de dado texto – **string** – para o tipo de dado inteiro – **integer** – utilizando a função **settype**. Há várias funções disponíveis na linguagem PHP e, durante o curso, utilizaremos várias dessas funções. Porém, para visualizar todas as funções disponíveis na linguagem, você deve consultar o **Manual PHP** disponível na instalação do PHP Editor. Para acessar o manual vá até o menu: I**niciar >> Programas >> PHP Editor** e selecione o item **Manual PHP**.

Síntese 

Nesta aula vimos:

• Os tipos de dados suportados pela linguagem PHP;

• Como podemos transformar o tipo de dado de uma variável para outro tipo de dado;

• Testes com os tipos de dados da linguagem PHP.

Exercícios propostos 

1) Selecione a alternativa incorreta. A linguagem PHP suporta os seguintes tipos de dados:

a. Números

b. Textos

c. Datas

d. Lista de Valores

2) O tipo de dado que permite duas formas de atribuição é o (apenas uma alternativa está correta):

a. Integer

b. Double

c. Boolean

d. String

3) O tipo de dado da variável abaixo será (apenas uma alternativa está correta):

$numero = “1”;

a. Integer

b. Boolean

c. String

d. Double

4) Para utilizar caracteres de escape na impressão de instruções em PHP é necessário (apenas uma alternativa está correta):

a. Utilizar o comando HTML

b. Utilizar o comando HTML

c. Utilizar o comando HTML <pre></pre>

d. Todas as alternativas estão incorretas

5) A instrução abaixo irá (apenas uma alternativa correta):

$numero = (double) “1”;

a. Transformar a variável para String

b. Transformar a variável para Double

c. Transformar a variável para Integer

d. Transformar a variável para Boolean

Aula 4

TRABALHANDO COM VARIÁVEIS E CONSTANTES

4

Caro aluno(a)!

Nesta quarta aula vamos identificar a diferença entre variáveis e constantes. Estudaremos como utilizá-las nas aplicações desenvolvidas em PHP. Além disso, verificaremos quais são os tipos de variáveis disponíveis em PHP e como podemos trabalhar com elas. Por fim, veremos mais algumas funções disponíveis em PHP que podem nos auxiliam nos procedimentos com variáveis.

Bom estudo!

Objetivos da Aula 

Ao final desta aula, você deverá ser capaz de:

- Identificar os tipos de variáveis existentes em PHP;
- Utilizar variáveis para criar rotinas em PHP;
- Utilizar funções do PHP para trabalhar com as variáveis;
- Identificar a diferença entre variáveis e constantes;
- Utilizar constantes para criar rotinas em PHP;
- Testar o funcionamento das variáveis e constantes.

Conteúdos da Aula 

Acompanhe os conteúdos desta aula. Se você preferir, assinale-os à medida em que for estudando.

- Trabalhando com variáveis;
- Funções PHP para trabalhar com variáveis;
- Trabalhando com constantes;
- Exercícios propostos.

1 TRABALHANDO COM VARIÁVEIS 

No capítulo anterior, vimos que na linguagem PHP, não há necessidade de declarar as variáveis com um tipo de dado específico, pois o interpretador PHP utiliza a tipagem automática para atribuir um tipo de dado a uma variável. Dessa forma, podemos inicializar uma variável somente quando realmente necessitamos dela em um procedimento da aplicação que estamos desenvolvendo.

Relembrando o que foi mencionado no capítulo 2 – introdução a linguagem PHP -, o nome da variável deverá sempre possuir o símbolo: “$” no início. É justamente esse símbolo que indica ao interpretador PHP que uma variável está sendo declarada do símbolo até o ponto e vírgula que irá encerrar a instrução de declaração da variável.

Também devemos relembrar que as variáveis em PHP são “case-sensitive”, ou seja, o nome da variável tem distinção entre letras maiúsculas e minúsculas.

Agora, após relembrarmos os conceitos básicos a respeito das variáveis, vamos verificar quais são os tipos de variáveis disponíveis na linguagem PHP.

Além das variáveis básicas do PHP criadas pelos desenvolvedores, há outros tipos de variáveis que podem ser utilizadas. Em geral, são criadas pelo servidor WEB e o interpretador PHP para auxiliar nos procedimentos da aplicação. As variáveis aqui conceituadas como básicas são todas aquelas utilizadas nos exemplos criados até agora no curso. A partir de agora conheceremos outros tipos de variáveis que poderemos utilizar em nossos exemplos e posteriores aplicações.

1.1 Variáveis enviadas pelo navegador

Quando utilizamos páginas PHP e HTML para fazer interação entre o usuário e a aplicação, podemos utilizar as variáveis enviadas pelo navegador. Há dois tipos de variáveis enviadas pelo navegador:

1. POST = são as variáveis enviadas pelos formulários em HTML

2. GET = são as variáveis enviadas ou por formulários HTML ou pelo endereço do navegador - URL no endereço do navegador.

Para verificar a diferença entre as duas variáveis, vamos analisar o exemplo a seguir. Neste exemplo criaremos um formulário HTML em uma página PHP para passar um valor por meio de uma variável e depois imprimir o valor da variável na tela. Utilizaremos o mesmo teste para variáveis do tipo POST e GET. Vamos ao exemplo.

Página1: **exemplo11_form.php**

```
<HTML>
<HEAD>
 <TITLE>Exemplo 11 - Form</TITLE>
</HEAD>
<BODY>
<form name=”form1” action=”exemplo11.php” method=”post”>
      Nome: <input type=”text” name=”nome”><br>
       <input type=”submit” name=”btnEnviar” value=”Enviar
valor”>
</form>
</BODY>
</HTML>
```

Página 2: **exemplo11.php**

```
<HTML>
<HEAD>
 <TITLE>Exemplo 11</TITLE>
</HEAD>
<BODY>
<?
  $nome = $_POST[“nome”];
  print “O nome informado é: $nome <br>”;
?>
</BODY>
</HTML>
```

Vamos agora analisar o exemplo. A página exemplo11_form.php criada, possui apenas um formulário HTML com um único campo texto chamado: nome. O valor informado nesse campo texto será enviado para a página: exemplo11.php**,** após clicar no botão: **Enviar Valor**. O comando que informa para onde será enviado o valor, está definido no comando: `action="exemplo11.php"` do comando `<form></form>` do formulário HTML. O tipo de variável a ser enviado para a página é definido no comando: `method="post"` do comando **<form></form>** do formulário HTML. Nesse exemplo, então, enviaremos o valor por meio de uma variável do tipo **POST**. Porém, se o desenvolvedor desejar enviar o valor por meio de uma variável do tipo **GET,** utilizando esse mesmo formulário, poderá fazê-lo trocando o texto **"post"** por **"get"**.

Analisando a página 2, exemplo11.php, identificamos na primeira linha, após o delimitador PHP, a instrução: `$nome = $_POST["nome"]`. Nesse comando estamos pegando o valor da variável **POST,** enviada pelo navegador, e atribuindo a variável `$nome` para depois imprimir na tela. Também é possível imprimir direto o conteúdo da variável enviada pelo navegador, para isso, basta alterar o comando para:

print $_POST["nome"];

Ao executar o exemplo acima, veremos que, após clicar no botão: Enviar valor, o navegador irá para a página exemplo11.php e mostrará o valor informado no campo de texto do formulário da página exemplo11_form.php

Vamos agora utilizar o mesmo exemplo para testar as variáveis do tipo: **GET**. Para isso, altere o `method="post"` do formulário para `method="get"`. Execute novamente o teste. Qual foi o resultado?

Nesse caso, não foi mostrado o valor informado, certo? Isso mesmo, porque a variável agora está indo por meio do tipo: **GET**. Verifique agora a barra de endereços do navegador da internet, ele deverá mostrar o seguinte comando:

http://localhost/apostila_php/exemplo11.php?nome=12&btnEnviar=Enviar+valor

Por que está mostrando os dados com o tipo: **GET** e com o tipo: **POST**?

Porque o tipo de variável **GET** envia os valores por meio do endereço do navegador.

E como fazemos para mostrar o valor enviado pelo navegador na página 2, exemplo11.php?

Devemos alterar a instrução: `$_POST["nome"]` para `$_GET["nome"]`, assim

o nome será mostrado corretamente na página exemplo11.php. Faça o teste novamente e valide essa informação.

Esses tipos de variáveis serão amplamente utilizados para o desenvolvimento de aplicações que fazem interação com usuários, porém há duas diferenças muito importantes entre os dois tipos de variáveis e que devem ser levadas em consideração na análise de qual tipo de variável utilizar:

1. Os tipos de variável **GET** são enviados pelo endereço do navegador e por isso todos os dados ficam expostos na barra de endereços. Assim evite utilizar esse tipo de variável quando houver dados sigilosos envolvidos.
2. A barra de endereço do navegador possui limitação de tamanho, máximo 255 caracteres, portanto, os valores dos tipos de variável **GET** não poderão ultrapassar esse tamanho. Se ultrapassar, os dados não serão enviados por completo, serão enviados somente os dados até o tamanho limite estabelecido pelo navegador.

Analisando essas informações, podemos definir que o tipo de variável indicado para formulários em HTML é o tipo: **POST**. O **GET** poderá ser utilizado em outros procedimentos da aplicação. Veremos mais detalhes sobre esses dois tipos de variáveis no decorrer do curso.

1.2 Variáveis de ambiente

A linguagem PHP possui várias variáveis de ambiente que podem auxiliar durante o desenvolvimento de uma aplicação. Para visualizar o conteúdo de uma variável de ambiente, basta utilizar a instrução: `$_SERVER["nome_da_variavel"]`. Há, por exemplo, a variável: `$_SERVER["PHP_SELF"]` que possui o nome e o caminho do próprio arquivo como valor. Há também outras variáveis que possuem dados sobre o navegador do usuário, o servidor HTTP, e até a versão do PHP instalada no servidor.

Para visualizar todas as variáveis de ambiente disponíveis, podemos utilizar a função phpinfo() ou a opção: “View phpinfo() screen”, disponível na página principal do VertrigoServ. As variáveis de ambiente estão listadas abaixo do item: “**Apache Environment**”.

1.3 Variáveis de sessão

A linguagem PHP disponibiliza também variáveis de sessão que permitem o controle da sessão do usuário. Esse tipo de variável é comumente utilizado em aplicações que possuem autenticação de usuário.

Para visualizar o conteúdo de uma variável de sessão, basta utilizarmos a instrução: `$_SESSION["nome_da_variavel"]`.

Estudaremos com mais detalhes sobre as variáveis de sessão no capítulo 8 – Controle de Sessão.

2 FUNÇÕES PHP PARA TRABALHAR COM VARIÁVEIS 

A linguagem PHP disponibiliza várias funções que auxiliam no desenvolvimento de aplicações. Nesse tópico, estudaremos as principais funções da linguagem que permitem manipular as variáveis criadas em PHP.

2.1 Funções que testam o tipo de dado de uma variável

As funções que testam o tipo de dado de uma variável são:

- **is_int** = testa se o tipo de dado da variável é inteiro;
- **is_integer** = testa se o tipo de dado da variável é inteiro;
- **is_real** = testa se o tipo de dado da variável é real – número com casas decimais;
- **is_long** = testa se o tipo de dado da variável é inteiro longo;
- **is_float** = testa se o tipo de dado da variável é float – número com casas decimais;
- **is_string** = testa se o tipo de dado da variável é string;
- **is_array** = testa se a variável é um array;
- **is_object** = testa se a variável é um objeto.

Para utilizar essas funções, deveremos seguir o modelo abaixo:

nome_funcao(nome_variavel);

O valor de retorno de todas as funções é booleano, ou seja, verdadeiro ou falso – true | false ou 1 | 0. O valor verdadeiro será retornado quando o tipo de dado da variável está de acordo com a função utilizada. Vamos executar o exemplo a seguir e verificar o funcionamento da função is_string:

Página: **exemplo12.php**

```
<HTML>
<HEAD>
 <TITLE>Exemplo 12</TITLE>
</HEAD>
<BODY>
<?
  $variavelString = "Texto da variávelString";
  print is_string($variavelString);
?>
</BODY>
</HTML>
```

Ao executarmos o exemplo12.php**,** será impresso na tela o valor 1. Esse valor é o retorno da função is_string e significa que a variável: `$variavelString` possui o tipo de dado: String.

2.2 Função que destrói uma variável

Toda vez que criamos uma variável, cria-se um espaço em memória para ser utilizado pela aplicação. Dessa forma, quando trabalhamos com várias variáveis em uma aplicação, todas estarão consumindo espaço em memória.

Portanto, é muito importante ao trabalharmos com variáveis, destruímos ou, na linguagem do computador, desalocarmos a variável da memória quando já não precisamos dela. A função que permite desalocar uma variável da memória é: unset(nome_variavel).

Após utilizar a função unset(), a variável será destruída, portanto, os valores da

variável serão perdidos. Assim, somente utilize essa função quando tiver certeza que a variável não será mais utilizada por algum procedimento dentro da aplicação.

2.3 Funções que verificam se uma variável possui valor

Existem duas funções que verificam se uma variável possui valor:

- **isset(nome_da_variavel)**
- **empty(nome_da_variavel)**

Como na maioria das funções PHP, o retorno dessas duas funções também é **booleano**.

Há, porém, uma diferença entre as duas variáveis que precisamos analisar antes de utilizá-las. Na primeira função – a função **isset** – é verificado se a variável já foi criada na aplicação, ou seja, se já foi utilizada alguma vez pela aplicação. Caso a variável não tenha sido criada ainda, retornará o valor **0** ou **false**. Na segunda função – a função **empty** – será verificado se a variável possui valor vazio ou nulo, porém a variável já deve estar criada na aplicação. Vamos executar o exemplo que segue para compreender melhor essa diferença entre as duas funções:

Página: **exemplo13.php**

```
<HTML>
<HEAD>
<TITLE>Exemplo 13</TITLE>
</HEAD>
<BODY>
<?
$variavel1 = "";
// verifica se a $variavel1 possui valor vazio ou nulo
print "variavel1 = " . empty($variavel1);
print "<br>";
//destroi a $variavel1
unset($variavel1);
```

```
//verifica se existe a $variavel1
print "variavel1 = " . isset($variavel1);
?>
</BODY>
</HTML>
```

3 TRABALHANDO COM CONSTANTES 

Constantes podem ser definidas como uma variável que possui um valor constante, ou seja, o valor será definido uma única vez e jamais será alterado durante a execução da aplicação. Essa é a principal diferença entre uma constante e uma variável. Como o próprio nome indica, as constantes possuem valores que jamais serão alterados por algum procedimento dentro da aplicação. Já as variáveis podem ter o seu valor alterado, dependendo do tipo de ação que for executado pela aplicação.

As constantes geralmente são utilizadas para receber valores de configurações / parâmetros que serão utilizados pelo sistema. Um exemplo de constante pode ser um login e senha para conexão da aplicação com o banco de dados. Esse dado sempre será o mesmo durante a execução de toda a aplicação, portanto, poderá estar definido em uma constante.

Para definir uma constante na aplicação, utilizamos a função: define(nome_da_constante,valor_da_constante).

A linguagem PHP possui algumas constantes pré-definidas que permitem verificar as configurações do ambiente onde o interpretador PHP está instalado. Algumas constantes disponíveis possuem como valor: a versão do PHP, o Sistema Operacional do servidor, o arquivo em execução, entre outras informações.

Para visualizar todas as constantes pré-definidas disponíveis, podemos utilizar a função **phpinfo()** ou então utilizar a opção: "**View phpinfo() screen**", disponível na página principal do VertrigoServ.

Veja o exemplo que mostra como definir e utilizar uma constante dentro das páginas PHP:

Página: **exemplo14.php**

```
<HTML>
<HEAD>
 <TITLE>Exemplo 14</TITLE>
</HEAD>
<BODY>
<?
// definir a constante taxa de juros
define(“taxaJuros”,0.5);

// imprime o valor da constante na tela
print “taxa de Juros = “ . taxaJuros;
print “<br>”;

// utiliza a constante para um cálculo
$valor = 100.00*taxaJuros;

// imprime o resultado do cálculo com a constante
print “valor = “ . $valor;
?>
</BODY>
</HTML>
```

Síntese 

Nesta aula vimos:

- Os tipos de variáveis disponíveis na linguagem PHP;
- Como trabalhar com as variáveis;
- Funções PHP que auxiliam nos procedimentos com variáveis;
- A diferença entre variáveis e constantes;
- Como trabalhar com as constantes;
- Testes com variáveis e constantes.

Exercícios propostos 

1) Os tipos de variáveis mais utilizados no desenvolvimento de aplicações com PHP são (é permitida múltipla escolha):

a. Sessão

b. Enviadas pelo Navegador

c. Ambiente

d. Memória

2) Complete a frase abaixo com uma das alternativas:

As variáveis que utilizam a barra de endereços do navegador é a ....

a. Sessão

b. Post

c. Get

d. Servidor

3) O tipo de variável que possui os dados da configuração do PHP é (apenas uma alternativa correta):

a. Ambiente

b. Post

c. Sessão

d. Servidor

4) A função que verifica se a variável está setada e com o valor vazio é a (apenas uma alternativa correta):

a. is_string

b. unset

c. empty

d. isset

5) A principal diferença entre uma variável e uma constante é (apenas uma alternativa correta):

a. A constante somente aceita um tipo de dado

b. A constante tem limitação de valor

c. O valor da constante não muda nunca na aplicação

d. A constante não pode ser utilizada em algumas instruções

Aula 5

OPERADORES DA LINGUAGEM PHP

5

Caro aluno(a)!

Nesta quinta aula você conhecerá os tipos de operadores disponíveis na linguagem PHP e a diferença entre eles. Além disso, analisaremos o que é uma expressão condicional e quando podemos utilizá-las dentro de nossas aplicações. Por fim, realizaremos alguns testes para identificar melhor a funcionalidade dos operadores e também da expressão condicional.

Bons Estudos!

Objetivos da Aula

Ao final desta aula, você deverá ser capaz de:

- Identificar os tipos de operadores da linguagem PHP;
- Utilizar os tipos de operadores;
- Identificar o que é uma expressão condicional;
- Utilizar a expressão condicional.

Conteúdos da Aula

Acompanhe os conteúdos desta aula. Se você preferir, assinale-os à medida em que for estudando.

- Conhecendo os Tipos de Operadores;
- Conhecendo a Expressão Condicional;
- Exercícios Propostos.

1 CONHECENDO OS TIPOS DE OPERADORES 

A linguagem PHP, assim como outras linguagens de programação, disponibiliza vários tipos de operadores para auxiliar na execução de procedimentos dentro das aplicações.

Nesse capítulo, verificremos quais são os tipos de operadores e como podemos utilizá-los em nossas páginas PHP.

1.1 Operadores Aritméticos

Os operadores aritméticos são aqueles que auxiliam em cálculos e expressões matemáticas. Esses operadores somente poderão ser utilizados entre variáveis com os tipos de dados números inteiros e/ou números com casas decimais – números reais.

Observe quais são os operadores aritméticos disponíveis em PHP.

+	adição
-	subtração
*	multiplicação
/	divisão
%	módulo

O exemplo demonstra a utilização dos operadores aritméticos nas páginas em PHP:

 Página: **exemplo15.php**

```
<HTML>
<HEAD>
 <TITLE>Exemplo 15</TITLE>
</HEAD>
<BODY>
<?
```

```
// utilizando operações matemáticas
      $numero1 = 10;
      $numero2 = 3;

      // imprime o valor das variáveis
      print "valor da variável numero1 = " . $numero1;
      print "<br>";
      print "valor da variável numero2 = " . $numero2;
      print "<br><br>";

      // faz a soma entre as variáveis e mostra o resultado
      print "Soma = " . ($numero1+$numero2);
      print "<br>";

      // faz a diferença entre as variáveis e mostra o resultado
      print "Subtração = " . ($numero1-$numero2);
      print "<br>";

      // faz a multiplicação entre as variáveis e mostra o resultado
      print "Multiplicação = " . ($numero1*$numero2);
      print "<br>";

      // faz a divisão entre as variáveis e mostra o resultado
      print "Divisão = " . ($numero1/$numero2);
      print "<br>";

      // faz a divisão entre as variáveis e mostra o resto da divisão
      print "Módulo - Resto da Divisão = " . ($numero1%$numero2);
      print "<br>";
      ?>
      </BODY>
      </HTML>
```

1.2 Operador String

Para os tipos de dados **string,** há apenas um operador disponível, o que permite a concatenação entre variáveis do tipo **string**. Concatenar duas variáveis do tipo **string** significa juntar o valor das duas variáveis.

O quadro a seguir mostra o símbolo que representa o operador de concatenação em PHP.

.	concatenação

Se você observar novamente os últimos exemplos desenvolvidos, utilizamos o operador de concatenação para unir textos com variáveis. O exemplo abaixo mostra a concatenação entre duas variáveis do tipo texto:

Página: **exemplo16.php**

```
<HTML>
<HEAD>
 <TITLE>Exemplo 16</TITLE>
</HEAD>
<BODY>
<?
// utilizando operador String
$texto1 = "Vamos testar ";
$texto2 = "o operador de concatenação";

// imprime o valor das variáveis concatenadas
print $texto1 . $texto2;
?>
</BODY>
</HTML>
```

1.3 Operadores de Atribuição

Os operadores de atribuição têm como função retornar um valor atribuído de acordo com a operação indicada. As operações podem utilizar operadores aritméticos ou ainda o operador: **string**.

A atribuição feita é sempre por valor, e não por referência. Isso significa que a atribuição irá alterar definitivamente o valor da variável armazenada em memória.

Observe no quadro a seguir os operadores de atribuição disponíveis em PHP.

=	atribuição simples
+=	atribuição com adição
-=	atribuição com subtração
*=	atribuição com multiplicação
/=	atribuição com divisão
%=	atribuição com módulo
.=	atribuição com concatenação

Na maioria dos exemplos que desenvolvemos até agora, utilizamos o operador básico de atribuição para atribuir um valor as variáveis. No exemplo abaixo, vamos utilizar outros dois operadores de atribuição para verificarmos o seu funcionamento:

Página: **exemplo17.php**

```
<HTML>
<HEAD>
 <TITLE>Exemplo 17</TITLE>
</HEAD>
<BODY>
<?
// utilizando operadores de atribuição
$texto = " O valor da variável numero é = ";
$numero = 5;

// utilizando o operador de atribuição - concatenação
```

```
$texto .= $numero;

    // imprime o valor da variável $texto
    print $texto;
    print "<BR>";

    // utiliza o operador de atribuição – adição
    $numero += 2;

    // imprime o valor da variável $numero
    print $numero;
    ?>
    </BODY>
    </HTML>
```

1.4 Operadores Lógicos

Os operadores lógicos são geralmente utilizados em expressões e comparações entre condições ou variáveis que retornam valores **booleanos**.

Os operadores lógicos disponíveis em PHP são os que mostramos no quadro a seguir.

!	não (inversão)
&&	"e" lógico
||	"ou" lógico

Nos exemplos dos próximos capítulos e também no desenvolvimento de nosso estudo de caso, muitas vezes utilizanremos os operadores lógicos para validar valores de variáveis e condições.

1.5 Operadores de Comparação

Os operadores de comparação são utilizados para comparar valores entre

variáveis e expressões. O retorno de uma comparação será sempre um valor do tipo: **booleano**.

Veja quais são os operadores de comparação disponíveis em PHP.

==	igual a
!=	diferente de
<	menor que
>	maior que
<=	menor ou igual a
>=	maior ou igual a

Nos exemplos dos próximos capítulos e também no desenvolvimento de nosso estudo de caso, muitas vezes utilizando os operadores de comparação para comparar os valores de variáveis e também expressões.

1.6 Operadores de Incremento e Decremento

Os operadores de atribuição têm como função aumentar (incremento) ou diminuir (decremento) o valor de uma variável. São operadores sempre utilizados em tipos de dados numéricos e o valor da variável terá sempre o incremento/decremento de 1. Por exemplo: se uma variável possui o valor 10 e utilizamos o operador de incremento **após** a utilização do operador, a variável terá o valor 11. Se utilizássemos o decremento nessa mesma variável, o valor seria diminuído para 9.

O incremento/decremento será feito de acordo com a ordem que disponibilizamos o operador e a variável. Se colocarmos o operador de incremento/decremento **antes** da variável, a ação do operador será feita antes de mostrar a variável, caso contrário, se o operador for colocado **depoi**s da variável, a ação do operador será feita depois de mostrar a variável.

Observe os operadores de incremento e decremento disponíveis em PHP.

++	Incremento
--	Decremento

Assim como os operadores lógicos e de atribuição, utilizaremos bastante os

operadores de incremento e decremento nos exemplos dos próximos capítulos. No exemplo que segue já podemos ter uma noção de como funcionam estes operadores:

Página: **exemplo18.php**

```
<HTML>
<HEAD>
 <TITLE>Exemplo 18</TITLE>
</HEAD>
<BODY>
<?
// utilizando operadores de incremento e decremento
$numero = 5;

// imprime o valor da variável $numero utilizando o operador de incremento
print $numero++;
print "<BR>";

// imprime o valor da variável $numero utilizando o operador de decremento
print $numero--;
print "<BR>";
?>
</BODY>
</HTML>
```

2 CONHECENDO A EXPRESSÃO CONDICIONAL

A expressão condicional é considerada um tipo de operador. Também é conhecida como um operador de seleção ternário.

A função da expressão condicional é verificar uma dada condição e, dependendo do valor da condição retornado – valor booleano: true ou false – a expressão executará

uma instrução.

A sintaxe da expressão condicional é:

(condição)?(instrução1):(instrução2)

O interpretador PHP interpreta a expressão condicional da seguinte maneira:

1. Avalia a condição feita;
2. Se o resultado da condição for verdadeiro – true ou 1 - a expressão condicional irá mostrar o valor da instrução1;
3. Se o resultado da condição for falso – false ou 0 – a expressão condicional irá mostrar o valor da instrução2.

A expressão condicional é também conhecida pelos desenvolvedores por “if rápido”. O “if” é uma estrutura de controle que também valida condições. Estudaremos mais sobre o “if” no próximo capítulo.

No exemplo a seguir podemos verificar o funcionamento da expressão condicional. Também utilizaremos um operador de comparação, o operador de igualdade:

Página: **exemplo19.php**

```
<HTML>
<HEAD>
<TITLE>Exemplo 19</TITLE>
</HEAD>
<BODY>
<?
     // testando a expressão condicional
     $numero1 = 5;
     $numero2 = 4;

/* utiliza a expressão condicional para verificar se os valores das variáveis: $numero1 e $numero2 são iguais e depois imprime um texto de acordo com o retorno da condição / comparação feita */
```

```
print ($numero1==$numero2)?"Valores iguais":"Valores dife-
rentes";
     print "<BR>";
     ?>
     </BODY>
     </HTML>
```

Síntese 

Nesta aula estudamos:

• Os tipos de operadores disponíveis na linguagem PHP;

• Como e quando utilizamos os tipos de operadores;

• O que é uma expressão condicional;

• Como e quando podemos utilizar uma expressão condicional;

• Realizamos testes para validar a funcionalidade dos tipos de operadores e também da expressão condicional.

Exercícios propostos 

1) Os operadores que permitem a realização de cálculos matemáticos são (múltipla escolha):

a. Aritméticos

b. String

c. Atribuição

d. Incremento

2) Complete a frase com uma das alternativas abaixo:

A concatenação é uma operação permitida pelo operador ... e tem como função ....

a. Aritmético, adicionar variáveis

b. Atribuição, atribuir valor às variáveis

c. String, unir valores das variáveis

d. Decremento, separar o valor da variável em dois

3) O operador que retorna o resto de uma divisão é (apenas uma alternativa correta):

a. Aritmético – Divisão

b. Aritmético – Módulo

c. Aritmético - Incremento

d. Aritmético - Atribuição

4) Os operadores de atribuição são utilizados para (múltipla escolha):

a. Atribuir valores as variáveis

b. Atribuir tipo de dados em variáveis

c. Executar uma operação matemática em uma variável

d. Executar uma operação matemática somente após a atribuição de um valor

5) A expressão condicional pode ser definida como (é permitida múltipla escolha):

a. Um operador aritmético

b. Um operador de atribuição

c. Um comando de seleção

d. Um operador de seleção ternário

Aula 6

ESTRUTURAS DE CONTROLE

6

Tutu Virtual

Caro aluno(a)!

Nesta sexta aula você estudará sobre as estruturas de controle disponíveis na linguagem PHP. Vamos aprender quais são essas estruturas e quando podemos utilizá-las dentro das nossas aplicações. Também desenvolveremos vários exemplos com as estruturas de controle para testar e compreender melhor a função de cada uma.

Bons Estudos!

Objetivos da Aula 

Ao final desta aula, você deverá ser capaz de:

• Identificar quais são as estruturas de controle do PHP;

• Identificar como e quando utilizar as estruturas de controle;

•Testar o funcionamento das estruturas de controle.

Conteúdos da Aula 

Acompanhe os conteúdos desta aula. Se você preferir, assinale-os à medida em que for estudando.

• Introdução às Estruturas de Controle;

• Estruturas de Seleção;

• Estruturas de Repetição;

• Estruturas de Quebra de Fluxo;

• Exercícios Propostos.

1 INTRODUÇÃO ÀS ESTRUTURAS DE CONTROLE 

As estruturas de controle disponíveis na linguagem PHP são comuns na maioria das linguagens de programação. Se você já possui o conhecimento de alguma linguagem de programação, identificará várias características semelhantes nas estruturas de controle do PHP.

A sintaxe das estruturas de controle do PHP também é bem similar à sintaxe de outras linguagens. Se você já trabalhou com a linguagem de programação C ou Java, vai verificar que a sintaxe das estruturas de controle é bem semelhante.

Nos próximos tópicos desse capítulo, estudaremos os três tipos de estruturas de controle disponíveis na linguagem PHP: Estruturas de Seleção, Estruturas de Repetição e Estruturas de Quebra de Fluxo. Na sintaxe e também nos exemplos, veremos que, para os dois primeiros tipos de estruturas, existe uma característica em comum, os **blocos de comando**.

Um bloco de comando consiste em um agrupamento de instruções PHP dentro de uma estrutura de controle. Esses blocos são delimitados pelos caracteres: “{” que representa o início do bloco e “}” que representa o fim do bloco de comando. Dependendo da estrutura de controle utilizada e também do procedimento que será executado dentro da estrutura de controle, poderemos ter: **nenhum**, **um** ou **mais blocos de comando**. A existência de um bloco de comando será dispensada quando o procedimento tiver apenas uma instrução de comando.

A seguir, verificaremos detalhadamente quais são as estruturas de controle e qual a aplicação de cada uma delas.

2 ESTRUTURAS DE SELEÇÃO 

As estruturas de seleção, também conhecidas por estruturas condicionais, têm como função validar condições e comparar o resultado das condições. Após executar as funções de validação e comparação, as estruturas de seleção irão executar os blocos de comando, definidos de acordo com o resultado da comparação.

A expressão condicional, que verificamos no capítulo anterior, pode também ser considerada como uma estrutura de controle de seleção.

Existem dois tipos de estruturas de seleção: **If** e **Switch**. A seguir vamos verificar quais as diferenças e sintaxe de cada uma delas.

2.1 If

O If é considerado a estrutura de controle mais comum das linguagens de programação e é muito utilizado no desenvolvimento das aplicações.

Possui um sintaxe bem simples:

```
IF(condição)
     Instrução;
```

Ou

```
IF(condição){
     Instrução1;
     Instrução2;
     Etc;
}
```

A primeira sintaxe é utilizada quando temos apenas uma instrução PHP a ser executada após a validação da condição. Não é necessário, portanto, criar um bloco de comando para agrupar e executar as instruções. Na segunda sintaxe, temos mais de uma instrução PHP a ser executada, portanto, é necessária a criação de um bloco de comando para agrupar as instruções e executá-las conforme a seqüência definida.

O If sempre executará o bloco de comando ou a instrução única se a condição entre parênteses retornar um resultado booleando verdadeiro. Caso contrário, o bloco de comando ou a instrução única não serão executadas.

Veja o exemplo.

Página: **exemplo20.php**

```
<HTML>
<HEAD>
 <TITLE>Exemplo 20</TITLE>
</HEAD>
<BODY>
<?
// testando o if
$numero1 = 5;
$numero2 = 4;

/* utiliza o if para verificar se os valores das variáveis: $numero1 e $numero2 são diferentes e depois imprime um texto caso o retorno da condição seja verdadeiro – neste exemplo estamos também utilizando o operador de comparação - desigualdade  */
     if ($numero1!=$numero2){
               print ”As variáveis possuem valores diferentes”;
               print “<BR>”;
     }
     ?>
     </BODY>
     </HTML>
```

Analisando o resultado do teste, verificaremos que a mensagem: “**As variáveis possuem valores diferentes**” foi impressa na tela, pois realmente elas possuem valores diferentes. Utilizando o operador de comparação de desigualdade, a condição retornou um valor booleano verdadeiro, pois as variáveis possuem valores diferentes.

Vamos agora utilizar o mesmo exemplo, porém alterando a condição para verificar se os valores das variáveis são iguais. Para isso, vamos utilizar o operador de comparação de igualdade:

Página: **exemplo21.php**

```
<HTML>
<HEAD>
 <TITLE>Exemplo 21</TITLE>
</HEAD>
<BODY>
<?
// testando o if
$numero1 = 5;
$numero2 = 4;

/* utiliza o if para verificar se os valores das variáveis: $numero1 e $numero2 são iguais e depois imprime um texto caso o retorno da condição seja verdadeiro – neste exemplo estamos também utilizando o operador de comparação - igualdade  */
if ($numero1==$numero2){
            print ”As variáveis possuem valores iguais”;
            print “<BR>”;
}
?>
</BODY>
</HTML>
```

Quando executamos o exemplo, agora, não foi mostrada nenhuma mensagem na tela, por quê? Porque a condição retornou um valor falso, ou seja, o valor das duas variáveis não é verdadeiro.

Certo, mas neste caso então, como fazemos para mostrar alguma mensagem quando o valor da condição é falso? Podemos utilizar a instrução: “else”. A sintaxe do IF utilizando a instrução “else” é:

```
IF(condição)
     Instrução;
Else
     Instrução;
```

Ou para utilizar com bloco de comando:

```
IF(condição){
     Instrução1;
     Instrução2;
     Etc;
     } Else {
     Instrução1;
     Instrução2;
     Etc;
}
```

As instruções do Else serão executadas somente quando o valor da condição do If for falso. Podemos ainda utilizar um encadeamento de condições quando precisamos analisar uma série de condições no mesmo procedimento. Esse encadeamento é uma mistura entre IF's e Else´s dentro do código. Veja como fica a sintaxe:

```
IF(condição1){
     Instrução1;
     Instrução2;
     Etc;
     } Else IF(condição2) {
     Instrução1;
     Instrução2;
     Etc;
```

```
} Else IF(condição3) {
     Instrução1;
     Instrução2;
     Etc;
} Else IF(condiçãoN) {
     Instrução1;
     Instrução2;
     Etc;
} Else {
     Instrução1;
     Instrução2;
     Etc;
}
```

Mostramos na sintaxe acima, que não existe um limite para o encadeamento de if´s. Esse tipo de procedimento pode ser utilizado de acordo com a quantidade de condições e comparações que devem ser executadas pela aplicação. É importante lembrar que as instruções do Else If serão executadas somente se a condição do IF tiver valor verdadeiro, caso contrário, as instruções executadas serão as do Else final, que não possui um IF com uma condição para comparar, por isso é executado quando o valor da condição de todos os If´s for falso.

Vamos testar o exemplo abaixo para compreender melhor o funcionamento do IF encadeado:

Página: **exemplo22.php**

```
<HTML>
<HEAD>
<TITLE>Exemplo 22</TITLE>
</HEAD>
<BODY>
<?
```

```
// testando o IF encadeado
     $numero1 = 5;
     $numero2 = 4;
     $numero3 = 5;

// utilizando o If encadeado
     if ($numero1==$numero2){
          print ”As variáveis: numero1 e numero2 possuem
valores iguais”;
          print “<BR>”;
     } else if ($numero1==$numero3){
          print ”As variáveis: numero1 e numero3 possuem
valores iguais”;
          print “<BR>”;
     } else if ($numero2==$numero3){
          print ”As variáveis: numero2 e numero3 possuem
valores iguais”;
          print “<BR>”;
     } else {
          print ”As três variáveis possuem valores diferen-
tes”;
          print “<BR>”;
     }
     ?>
     </BODY>
</HTML>
```

2.2 Switch

A estrutura de seleção Switch funciona de maneira semelhante aos If´s encadeados. Essa estrutura é muito utilizada quando é necessário comparar o valor de uma variável ou então de uma mesma condição com diversos valores.

Se utilizarmos o IF encadeado para testar uma variável com diversos valores, teremos sempre que montar um IF com vários Else If até completar todos os valores que desejamos testar. Utilizando o Switch para esse mesmo procedimento, o código ficará mais enxuto e terá o mesmo efeito. Vamos analisar a sintaxe do switch:

```
Switch(valor_a_ser_comparado){
     Case valor1:
          Instrução;
          Break;
     Case valor2:
          Instrução;
          Break;
     Default:
          Instrução;
          Break;
}
```

Na sintaxe acima, os itens em negrito são os que deveremos alterar de acordo com o procedimento que será executado. Onde está o “valor_a_ser_comparado” será indicada a variável que deseja comparar o valor ou então a condição que deseja comparar o valor resultante da condição. Onde está o valor1 e o valor2 serão indicados os valores que serão comparados.

As instruções a serem executadas serão as que indicam que a condição é verdadeira. Exemplo: se o valor_a_ser_comparado é igual ao valor1, então a instrução que será executada é a instrução abaixo do case valor1. Caso nenhum case possua o valor do valor a ser comparado, a instrução a ser executada será a instrução que está abaixo do Default. O comando Default é semelhante ao Else na estrutura do If.

Para compreendermos melhor o funcionamento do **switch**, vamos executar o exemplo a seguir:

Página: **exemplo23.php**

```
<HTML>
```

```
<HEAD>
<TITLE>Exemplo 23</TITLE>
</HEAD>
<BODY>
<?
// testando o switch
$numero = 10;

// utilizando o switch com variável
switch($numero){
               case 0:
                    print “O valor da variável é 0”;
                    break;
               case 5:
                    print “O valor da variável é 5”;
                    break;
               case 10:
                    print “O valor da variável é 10”;
                    break;
               default:
                    print “O valor da variável é diferente
de 0,5 e 10”;
                    break;
          }
     print “<br><br>”;

// utilizando o switch com condição
     switch($numero/2){
               case 0:
                    print “A divisão variável por 2 é 0”;
                    break;
               case 5:
```

```
                    print "A divisão variável por 2 é 5";
                    break;
               default:
                    print "A divisão variável por 2 é di-
ferente de 0 e 5";
                    break;
     }
     ?>
     </BODY>
     </HTML>
```

3 ESTRUTURAS DE REPETIÇÃO 

As estruturas de repetição são muito utilizadas quando precisamos executar um bloco de comandos várias vezes até que uma dada condição fique com o valor falso. Existem três tipos de estruturas de repetição: While, Do...While e For. Todas possuem a mesma funcionalidade, repetir as instruções do bloco de comandos até que uma dada condição fique com o valor falso, porém as três fazem a validação da condição de forma diferente.

Vamos analisar cada estrutura de repetição separadamente para compreender melhor as diferenças entre elas.

3.1 While

A estrutura de repetição while possui a seguinte sintaxe:

```
While(condição)
     Instrução;
```

Ou utilizando bloco de comandos:

```
While(condição){
     Instrução1;
     Instrução2;
     InstruçãoN;
}
```

Tal qual o If, a condição entre os parênteses do while devem possuir um valor verdadeiro para que as instruções sejam executadas. Dessa forma, somente se o resultado da condição for verdadeiro é que as instruções serão executadas.

Para utilizar uma estrutura de repetição, é importante que, em algum momento, o valor da condição testado se torne falso, caso contrário, a aplicação ficará em eterno "loop", ou seja, estará executando as instruções eternamente. Isso acontece porque as instruções somente serão executadas quando o valor da condição for verdadeiro. Se a condição permanecer eternamente com o valor verdadeiro a aplicação estará executando as instruções infinitamente. Essa característica é comum para todas as estruturas de repetição.

No caso específico do while, a condição é testada no início da instrução while e, em seguida, é feita a execução das instruções. Se a condição permanecer com o valor verdadeiro, as instruções serão executadas novamente e assim sucessivamente, até que o valor da condição se torne falso.

Veja no exemplo abaixo a funcionalidade da estrutura while:

Página: **exemplo24.php**

```
<HTML>
<HEAD>
<TITLE>Exemplo 24</TITLE>
</HEAD>
     <BODY>
     <?
     // testando o while
     $numero = 1;
```

```
     // utilizando while para testar se o valor da variável
$numero é menor que 10
     while($numero<10){
               print "O valor da variável numero é " .$nu-
mero . "<br>";

               // utilizando o operador de incremento para
aumentar o valor
               ++$numero;
               // imprime o novo valor da variável $numero
               print "O valor da variável numero agora é "
.$numero . "<BR>";

               // verifica se o valor da variável é 10 e im-
prime o fim do loop
               if($numero==10)
                    print "Fim do Loop";
          }
     ?>
     </BODY>
</HTML>
```

3.2 Do ... while

A estrutura de repetição do...while é semelhante à estrutura while. A única **diferença** entre elas é que o **while testa a condição no ínicio do comando** e a estrutura **do...while** testa a **condição no final do comando**. Isso significa que, se a condição estiver com o valor **falso**, utilizando **do...while** as instruções serão executadas uma vez. Utilizando while, se a condição estiver com o valor falso, as instruções não serão executadas.

Veja a sintaxe do do...while:

```
Do{
     Instrução;
}while(condição);
```

Diferente das demais estruturas de controle, o do...while obriga a utilização do bloco de comando em sua sintaxe.

Vamos ao exemplo utilizando a estrutura do...while:

Página: **exemplo25.php**

```
     <HTML>
     <HEAD>
      <TITLE>Exemplo 25</TITLE>
     </HEAD>
     <BODY>
     <?
     // testando o do...while
     $numero = 10;

// utilizando do...while para testar se o valor da variável
$numero é menor que 10
     do{
               print "O valor da variável numero é " .$nu-
mero . "<br>";
// utilizando o operador de incremento para aumentar o va-
lor
++$numero;

// imprime o novo valor da variável $numero
          print "O valor da variável numero agora é " .$nu-
mero . "<BR>";
```

```
// verifica se o valor da variável é maior ou igual 10 e im-
prime o fim do loop
                if($numero>=10)
                     print "Fim do Loop";
     }while($numero<10);
     ?>
     </BODY>
</HTML>
```

3.3 For

A estrutura de repetição For possui a mesma função que o While e o Do...While, porém sua estrutura e sintaxe são bem diferentes.

O For também é uma condição e executa as instruções do bloco de comando, caso o valor da condição seja verdadeiro. Porém no For, a condição é baseada em uma variável contadora, o que o torna muito útil quando temos uma lista de valores para ser mostrado ou trabalhado e temos o conhecimento do início e fim dessa lista.

É imprescindível que saibamos o início e o fim da lista, pois estes dados são utilizados na sintaxe do for. Vamos analisar a sintaxe do for:

```
For(inicialização;condição;incremento ou decremento do con-
tador)
     Instrução;
```

Ou utilizando bloco de comandos:

```
For(inicialização;condição;incremento ou decremento do conta-
dor){
Instrução1;
     InstruçãoN;
}
```

Entre as várias utilidades que a estrutura For nos possibilita, a principal é a facilidade para trabalhar com variáveis do tipo array. Como já vimos no capítulo sobre os tipos de dados, um array é uma variável com várias posições para armazenamento de valores. Sempre sabemos a quantidade de posições que o array possui, portanto temos as informações de tamanho dessa lista de variáveis. No exemplo abaixo, iremos compreender melhor a utilização do **For** para mostrar os valores das posições de um **Array**:

Página: **exemplo26.php**

```
<HTML>
<HEAD>
 <TITLE>Exemplo 26</TITLE>
</HEAD>
<BODY>
<?
// testando o for
$arrayNomes[0] = “João”;
$arrayNomes[1] = “José”;
$arrayNomes[2] = “Maria”;
$arrayNomes[3] = “Felisberto”;

// utilizando o for para mostrar os valores do $arrayNomes
for($contador=0;$contador<=3;$contador++){
          print $arrayNomes[$contador] . “<br>”;
}
?>
</BODY>
</HTML>
```

4 ESTRUTURAS DE QUEBRA DE FLUXO 

As estruturas de quebra de fluxo são geralmente utilizadas para dar suporte às outras estruturas de controle. São utilizadas para quebrar o fluxo de um procedimento, parar a execução de uma instrução que está dentro de um “loop”, por exemplo, e dar continuidade nas instruções seguintes.

Existem dois tipos de estruturas de quebra de fluxo: Break e Continue. A seguir vamos analisar o comportamento dessas duas estruturas.

4.1 Break

A estrutura de controle break pode ser utilizada como suporte pelas estruturas de repetição: While, Do...While e For e também na estrutura de seleção: Switch.

Quando o interpretador PHP encontra o Break dentro de uma estrutura, imediatamente ele pára a execução das instruções e segue para as demais instruções definidas.

Como já vimos no exemplo do Switch, o Break serve para parar as comparações de valores de uma variável ou condição. Com as estruturas de repetição acontece a mesma ação. A instrução seguinte ao break não é executada e o fluxo de instruções após a repetição é executado.

O exemplo a seguir queremos imprimir na tela somente os números múltiplos de 2. Os números que não são múltiplos de 2 não serão impressos na tela.

Página: **exemplo27.php**

```
<HTML>
<HEAD>
 <TITLE>Exemplo 27</TITLE>
</HEAD>
<BODY>
<?
```

```
// testando a quebra de fluxo com Break
// utilizando o for para mostrar os números múltiplos de
2
     for($numero=0;$numero<=10;$numero++){
               if( ($numero%2) == 0 ) {
                    print $numero . " é múltiplo de
2<br>";
                    break;
                    print "Mensagem após o Break<BR>";
               }
     }
     ?>
     </BODY>
</HTML>
```

Ao executarmos o exemplo, apenas uma linha foi mostrada, a linha do número 0. Porque isto aconteceu? Porque, após a impressão da mensagem na tela, o interpretador encontrou a estrutura de quebra de fluxo Break. Quando o interpretador encontra o Break, pára imediatamente de executar as instruções da estrutura onde o Break se encontra e passa a executar as instruções que estão na seqüência dessa estrutura.

A seguir analisaremos a estrutura de quebra de fluxo **Continue** e utilizaremos esse mesmo exemplo para analisar a diferença entre as duas estruturas.

4.2 Continue

A estrutura de controle Continue também dá suporte às estruturas de controle de repetição. A **diferença** entre o **Break** e o **Continue:** em que o Continue pára a execução da estrutura de repetição no momento o interpretador encontra o Continue, porém ao invés de sair da estrutura de repetição, a execução é direcionada para o início da estrutura. Utilizando o for como exemplo, a execução é direcionada para a primeira linha do For onde é testada a condição da variável contadora.

Vamos utilizar o mesmo exemplo do Break, alterando-o para Continue e verificar a diferença entre eles.

Página: **exemplo28.php**

```
<HTML>
<HEAD>
 <TITLE>Exemplo 28</TITLE>
</HEAD>
<BODY>
<?
// testando a quebra de fluxo com Continue
// utilizando o for para mostrar os números múltiplos de
2
for($numero=0;$numero<=10;$numero++){
          if( ($numero%2) == 0 ) {
               print $numero . " é múltiplo de
2<br>";
               continue;
               print "Mensagem após o Continue<BR>";
          }
}
?>
</BODY>
</HTML>
```

Após executarmos o exemplo novamente, trocando o break pelo continue, podemos observar que os números múltiplos de 2 foram corretamente impressos na tela. Porém a mensagem que fica abaixo do continue, assim como quando utilizamos o break, nunca será executada.

Síntese 

Nesta aula estudamos:

• As estruturas de controle disponíveis na linguagem PHP;

• Como e quando podemos utilizar as estruturas de controle nas aplicações;

• Desenvolvemos testes para validar a funcionalidade das estruturas de controle.

Exercícios propostos 

1) As estruturas de controle disponíveis no PHP são (é permitida múltipla escolha):

a. Repetição

b. Atribuição

c. Seleção

d. Quebra de Fluxo

2) A estrutura de controle If é uma estrutura do tipo (apenas uma alternativa está correta):

a. Condição

b. Seleção

c. Atribuição

d. Repetição

3) A estrutura de controle Switch se assemelha a outra estrutura de controle. Qual é a estrutura?

a. For

b. If

c. Break

d. While

4) A estrutura de repetição que testa o valor da condição no final é:

a. While

b. For

c. Do ... While

d. Switch

5) As estruturas de quebra de fluxo são utilizadas para (é permitida múltipla escolha):

a. Finalizar uma instrução

b. Dar suporte às outras estruturas de controle

c. Alterar o fluxo das estruturas de repetição

d. Interromper a execução da página

Aula 7

CLASSES E OBJETOS

7

Caro aluno(a)!

Nesta aula você estudará sobre programação orientada a objetos na linguagem PHP.

Revisaremos os conceitos básicos da Orientação a Objetos; verificaremos a aplicação desses conceitos na linguagem PHP e, ao final, construiremos alguns exemplos utilizando a programação orientada a objetos. O conteúdo visto nesta aula será bastante aplicado no nosso estudo de caso.

Bons Estudos!

Objetivos da Aula 

Ao final desta aula, você deverá ser capaz de:

- Identificar os conceitos básicos de Orientação a Objetos;
- Identificar a sintaxe básica para trabalhar com Orientação a Objetos em PHP;
- Testar exemplos de programação Orientada a Objetos.

Conteúdos da Aula 

Acompanhe os conteúdos desta aula. Se você preferir, assinale-os à medida em que for estudando.

- Revisando os conceitos básicos da Orientação a Objetos;
- Trabalhando com Classes e Objetos;
- Exercícios Propostos.

1 REVISANDO CONCEITOS BÁSICOS DA ORIENTAÇÃO A OBJETOS 

O foco do nosso aprendizado neste curso é conhecer a linguagem PHP e as características principais para desenvolvermos aplicações para Internet. O tema **Orientação a Objetos**, que verificaremos neste capítulo, abordará alguns conceitos básicos da Orientação a Objetos e também demonstrará como podemos, por meio da linguagem PHP, aplicar seus conceitos no desenvolvimento de aplicações.

Esses conceitos serão suficientes para compreendermos como a linguagem PHP implementa a programação orientada a objetos.

1.1 Classes e Objetos

Na definição básica da orientação a objetos, uma classe pode ser definida como a estrutura que cria novos objetos. Isso significa que todos os objetos utilizados em um sistema são criados por meio da estrutura das classes.

O que são os objetos e para que servem? Qual a vantagem da utilização dos objetos em uma aplicação? Vamos responder essas questões em etapas.

Para compreendermos o que é um objeto, vamos relembrar o conceito de variável: utilizada em uma aplicação para armazenar um valor, temporariamente, em memória, durante a execução de uma aplicação. Essa variável poderá ter o seu valor alterado, porém isto varia de acordo com os procedimentos definidos na aplicação.

Vimos que podemos utilizar variáveis do tipo array para armazenar mais de um valor em memória. Criamos um array com algumas posições e atribuímos valores nessas posições para executar diversos procedimentos durante a execução da aplicação.

Da mesma forma, os objetos também serão armazenados em memória, porém os objetos possuem características diferentes. Vamos imaginar que estamos desenvolvendo um sistema acadêmico onde há a manutenção do cadastro de alunos. Por meio dessa manutenção é possível: cadastrar novos alunos, alterar cadastros de alunos existentes, apagar e visualizar o cadastro completo do aluno.

Por meio da funcionalidade: **Visualizar Cadastro Completo do Aluno** será

visualizar todos os dados do cadastro do aluno. Nessa tela serão mostrados os seguintes dados: Matrícula, Nome, Endereço, Telefone do Aluno. Se quisermos trabalhar com os dados de um Aluno nessa tela, permitir a alteração dos dados, por exemplo, teríamos que armazenar todos os dados em variáveis. Porém essas variáveis e seus valores estariam armazenados na memória em locais diferentes, ou seja, não estariam agrupadas em um único local para acesso. Isso poderia aumentar o tempo na busca pelos dados das variáveis em memória, principalmente quando existirem muitos campos no cadastro de Alunos. Esse problema poderia ser solucionado se todos os dados estivessem armazenados agrupados na memória.

Quando utilizamos objetos para esses casos, conseguimos agrupar as variáveis em um único local, permitindo rápido acesso ao dado armazenado em memória. No exemplo do Aluno, utilizando o conceito de objetos, criaríamos um objeto Aluno, que seria armazenado em memória e, dentro desse objeto, seriam agrupados e armazenados os valores de suas variáveis. Em orientação a objetos, as **variáveis agrupadas** dos objetos são chamadas de **atributos**.

Assim, o objeto Aluno teria os seguintes atributos: Matrícula, Nome, Endereço e Telefone, que estariam agrupados em um único local em memória, facilitando o acesso aos dados armazenados e também a manipulação desses dados. A vantagem, nesse caso, é permitir o agrupamento dos dados do Aluno em um único local, fazendo grande diferença no desempenho de uma aplicação.

Além disso, utilizar objetos nos permite ainda outro recurso, a possibilidade de definir algumas funcionalidades específicas para cada objeto.

No exemplo do objeto Aluno, definimos como atributos desse objeto: Matrícula, Nome, Endereço e Telefone. Os valores dos atributos serão os dados do Aluno armazenados no sistema e, para cada registro de Aluno no sistema existirá um objeto Aluno. Como os atributos estão agrupados em um único local, é possível definir funcionalidades para o tratamento dos dados destes atributos.

Analisando novamente as funcionalidades da manutenção do cadastro de Alunos, há uma funcionalidade que permite a alteração de qualquer um dos registros de Alunos existentes no sistema. Como é permitida a alteração de qualquer registro, identificamos que essa funcionalidade é igual para todos os objetos Aluno, certo? O sistema tem que permitir que todos os objetos Aluno possam ter os dados de seus

atributos alterados.

Vamos analisar outra funcionalidade, a que permite apagar os dados dos alunos. Essa funcionalidade também permitirá apagar qualquer registro de aluno, salvo algumas restrições que podem ser definidas na aplicação. Logo, a funcionalidade **remover registro de aluno** também é comum para todos os objetos Aluno.

Após essa análise, podemos identificar os seguintes itens:

1. Todos os objetos Aluno terão os atributos: Matrícula, Nome, Endereço e Telefone, dados que são comuns entre todos os registros de alunos do sistema
2. Todos os objetos Aluno poderão ter as funcionalidades: Alterar Dados do Aluno, Remover Dados do Aluno que também são funcionalidades comuns para todos os registros de alunos do sistema

Então, além de permitir armazenar os dados de seus atributos em único local para rápido acesso, um objeto também permite a definição de funcionalidades para o tratamento dos valores dos seus atributos. Essa é uma das principais vantagens da utilização de objetos em aplicações: manter os atributos e funcionalidades de uma informação, no nosso exemplo a informação Aluno, agrupados em uma única estrutura: o objeto.

Conhecendo o que é um objeto e quais as vantagens na sua utilização, vamos relembrar a definição da classe: pode ser definida como uma estrutura que cria objetos.

É por meio da classe que vamos conseguir criar objetos em nossas aplicações. A classe então possuirá a estrutura do objeto que vai criar. Vamos utilizar o exemplo do objeto Aluno para verificar o formato de uma classe:


```
Classe Aluno
     Atributos:
          Matrícula
          Nome
          Endereço
          Telefone
```

```
Funcionalidades:
     Cadastra Aluno
     Altera dados do Aluno
     Apaga dados do Aluno
     Retorna dados do Aluno
```

Analisando o formato da classe aluno, verificamos que foram definidos os atributos utilizados pelo objeto Aluno e também suas funcionalidades. A função da classe é ter a estrutura final de como o objeto deverá ser criado na memória. Assim, a classe deve ter a relação de todos os atributos e funcionalidades que o objeto vai ter depois de criado. Dessa forma, quando precisarmos criar um objeto Aluno, sempre iremos chamar a classe Aluno para saber qual é a estrutura do Aluno que deve ser criada.

Resumindo, podemos definir classe e objeto da seguinte forma:

• Classe é o formato da estrutura que um objeto vai possuir. Sua função é criar objetos baseados nesse formato.

• Objetos são criados por meio de uma classe e permitem agrupar dados de atributos e funcionalidades de uma informação. Os objetos irão assumir o formato de estrutura definido na classe que o criou.

1.2 Estrutura das Classes

Uma classe é composta dos seguintes itens:

• Atributos

• Método Construtor (parâmetros do método)

• Método Destrutor (parâmetros do método)

• Métodos de Funcionalidades (parâmetros do método)

No tópico anterior, vimos que os atributos são as posições em memória dos objetos que armazenam os dados. Os atributos de uma classe devem ser definidos de acordo com os dados dos objetos que desejamos armazenar e/ou manipular.

Quando definimos a estrutura de uma classe, podemos incluir um método construtor. O método construtor sempre será executado toda vez que a classe for

utilizada para criar um novo objeto. Uma classe pode ter mais de um método construtor, mas para isso deverá ser aplicado o conceito de sobrecarga de métodos dos conceitos da Orientação a Objetos.

Fique sabendo

A sobrecarga de métodos é um recurso da Orientação a Objetos que permite a criação de um método com o nome de um outro já existente, porém com parâmetros diferentes – quantidade de parâmetros que o método irá receber e/ou tipo de dados desses parâmetros.

O método destrutor, por sua vez, é sempre executado quando um objeto é apagado da memória. Esta ação acontece quando a aplicação ou um procedimento específico da aplicação é executado e a utilização do objeto em memória não é mais necessária. Para evitar que os objetos fiquem em excesso na memória – objetos que não serão mais utilizados pela aplicação no procedimento – são destruídos e removidos da memória. Assim, quando a ação é executada, o método destrutor da classe é executado.

Os métodos de funcionalidades de uma classe são aqueles que possuem as funcionalidades de um objeto. Para cada funcionalidade do objeto, teremos um método de funcionalidade definido dentro da classe. A ação que o método irá executar também será definida dentro da classe, pois a ação da funcionalidade já estará completa e disponível para ser utilizada pelo objeto, assim que o objeto for criado.

Todos os métodos podem receber parâmetros da aplicação/procedimento que o chamou. Os parâmetros são opcionais e podem ser utilizados para auxiliar na função que o método irá desempenhar. Os parâmetros geralmente são variáveis que a aplicação/procedimento utiliza para o auxílio da execução de alguma função. Vamos considerar, por exemplo, que temos uma classe que cria objetos e realiza cálculos matemáticos. Nessa classe há o método: soma que recebe dois números como parâmetro e retorna a soma desses dois números. Nesse caso, os números foram passados por parâmetro para auxiliar na execução da função do método: realizar a soma dos números recebidos por parâmetro.

No próximo tópico, verificaremos qual é a sintaxe para a definição das classes

e como é feita a criação dos objetos por meio de uma classe.

2 TRABALHANDO COM CLASSES E OBJETOS 

Quando definimos uma classe em PHP, devemos utilizar a seguinte sintaxe:

```
class NomeDaClasse{

      [modificador] atributo1;
      [modificador] atributoN;

      [modificador] function método1([parâmetros]){
            Instrução1;
            InstruçãoN;
      }

      [modificador] function método1([parâmetros]){
            Instrução1;
            InstruçãoN;
      }
}
```

Observemos que existem alguns itens entre colchetes, são os modificadores e os parâmetros, representados entre colchetes, pois a definição deles é opcional. Isso significa que não é necessário informar estes itens, quando definimos uma classe. Utilizamos parâmetros nos métodos somente quando este precisa realizar alguma função que depende do valor de uma variável externa, caso contrário, a passagem de parâmetros para o método é descartada.

Os modificadores indicados na sintaxe da classe são os comandos que definem a visibilidade dos itens da classe e, conseqüentemente, do objeto criado por meio dela. A visibilidade identificará como os atributos e métodos serão utilizados por toda a aplicação. Há três tipos de visibilidade:

1. **Public** – deixa os atributos/métodos públicos para toda a estrutura da aplicação.
2. **Protected** – permite que somente algumas páginas e objetos do sistema tenham acesso aos atributos e métodos da classe, que ficam protegidos e poderão ser acessados somente por páginas e objetos que fazem parte do mesmo pacote – mesmo caminho físico onde se encontra o arquivo da classe.
3. **Private** – tornará os atributos/métodos privados. Isso significa que somente a própria classe/objeto poderá acessar e manipular os atributos e métodos definidos como privados.

Geralmente, os atributos de uma classe são definidos como privados – utilizam o modificador private no início da declaração - porque, segundo o conceito de encapsulamento da orientação a objetos, somente o próprio objeto poderá manipular seus atributos, mantendo a integridade dos dados dos objetos e evitando que procedimentos alheios ao objeto prejudiquem os dados armazenados.

Vamos, agora, criar nossa primeira classe em PHP, utilizando o exemplo a seguir.

Página: **class.operacao.php**

```
<?
// declara a classe
// o objeto que for criado por meio desta classe assumirá
a visibilidade e nome // da classe
class Operacao{

     //declara um atributo
     private $resultado;

     // declara o método adição
// o método recebe dois números para executar o cálculo e
retornar o
// resultado
```

```
Public function adicao($numero1, $numero2){
                $this->resultado = $numero1 + $numero2;
                return $this->resultado;
            }

    }
    ?>
```

Podemos executar o arquivo da classe no VertrigoServ para validar se não há erro, porém não será mostrada nenhuma mensagem na tela, porque criamos apenas o formato da classe, ou seja, a estrutura que um objeto irá possuir. Para testarmos o funcionamento dessa classe, vamos criar outro arquivo de exemplo que utilizará a estrutura da classe para criar objetos: Operacao. Quando utilizamos uma classe para criar um objeto, dizemos que instanciamos um novo objeto. Todos os objetos são instanciados por meio de uma classe já definida. Esse é também um conceito da Orientação a Objetos.

Vamos agora criar um novo arquivo de exemplo para testarmos o funcionamento da classe: Operacao. Nesse arquivo vamos disponibilizar um formulário HTML com dois campos para que possamos informar dois números. Depois iremos submeter os valores dos campos para esse mesmo arquivo – vamos utilizar a função que verifica se existe uma variável setada, nesse caso, as duas variáveis que serão enviadas pelo navegador – e instanciar um objeto Operacao para utilizar o método: adicao desse objeto.

Página: **exemplo29_form.php**

```
<HTML>
<HEAD>
 <TITLE>Exemplo 29</TITLE>
</HEAD>
<BODY>
<!--
Este é um comentário em HTML!
```

```
Quando queremos que o formulário envie as variáveis para a
mesma página, não especificamos o atributo: action do co-
mando form
-->
<form name=”form1”  method=”post”>
      Número 1: <input type=”text” name=”numero1”><br>
      Número 2: <input type=”text” name=”numero2”><br>
     <input type=”submit” name=”btnEnviar” value=”Calcular
Soma”>
</form>

<?
//verifica se existe a variável de navegador numero1
if(isset($_POST[‘numero1’])){

  //cria novas variáveis com base nas variáveis de navega-
dor
  $num1 = $_POST[‘numero1’];
  $num2 = $_POST[‘numero2’];

   //verifica se as variáveis possuem valor diferente de
vazio
  if(!empty($num1) && !empty($num2)){

       // inclui o arquivo que possui a estrutura da classe
para permitir a
        // criação do objeto
        require_once(“class.operacao.php”);

        // cria o objeto
        $objOperacao = new Operacao();
        // a partir de agora a variável: $objOperacao é um
```

```
objeto que possui a estrutura
       // da classe operação
      // vamos então chamar o método adicao e obter o re-
sultado da soma e mostrar
      // o resultado na tela
      print $objOperacao->adicao($num1,$num2) . "<br>";

       // como já não precisamos do objeto: $objOperacao,
vamos destruí-lo
      Unset($objOperacao);
  }

}
?>
</BODY>
</HTML>
```

Ao analisarmos o código fonte do exemplo29.php, podemos identificar três comandos que ainda não foram utilizados:

1. require_once(arquivo) – esse comando permite incluir um arquivo php dentro de outro arquivo. Permite estruturarmos e organizarmos melhor os arquivos da aplicação. Existem outros dois comandos semelhantes ao require_once(): include() e require().
2. new NomeClasse() – permitirá a criação de um objeto com base na estrutura da classe identificada.
3. -> - utilizado somente pelos objetos para chamar e executar as instruções de um método.

As demais instruções utilizadas no exemplo29.php já foram vistas nos capítulos anteriores.

Ao executarmos o exemplo29.php no VertrigoServ, será mostrado o formulário HTML com dois campos de texto para preenchimento. Após informarmos os números e clicarmos no botão: Calcular Soma, as validações serão feitas, o objeto Operacao

será criado e o método: adicao executado. Como o método adicao retorna o resultado, podemos imprimir o resultado na tela. Ao final, como não iremos mais utilizar o objeto: Operação, podemos destruí-lo, removendo-o da memória.

Síntese 

Nesta aula fizemos uma revisão sobre os conceitos básicos da Orientação a Objetos. Estudamos a sintaxe básica da programação orientada a objetos em PHP e analisamos alguns exemplos da aplicação da programação orientada a objetos em PHP.

Exercícios propostos 

1) A diferença entre uma classe e um objeto é (apenas uma alternativa está correta):

a. O objeto é armazenado em memória e a classe não

b. A classe possui a estrutura de um objeto

c. O objeto possui a estrutura de uma classe

d. A classe armazena variáveis na memória

2) A sintaxe correta para criar um objeto é (apenas uma alternativa está correta):

a. $objeto = TipoObjeto();

b. objeto = new TipoObjeto();

c. objeto = TipoObjeto()

d. $objeto = new TipoObjeto();

3) Exercício prático:

Desenvolva os métodos: subtração, multiplicação e divisão para a classe: class.operacao.php e altere o formulário do exemplo29.php para permitir que o usuário selecione a operação que deseja realizar.

Aula 8

CONTROLE DE SESSÃO

8

Caro aluno(a)!

Nesta oitava aula você estudará sobre o controle de sessão em PHP. Verificaremos o que é uma sessão e como podemos utilizá-la em aplicações para Internet. O controle de sessão é muito utilizado, principalmente por aplicações que possuem restrição de acesso de usuários.

Bons Estudos!

Objetivos da Aula 

Ao final desta aula, você deverá ser capaz de:

- Identificar como é o funcionamento das sessões;
- Testar o controle de sessão;
- Utilizar o controle de sessão em suas aplicações.

Conteúdos da Aula 

Acompanhe os conteúdos desta aula. Se você preferir, assinale-os à medida em que for estudando.

- Introdução ao controle de sessão;
- Testando o controle de sessão em PHP;
- Exercícios propostos.

1 INTRODUÇÃO AO CONTROLE DE SESSÃO 

No primeiro capítulo deste livro-texto, vimos como utilizar a arquitetura cliente-servidor quando acessamos aplicações na Internet. Assim, todas as aplicações desenvolvidas para Internet estarão instaladas em um servidor que disponibilizará a aplicação aos usuários, por meio da conexão entre a máquina cliente e o servidor.

Quando o usuário abre o navegador de Internet e faz a primeira requisição de acesso à aplicação no servidor, é criada uma sessão de conexão entre a máquina cliente e o servidor, mantida pela própria máquina cliente até a finalização do acesso à aplicação. A sessão será encerrada somente quando o usuário fechar a aplicação ou por algum imprevisto na conexão cliente-servidor.

O controle de sessão é muito utilizado por aplicações que possuem acesso restrito. Utilizaremos alguns exemplos que demonstram esse tipo de recurso.

No capítulo 4 deste livro-texto – Trabalhando com Variáveis e Constantes - tivemos uma pequena introdução às variáveis de sessão. Quando uma conexão cliente-servidor é estabelecida, a sessão do cliente é criada. Após a criação da sessão, é possível utilizá-la para criar variáveis.

A sessão é uma identificação da conexão do cliente com o servidor e todas as variáveis de sessão criadas estarão automaticamente relacionadas com essa que será mantida pela máquina cliente. Caso seja encerrada, todas as variáveis relacionadas com a sessão também serão encerradas. Dessa forma, quando trabalhamos com variáveis de sessão é necessário sempre verificar se ela está ativa e depois, se a variável de sessão existe. A sintaxe para criação de uma variável de sessão é:

$_SESSION[‘nome_da_variavel’] = valor_da_variavel

A variável de sessão, assim como as outras, também pode receber qualquer valor, pois o interpretador PHP utilizará a tipagem automática para atribuir o tipo de dado à variável.

Para utilizarmos variáveis de sessão dentro de uma aplicação, precisamos fazer uso das seguintes funções do PHP:

• session_start() = inicia a sessão que permitirá a manipulação de variáveis de sessão. Função sempre utilizada quando houver procedimentos com variáveis

de sessão.

• session_is_registered(nome_da_variavel) - verifica se a variável de sessão existe na sessão. Essa função irá retornar um valor booleano - verdadeiro ou falso.

• session_destroy() = finaliza todas as variáveis de sessão registradas.

Se, por algum motivo, desejarmos encerrar uma única variável de sessão, deveremos utilizar a função unset(nome_da_variavel), pois a função session_destroy(), finaliza todas as variáveis mantidas na sessão.

2 TESTANDO O CONTROLE DE SESSÃO EM PHP 

Para testarmos o controle de sessão em PHP, utilizaremos três arquivos de exemplo: exemplo30.php, que possuirá um formulário HTML de autenticação de usuários; o exemplo31.php, que fará a validação dos dados informados no formulário HTML, criará uma variável de sessão e redirecionará para o exemplo32.php. O exemplo32.php estará validando a criação da variável de sessão e mostrando uma mensagem na tela.

Página: **exemplo30_form.php**

```
<HTML>
<HEAD>
 <TITLE>Exemplo 30</TITLE>
</HEAD>
<BODY>
<form name=”form1” action=”exemplo31.php”  method=”post”>
      Login: <input type=”text” name=”login”><br>
      Senha: <input type=”password” name=”senha”><br>
       <input type=”submit” name=”btnEnviar” value=”Autenticar
Usuário”>
</form>
</BODY>
</HTML>
```

Página: **exemplo31_form.php**

```
<?
//inicia a sessão
session_start();

if(isset($_POST['login'])){

  $login = $_POST['login'];
  $senha = $_POST['senha'];

  //verifica se o login e senha informados estão corretos
  if($login == "user" && $senha == "1234"){

    //cria a variável de sessão
    $_SESSION['user'] = $login;
  }

}
// redireciona para a tela exemplo32.php
header("location:exemplo32.php");
?>
```

Página: **exemplo32_form.php**

```
<?
//inicia a sessão
session_start();
?>
<HTML>
<HEAD>
<TITLE>Exemplo 32</TITLE>
</HEAD>
<BODY>
```

```
<?
//verifica se a variável de sessão foi criada
if(session_is_registered("user")){
  print "Sessão criada com sucesso.";
} else {
  print "A sessão não foi criada, login e senha incorre-
tos";
}
//destroi a sessão
session_destroy();
?>
<br><br>
<a href ="exemplo30.php">Voltar para tela de Autenticação</
a>
</BODY>
</HTML>
```

Teste os exemplos informando o login e senha corretos, definido no exemplo31.php e também incorretos para validar a criação da variável de sessão. Uma sessão pode ter "N" variáveis de sessão, porém, por medida de segurança, não é aconselhável manter muitos dados em sessão.

Antes de finalizarmos esse capítulo, vamos analisar uma nova instrução utilizada nos exemplos acima:

header("location:exemplo32.php");

O comando header pode ser utilizado para vários procedimentos. No exemplo acima, utilizamos o comando para que, ao final da execução do exemplo31.php, o usuário fosse redirecionado para o exemplo32.php. O item "location:" é o subcomando do header que permite esse tipo de recurso. Porém somente será possível utilizar este redirecionamento quando nenhum tipo de impressão foi feito na tela. Se, por acaso, não for feita nenhuma impressão na tela, mas houver algum comando HTML como o <HTML></HTML> ou outro comando HTML qualquer, o recurso não vai funcionar e será retornado um erro. Portanto, utilize esse recurso somente quando houver

certeza de que não será impresso nada na tela do usuário.

Outra dica importante é com relação à função: session_start(). Se você verificar o exemplo32.php, a função está logo no topo da página, antes mesmo do comando HTML: <HTML></HTML>. O session_start() também tem o mesmo tipo de requisito que o comando header, ou seja, deve ser utilizado antes de qualquer impressão na tela, mesmo que sejam apenas comandos HTMLs.

Síntese 

Nesta aula estudamos como funciona o Controle de Sessão; introduzimos alguns conteúdos sobre Controle de Sessão em PHP e analisamos exemplos de testes para verificar como funcionam as sessões em PHP.

Exercícios propostos 

1) A sessão pode ser definida como (apenas uma alternativa correta):

a. Uma variável

b. Uma constante

c. Uma identificação de conexão

d. Uma estrutura de controle

2) Quando utilizamos variáveis de sessão, conseqüentemente estas variáveis estarão armazenadas:

a. No banco de dados

b. No HD – disco físico

c. Na memória

d. Na sessão da máquina cliente

3) A sessão é criada quando:

a. Um usuário abre o navegador

b. Um usuário faz uma autenticação correta em um sistema

c. Uma máquina cliente faz uma requisição ao servidor

d. Uma máquina cliente efetiva uma conexão de acesso com o servidor

4) Quando precisamos realizar operações com variáveis de sessão devemos sempre utilizar a função:

a. Session_is_registered()

b. Session_destroy()

c. $_SESSION[]

d. Session_start()

5) A função que encerra todas as variáveis de sessão mantidas na sessão da máquina cliente é a:

a. Session_is_registered()

b. Session_destroy()

c. $_SESSION[]

d. Session_start()

Aula 9

TRABALHANDO COM BANCO DE DADOS – MYSQL

9

Tutu Virtual

Caro aluno(a)!

Nesta nona aula estudaremos as principais formas de integração do PHP com o banco de dados MySQL. Identificaremos as principais funções para conexão e manipulação dos dados de um banco de dados e os procedimentos necessários para a conexão com esse banco.

Bons Estudos!

Objetivos da Aula 

Ao final desta aula, você deverá ser capaz de:

• Identificar quais são as funções necessárias para trabalhar

com o MySQL;

• Identificar os passos para fazer uma conexão com o MySQL;

• Testar a integração do PHP com o MySQL.

Conteúdos da Aula 

Acompanhe os conteúdos desta aula. Se você preferir, assinale-os à medida em que for estudando.

• Introdução;

• Conexão com o MySQL;

• Executando comandos de manipulação de dados;

• Executando comandos de seleção de dados;

• Exercícios Propostos.

1 INTRODUÇÃO 

Grande parte das aplicações disponíveis na Internet, atualmente, possuem conteúdo dinâmico, ou seja, possuem o conteúdo armazenado em um banco de dados. Manter o conteúdo atualizado desse tipo de aplicação é muito mais fácil e rápido do que manter o conteúdo de aplicações estáticas, desenvolvidas apenas com HTML, pois não é necessário alterar o leaiute da página a cada alteração de conteúdo. Basta que esteja definido onde serão publicados os conteúdos armazenados em banco de dados e, conforme são atualizados serão disponibilizados automaticamente no local definido.

Para que essa facilidade na manutenção de conteúdos se torne real, é necessário utilizar uma linguagem de programação de aplicações para Internet que tenha suporte e integração a um banco de dados.

A linguagem PHP, atualmente na versão 5, possui suporte e integração em grande parte dos bancos de dados. Além disso, a combinação: PHP e MySQL tem sido um par perfeito desde o início da popularidade do PHP.

Assim, nesse capítulo, estudaremos as formas de integração que a linguagem PHP disponibiliza para podermos desenvolver aplicações com conteúdo dinâmico armazenado no banco de dados MySQL.

Se você fez a instalação do VertrigoServ, também já tem instalado e configurado em sua máquina a versão 5 do MySQL.

1.1 Conectando com o MySQL

A linguagem PHP disponibiliza várias funções que permitem a integração com o banco de dados MySQL. Identificar essas funções no manual do PHP é bem simples, pois todas elas iniciam com o termo: mysql.

A conexão com o servidor de banco de dados MySQL, em PHP, é feita por meio da função mysql_connect, que tem a seguinte sintaxe:

```
mysql_connect(nome_servidor[:porta], login_usuario, senha_usuario);
```

Analisando a sintaxe da função de conexão com o servidor, podemos verificar

que é necessária a passagem de três parâmetros:

1. O nome do servidor onde está instalado o MySQL e, caso o MySQL não esteja instalado na porta padrão, deve-se informar, após o nome do servidor, o número da porta;
2. O login do usuário que possui acesso ao servidor MySQL e também ao banco de dados que se deseja conectar;
3. E a senha do usuário que está conectando no servidor MySQL.

A função de conexão irá retornar um valor inteiro, que é o identificador da conexão estabelecida e deverá ser armazenado numa variável para ser utilizado depois na seleção do banco de dados e também nos procedimentos de manipulação com o banco de dados.


Veja o exemplo de conexão com o banco de dados:

```
$conexao = mysql_connect("localhost", "root", "vertrigo");
```

Utilizando o exemplo acima, se a conexão for bem sucedida, ou seja, se existir um servidor MySQL chamado: localhost que possua o usuário e a senha fornecida, o identificador da conexão com o servidor ficará armazenado na variável $conexão.

Após efetuar com sucesso a conexão com o MySQL, será necessário selecionar o banco de dados que possui os dados da aplicação. Essa seleção é feita por meio da função int mysql_db_query, que possui a seguinte sintaxe:

```
mysql_db_query(nome_banco, identificador_conexao);
```

A função de seleção do banco de dados que irá retornar é 0, se não encontrar o banco de dados no servidor especificado pelo identificador da conexão, e 1, em caso de sucesso. Veja o exemplo de seleção do banco de dados revenda – que será utilizado no nosso estudo de caso:


```
mysql_select_db("revenda", $conexao);
```

Caso a seleção do banco de dados seja feita com sucesso, será possível realizar qualquer manipulação nos dados desse banco de dados, desde que o usuário utilizado para a conexão com o servidor tenha permissão para realizar tais operações.

1.2 Executando comandos de manipulação de dados

Para executar comandos de manipulação de dados no MySQL, por meio da

linguagem PHP, é necessário que a conexão com o MySQL e a seleção do banco de dados já tenha sido efetuada com sucesso. Caso contrário, a tentativa de manipulação irá retornar um erro.

Para manipular dados em um banco de dados, utilizamos os seguintes comandos SQL:

1. INSERT – para inserir registros
2. UPDATE – para alterar registros
3. DELETE – para excluir registros

Não entraremos em detalhes sobre a sintaxe e os padrões da linguagem SQL aqui, pois esse conhecimento é um pré-requisito para cursar esta disciplina.

Os comandos SQL serão executados na linguagem PHP por meio da função: mysql_query cuja sintaxe é:

```
mysql_query("comando_sql", identificador_da_conexao);
```

O retorno da função será 0, em caso de falha, e 1, em caso de sucesso. Quando a função for executada com sucesso, significa que instrução SQL foi executada corretamente pelo servidor MySQL e a manipulação dos dados foi executada com sucesso.

1.3 Executando comandos de seleção de dados

Os comandos de seleção de dados também são executados por meio da função mysql_query. A diferença nesse caso, é que a função terá um valor diferente de 0 e 1. O valor de retorno, nesse caso, é um identificador que possui a identificação do resultado da seleção realizada. Para visualizarmos o resultado da seleção, será necessário utilizarmos a função mysql_result, cuja sintaxe é:

```
mysql_result(identificador_resultado, linha_registro, campo_registro);
```

O identificador do resultado é obtido por meio do retorno da função mysql_query. A linha especifica a linha de registro a ser exibida, já que uma query SELECT pode retornar diversas linhas de registros, e campo é o identificador do campo a ser exibido. Vejamos um exemplo:

EX.

```
$consulta = "SELECT nome, email FROM usuario WHERE nome
LIKE 'João'";
```

```
$resultado = mysql_query($consulta, $conexao);
Print "Nome: " . mysql_result($resultado,0,"nome") . "<br>
";
Print "e-mail: " . mysql_result($resultado,0,"email") .
"<br>";
```

A função mysql_result não é a única que permite o tratamento do resultado de um comando de seleção. Inclusive, para resultado com muitas linhas de registro, podemos dizer que ela é desaconselhável. Por isso, é aconselhável utilizar outra função, por exemplo, a função mysql_fetch_row, que possui a seguinte sintaxe:

```
mysql_fetch_row(identificador_resultado);
```

O retorno dessa função será um array de registros. Assim torna-se mais fácil tratar um resultado com várias linhas, e sem utilizar os nomes dos campos na rotina de tratamento do resultado. Vamos analisar o mesmo exemplo, porém agora utilizando a função mysql_fetch_row:

```
$consulta = "SELECT nome, email FROM usuario";
$resultado = mysql_query($consulta, $conexao);
while ($linha = mysql_fetch_row($resultado)) {
print "Nome: " . $linha[0] . "<BR>";
print "e-mail: " . $linha[1] . "<BR>";
}
```

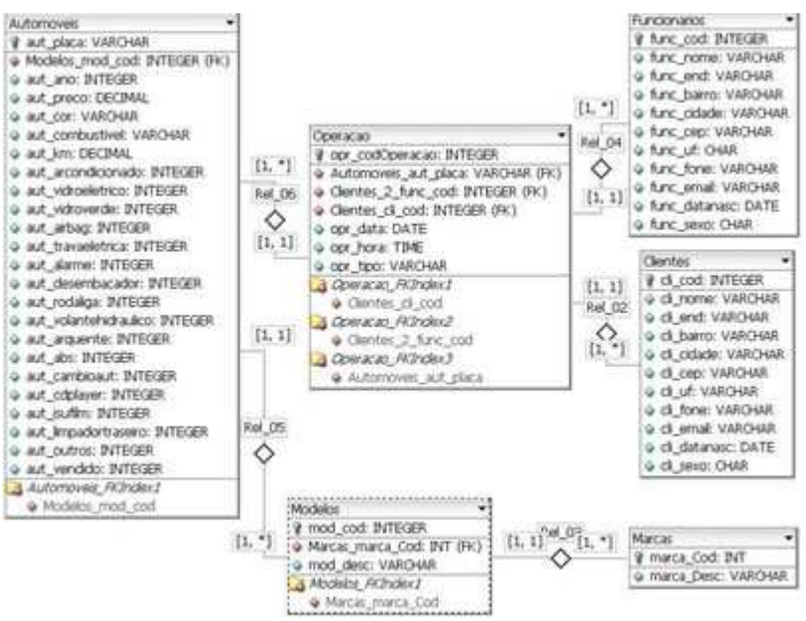
Nessa aula conhecemos apenas algumas funções básicas que permitem a integração do PHP com o banco de dados MySQL, porém existem outras funções que auxiliam nessa integração. Na próxima aula desenvolveremos um estudo de caso e aplicaremos os conceitos de integração com o banco de dados MySQL vistos nesse capítulo.

Síntese

Nesta aula estudamos as principais funções para integração entre o PHP e o MySQL. Também estudamos os procedimentos que devem ser executados para a conexão com o MySQL, além de termos analisado exemplos de integração e manipulação dos dados de um banco de dados MySQL.

Exercícios propostos

Para resolver os exercícios seguintes, é obrigatória a criação do banco de dados revenda do modelo ER abaixo. Este banco de dados também será utilizado no estudo de caso da próxima aula.



1) Crie um arquivo em PHP que faça a conexão com o banco de dados MySQL, selecione o banco de dados revenda e em seguida, utilize as funções de integração para executar os seguintes comandos SQL:

a. Inserir os seguintes registros na tabela Marcas:

i. 1 – Citroen

ii. 2 – Audi

iii. 3 – Renault

b. Selecionar e imprimir todos os registros inseridos na tabela Marcas.

c. Alterar a descrição do registro número 3 da tabela Marcas para Pegeout.

d. Selecionar e imprimir todos os registros inseridos na tabela Marcas depois da alteração.

Aula 10

ESTUDO DE CASO: REVENDA DE AUTOMÓVEIS

10

Caro aluno(a)!

Nesta décima aula reveremos todos os conceitos vistos até agora e consolidaremos nossos conhecimentos com o início do desenvolvimento de uma aplicação para revenda de automóveis.

Verificaremos os passos principais para o desenvolvimento de uma aplicação e efetivaremos uma funcionalidade completa para manutenção dos dados de uma tabela no banco de dados. Ao finalizar essa aula, ficará como desafio para você, desenvolver o restante da aplicação seguindo as dicas e o conhecimento adquirido na disciplina.

Bons Estudos e bom desenvolvimento!

Objetivos da Aula

Ao final desta aula, você deverá ser capaz de:

- Consolidar os conhecimentos adquiridos na disciplina;
- Iniciar o desenvolvimento de uma aplicação de revenda de automóveis.

Conteúdos da Aula

Acompanhe os conteúdos desta aula. Se você preferir, assinale-os à medida em que for estudando.

- Características da Aplicação;
- Desenvolvendo as Classes de Conexão;
- Desenvolvendo as Classes de Negócio;
- Desenvolvendo a interface com usuário.

1 CARACTERÍSTICAS DA APLICAÇÃO

A aplicação que iniciaremos nessa aula refere-se ao modelo ER, da pág. , da aula anterior. A aplicação tem como objetivo atender as necessidades de uma revenda de automóveis, cuja intenção é desenvolver um site na Internet que permita ao usuário selecionar marcas e/ou modelos de automóveis e visualizar uma lista de automóveis disponíveis para venda.

Verificaremos quais são os primeiros passos para desenvolver a aplicação, criar as principais classes do sistema, a tela principal e a manutenção dos registros de uma das tabelas na administração do site. Ao finalizarmos esses itens, você já será capaz de dar continuidade na aplicação e desenvolver os demais recursos.

Ao final do capítulo, algumas sugestões de funcionalidades serão apresentadas para enriquecer a aplicação da revenda de automóveis.

2 DESENVOLVENDO AS CLASSES DE CONEXÃO

A primeira tarefa que devemos realizar, para desenvolvermos a aplicação da revenda de automóveis é a criação das classes que permitirão criar os objetos de conexão e acesso aos dados para seleção e manipulação.

Criaremos, então, duas classes:

1. A classe: class.conexaobanco.php, responsável pela conexão com o MySQL
2. E a classe: class.acessobanco.php, cujos métodos serão operações de manipulação e seleção que podem ser executadas no banco de dados.

Essas duas classes serão o coração da nossa aplicação, pois todos os objetos do sistema as utilizarão para fazer a conexão e seleção/manipulação dos dados no banco de dados.

Vamos então analisar o código fonte da classe: class.conexaobanco.php:

```
<?php
class ConexaoBanco {
//Propriedade Estática referenciando um tipo da mesma Classe
```

```
     static $instancia = false;

     private $server = "localhost";
     private $user = "root";
     private $password = "vertrigo";
     private $conexao;

     //Construtor Private - Não é possível utilizar new em out-
ras classes
     private function __construct() {

          //faz a conexo com o banco de dados
          $this->conexao = mysql_connect($this->server, $this-
>user, $this->password);
     }

     //Metodo para recuperar instancia
     public function getInstancia() {
          if            (!ConexaoBanco::$instancia)            {

     ConexaoBanco::$instancia = new ConexaoBanco();//chamando
construtor
          }
     return ConexaoBanco::$instancia;
     }

public function getConexao(){
          return $this->conexao;
     }

     //destrutor - destrói conexão com o banco
     function __destruct(){
```

```
            mysql_close($this->conexao);
        }
}
?>
```

Analisando o código fonte da classe, observamos haver um método construtor, que farár a conexão com o MySQL, por meio dos dados dos atributos privados – já selecionados com os dados de conexão com o MySQL. Nessa classe, existe também o método destrutor, que fechará a conexão com o banco de dados, quando o objeto for finalizado/destruído. A função que fecha a conexão com o banco de dados é a mysql_close.

Os outros dois métodos da classe, o getConexao: retorna o identificador da conexão realizada e o getInstancia sempre retornará o mesmo objeto em memória para os demais objetos da aplicação. Esse método foi configurado dessa forma para prevenir o problema de estouro de “pool” no banco de dados. O problema ocorre quando muitas conexões com o banco de dados são inicializadas e não são finalizadas. Essas conexões que ficam abertas irão consumir os recursos do banco até esgotar e isso gerará o famoso estouro de “pool”. No nosso caso, utilizando o objeto conexaobanco, haverá sempre uma única conexão e, ao final da aplicação, essa conexão será devidamente finalizada.

Agora vamos analisar o código-fonte da classe: class.acessobanco.php:

```
<?php
require('class.conexaobanco.php');

class AcessoBanco{

private $base = "revenda";
private $conexao;

//método construtor
public function AcessoBanco(){
```

```
//conexão com o banco de dados
          $oConexaoBanco = ConexaoBanco::getInstancia();
          $this->conexao = $oConexaoBanco->getConexao();

     }


     //método que retorna o valor do campo informado no parmet-
ro
     public function getValorCampo($paramCampo, $paramFrom,
$paramWhere){

          $valorCampo = "";

          //comando SQL que busca o valor do campo informado
          $comandoSql = "SELECT " . $paramCampo . " FROM " .
$paramFrom . "  WHERE " . $paramWhere;

          //executa o comando SQL no banco de dados
          $resultado = mysql_db_query ($this->base, $comandoSql,
$this->conexao);

//pega a qtde de registros retornados
          $qtdeRegistros = mysql_num_rows ($resultado);

          if ($qtdeRegistros > 0) {
               $valorCampo = mysql_result($resultado,0,$paramCa
mpo);
          }
return $valorCampo;

     }
```

```
//método que seleciona registros no banco de dados
     public function selectRegistro($paramTabela, $paramCampos,
$paramCondicao = “”, $paramOrdenacao = “”){

          //comando SQL que seleciona os dados no banco de da-
dos
          $comandoSql = “SELECT “ . $paramCampos . “ FROM “ .
$paramTabela;

          //inclui condio quando existir
          if (!empty($paramCondicao)){
               $comandoSql .= “ WHERE “ . $paramCondicao;
          }

          //inclui ordenação quando existir
          if (!empty($paramOrdenacao)){
               $comandoSql .= “ ORDER BY “ . $paramOrdenacao;
          }

          //executa o comando SQL no banco de dados
          $resultado = mysql_db_query ($this->base, $comandoSql,
$this->conexao);

          $i = 0;

          $arrayRegistro = array();

          //seta o array de Registros com os registros retorna-
dos
          while ($registro = mysql_fetch_array ($resultado)) {
               $arrayRegistro[$i] = $registro;
               $i++;
```

```
}

          return $arrayRegistro;

     }


     //método que insere registros no banco de dados e retorna
o último id inserido
     public function insertRegistro($paramTabela, $paramCampos,
$paramValores){

          $id = 0;

          //comando SQL que insere os dados no banco de dados
          $comandoSql = "INSERT INTO " . $paramTabela . " (" .
$paramCampos . ")  VALUES(" . $paramValores . ") ";

          //executa o comando SQL no banco de dados
          $resultado = mysql_db_query ($this->base, $comandoSql,
$this->conexao);

          //verifica se deve ser retornado o id
          $id = mysql_insert_id($this->conexao);

          //verifica se teve erro no query
          if (mysql_errno($this->conexao)){
                echo mysql_error($this->conexao);
          }

          return $id;
}
```

```
//método que altera registros no banco de dados
     public function updateRegistro($paramTabela, $paramCampos,
$paramCondicao){
          $resultado = 0;

          //comando SQL que altera os dados no banco de dados
          $comandoSql = "UPDATE " . $paramTabela . " SET " .
$paramCampos . "  WHERE " . $paramCondicao;

          //executa o comando SQL no banco de dados
          $resultado = mysql_db_query ($this->base, $comandoSql,
$this->conexao);

          //verifica se teve erro no query
          if (mysql_errno($this->conexao)){
               echo mysql_error($this->conexao);
          }

          return $resultado;

     }


     //método que altera registros no banco de dados
     public function deleteRegistro($paramTabela, $paramCondi-
cao){

          //comando SQL que apaga os dados no banco de dados
          $comandoSql = "DELETE FROM " . $paramTabela . " WHERE
" . $paramCondicao;

          //executa o comando SQL no banco de dados
```

```
$resultado = mysql_db_query ($this->base, $comandoSql, $this-
>conexao);

      }
}

?>
```

A classe acessobanco é a classe responsável pela seleção do banco de dados e por todo tipo de manipulação e seleção dos dados no banco. Essa classe possui um método para cada tipo de comando SQL, permitindo flexibilidade e organização dos comandos SQL em um único local. Observando os métodos criados, identificamos uma palavra que não tinha sido mencionada até o momento, a palavra static, que também faz parte do conceito de orientação a objetos e indica que não é necessária a criação de um objeto em memória para utilizar os métodos da classe. Isso significa que não precisamos utilizar o comando new Objeto() para utilizar o método static, basta apenas informar qual é a classe que possui o método static e chamá-lo.

Veja, a seguir, como é feita a chamada de um método static de uma classe:

```
NomeDaClasse::NomeDoMetodo();
```

Nas classes de negócio utilizaremos bastante a chamada de métodos static.

3 DESENVOLVENDO AS CLASSES DE NEGÓCIO 

Criadas as classes de conexão e acesso ao banco, precisaremos desenvolver as classes de negócio – aquelas que criarão os objetos no sistema para manipular as informações – ao final, salvar no banco de dados.

Neste livro-texto visualizaremos as classes de negócio que criarão objetos Marcas, porém existem outras classes na aplicação que possuem a mesma estrutura e que serão disponibilizadas para os testes e desenvolvimento dos demais recursos da aplicação.

As classes de negócio são divididas em duas:

1. Classe que cria o objeto “Bean” – esse termo é muito utilizado em

programação orientada a objetos e identifica as classes que possuem a estrutura das tabelas definidas no banco de dados. Para cada tabela no banco de dados será criada uma classe com a estrutura dessa tabela, cujos campos serão os atributos da classe. Para cada atributo dessa classe existirão dois métodos: um que atribui um valor para o atributo e outro que retorna o valor do atributo. São os chamados métodos: set e get da orientação a objetos.

2. Classe com a Regra de Negócio do objeto “Bean” – essa classe possuirá os métodos/funcionalidades de negócio disponíveis para o objeto “Bean” correspondente. Essa classe possuirá métodos static, para chamá-los, não será necessária a criação do objeto Regra de Negócio.

Vamos analisar o código-fonte das classes “Bean” e Regra de Negócio, criadas por meio da tabela: Marcas do banco de dados.

Classe Bean – class.marca.php

```
<?php
class Marca{

//atributos
private $cod;
private $desc;

//método construtor da classe
public function Marca(){
     $this->cod = 0;
     $this->desc = “”;
}

//métodos da classe

//cod
public function setCod($paramCod){
     $this->cod = $paramCod;
}
```

```
public function getCod(){
          return $this->cod;
     }

     //desc
     public function setDesc($paramDesc){
          $this->desc = $paramDesc;
     }

     public function getDesc(){
          return $this->desc;
     }
}

?>
```

Analisando a classe Bean da tabela: Marcas, identificamos uma estrutura bem simples – possui dois atributos e quatro métodos. Para cada atributo da classe há um método: set, que atribui valor ao atributo, e um método: get, que retorna o valor do atributo. Para todas as tabelas do sistema será necessário criar uma classe semelhante, lembrando sempre que os atributos da classe são os campos da tabela em questão.

Vamos agora analisar a classe Regra de Negócio da tabela: Marcas.

Classe Regra de Negócio – class.marcaRN.php

```
<?php
// inclusão do arquivo de classe Marca
require_once('class.marca.php');
//inclui o arquivo da classe AcessoBanco
require_once("class.acessobanco.php");
```

```
class MarcaRN{


     //retorna uma lista de objetos Marca
     public static function getListaObjeto($paramCondicao = "",
$paramOrderBy = "marca_desc"){

          //instancia o objeto AcessoBanco
          $acessoBanco = new AcessoBanco();

          //seta os valores para passar por parâmetro
          $tabela = "marcas";
          $campos = "marca_cod, marca_desc";
          $condicao = $paramCondicao;

          //pega a lista de registros
          $resultado  =  $acessoBanco->selectRegistro($tabela,
$campos, $condicao, $paramOrderBy);

class MarcaRN{


     //retorna uma lista de objetos Marca
     public static function getListaObjeto($paramCondicao = "",
$paramOrderBy = "marca_desc"){

          //instancia o objeto AcessoBanco
          $acessoBanco = new AcessoBanco();

          //seta os valores para passar por parâmetro
          $tabela = "marcas";
          $campos = "marca_cod, marca_desc";
```

```
$condicao = $paramCondicao;

          //pega a lista de registros
          $resultado   =   $acessoBanco->selectRegistro($tabela,
$campos, $condicao, $paramOrderBy);

          //gera a lista de objetos
          $lista = self::geraArrayObjeto($resultado);

          //destroi objeto
          unset($acessoBanco);

          return $lista;

     }


     //método que recebe um array de registros e retorna um ar-
ray de objetos
     private static function geraArrayObjeto($paramLista){

          $i = 0;
          $marcas = array();

          foreach ($paramLista as $indiceArray => $array) {

               //instancia o objeto
               $marca = new Marca();
               $marca->setCod($array[“marca_cod”]);
               $marca->setDesc($array[“marca_desc”]);

               $marcas[$i] = $marca;
```

```
            $i++;
      }

            return $marcas;

      }

      //retorna um objeto
      public static function getObjeto($paramCod){

            //instancia o objeto AcessoBanco
            $acessoBanco = new AcessoBanco();

            //seta os valores para passar por parmetro
            $tabela = "marcas";
            $campos = "marca_cod, marca_desc";
            $condicao = " marca_cod = " . $paramCod;

            //pega a lista de registros Categoria
            $resultado  =  $acessoBanco->selectRegistro($tabela,
$campos, $condicao);

            $marca = "";

            //verifica se retornou um resultado
            if(sizeof($resultado)>0){
                  $marca = new Marca();
                  $marca->setCod($resultado[0]["marca_cod"]);
                  $marca->setDesc($resultado[0]["marca_desc"]);
            }
```

```
     //destroi objeto
          unset($acessoBanco);

          return $marca;

     }


     public static function alterObjeto(&$Objeto){

          //instancia o objeto AcessoBanco
          $acessoBanco = new AcessoBanco();

          //seta os valores para passar por parmetro
          $tabela = "marcas";
          $campos = "marca_desc = '" . $Objeto->getDesc() .
"'";
          $condicao = "marca_cod = " . $Objeto->getCod();

          //altera o Banco de Dados
          $resultado = $acessoBanco->updateRegistro($tabela,
$campos, $condicao);

          //destroi objeto
          unset($acessoBanco);

          }

          //altera o objeto

     public static function addObjeto(&$Objeto){
```

```
     //instancia o objeto AcessoBanco
          $acessoBanco = new AcessoBanco();

          //seta os valores para passar por parmetro
          $tabela = "marcas";
          $campos = "marca_cod, marca_desc";
          $valor = $Objeto->getCod() . ",'" . $Objeto->getDesc()
. "'";

          //altera o Banco de Dados
          $resultado   =   $acessoBanco->insertRegistro($tabela,
$campos, $valor);

          //destroi objeto
          unset($acessoBanco);

     }

     //deleta o objeto
     public static function delObjeto(&$Objeto){

          //instancia o objeto AcessoBanco
          $acessoBanco = new AcessoBanco();

          //seta os valores para passar por parmetro

     $tabela = "marcas";
          $condicao = "marca_cod = " . $Objeto->getCod();

          //altera o Banco de Dados
          $resultado   =   $acessoBanco->deleteRegistro($tabela,
$condicao);
```

```
        //destroi objeto
             unset($acessoBanco);

        }

}

?>
```

A classe regra de negócio é a que permitirá a manipulação dos registros da tabela no banco de dados. No caso da classe: MarcaRN, criaremos métodos específicos para a manipulação dos dados da tabela: Marcas do banco de dados revenda.

Analisando o código fonte da classe, identificamos método para cada tipo de manipulação no banco de dados. Há dois métodos de seleção no banco: o getListaObjeto que irá retornar uma lista de objetos Marca, de acordo com a quantidade de registros existente no banco de dados e o getObjeto, que retorna um único objeto Marca. Esse segundo método é utilizado para pegar um registro específico da tabela por meio da chave primária – campo que identifica o registro no banco de dados.

Como estamos programando orientado a objetos e utilizando um banco de dados relacional para armazenar os dados, é necessário fazer um mapeamento objeto - relacional para pegar os registros que estão no banco de dados relacional e transformá-los em objetos dentro da aplicação. Assim, o método: geraArrayObjeto é o responsável pela criação da lista de objetos Marcas, com base no resultado da consulta de seleção executada no banco de dados.

Os demais métodos da classe referem-se às operações de manipulação dos registros no banco de dados. Existe um método para cada operação – comando SQL. Nesses métodos também utilizamos o mapeamento objeto – relacional, porém nesse momento transformaremos os objetos em memória nos registros do banco de dados.

Como todos os métodos da classe de negócio se referem à manipulação e seleção de registros no banco de dados, a classe: acessobanco, criada no tópico anterior, será a ponte entre o objeto e o banco de dados.

Assim, todas as tabelas do banco de dados revenda terão duas classes

correspondentes na aplicação: a classe Bean, que possuirá a estrutura da tabela, e a Regra de Negócio, com métodos que permitirão a seleção e manipulação dos registros no banco de dados.

Vamos agora criar as classes Bean e Regra de Negócio para as tabelas: Modelos e Automoveis. Verifique que as estruturas dessas classes são semelhantes às classes da tabela: Marcas. A única diferença entre elas, em relação à quantidade de atributos é o mapeamento objeto – relacional, que utilizará dados da tabela a que a classe se refere.

Classe Bean – class.modelo.php

```
<?php

class Modelo{

     //atributos
     private $cod;
     private $desc;
     private $marca_cod;

     //método construtor da classe
     public function Modelo(){
          $this->cod = 0;
          $this->desc = “”;
          $this->marca_cod = 0;
     }

     //métodos da classe

     //cod
     public function setCod($paramCod){
               $this->cod = $paramCod;
```

```
    }

    public function getCod(){
        return $this->cod;
    }

    //desc
    public function setDesc($paramDesc){
        $this->desc = $paramDesc;
    }

    public function getDesc(){
        return $this->desc;
    }

    //marca_cod

    public function setMarcaCod($paramMarcaCod){
        $this->marca_cod = $paramMarcaCod;
    }

    public function getMarcaCod(){
        return $this->marca_cod;
    }
  }

?>
```

Classe Regra de Negócio: class.modeloRN.php

```
<?php
// inclusãoo do arquivo de classe Modelo
require_once('class.modelo.php');
```

```
	//inclui o arquivo da classe AcessoBanco
	require_once("class.acessobanco.php");

	class ModeloRN{


	//retorna uma lista de objetos modelo
	public static function getListaObjeto($paramCondicao = "",
$paramOrderBy = "mod_desc"){

		//instancia o objeto AcessoBanco
		$acessoBanco = new AcessoBanco();

		//seta os valores para passar por parâmetro
		$tabela = "modelos";
		$campos = "mod_cod, mod_desc, marca_cod";
		$condicao = $paramCondicao;

		//pega a lista de registros
		$resultado  =  $acessoBanco->selectRegistro($tabela,
$campos, $condicao, $paramOrderBy);

		//gera a lista de objetos
		$lista = self::geraArrayObjeto($resultado);

		//destroi objeto
		unset($acessoBanco);

		return $lista;

	}
```

```
    //método que recebe um array de registros e retorna um ar-
ray de objetos
    private static function geraArrayObjeto($paramLista){

        $i = 0;
        $modelos = array();

        foreach ($paramLista as $indiceArray => $array) {

            //instancia o objeto
            $modelo = new modelo();
            $modelo->setCod($array[“mod_cod”]);
            $modelo->setDesc($array[“mod_desc”]);
            $modelo->setMarcaCod($array[“marca_cod”]);

            $modelos[$i] = $modelo;
            $i++;

        }

        return $modelos;

    }

    //retorna um objeto
    public static function getObjeto($paramCod){

        //instancia o objeto AcessoBanco
        $acessoBanco = new AcessoBanco();

        //seta os valores para passar por parmetro
        $tabela = “modelos”;
```

```
            $campos = "mod_cod, mod_desc,marca_cod";
            $condicao = " mod_cod = " . $paramCod;

            //pega a lista de registros Categoria
            $resultado  =  $acessoBanco->selectRegistro($tabela,
$campos, $condicao);

            $modelo = "";

            //verifica se retornou um resultado
            if(sizeof($resultado)>0){
                  $modelo = new modelo();
                  $modelo->setCod($resultado[0]["mod_cod"]);
                  $modelo->setDesc($resultado[0]["mod_desc"]);
                  $modelo->setMarcaCod($resultado[0]["marca_
cod"]);
            }

            //destroi objeto
            unset($acessoBanco);

      return $modelo;

      }


      public static function alterObjeto(&$Objeto){

            //instancia o objeto AcessoBanco
            $acessoBanco = new AcessoBanco();

            //seta os valores para passar por parmetro
```

```
            $tabela = "modelos";
            $campos = "mod_desc = '" . $Objeto->getDesc() . "'.";
            $campos = "marca_cod = '" . $Objeto->getMarcaCod() .
"";
            $condicao = "mod_cod = " . $Objeto->getCod();

            //altera o Banco de Dados
            $resultado   =   $acessoBanco->updateRegistro($tabela,
$campos, $condicao);

            //destroi objeto
            unset($acessoBanco);

      }

      //altera o objeto
      public static function addObjeto(&$Objeto){

            //instancia o objeto AcessoBanco
            $acessoBanco = new AcessoBanco();

            //seta os valores para passar por parmetro
            $tabela = "modelos";
            $campos = "mod_cod, mod_desc,marca_cod";
            $valor = $Objeto->getCod() . ",'" . $Objeto->getDesc()
. "'," . $Objeto->getMarcaCod() . "";

            //altera o Banco de Dados
            $resultado   =   $acessoBanco->insertRegistro($tabela,
$campos, $valor);

            //destroi objeto
```

```
        unset($acessoBanco);

        }

        //deleta o objeto
        public static function delObjeto(&$Objeto){

                //instancia o objeto AcessoBanco
                $acessoBanco = new AcessoBanco();

                //seta os valores para passar por parmetro
                $tabela = "modelos";
                $condicao = "mod_cod = " . $Objeto->getCod();

                //altera o Banco de Dados
                $resultado  =  $acessoBanco->deleteRegistro($tabela,
$condicao);

                //destroi objeto
                unset($acessoBanco);

        }

}

?>
```

Classe Bean: class.automovel.php

```
<?php

class Automovel{
```

```
//atributos
private $placa;
private $mod_cod;
private $ano;
private $preco;
private $cor;
private $combustivel;
private $km;
private $arcondicionado;
private $vidroeletrico;
private $vidroverde;
private $airbag;
private $travaeletrica;
private $alarme;
private $desembacador;
private $rodaliga;
private $volantehidraulico;
private $arquente;
private $abs;
private $cambioaut;
private $cdplayer;
private $isufilm;
private $limpadortraseiro;
private $outros;
private $vendido;

//método construtor da classe
public function Automovel(){
     $this->placa = "";
     $this->mod_cod = 0;
     $this->ano = 0;
     $this->preco = 0;
```

```
$this->cor = “”;
     $this->combustivel = “”;
     $this->km = 0;
     $this->arcondicionado = 0;
     $this->vidroeletrico = 0;
     $this->vidroverde = 0;
     $this->airbag = 0;
     $this->travaeletrica = 0;
     $this->alarme = 0;
     $this->desembacador = 0;
     $this->rodaliga = 0;
     $this->volantehidraulico = 0;
     $this->arquente = 0;
     $this->abs = 0;
     $this->cambioaut = 0;
     $this->cdplayer = 0;
     $this->isufilm = 0;
     $this->limpadortraseiro = 0;
     $this->outros = 0;
     $this->vendido = 0;
}

//métodos da classe

//placa
public function setPlaca($paramPlaca){
     $this->placa = $paramPlaca;
}

public function getPlaca(){
     return $this->placa;
}
```

```
}

//mod_cod
public function setModCod($paramModCod){
     $this->mod_cod = $paramModCod;
}

public function getModCod(){
     return $this->mod_cod;
}

//ano
public function setAno($paramAno){
     $this->ano = $paramAno;
}

public function getAno(){
     return $this->ano;
}

//preco
public function setPreco($paramPreco){
     $this->preco = $paramPreco;
}

public function getPreco(){
     return $this->preco;
}

//cor
public function setCor($paramCor){
     $this->cor = $paramCor;
```

```
}

public function getCor(){
     return $this->cor;
}

//combustivel
public function setCombustivel($paramCombustivel){
     $this->combustivel = $paramCombustivel;
}

public function getCombustivel(){
     return $this->combustivel;
}

//km
public function setKm($paramKm){
     $this->km = $paramKm;
}

public function getKm(){
     return $this->km;
}

//arcondicionado
public function setArCondicionado($paramArCondicionado){
     $this->arcondicionado = $paramArCondicionado;
}

public function getArCondicionado(){
     return $this->arcondicionado;
}
```

```
//vidroeletrico
public function setVidroEletrico($paramVidroEletrico){
     $this->vidroeletrico = $paramVidroEletrico;
}

public function getVidroEletrico(){
     return $this->vidroeletrico;
}

//vidroverde
public function setVidroVerde($paramVidroVerde){
     $this->vidroverde = $paramVidroVerde;
}

public function getVidroVerde(){
     return $this->vidroverde;
}

//airbag
public function setAirBag($paramAirBag){
     $this->airbag = $paramAirBag;
}

public function getAirBag(){
     return $this->airbag;
}

//travaeletrica
public function setTravaEletrica($paramTravaEletrica){
     $this->travaeletrica = $paramTravaEletrica;
}
```

```
public function getTravaEletrica(){
     return $this->travaeletrica;
}

//alarme
public function setAlarme($paramAlarme){
     $this->alarme = $paramAlarme;
}

public function getAlarme(){
     return $this->alarme;
}

//desembacador
public function setDesembacador($paramDesembacador){
     $this->desembacador = $paramDesembacador;
}

public function getDesembacador(){
     return $this->desembacador;
}

//rodaliga
public function setRodaLiga($paramRodaLiga){
     $this->rodaliga = $paramRodaLiga;
}

public function getRodaLiga(){
     return $this->rodaliga;
}

//volantehidraulico
```

```
    public function setVolanteHidraulico($paramVolanteHidrauli
co){
        $this->volantehidraulico = $paramVolanteHidraulico;
    }

    public function getVolanteHidraulico(){
        return $this->volantehidraulico;
    }

    //arquente
    public function setArQuente($paramArQuente){
        $this->arquente = $paramArQuente;
    }

    public function getArQuente(){
        return $this->arquente;
    }

    //abs
    public function setAbs($paramAbs){
        $this->abs = $paramAbs;
    }

    public function getAbs(){
        return $this->abs;
    }

    //cambioaut
    public function setCambioAut($paramCambioAut){
        $this->cambioaut = $paramCambioAut;
    }
```

```
	public function getCambioAut(){
		return $this->cambioaut;
	}

	//cdplayer
	public function setCdPlayer($paramCdPlayer){
		$this->cdplayer = $paramCdPlayer;
	}

	public function getCdPlayer(){
		return $this->cdplayer;
	}

	//isufilm
	public function setIsufilm($paramIsufilm){
		$this->isufilm = $paramIsufilm;
	}

	public function getIsufilm(){
		return $this->isufilm;
	}

	//limpadortraseiro
	public function setLimpadorTraseiro($paramLimpadorTraseiro
){
		$this->limpadortraseiro = $paramLimpadorTraseiro;
	}

	public function getLimpadorTraseiro(){
		return $this->limpadortraseiro;
	}
```

```
	//outros
	public function setOutros($paramOutros){
		$this->outros = $paramOutros;
	}

	public function getOutros(){
		return $this->outros;
	}

	//vendido
	public function setVendido($paramVendido){
		$this->vendido = $paramVendido;
	}

	public function getVendido(){
		return $this->vendido;
	}
}

?>
```

Classe Regra de Negócio: class.automovelRN.php

```
	<?php
	// inclusãoo do arquivo de classe Automovel
	require_once('class.automovel.php');
	//inclui o arquivo da classe AcessoBanco
	require_once("class.acessobanco.php");

	class AutomovelRN{

	//retorna uma lista de objetos automovel
```

```
    public static function getListaObjeto($paramCondicao = "",
$paramOrderBy = "aut_placa"){

        //instancia o objeto AcessoBanco
        $acessoBanco = new AcessoBanco();

        //seta os valores para passar por parâmetro
        $tabela = "automoveis";
        $campos  =  "aut_placa,mod_cod,aut_ano,aut_preco,aut_
cor,aut_combustivel,aut_km,aut_arcondicionado,";
        $campos   .=   "aut_vidroeletrico,aut_vidroverde,aut_
airbag,aut_travaeletrica,aut_alarme,aut_desembacador,";
        $campos  .=  "aut_rodaliga,aut_volantehidraulico,aut_
arquente,aut_abs,aut_cambioaut,aut_cdplayer,aut_isufilm,";
        $campos  .= "aut_limpadortraseiro,aut_outros,aut_ven-
dido";
        $condicao = $paramCondicao;

        //pega a lista de registros
        $resultado   =   $acessoBanco->selectRegistro($tabela,
$campos, $condicao, $paramOrderBy);

        //gera a lista de objetos
        $lista = self::geraArrayObjeto($resultado);

        //destroi objeto
        unset($acessoBanco);

        return $lista;

    }
```

```
     //método que recebe um array de registros e retorna um ar-
ray de objetos
     private static function geraArrayObjeto($paramLista){

          $i = 0;
          $automovels = array();

          foreach ($paramLista as $indiceArray => $array) {

               //instancia o objeto
               $automovel = new automovel();
               $automovel->setPlaca($array[“aut_placa”]);
               $automovel->setModCod($array[“mod_cod”]);
               $automovel->setAno($array[“aut_ano”]);
               $automovel->setPreco($array[“aut_preco”]);
               $automovel->setCor($array[“aut_cor”]);
               $automovel->setCombustivel($array[“aut_combus-
tivel”]);
               $automovel->setKm($array[“aut_km”]);
               $automovel->setArCondicionado($array[“aut_ar-
condicionado”]);
               $automovel->setVidroEletrico($array[“aut_vidro-
eletrico”]);
               $automovel->setVidroVerde($array[“aut_vidrov-
erde”]);
               $automovel->setAirBag($array[“aut_airbag”]);
               $automovel->setTravaEletrica($array[“aut_tra-
vaeletrica”]);
               $automovel->setAlarme($array[“aut_alarme”]);
               $automovel-
>setDesembacador($array[“aut_desembacador”]);
               $automovel->setRodaLiga($array[“aut_rodaliga”]);
```

```
                $automovel->setVolanteHidraulico($array["aut_
volantehidraulico"]);
                $automovel->setArQuente($array["aut_arquente"]);
                $automovel->setAbs($array["aut_abs"]);
                $automovel->setCambioAut($array["aut_cambio-
aut"]);
                $automovel->setCdPlayer($array["aut_cdplayer"]);
                $automovel->setIsufilm($array["aut_isufilm"]);
                $automovel->setLimpadorTraseiro($array["aut_
limpadortraseiro"]);
                $automovel->setOutros($array["aut_outros"]);
                $automovel->setVendido($array["aut_vendido"]);
                $automovels[$i] = $automovel;
                $i++;

          }

          return $automovels;

}


     //retorna um objeto
     public static function getObjeto($paramCod){

          //instancia o objeto AcessoBanco
          $acessoBanco = new AcessoBanco();

          //seta os valores para passar por parmetro
          $tabela = "automoveis";
          $campos
```

```
          "aut_placa,mod_cod,aut_ano,aut_preco,aut_cor,aut_
combustivel,aut_km,aut_arcondicionado,";
          $campos   .=   "aut_vidroeletrico,aut_vidroverde,aut_
airbag,aut_travaeletrica,aut_alarme,aut_desembacador,";
          $campos  .=  "aut_rodaliga,aut_volantehidraulico,aut_
arquente,aut_abs,aut_cambioaut,aut_cdplayer,aut_isufilm,";
          $campos  .= "aut_limpadortraseiro,aut_outros,aut_ven-
dido";
          $condicao = " aut_placa = " . $paramCod;

          //pega a lista de registros Categoria
          $resultado   =   $acessoBanco->selectRegistro($tabela,
$campos, $condicao);

          $automovel = "";

          //verifica se retornou um resultado

          if(sizeof($resultado)>0){
               $automovel = new automovel();
               $automovel->setPlaca($resultado[0]["aut_
placa"]);
               $automovel->setModCod($resultado[0]["mod_cod"]);
               $automovel->setAno($resultado[0]["aut_ano"]);
               $automovel->setPreco($resultado[0]["aut_
preco"]);
               $automovel->setCor($resultado[0]["aut_cor"]);
               $automovel-
>setCombustivel($resultado[0]["aut_combustivel"]);
               $automovel->setKm($resultado[0]["aut_km"]);
               $automovel-
>setArCondicionado($resultado[0]["aut_arcondicionado"]);
```

```
                $automovel->setVidroEletrico($resultado[0]["aut
_vidroeletrico"]);
                $automovel->setVidroVerde($resultado[0]["aut_
vidroverde"]);
                $automovel->setAirBag($resultado[0]["aut_air-
bag"]);
                $automovel->setTravaEletrica($resultado[0]["aut
_travaeletrica"]);
                $automovel->setAlarme($resultado[0]["aut_
alarme"]);
                $automovel->setDesembacador($resultado[0]["aut_
desembacador"]);
                $automovel->setRodaLiga($resultado[0]["aut_ro-
daliga"]);
                $automovel->setVolanteHidraulico($resultado[0][
"aut_volantehidraulico"]);
                $automovel->setArQuente($resultado[0]["aut_ar-
quente"]);
                $automovel->setAbs($resultado[0]["aut_abs"]);
                $automovel->setCambioAut($resultado[0]["aut_
cambioaut"]);
                $automovel->setCdPlayer($resultado[0]["aut_cd-
player"]);
                $automovel->setIsufilm($resultado[0]["aut_isu-
film"]);
                $automovel-
->setLimpadorTraseiro($resultado[0]["aut_limpadortraseiro"]);
                $automovel->setOutros($resultado[0]["aut_out-
ros"]);
                $automovel->setVendido($resultado[0]["aut_ven-
dido"]);
          }
```

```
        //destroi objeto
             unset($acessoBanco);

             return $automovel;

        }

}

?>
```

4 DESENVOLVENDO A INTERFACE COM O USUÁRIO 

Criadas as classes Bean e Regra de Negócio, podemos desenvolver as páginas PHP que possuirão a interface da aplicação com o usuário. Essas páginas utilizarão as classes Bean e Regra de Negócio para trabalhar com os objetos dentro da aplicação. Desenvolveremos quatro páginas de interface:

1. index.php – página principal da aplicação que permitirá ao usuário selecionar uma Marca, um Modelo relacionado à Marca selecionada e verificar a lista de carros relacionadas ao Modelo para consulta.
2. admin/marca_lst.php – página da administração que permitirá ao administrador da aplicação visualizar todos os registros armazenados na tabela: Marcas do banco de dados.
3. admin/marca_frm.php – página com o formulário que permitirá o cadastro e alteração dos registros da tabela: Marcas.
4. admin/marca_action.php – página que irá alterar os dados do objeto de acordo com a ação feita no formulário: cadastro, alteração ou remoção e chamar a classe Regra de Negócio para efetivar a alteração dos dados do objeto no banco de dados.

Seguem os códigos-fonte de cada página:

Página: index.php

```
<html>
<head>
<title>Revenda de Automóveis</title>
<meta http-equiv=”Content-Type” content=”text/html; charset=iso-
8859-1”>
</head>

<body>
<?
//inclui as classes
require_once(“classes/class.marcaRN.php”);
require_once(“classes/class.modeloRN.php”);
require_once(“classes/class.automovelRN.php”);
?>
<form name=”form1” method=”post”>
<table width=”100%” border=”0” cellpadding=”0” cellspacing=”0”>
     <tr>
          <td width=”20%” valign=”top”>
               <!-- seleção de marcas e modelos -->
               Marcas:
               <select   name=”marca_cod”   onChange=”document.
form1.submit();”>
                    <option  value=”0”>Selecione  a  Marca</op-
tion>
                    <?
                    $marca_cod=0;
                    if(isset($_POST[‘marca_cod’])){
                         $marca_cod = $_POST[‘marca_cod’];
                    }

                    $lista = MarcaRN::getListaObjeto(“”,””);
```

```
foreach($lista as $oMarca){
                    ?>
                         <option value="<?=$oMarca->getCod()?>"
<?  ($marca_cod==$oMarca->getCod())?print  "selected":print  "";
?> ><?=$oMarca->getDesc()?></option>
                    <?
                    }
                    ?>
               </select>
               <br><br>
               Modelos:
               <select name="mod_cod" onChange="document.form1.
submit();">
                    <option  value="0">Selecione  o  Modelo</op-
tion>
                    <?
                    $condicao = "";
                    if(isset($_POST['marca_cod'])){
                         $condicao  =  "  marca_cod  =  "  .  $_
POST['marca_cod'];
                    }

                    $mod_cod=0;
                    if(isset($_POST['mod_cod'])){
                         $mod_cod = $_POST['mod_cod'];
                    }

                    print $condicao . "<br>";

                    $lista = ModeloRN::getListaObjeto($condicao
,"");
                    foreach($lista as $oModelo){
```

```
     ?>
                         <option       value="<?=$oModelo->get-
Cod()?>"  <?  ($mod_cod==$oModelo->getCod())?print  "selected":
print ""; ?> ><?=$oModelo->getDesc()?></option>
                    <?
                    }
                    ?>
               </select>
          </td>
          <td width="5%"> </td>
          <td>
               <!-- lista de automótveis -->
               <table  width="100%"  align="center"  border="1"
bordercolor="#000000" cellpadding="3" cellspacing="0">
                    <tr bgcolor="#EAEAEA">
                         <td>Placa</td>
                         <td>Ano</td>
                         <td>Preco</td>
                    </tr>
                    <?
                    $condicao = " mod_cod = " . $mod_cod;
                    $lista = AutomovelRN::getListaObjeto($condi
cao,"");
                    foreach($lista as $oAutomovel){
                         ?>
                         <tr>
                         <td><?=$oAutomovel->getPlaca()?></td>
                         <td><?=$oAutomovel->getAno()?></td>
                         <td><?=$oAutomovel->getPreco()?></td>
                         </tr>
                         <?
```

```
}//foreach
                    ?>
               </table>
          </td>
     </tr>
</table>
</form>
</body>
</html>
```

Página: admin/marca_lst.php

```
<?
/* Indica que o arquivo não deve ficar em cache (força o seu re-
processamento) */

require_once("classes/class.marcaRN.php");

?>
     <input name="btnNovo" type="button" value="Nova Marca"
onClick="location.href='marca_frm.php'" /><br><br>
     <table width="100%" align="center" border="1" bordercol-
or="#000000" cellpadding="3" cellspacing="0">
          <tr bgcolor="#EAEAEA">
               <td>Código</td>
               <td>Descrição</td>
               <td> </td>
          </tr>
          <?

          $lista = MarcaRN::getListaObjeto("","");

          foreach($lista as $oMarca){
```

```
?>
                <tr>
                     <td><?=$oMarca->getCod()?></td>
                     <td><?=$oMarca->getDesc()?></td>
                     <td><input name=”btnAlterar” type=”button”
value=”Alterar”              onClick=”location.href=’marca_frm.
php?cod=<?=$oMarca->getCod()?>’” /></td>
                </tr>
                <?
           }//foreach
           ?>
     </table>
<?
//Libera objetos
unset($lista);
?>

Página: admin/marca_frm.php
<?
require_once(“classes/class.marcaRN.php”);

// verifica se é cadastro ou alteração
if (isset($_GET[“cod”])){
   // alteração
     $cod = $_GET[“cod”];
     $oMarca = MarcaRN::getObjeto($cod);
} else {
  // cadastro
     $cod = 0;
     $oMarca = new Marca();
}
```

```
$desc = $oMarca->getDesc();

//Libera objetos
unset($oMarca);

?>
     <form method=”post” name=”frm” action=”marca_action.php”>
          <table width=”95%” align=”center” border=”0” cellpad-
ding=”0” cellspacing=”0”>
            <tr>
               <td  width=”30%”  align=”right”>Código:   </
td>
               <td><input   name=”cod”   type=”text”   size=”10”
maxlength=”5” value=”<?=$cod?>” /></td>
            </tr>
            <tr>
               <td width=”30%” align=”right”>Descrição:  </
td>
               <td><input  name=”desc”  type=”text”  size=”40”
maxlength=”100” value=”<?=$desc?>” /></td>
            </tr>
            <tr>
              <td width=”30%” align=”right”> </td>
              <td> </td>
            </tr>
            <tr>
               <td width=”30%”> </td>
               <td>
          <input name=”btnSalvar” type=”Submit” value=”Salvar”>
          <input        name=”btnCancelar”        type=”button”
value=”Cancelar” onClick=”location.href=’marca_lst.php’”>
          <? if($cod > 0) { ?>
```

```
<input name=”btnApagar” type=”Submit” value=”Apagar” />
                    <? } ?>
                </td>
            </tr>
       </table>
     </form>

Página: admin/marca_action.php
<?
require_once(“classes/class.marcaRN.php”);

//Carrega ou cria objeto
if (isset($_POST[“cod”])){
     $cod = $_POST[“cod”];
     $oMarca = MarcaRN::getObjeto($cod);
}

//variavel booleana que irá verificar se é um novo registro
$ehNovoRegistro = false;

//caso não exista o objeto cria um novo objeto pois é um novo
registro
if(empty($oMarca)){
     $oMarca = new Marca();
     $ehNovoRegistro = true;
}
//Verifica se foi clicado no botão apagar
if  ((isset($_POST[“btnApagar”]))  &&  ($_POST[“btnApagar”]  ==
“Apagar”)){
     if ($cod > 0){
          MarcaRN::delObjeto($oMarca);
```

```
}
}else{
     $oMarca->setCod($_POST[“cod”]);
     $oMarca->setDesc($_POST[“desc”]);

     if ($ehNovoRegistro){
          // adiciona o registro
          MarcaRN::addObjeto($oMarca);
     } else {
          //altera o registro
          MarcaRN::alterObjeto($oMarca);
     }
}
unset($oMarca);

header(“location:marca_lst.php”)
?>
```

Para testar as páginas da aplicação, é necessário incluir alguns registros nas tabelas: Marcas, Modelos e Automóveis no banco de dados. Os registros da tabela: Marcas poderão ser inseridos por meio das telas da administração.

A partir de agora, você poderá dar continuidade ao desenvolvimento da aplicação. Para isso, basta criar as classes Bean e Regra de Negócio para as tabelas que faltam. Além disso, você poderá melhorar a administração da aplicação, incluindo uma tela de autenticação para os funcionários da revenda. Se você tiver dúvidas de como desenvolver esse recurso, retorne ao capítulo 8 do livro-texto e verifique o exemplo que desenvolvemos.

Ainda na administração poderão ser desenvolvidas as telas de manutenção das outras tabelas do banco de dados, isso dará bastante flexibilidade para os mantenedores da aplicação.

Síntese 

Nesta aula estudamos os passos para o desenvolvimento de uma aplicação para Internet, utilizando PHP e banco de dados MySQL.

CONCLUSÃO

Atualmente, é quase impossível que uma empresa não possua uma arquitetura cliente-servidor e sistemas disponíveis para acesso por meio de um navegador de Internet. Esse fato tem se tornado cada vez mais constante, pois manter as aplicações agrupadas em um local torna-se mais rentável para as empresas, que economizarão tempo e recursos com manutenção em aplicações instaladas em diversas máquinas. Isso sem contar os custos com licenças que os sistemas desktop podem acrescentar ao orçamento das empresas.

Também é fato que as aplicações disponíveis na Internet têm evoluído muito e em grande escala. As empresas que fornecem aplicações e softwares diversos possuem cada vez mais demanda para o profissional que está atualizado com este tipo de tecnologia.

Portanto, é imprescindível que os profissionais de informática, tenham conhecimento básico da programação para a Internet. Este conhecimento irá possibilitar ao profissional desenvolver aplicações dinâmicas e interativas e disponibilizá-las na Internet.

Assim, esse módulo teve como objetivo principal introduzir os conhecimentos básicos sobre programação de aplicações para a Internet, utilizando a linguagem PHP.

Respostas dos Exercícios 

Aula 1

1) C

2) D

3) C

4) D

5) A, C

Aula 2

1) C

2) A

3) B

4) A,C

5) B,D

Aula 3

1) C

2) D

3) C

4) C

5) B

Aula 4

1) A,B,C

2) C

3) A

4) C

5) C

Aula 5

1) A,C,D

2) C

3) B

4) A,C

5) C,D

Aula 6

1) A,C,D

2) B

3) B

4) C

5) A,B,C

Aula 7

1) B

2) D

3)

class.operacao.php

```
<?
// declara a classe
// o objeto que for criado por meio desta classe assumirá a visi-
bilidade e nome
// da classe
class Operacao{

                //declara um atributo
                private $resultado = 0;
                // declara o método adição
                // o método recebe dois números para executar o
cálculo e
                // retornar o resultado
```

```
Public function adicao($numero1, $numero2){
                    $this->resultado = $numero1 + $numero2;
                    return $this->resultado;
               }


               // declara o método subtração
               Public function subtracao($numero1, $numero2){
                    $this->resultado = $numero1 - $numero2;
                    return $this->resultado;
               }


               // declara o método multiplicação
               Public function multiplicacao($numero1, $nume-
ro2){
                    $this->resultado = $numero1 * $numero2;
                    return $this->resultado;
               }


               // declara o método divisão
               Public function divisao($numero1, $numero2){
                    // faz o tramento para não permitir divisão
por zero
                    if($numero2>0){
                         $this->resultado = $numero1 / $nume-
ro2;
                    }
                    return $this->resultado;
               }
```

```
}
?>
```

exemplo29.php

```
<HTML>
<HEAD>
 <TITLE>Exemplo 29</TITLE>
</HEAD>
<BODY>
<!--
Este é um comentário em HTML!
Quando queremos que o formulário envie as variáveis para a mesma
página, não especificamos o atributo: action do comando form
-->
<form name=”form1”  method=”post”>
      Número 1: <input type=”text” name=”numero1”><br>
      Número 2: <input type=”text” name=”numero2”><br>
         <input  type=”submit”  name=”btnEnviar”  value=”Calcular
Soma”>
</form>

<?
//verifica se existe a variável de navegador numero1
if(isset($_POST[‘numero1’])){

  //cria novas variáveis com base nas variáveis de navegador
  $num1 = $_POST[‘numero1’];
  $num2 = $_POST[‘numero2’];

     //verifica se as variáveis possuem valor diferente de vazio
       if(!empty($num1) && !empty($num2)){
```

```
      //inclui o arquivo que possui a estrutura da classe para
permitir a criação
      //do objeto
      require_once("class.operacao.php");

      //cria o objeto
      $objOperacao = new Operacao();

      // a partir de agora a variável: $objOperacao é um objeto
que possui a estrutura
      // da classe operação
      // vamos então chamar o método adicao e obter o resultado
da soma e mostrar
      // na tela
      print $objOperacao->adicao($num1,$num2) . "<br>";


      // como já não precisamos do objeto: $objOperacao, vamos
destruí-lo
      unset($objOperacao);
  }
}
?>
</BODY>
</HTML>
```

Aula 8

1) C

2) D

3) D

4) D

5) B

Aula 9

1)

```
<HTML>
<HEAD>
 <TITLE>Exemplo 33</TITLE>
</HEAD>
<BODY>
<?
//conecta com o banco de dados MySQL
$conexao = mysql_connect(“localhost”, “root”, “vertrigo”);

//seleciona o banco de dados revenda
mysql_db_query(“revenda”,$conexao);


//apaga todos os registros da tabela marcas
mysql_query(“DELETE FROM marcas”);


//insere os registros na tabela marcas:
mysql_query(“INSERT INTO marcas VALUES(1,’Citroen’)”);
mysql_query(“INSERT INTO marcas VALUES(2,’Audi’)”);
mysql_query(“INSERT INTO marcas VALUES(3,’Renault’)”);

//seleciona e mostra os registros
$consulta = “SELECT marca_cod, marca_desc FROM marcas”;
$resultado = mysql_query($consulta, $conexao);
while ($linha = mysql_fetch_row($resultado)) {
      print “Código: “ . $linha[0] . “<BR>”;
      print “Descrição: “ . $linha[1] . “<BR><BR>”;
}
```

```
print “Alteração de Registro<br><br>”;

//altera o registro 3 da tabela
mysql_query(“UPDATE marcas SET marca_desc = ‘Pegeout’ WHERE mar-
ca_cod = 3”);

print “Mostra os registros novamente<br><br>”;

//seleciona e mostra os registros
$consulta = “SELECT marca_cod, marca_desc FROM marcas”;
$resultado = mysql_query($consulta, $conexao);
while ($linha = mysql_fetch_row($resultado)) {
      print “Código: “ . $linha[0] . “<BR>”;
      print “Descrição: “ . $linha[1] . “<BR><BR>”;
}

?>
</BODY>
</HTML>
```

REFERÊNCIAS BIBLIOGRÁFICAS 

1. CONVERSE, Tim; PARK, Joyce. **PHP – A Bíblia**. São Paulo: Campus / Elsevier, 2003.

2. WIKIPÉDIA, a enciclopédia livre. **Artigos Eletrônicos**. Disponível em: http://pt.wikipedia.org/

3. Internet World Stats. Disponível em: http://www.internetworldstats.com/

Autor: Rosemary Francisco

Programação de Págia WEB - PHP: Material didático / Rosemary Francisco

Design institucional: Thiago Vedoi de Lima; Cristiane de Oliveira - Joinville: Tupy Virtual, 2007

Ficha catalográfica elaborada pela Biblioteca Universitária Tupy Virtual

Créditos

SOCIESC – Sociedade Educacional de Santa Catarina

Tupy Virtual – Ensino a Distância

Rua Albano Schmidt, 3333 – Joinville – SC – 89206-001
Fone: (47)3461-0166
E-mail: ead@sociesc.org.br
Site: www.sociesc.org.br/portalead

Diretor Geral
Sandro Murilo Santos

Diretor de Administração
Vicente Otávio Martins de Resende

Diretor de Ensino, Pesquisa e Extensão
Roque Antonio Mattei

Diretor do Instituto Superior Tupy
Wesley Masterson Belo de Abreu

Diretor da Escola Técnica Tupy
Luiz Fernando Bublitz

Coordenador da Escola Técnica Tupy
Alexssandro Fossile
Alan Marcos Blenke

Coordenador do Curso
Juliano Prim

Coordenador de Projetos
José Luiz Schmitt

Revisora Pedagógica
Nádia Fátima de Oliveira

EQUIPE TUPY VIRTUAL

Raimundo Nonato Gonçalves Robert
Wilson José Mafra
Thiago Vedoi de Lima
Cristiane Oliveira

Design Gráfico
Thiago Vedoi de Lima

Equipe Didático-Pedagócia
Rosemary Francisco

EDIÇÃO – MATERIAL DIDÁTICO

Professor Conteudista
Rosemary Francisco

Design Institucional
Thiago Vedoi de Lima
Cristiane Oliveira

Ilustração Capa
Thiago Vedoi de Lima

Projeto Gráfico
Equipe Tupy Virtual

Revisão Ortográfica
Nádia Fátima de Oliveira