ESCOLA
Alcides Maya

Linguagem de Programação IV (PHP)

1 ACESSANDO O MYSQL

Neste documento todos os exemplos referentes a acesso de bancos de dados utilizarão o gerenciador de banco de dados MySQL, que pode ser 'baixado' gratuitamente no site http://www.mysql.com.

Para interagir com uma base de dados SQL existem três comando básicos que devem ser utilizados: um que faz a conexão com o servidor de banco de dados, um que seleciona a base de dados a ser utilizada e um terceiro que executa uma 'query' SQL.

1.1 Conexão com o servidor

A conexão com o servidor de banco de dados MySQL em PHP é feita através do comando mysql_connect, que possui a seguinte sintaxe:

```
resource mysql_connect(string host, string login, string senha);
```

Os parâmetros são bastante simples: o endereço do servidor (host), o nome do usuário (login) e a senha para conexão. A função retorna um identificador da conexão estabelecida e deverá ser armazenado em uma variável para utilização posterior. No nosso exemplo, temos como servidor de banco de dados a mesma máquina que roda o servidor http. Como login utilizaremos o usuário 'root' e a senha '12345':

```
$conexao= mysql_connect('localhost', 'root', '12345');
```

Assim, se a conexão for bem sucedida (existir um servidor no endereço especificado que possua o usuário e senha fornecida), o identificador da conexão fica armazenado na variável $conexao.

1.2 Seleção do banco de dados

Uma vez conectado, é preciso selecionar o banco de dados existente no servidor com o qual desejamos trabalhar. Isso é feito através da função mysql_select_db, que possui a seguinte sintaxe:

```
bool mysql_select_db(string nome_base, int conexao);
```

O nome da base de dados a selecionar é o primeiro parâmetro fornecido, seguido pelo identificador da conexão. Se este for omitido, o interpretador PHP tentará utilizar a última conexão estabelecida. Recomenda-se sempre explicitar este valor para facilitar a legibilidade do código e evitar erros nas execuções SQL em caso de múltiplas conexões. No nosso exemplo a base de dados a ser selecionada possui o nome 'ged':

```
mysql_select_db('ged', $conexao);
```

Após a execução desse comando qualquer consulta executada para aquela conexão utilizará a base de dados selecionada.

1.3 Execução de queries SQL

Após estabelecida a conexão e selecionada a base de dados a ser utilizada, quase toda a interação com o servidor MySQL pode ser feita através de consultas escritas em SQL, com o comando mysql_query, que utiliza a seguinte sintaxe:

```
resource mysql_query(string consulta, int conexao);
```

A função retorna FALSE em caso de erro ou resource em caso de sucesso. Sucesso aqui significa que a consulta está sintaticamente correta e foi executada no servidor. Nenhuma informação sobre o resultado é retornada deste comando, ou até mesmo se o resultado é o esperado. No caso da consulta ser um comando SELECT ou outro comando SQL que retorne um resultset, o valor de retorno é um valor resource que identifica o resultado, que poderá ser tratado por uma das funções de acesso a dados, como mysql_fetch_array ou mysql_fetch_object, dentre outras. Para os demais comandos SQL a função retorna TRUE em caso de sucesso ou FALSE em caso de erro. A string de consulta não deve conter ponto-e-vírgula no final do comando e o identificador da conexão é opcional. Vejamos o seguinte exemplo para uma consulta SQL em uma suposta tabela de alunos.

```
$comando= 'SELECT * FROM alunos';
$resultado= mysql_query($comando, $conexao);
```

Os possíveis dados retornados pelo comando SQL são colocados em memória, em local definido pelo banco de dados. Apenas o endereço deste local é retornado e armazenado na variável $resultado, que será utilizada posteriormente para o acesso a estes dados.

Vamos ver agora um exemplo de um suposto cadastro nesta tabela de alunos.

```
$comando= "INSERT INTO alunos (
              nome,
              email
       ) VALUES (
              'Maria',
              'maria@site.com'
       )";
mysql_query($comando, $conexao);
```

Repare que neste exemplo não foi necessário armazenar o retorno de mysql_query pois o comando SQL INSERT não retorna um recordset.

1.4 Pegando resultados de uma consulta SELECT

Para acessar os dados retornados por uma consulta SQL SELECT podemos utilizar diversas funções. Dentre elas veremos dois exemplos: mysql_fetch_array, que é a mais comum para esta tarefa e a mysql_fetch_object, para um estilo de codificação orientado a objetos. As sintaxes de ambas são:

```
array mysql_fetch_array(resource resultado);
```

ou

```
object mysql_fetch_object(resource resultado);
```

Estas duas funções acessam o local de memória onde os dados foram colocados e trazem somente um registro por vez.

Quando não existirem mais registros a serem retornados as funções retornam FALSE. Pode-se utilizá-las em conjunto com laço while para controlar quantas vezes elas devem ser chamadas. Vejamos um exemplo para cada uma delas. Utilizaremos a consulta realizada na suposta tabela de alunos.

Exemplo com mysql_fetch_array:

```
$comando= 'SELECT * FROM alunos';
$resultado= mysql_query($comando, $conexao);
while($dado= mysql_fetch_array($resultado)){
 echo "Nome: {$dado['nome']}, Email: {$dado['email']}<br />";
}
```

Exemplo com mysql_fetch_object:

```
$comando= 'SELECT * FROM alunos';
$resultado= mysql_query($comando, $conexao);
while($dado= mysql_fetch_object($resultado)){
 echo "Nome: {$dado->nome}, Email: {$dado->email}<br />";
}
```

***Dica**: Repare que utilizamos vetores e objetos dentro de strings de aspas duplas, que são interpretadas. Acostume-se a delimitar com chaves toda e qualquer variável que você colocar dentro de strings de aspas duplas. Em alguns casos omitir as chaves causa erro de sintaxe.*

Nos exemplos acima foi executada uma consulta SQL SELECT que retorna um recordset. Os dados que foram colocados em memória pelo banco de dados são identificados pelo valor armazenado na variável $resultado. Como não sabemos quantos registros estão na memória utilizamos um laço while para percorrer a lista de registros. A cada iteração do while um novo registro é retornado para a variável $dado. No código do while escrevemos os dados na tela seguindo a sintaxe de cada função. Na mysql_fetch_array, conforme o nome da função indica, acessamos os dados como um array associativo, onde o índice do vetor é o nome da coluna que se quer acessar. No exemplo com mysql_fetch_object acessamos os dados pela sintaxe orientada a objetos, onde o nome da coluna é chamado como um atributo.

2 ORIENTAÇÃO A OBJETOS

Neste capítulo veremos como o PHP implementa a orientação a objetos, que é essencial no mercado de trabalho nos dias de hoje.

2.1 Classes

A classe é uma estrutura estática utilizada para descrever objetos mediante atributos (propriedades) e métodos (funcionalidades). A classe é um modelo para a criação desses objetos.

Podem ser classes: entidades do negócio da aplicação (pessoa, conta, pedido, etc.), entidades de interface (janela, botão, painel, etc.), dentre outras (conexão a banco de dados, interpretação de arquivos texto, etc.)

No PHP utilizamos a seguinte sintaxe para criação de uma classe:

```
class NomeDaClasse{
   ...
}
```

As classes são orientadas ao assunto, ou seja, cadas classe é responsável por um assunto diferente e possui responsabilidade som o mesmo. Ela deve proteger o acesso ao seu conteúdo por meio de mecanismos como encapsulamento (veremos adiante). Desta forma, criamos sistemas mais confiáveis e robustos.

Recomenda-se que cada classe esteja em um arquivo dedicado, ou seja, um arquivo PHP onde exista somente o código desta classe. Para nos ajudar a carregar estas classes em nossos sistemas ou sites, a comunidade PHP procura padronizar até mesmo os nomes dos arquivos das classes conforme o exemplo a seguir:

class.NomeDaClasse.php

ou

NomeDaClasse.class.php

Repare que com o nome de arquivo padronizado podemos facilmente incluir os arquivos das classes dentro dos arquivos de nosso sistema. Podemos inclusive criar uma função de carregamento automático de classes, que veremos mais adiante.

Os componentes de uma classe devem ser declarados na seguinte ordem: Atributos, declarados com o operador var (que é obsoleto) ou por um modificador de acesso. Mais adiante veremos detalhes sobre estes modificadores de acesso. Em seguida, declaramos os métodos precedidos também por um modificador de acesso e pelo operador function. Vamos a um exemplo de uma classe de aluno, onde supostamente manipularíamos a tabela de alunos do capítulo 1 (Acessando o MySQL)

```
<?php
 class Aluno{
       public $nome;
       public $email;

       public function EnviaMail($assunto, mensagem){
             mail($this->email, $assunto, $mensagem);
       }
 }
?>
```

Neste exemplo criamos uma classe chamada Aluno com dois atributos ($nome e $email) e uma funcionalidade (EnviaMail). Não se preocupe com o modificador de acesso public que está declarado. Veremos ele mais adiante. Repare que no método EnviaMail() tivemos que utilizar um dos atributos da classe, que seria o e-mail do aluno. Dentro de uma classe, para referenciar um de seus membros utilizamos a pseudovariável $this, que representa o próprio objetos que está sendo manipulado.

***Dica**: procure no site www.php.net a sintaxe e opções/restrições para a função mail, que é uma forma simples de enviar e-mails via PHP.*

2.2 Objeto

Um objeto é uma estrutura dinâmica originada com base em uma classe. Após a utilização de uma classe para criar diversas estruturas iguais a ela, que interagem no sistema e possuem dados nela armazenados, dizemos que estamos criando objetos, ou mesmo instanciando objetos de uma classe.

Um objeto existe durante um dado instante de tempo, que vai da sua criação até sua destruição. São objetos da classe Aluno: João, Maria, Pedro, etc... Todos têm uma estrutura igual (a classe Aluno), mas propriedades com valores diferentes, caracterizando cada um de forma distinta.

Para instanciar um objeto de uma determinada classe, devemos declarar uma variável qualquer que será nosso objeto e lhe atribuímos o operador new seguido pelo nome da classe que desejamos instanciar. Veja o exemplo abaixo onde criamos uma instância da classe Aluno:

```
$a= new Aluno();
```

A partir deste momento temos uma instância da classe aluno na variável $a. Podemos agora manipular os atributos e funcionalidades da classe utilizando o objeto. Veja no exemplo:

```
$a->nome= 'João';
$a->email= 'joao@site.com';
$a->EnviaMail('Olá!', 'Bem vindo(a) a nossa escola!');
```

Repare que acessamos os membros da classe Aluno através do objeto instanciado na variável $a. Para isso utilizamos o operado de orientação a objetos ->.

2.3 Construtores e destrutores

Um construtor é um método especial utilizado para definir o comportamento inicial de um objeto, ou seja, o comportamento no momento de sua criação. O método construtor é executado automaticamente no momento em que instanciamos um objeto por meio do operador new. Assim, não devemos retornar nenhum valor por meio do método construtor porque o mesmo retorna por definição o próprio objeto que está sendo instanciado.

Caso não seja definido um método construtor o interpretador do PHP irá criar um para nós em memória e inicializar todos os atributos com null. Normalmente não é o comportamento que queremos, por isso recomenda-se sempre declarar o método construtor e inicializar todos os atributos com seus respectivos valores iniciais conforme o tipo de dado de cada um.

O método construtor deve se chamar obrigatoriamente __construct. Veja no exemplo abaixo como ficará nossa classe Aluno com o método construtor.

```
<?php
 class Aluno{
       public $nome;
       public $email;

       public function __construct(){
             $this->nome= '';
             $this->email= '';
       }

       public function EnviaMail($assunto, mensagem){
             mail($this->email, $assunto, $mensagem);
             }
       }
 ?>
```

Repare que os dois atributos são strings, por isso foram inicializados com uma string vazia. Lembre-se! Sempre inicialize os atributos de uma classe conforme seu tipo de dado. Isso é uma boa prática de programação PHP e você só terá benefício em utilizá-la!

Um destrutor é um método especial executado automaticamente quando um objeto é desalocado da memória, quando atribuímos null ao objeto, quando utilizamos a função unset() sobre ele ou quando o programa é finalizado. O método destrutor pode ser utilizado para finalizar conexões, apagar arquivos temporários criados durante o ciclo de vida do objeto, dentre outras circunstâncias.

O método construtor é opcional e se for declarado deve se chamar obrigatoriamente __destruct. Veja no exemplo abaixo como ficaria nossa classe Aluno utilizando um destrutor.

```
<?php
 class Aluno{
       public $nome;
       public $email;

       public function __construct(){
             $this->nome= '';
             $this->email= '';
       }

       public function EnviaMail($assunto, mensagem){
             mail($this->email, $assunto, $mensagem);
       }

       public function __destruct(){
             echo "O objeto do aluno {$this->nome} foi finalizado...";
       }
 }
?>
```

Repare que o destrutor irá escrever uma frase na tela informando que o objeto foi destruído. Não faça isso em uma classe real.

2.4 Modificadores de acesso

Os modificadores de acesso determinam como os membros de uma classe serão acessíveis quando estiverem instanciados em objetos. O PHP implementa três modificadores: public, protected e private.

Modificador public: Define que a acessibilidade do membro será total, ou seja, qualquer um, de qualquer lugar, pode ter acesso ao membro. Por exemplo: Na classe Aluno o atributo $nome foi declarado como público, sendo assim, ele é acessível de dentro da própria classe, de classes derivadas (veremos isso mais adiante) e de dentro do programa. Todo mundo consegue manipular o atributo (e isso não é uma boa ideia!).

Modificador protected: Define que a acessibilidade do membro será parcial. Ele será acessível somente de dentro da própria classe ou de classes derivadas.

Modificador private: Define que a acessibilidade do membro será restrita. Ele será acessível somente de dentro da própria classe. Por exemplo: Se modificarmos a declaração do atributo $nome da classe Aluno para private, ele somente será

acessível de dentro da própria classe, ficando o programa ou classes derivadas proibidos de ler ou modificar este atributo (e isso realmente é uma boa ideia!).

Os modificadores de acesso são parte fundamental de um conceito importantíssimo da orientação a objetos. O encapsulamento.

2.5 Encapsulamento

Um dos recursos mais interessantes na orientação a objetos é o encapsulamento, um mecanismo que provê proteção de acesso aos membros internos de um objeto. Lembre-se que cada classe possui responsabilidade sobre os atributos que contém. Dessa forma, os atributos de uma classe devem ser tratados exclusivamente por métodos dela mesma, que são implementações projetadas para manipular estes atributos de forma correta. Os atributos nunca devem ser acessados diretamente de fora do escopo de uma classe, pois dessa forma a classe não fornece mais garantias sobre os valores que contém, perdendo assim a responsabilidade sobre eles.

Dois passos são necessários para atingirmos o encapsulamento.

1) Defina a acessibilidade dos atributos como private. Assim somente a própria classe pode manipulá-los.

2) Implemente métodos para manipulação destes atributos, garantindo que os dados passados a eles sejam tratados e garantidos quanto a seu escopo antes de serem armazenados nos atributos.

Chamamos o passo dois acima de ‘implementação dos gets e sets’.

Os métodos ‘gets’ tem a única e exclusiva funcionalidade de pegar o valor atual de um determinado atributo. Já os métodos ‘sets’ tem a única e exclusiva funcionalidade de atribuir um valor a um determinado atributo, mas antes deve validar esse valor da forma mais consistente possível.

***Observação**: Nesta apostila não faremos validações de dados nos métodos ‘sets’ pois fogem do escopo da disciplina.*

Vamos agora modificar a classe Aluno para prover o encapsulamento. Lembre-se dos dois passos necessários e identifique cada um deles no código abaixo:

```
<?php
    class Aluno{
        private $nome;
        private $email;

        public function __construct(){
            $this->nome= '';
            $this->email= '';
        }

        public function GetNome(){
            return $this->nome;
        }

        public function SetNome($valor){
            $this->nome= $valor;
        }

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

        public function SetEmail($valor){
            $this->email= $valor;
        }

        public function EnviaMail($assunto, mensagem){
            mail($this->email, $assunto, $mensagem);
        }

        public function __destruct(){
            echo "O objeto do aluno {$this->nome} foi finalizado...";
        }
    }
?>
```

Repare que os dois atributos da classe foram definidos com o modificador de acesso private e que foram criados métodos para pegar e atribuir valores a estes atributos. Não é sempre necessário criar os dois métodos para cada atributo. Você deve decidir se determinado atributo precisa ser pego e modificado pelo programa e assim criar os gets e sets conforme necessário.

Repare também nos dois métodos sets (SetNome e SetEmail). Os dados passados pelo parâmetro $valor de ambos os métodos deveriam ter sido validados antes de serem atribuídos. Poderíamos ter verificado se o $valor é uma string, qual o seu tamanho, se o e-mail está no formato correto, etc.

***Dica**: No site www.php.net você encontra funções que testam o tipo de dados de uma variável, assim como tamanho de string e formas de detectar um padrão em strings, para verificar formato correto de e-mail.*

Vamos ver agora como um programa que utilizaria essa nova classe Aluno

```
<?php
 require('class.Aluno.php');
 $a= new Aluno();
 $a->SetNome('João');
 $a->SetEmail('joao@site.com');
 $a->EnviaEmail('Olá!', 'Bem vindo a nossa escola!');

 echo "Foi enviado um e-mail para o aluno: {$a->GetNome()}";
?>
```

2.6 Herança

A utilização da orientação a objetos e do encapsulamento do código em classes nos orientam em direção a uma maior organização, mas um dos maiores benefícios que encontramos na utilização desse paradigma é o reuso. A possibilidade de reutilizar partes de código já definidas é o que nos dá maior agilidade no dia a dia, além de eliminar a necessidade de eventuais duplicações ou reescritas de código.

Quando falamos em herança, a primeira imagem que nos aparece na memória é a de uma árvore genealógica com avós, pais, filhos e nas características que são transmitidas geração após geração. O que devemos levar em consideração sobre herança em orientação a objetos é o compartilhamento de atributos e comportamentos entre as classes de uma mesma hierarquia (árvore). As classes inferiores da hierarquia automaticamente herdam todas os atributos e métodos das classes superiores (desde que os modificadores de acesso destes atributos ou métodos estejam definidos como protected ou public).

Este recurso tem uma aplicabilidade muito grande, visto que é relativamente comum termos de criar novas funcionalidades em software. Utilizando a herança, em vez de criarmos uma estrutura totalmente nova (uma classe), podemos reaproveitar uma estrutura já existente que nos forneça uma base abstrata para o desenvolvimento , provendo recursos básicos e comuns.

Vamos imaginar nosso exemplo da classe Aluno. Um aluno é uma pessoa, e em um sistema real teríamos vários tipos de pessoa (aluno, professor, funcionário, etc.). Todas estas pessoas possuem nome, e-mail, cpf, endereço, etc. Não é necessário colocar todos estes atributos novamente em cada classe que representa estas pessoas. Imagine reescrever todos os atributos, gets, sets e métodos para cada tipo de pessoa. Imagine também se o padrão de CPF, que toda pessoa possui, muda. Teremos que recodificar o algoritmo que valida o CPF em todas as classes que manipulam pessoa.

Uma forma mais inteligente de resolver esse problema seria criarmos uma classe pessoa, que teria a tarefa de manipular todos os dados que são comuns a todas as pessoas (nome, e-mail, cpf, endereço, etc.), deixando os dados específicos para cada classe especializada manipular. Vamos modificar nosso exemplo adicionando uma classe de pessoa e fazendo com que a classe Aluno herde essa nova classe.

```
<?php
    class Pessoa{
          private $nome;
          private $email;

          public function __construct(){
                $this->nome= ‘’;
                $this->email= ‘’;
          }

          public function GetNome(){
                return $this->nome;
          }

          public function SetNome($valor){
                $this->nome= $valor;
          }

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

          public function SetEmail($valor){
                $this->email= $valor;
          }

    }
   ?>
```

Agora devemos reescrever a classe aluno:

```
<?php
 class Aluno extends Pessoa{
        private $matricula;

        public function __construct(){
                parent::__construct();
                $this->matricula= 0;
        }

        public function GetMatricula(){
                return $this->matricula;
        }

        public function SetMatricula($valor){
                $this->matricula= $valor;
        }

        public function EnviaMail($assunto, mensagem){
                mail($this->email, $assunto, $mensagem);
        }

        public function __destruct(){
                echo "O objeto do aluno {$this->nome} foi finalizado...";
        }
 }
?>
```

Vamos entender o que aconteceu.

Criamos uma classe chamada pessoa, onde assumimos que todos os tipos de pessoas em nosso sistema possuem em comum o nome e e-mail. Criamos também o método construtor e os métodos gets e sets para os atributos. Adicionaríamos também nesta classe qualquer funcionalidade genérica de uma pessoa.

Na classe aluno, o que chama a atenção primeiramente é o nome da classe, que está sendo seguido da palavra extends e do nome da classe Pessoa. O comando extends é o que indica que a classe Aluno está herdando a classe Pessoa. Tudo que for protected ou public na classe Pessoa agora existe também na classe Aluno! Para exemplificar, adicionamos também um novo atributo específico dos alunos, a matrícula. Repare também no construtor da classe Aluno. Um comando parent::__construct() foi adicionado. Vejamos porque: Você se lembra quando um construtor é executado? Bem, o construtor de uma classe é executado no momento em que instanciarmos um objeto dessa classe (usando o perador new). Como vamos instanciar Aluno no nosso sistema o construtor da classe Pessoa não será executado. Para evitar que os atributos da classe Pessoa sejam manipulados sem inicialização devemos garantir a execução do construtor da classe Pessoa chamando diretamente no construtor da classe Aluno. Um construtor de uma classe filha sempre deve chamar o construtor de sua classe pai.

Vamos ver um exemplo da utilização da classe aluno em um programa.

```
<?php
    require('class.Pessoa.php');
    require('class.Aluno.php');
    $a= new Aluno();
    $a->SetNome('João');
    $a->SetEmail('joao@site.com');
    $a->SetMatricula(12345);
    $a->EnviaEmail('Olá!', 'Bem vindo a nossa escola!');
    echo "Foi enviado um e-mail para o aluno: {$a->GetNome()}";
?>
```

Repare que carregamos o arquivo que contém a classe Pessoa, para que essa classe exista quando a herança com a classe Aluno for feita. A utilização no programa é transparente. Repare na utilização dos métodos SetNome e SetEmail. Eles foram declarados na classe Pessoa, mas nós instanciamos a classe Aluno, que não possui estes métodos declarados e sim os possui via herança.

Poderíamos criar também as classes Professor e Funcionario herdando de Pessoa.

2.7 Polimorfismo

O significado de polimorfismo nos remete a 'muitas formas'. Polimorfismo em orientação a objetos é o princípio que permite que classes derivadas de uma mesma superclasse tenham métodos iguais (com a mesma nomenclatura e parâmetros), mas comportamentos diferentes, redefinidos em cada uma das classes filhas. Poderíamos reescrever nosso exemplo colocando o método EnviaMail na classe Pessoa. Manteremos o método na classe cliente mas com uma funcionalidade diferente, que é enviar a mensagem com um rodapé diferenciado. Vamos ver como ficará. Tente identificar as modificações e o polimorfismo.

```
<?php
    class Pessoa{
          private $nome;
          private $email;

          public function __construct(){
                $this->nome= '';
                $this->email= '';
          }
          public function GetNome(){
                return $this->nome;
          }
          public function SetNome($valor){
                $this->nome= $valor;
          }
          public function GetEmail(){
                return $this->email;
          }
          public function SetEmail($valor){
                $this->email= $valor;
          }
          public function EnviaMail($assunto, mensagem){
                mail($this->email, $assunto, $mensagem);
          }

    }
?>
```

Classe aluno utilizando polimorfismo no método EnviaMail:

```
<?php
  class Aluno extends Pessoa{
          private $matricula;

          public function __construct(){
                parent::__construct();
                $this->matricula= 0;
          }
          public function GetMatricula(){
                return $this->matricula;
          }
          public function SetMatricula($valor){
                $this->matricula= $valor;
          }
          public function EnviaMail($assunto, mensagem){
                $msg= "{$mensagem}\n

                      Atenciosamente,
                            Escola Alcides Maya
                            www.alcidesmaya.com.br";
```

```
                mail($this->email, $assunto, $msg);
        }
        public function __destruct(){
                echo "O objeto do aluno {$this->nome} foi finalizado...";
        }
    }
?>
```

Quando instanciarmos uma classe de Professor ou Funcionario, que também seriam derivadas de Pessoa utilizaremos o EnviaMail da classe Pessoa. Mas quando instanciamos a classe Aluno e enviamos um e-mail para ele utilizaremos a EnviaMail da classe Aluno, ou seja, um método já definido na classe pai foi reescrito (mantendo o mesmo nome e parâmetros) com uma funcionalidade diferente.

3 EXEMPLOS

3.1 Exemplo de uma classe de conexão ao mysql

```
<?php
  class DB{
    private $conn;
    private $result;

    /**
     * REABRE a conexão ao banco de dados
     *
     */
    public function __construct(){
      if(!$this->conn= mysql_connect('localhost','root','12345')){
        throw new Exception('Erro ao conectar a base de dados');
      }
      if(!mysql_select_db('nome_da_base_dados',$this->conn)){
        throw new Exception('Erro ao selecionar a base de dados para uso');
      }
    }

    /**
     * Retorna esta conexão ao banco de dados
     *
     * @return resource
     */
    public function Conn(){
      return $this->conn;
    }

    /**
     * Fecha a conexão ao mysql
```

```
     *
     * @return bool
     */
    public function Close(){
        return mysql_close($this->conn);
    }

    /**
     * Inicia uma transação de banco de dados
     *
     */
    public function StartTransaction(){
        if(!mysql_query('START TRANSACTION',$this->Conn())){
            throw new Exception('Erro ao iniciar a transação');
        }
    }

    /**
     * Executa o ROLLBACK na transação atual
     *
     */
    public function Rollback(){
        if(!mysql_query('ROLLBACK',$this->Conn())){
            throw new Exception('Erro ao executar o cancelamento da transação');
        }
    }

    /**
     * Executa o COMMIT da transação atual
     *
     */
    public function Commit(){
        if(!mysql_query('COMMIT',$this->Conn())){
            throw new Exception('Erro ao salvar os dados da transação');
        }
    }

    /**
     * Executa um comando SQL no banco de dados
     *
     * @param string $query
     * @return resource
     */
    public function Sql($query){
        if(!$this->result = mysql_query($query,$this->Conn())){
            return false;
        }else{
            return $this->result;
        }
```

```
    }

    /**
     * Retorna uma linha de resultado por chamada
     *
     * @return object
     */
    public function Fetch(){
       return mysql_fetch_object($this->result);
    }

        /**
         * Retorna o número de linhas retornadas pela consulta SQL ou número de
linhas modificadas por comandos como UPDATE e DELETE
         *
         * @return int
         */
        public function NumRows(){
            return mysql_num_rows($this->result);
        }

        /**
         * Retorna a mensagem de erro do mysql
         *
         * @return string
         */
        public function Error(){
            return mysql_error($this->Conn());
        }

        /**
         * Retorna o último autoincrement gerado
         *
         * @return int
         */
        public function LastInsertId(){
            return mysql_insert_id($this->conn);
        }
    }
?>
```

Abaixo segue um exemplo de uma classe que manipula uma tabela de noticias:

```
<?php
  /**
   * Classe para manipulação de noticias
   *
   */
  class Noticia{
```

```
    private $codigo;
    /**
     * @var DataHora
     */
    private $datahora;
    private $titulo;
    private $texto;

    /**
     * Construtor da classe
     * Se id for informado carrega a noticia na instancia atual
     *
     * @param int $id
     */
    public function __construct($id= null){
       $this->codigo= 0;
       $this->datahora= null;
       $this->titulo= '';
       $this->texto= '';

       if($id !== null){
          $id= (int) $id;
          if($id <= 0){
             throw new Exception('ID de notícia inválida');
          }

          $query= "SELECT * FROM noticias WHERE id = {$id}";
          $db= new DB();
          $db->Sql($query);
          if($db->NumRows() == 0){
             throw new Exception('Notícia não encontrada');
          }
          $dado= $db->Fetch();
          $this->codigo= $dado->id;
          $this->datahora= new DataHora($dado->datahora);
          $this->titulo= $dado->titulo;
          $this->texto= $dado->texto;
       }
    }

    /**
     * Retorna o codigo
     *
     * @return int
     */
    public function getCodigo(){
       return $this->codigo;
    }
```

```
/**
 * Seta a data e hora
 *
 * @param DataHora $value
 */
public function setDataHora(DataHora $value){
   $this->datahora= $value;
}

/**
 * Retorna a data e hora
 *
 * @return DataHora
 */
public function getDataHora(){
   return $this->datahora;
}

/**
 * Seta o titulo
 *
 * @param string $value
 */
public function setTitulo($value){
   $value= Util::LimpaStringCompleta($value);
   $caracteres= strlen($value);
   if($caracteres < 4 && $caracteres > 50){
      throw new Exception('O título deve conter de 4 a 50 caracteres');
   }
   $this->titulo= $value;
}

/**
 * Retorna o titulo
 *
 * @return string
 */
public function getTitulo(){
   return Util::LimpaStringEscrita($this->titulo);
}

/**
 * Seta o texto
 *
 * @param string $value
 */
public function setTexto($value){
   //nao retira html/php/javascript
   $this->texto= $value;
```

```
    }

    /**
     * Retorna o texto
     *
     * @return string
     */
    public function getTexto(){
       return Util::LimpaStringEscrita($this->texto);
    }

    /**
     * Cadastra a noticia
     *
     */
    public function Cadastra(){
       $query= "INSERT INTO noticias (
                datahora,
                titulo,
                texto
               )VALUES(
                '{$this->datahora->DataHoraISO()}',
                '{$this->titulo}',
                '{$this->texto}'
               )";
       $db= new DB();
       if(!$db->Sql($query)){
          throw new Exception('Falha ao cadastrar a notícia');
       }
       $this->codigo= $db->LastInsertId();
    }

    /**
     * Edita a noticia
     *
     */
    public function Edita(){
       $query= "UPDATE noticias SET
                datahora= '{$this->datahora->DataHoraISO()}',
                titulo= '{$this->titulo}',
                texto= '{$this->texto}'
               WHERE id = {$this->codigo}";
       $db= new DB();
       if(!$db->Sql($query)){
          throw new Exception('Falha ao editar a notícia');
       }
    }

            /**
```

```
         * Remove a noticia
         *
         */
        public function Remove(){
            $query= "DELETE FROM noticias WHERE id = {$this->codigo} LIMIT 1";
            $db= new DB();
            if(!$db->Sql($query)){
                throw new Exception('Falha ao remover a notícia');
            }
        }
    }
?>
```

Abaixo um exemplo de uma classe simples para manipulação de datas. Esta classe é limitada pelo mktime() . Consulte o site www.php.net para ver as limitações do mktime().

```
<?php
  class DataHora{
    private $mktime;//mktime

    /**
     * Manipula data e hora.
     * Se o parâmetro foi informado, utilizará como data e hora
     * O formato deve ser ANO-MES-DIA HORA:MINUTO:SEGUNDO
     *
     * Ex.:
     * 2000-03-15 21:34:12 - data e hora
     * 2000-03-15 00:00:00 - somente data
     *
     * @param string|null $dh
     */
    public function __construct($dStart= null){
      if($dStart === null){
        $this->mktime= mktime();
      }else{
        if(!preg_match("/^([0-9]{4})-([0-9]{2})-([0-9]{2}) ([0-9]{1,2}):([0-9]{1,2}):([0-9]{1,2})$/", $dStart)){
          throw new Exception("Data em formato inválido: {$dStart}");
        }
        $this->mktime= $this->ConverteMktime($dStart);
      }
    }

    /**
     * Converte em mktime uma data/hora no formato ANO-MES-DIA HORA:MINUTO:SEGUNDO
     *
     * @param string $dh
     * @return int
     */
    protected function ConverteMktime($dStart){
```

```
            //separa a data da hora
            $dh= explode(' ',$dStart);
            $data= explode('-',$dh[0]);
            $hora= explode(':',$dh[1]);

            //verifica se a data é valida se for diferente de 0000-00-00
            if($dh[0] != '0000-00-00'){
                if(!checkdate($data[1],$data[2],$data[0])){
                    throw new Exception("Data invÃ¡lida: {$dh[0]}");
                }
            }else{
                return 0;
            }

            if(($hora[0] < 0 || $hora[0] > 23) || ($hora[1] < 0 || $hora[1] > 59) || ($hora[2] < 0 || $hora[2] > 59)){
                throw new Exception("Hora inválida: {$dh[1]}");
            }
            return mktime($hora[0],$hora[1],$hora[2],$data[1],$data[2],$data[0]);
        }

        /**
         * Data no formato DIA/MES/ANO
         *
         * @return string
         */
        public function DataPortugues(){
            return $this->mktime != 0 ? date('d/m/Y',$this->mktime) : '00/00/0000';
        }

        /**
         * Data no formato ANO-MES-DIA
         *
         * @return string
         */
        public function DataISO(){
            return $this->mktime != 0 ? date('Y-m-d',$this->mktime) : '0000-00-00';
        }

        /**
         * Data e hora no formato DIA/MES/ANO HORA:MINUTO:SEGUNDO
         *
         * @param bool $segundos
         * @return string
         */
        public function DataHoraPortugues($segundos= true){
            return $segundos ? $this->mktime != 0 ? date('d/m/Y H:i:s',$this->mktime) : '00/00/0000 00:00:00' : "{$this->DataPortugues()} {$this->HoraMinuto()}";
        }
```

```
/**
 * Data e hora no formato ANO-MES-DIA HORA:MINUTO:SEGUNDO
 *
 * @return string
 */
public function DataHoraISO(){
   return $this->mktime != 0 ? date('Y-m-d H:i:s',$this->mktime) : '0000-00-00 00:00:00';
}

/**
 * hora completa no formato HORA:MINUTO:SEGUNDO
 *
 * @return string
 */
public function HoraCompleta(){
   return $this->mktime != 0 ? date('H:i:s',$this->mktime) : '00:00:00';
}

/**
 * hora e minutos no formato HORA:MINUTO
 *
 * @return string
 */
public function HoraMinuto(){
   return $this->mktime != 0 ? date('H:i',$this->mktime) : '00:00';
}

/**
 * Retorna o dia do mes de 01 a 31
 *
 * @return string|int
 */
public function Dia(){
   return $this->mktime != 0 ? date('d',$this->mktime) : '00';
}

/**
 * Retorna o mes em numero 01 a 12
 *
 * @return string|int
 */
public function Mes(){
   return $this->mktime != 0 ? date('m',$this->mktime) : '00';
}

/**
 * Retorna o ano em 4 digitos
 *
```

```
     * @return string|int
     */
    public function Ano4(){
        return $this->mktime != 0 ? date('Y',$this->mktime) : '0000';
    }

    /**
     * Retorna o ano em 2 digitos
     *
     * @return string|int
     */
    public function Ano2(){
        return $this->mktime != 0 ? date('y',$this->mktime) : '00';
    }

    /**
     * Hora no formato 24 horas. De 00 a 23
     *
     * @return string|int
     */
    public function Hora24(){
        return $this->mktime != 0 ? date('H',$this->mktime) : '00';
    }

    /**
     * Hora no formato 12 horas. De 01 a 12
     *
     * @return string|int
     */
    public function Hora12(){
        return $this->mktime != 0 ? date('h',$this->mktime) : '00';
    }

    /**
     * Minuto da hora. De 00 a 59
     *
     * @return string|int
     */
    public function Minuto(){
        return $this->mktime != 0 ? date('i',$this->mktime) : '00';
    }

    /**
     * Segundos da hora. De 00 a 59
     *
     * @return string|int
     */
    public function Segundo(){
        return $this->mktime != 0 ? date('s',$this->mktime) : '00';
```

```
}

/**
 * Antes ou depois do meio dia. AM/PM
 *
 * @return string
 */
public function AmPm(){
   return date('A',$this->mktime);
}

/**
 * Numero de dias do mes. De 28 a 31
 *
 * @return int
 */
public function TotalDiasMes(){
   return $this->mktime != 0 ? date('t',$this->mktime) : 0;
}

/**
 * Numero da semana do ano. Semana começando na segunda
 *
 * @return int
 */
public function SemanaAno(){
   return $this->mktime != 0 ? date('W',$this->mktime) : 0;
}

/**
 * Dia da semana em número. 0 domingo 6 sabado
 *
 * @return int
 */
public function DiaSemana(){
   return $this->mktime != 0 ? date('w',$this->mktime) : -1;
}

/**
 * Dia do ano. de 1 a 366
 *
 * @return int
 */
public function DiaAno(){
   return $this->mktime != 0 ? date('z',$this->mktime) + 1 : 0;
}

/**
 * timezone da data. UTC/GMT/ETC...
```

```
     *
     * @return string
     */
    public function Timezone(){
       return date('e',$this->mktime);
    }

    /**
     * Se está ou não no horario de verão
     *
     * @return bool
     */
    public function HorarioVerao(){
       return date('I',$this->mktime) == 1 ? true : false;
    }

    /**
     * Se esta ou nao em um ano bissexto
     *
     * @return bool
     */
    public function AnoBissexto(){
       return date('L',$this->mktime) == 1 ? true : false;
    }

    /**
     * Retorna o mktime() da data
     *
     * @return int
     */
    public function MkTime(){
       return $this->mktime;
    }

    /**
     * Dia da semana em português. Domingo a sábado
     *
     * @return string
     */
    public function DiaSemanaPortugues(){
       switch($this->DiaSemana()){
          case 0:return "Domingo";       break;
          case 1:return "Segunda-feira"; break;
          case 2:return "Terça-feira";   break;
          case 3:return "Quarta-feira";  break;
          case 4:return "Quinta-feira";  break;
          case 5:return "Sexta-feira";   break;
          case 6:return "Sábado";        break;
       }
```

```
    }

    /**
     * Mes em português. Janeiro a dezembro
     *
     * @return string
     */
    public function MesPortugues(){
       switch($this->Mes()){
          case 1: return “Janeiro”;   break;
          case 2: return “Fevereiro”; break;
          case 3: return “Março”;     break;
          case 4: return “Abril”;     break;
          case 5: return “Maio”;      break;
          case 6: return “Junho”;     break;
          case 7: return “Julho”;     break;
          case 8: return “Agosto”;    break;
          case 9: return “Setembro”;  break;
          case 10:return “Outubro”;   break;
          case 11:return “Novembro”;  break;
          case 12:return “Dezembro”;  break;
       }
    }

    /**
     * Soma x dias a data atual
     *
     * @param int $dias
     */
    public function SomaDia($dias){
       if($dias < 0){
          throw new Exception(‘Número de dias deve ser maior ou igual a zero’);
       }
        $this->mktime= mktime($this->Hora24(),$this->Minuto(),$this->Segundo(),$this->Mes(),$this->Dia() + $dias,
$this->Ano4());
    }

    /**
     * Soma x meses a data atual
     *
     * @param int $meses
     */
    public function SomaMes($meses){
       if($meses < 0){
          throw new Exception(‘Número de meses deve ser maior ou igual a zero’);
       }
      $this->mktime= mktime($this->Hora24(),$this->Minuto(),$this->Segundo(),$this->Mes() + $meses,$this->Dia(),
$this->Ano4());
    }
```

```
        /**
         * Soma x anos a data atual
         *
         * @param int $anos
         */
        public function SomaAno($anos){
           if($anos < 0){
              throw new Exception('Número de anos deve ser maior ou igual a zero');
           }
             $this->mktime= mktime($this->Hora24(),$this->Minuto(),$this->Segundo(),$this->Mes(),$this->Dia(), $this->Ano4() + $anos);
        }

        /**
         * Soma x horas a data atual
         *
         * @param int $horas
         */
        public function SomaHora($horas){
           if($horas < 0){
              throw new Exception('Número de horas deve ser maior ou igual a zero');
           }
            $this->mktime= mktime($this->Hora24() + $horas,$this->Minuto(),$this->Segundo(),$this->Mes(),$this->Dia(), $this->Ano4());
        }

        /**
         * Soma x minutos a data atual
         *
         * @param int $min
         */
        public function SomaMinuto($min){
           if($min < 0){
              throw new Exception('Número de minutos deve ser maior ou igual a zero');
           }
             $this->mktime= mktime($this->Hora24(),$this->Minuto() + $min,$this->Segundo(),$this->Mes(),$this->Dia(), $this->Ano4());
        }

        /**
         * Soma x segundos a data atual
         *
         * @param int $seg
         */
        public function SomaSegundo($seg){
           if($seg < 0){
              throw new Exception('Número de segundos deve ser maior ou igual a zero');
           }
```

```
            $this->mktime= mktime($this->Hora24(),$this->Minuto(),$this->Segundo() + $seg,$this->Mes(),$this->Dia(),
$this->Ano4());
        }

        /**
         * Subtrai x dias a data atual
         *
         * @param int $dias
         */
        public function SubtraiDia($dias){
            if($dias < 0){
                throw new Exception('Número de dias deve ser maior ou igual a zero');
            }
             $this->mktime= mktime($this->Hora24(),$this->Minuto(),$this->Segundo(),$this->Mes(),$this->Dia() - $dias,
$this->Ano4());
        }

        /**
         * Subtrai x meses a data atual
         *
         * @param int $meses
         */
        public function SubtraiMes($meses){
            if($meses < 0){
                throw new Exception('Número de meses deve ser maior ou igual a zero');
            }
           $this->mktime= mktime($this->Hora24(),$this->Minuto(),$this->Segundo(),$this->Mes() - $meses,$this->Dia(),
$this->Ano4());
        }

        /**
         * Subtrai x anos a data atual
         *
         * @param int $anos
         */
        public function SubtraiAno($anos){
            if($anos < 0){
                throw new Exception('Número de anos deve ser maior ou igual a zero');
            }
              $this->mktime= mktime($this->Hora24(),$this->Minuto(),$this->Segundo(),$this->Mes(),$this->Dia(), $this-
>Ano4() - $anos);
        }

        /**
         * Subtrai x horas a data atual
         *
         * @param int $horas
         */
        public function SubtraiHora($horas){
```

```
            if($horas < 0){
                throw new Exception('Número de horas deve ser maior ou igual a zero');
            }
            $this->mktime= mktime($this->Hora24() - $horas,$this->Minuto(),$this->Segundo(),$this->Mes(),$this->Dia(),
$this->Ano4());
        }

        /**
         * Subtrai x minutos a data atual
         *
         * @param int $min
         */
        public function SubtraiMinuto($min){
            if($min < 0){
                throw new Exception('Número de minutos deve ser maior ou igual a zero');
            }
             $this->mktime= mktime($this->Hora24(),$this->Minuto() - $min,$this->Segundo(),$this->Mes(),$this->Dia(),
$this->Ano4());
        }

        /**
         * Subtrai x segundos a data atual
         *
         * @param int $seg
         */
        public function SubtraiSegundo($seg){
            if($seg < 0){
                throw new Exception('Número de segundos deve ser maior ou igual a zero');
            }
              $this->mktime= mktime($this->Hora24(),$this->Minuto(),$this->Segundo() - $seg,$this->Mes(),$this->Dia(),
$this->Ano4());
        }

        /**
         * Retorna a semana por extenso
         *
         * @return string
         */
        public function SemanaExtenso(){
            switch(date('w',$this->mktime)){
                case 0: return 'Domingo';break;
                case 1: return 'Segunda-feira';break;
                case 2: return 'TerÃ§a-feira';break;
                case 3: return 'Quarta-feira';break;
                case 4: return 'Quinta-feira';break;
                case 5: return 'Sexta-feira';break;
                case 6: return 'SÃ¡bado';break;
            }
        }
```

```
        /**
         * Monta uma frase com as informações de data e hora.
         * SEMANA, DIA de MES de ANO. [HORA:MINUTO[:SEGUNDO]]
         *
         * @param bool $hora Se deve mostrar a hora
         * @param bool $segundo Se deve mostrar os segundos da hora (caso mostre a hora)
         * @return string
         */
        public function FraseDataCompleta($hora= false,$segundo= false){
            return "{$this->SemanaExtenso()}, {$this->Dia()} de {$this->MesPortugues()} de {$this->Ano4()} ".($hora ? $segundo ? "- {$this->HoraCompleta()}" : "- {$this->HoraMinuto()}" : null);

        }

        /**
         * Compara duas datas. Utiliza o mktime das duas datas para comparação
         * Retorna:
         * -1 se a data form menor
         * 0 se for igual
         * 1 se for maior
         *
         * @param DataHora $data
         * @return int
         */
        public function Compara(DataHora $data){
           switch(true){
              case ($this->MkTime() < $data->MkTime()) : return -1;break;
              case ($this->MkTime() == $data->MkTime()): return 0; break;
              case ($this->MkTime() > $data->MkTime()) : return 1; break;
           }
        }

        /**
         * Verifica se as duas datas estão no mesmo mês e ano
         *
         * @param DataHora $data
         * @return bool
         */
        public function ComparaMesmoMesAno(DataHora $data){
           return ($this->Mes() == $data->Mes()) && ($this->Ano4() == $data->Ano4()) ? true : false;
        }


        /**
         * ## FUNCOES ESTATICAS
         */

        /**
```

```
        * Retorna a data e hora atual no formato ANO-MES-DIA HORA:MINUTO:SEGUNDO
        *
        * @return string
        */
            public static function Now(){
                return date('Y-m-d H:i:s');
            }

            /**
             * Retorna o mktime() atual
             *
             * @return int
             */
            public static function NowMkTime(){
                return mktime();
            }
        }
    ?>
```