O PDO é uma biblioteca do PHP escrita em C, orientada a objetos e apresenta um bom ganho de performance em relação as bibliotecas mysql_ e mysqli.

Singleton.

Esse padrão tem como objetivo garantir que exista apenas uma instância de uma determinada classe e também definir um ponto de acesso global a essa instância. Basicamente quem gerencia o objeto é ele mesmo, ou seja, sempre que for necessário instanciar uma novo objeto de uma determinada classe quem irá definir a necessidade ou não de se instanciar esse objeto é a própria classe através de um método estático *getInstance*(). Outro ponto importante é que essa classe deve possuir seu método construtor como privado, dessa maneira a mesma não poderá ser instanciada diretamente por outras classes, pois seu método *getInstance*() é quem verifica se já existe um objeto dessa classe na memória e caso exista ele retorna esse mesmo objeto.

Conexao.php

```
1   <?php
2   /*
3    * Constantes de parâmetros para configuração da conexão
4    */
5   define('HOST', 'localhost');
6   define('DBNAME', 'DB');
7   define('CHARSET', 'utf8');
8   define('USER', 'root');
9   define('PASSWORD', '123456');
10
11  class Conexao {
12
13   /*
14    * Atributo estático para instância do PDO
15    */
16   private static $pdo;
17
18   /*
19    * Escondendo o construtor da classe
20    */
21   private function __construct() {
22     //
23   }
24
25   /*
26    * Método estático para retornar uma conexão válida
```

```
27     * Verifica se já existe uma instância da conexão, caso não, configura uma
28 nova conexão
29     */
30    public static function getInstance() {
31      if (!isset(self::$pdo)) {
32        try {
33              $opcoes = array(PDO::MYSQL_ATTR_INIT_COMMAND => 'SET
34 NAMES UTF8', PDO::ATTR_PERSISTENT => TRUE);
35          self::$pdo = new PDO("mysql:host=" . HOST . "; dbname=" . DBNAME
36 . "; charset=" . CHARSET . ";", USER, PASSWORD, $opcoes);
37        } catch (PDOException $e) {
38          print "Erro: " . $e->getMessage();
39        }
40      }
        return self::$pdo;
      }
    }
```

Detalhes da classe de conexão com PDO

Nesse exemplo usamos as constantes do PHP para definir os dados de conexão com o banco de dados, *HOST*, *DBNAME*, *CHARSET*, *USER e PASSWORD*, mas os mesmos podem ser informados diretamente no momento em que se instância o objeto PDO.

Onde resume em apenas um atributo estático *$pdo* o qual vai conter um ponteiro para nosso objeto PDO, um método construtor privado como explicado acima para que a classe não seja instanciada e um método estático *getInstance*(), esse método é quem verifica a existência de uma instância em memória, caso não exista será instanciado um objeto PDO senão será retornado o objeto já existente. É sempre bom lembrar que atributos estáticos tem seu valor vinculado a classe e não aos objetos instanciados, dessa maneira um valor atribuído será persistido em memória até que seja alterado, independente de quantos objetos dessa classe existirem em memória.

Um ponto chave para esse script funcionar da maneira desejada é nesse atributo destacado em vermelho que foi informado na array *$opcoes*:

```
1 $opcoes = array(PDO::MYSQL_ATTR_INIT_COMMAND => 'SET NAMES
  UTF8', PDO::ATTR_PERSISTENT => TRUE);
```

Setando como *TRUE* esse atributo estamos informando ao PHP que essa conexão será persistida no pool do servidor mesmo após o final do script onde for chamada a conexão, sendo assim outras chamadas a esse objeto PDO serão reaproveitadas. Assim garantimos que somente uma conexão será realizada, uma vez que só a primeira solicitação irá instanciar um objeto PDO e as demais irão usufruir desse mesmo objeto.

Agora é só chamar o método estático para receber a conexão:

```
1 <?php
2 /* Require no script contendo a classe */
3 require_once 'conexao.php';
4
5 /* Variável recebendo instância do objeto PDO */
6 $pdo = Conexao::getInstance();
7 ?>
```

Habilitar o PDO

Antes de começarmos a trabalhar com o PDO, é necessário habilitar o driver do PDO e o driver referente ao banco que será utilizado. Para habilitar o PDO é bem simples, vá ate o seu arquivo php.ini que encontra-se dentro do diretório onde foi instalado o PHP e remova os comentários (;) das linhas abaixo.

Listagem 1: Habilitando PDO no Windows

```
extension=php_pdo.dll

extension=php_pdo_mysql.dll
```

Listagem 2: Habilitando PDO no Linux

```
extension=pdo.so

extension=pdo_mysql.so
```

PDO (PHP Data Object) - Definição

PDO_MYSQL é um driver que implementa a interface PHP Data Objects (PDO) - http://www.php.net/manual/pt_BR/intro.pdo.php para acesso do PHP ao MySQL 3.x, 4.x and 5.x.

PDO_MYSQL tem vantagens do nativo suporte a prepared statement $stmt presente no MySQL 4.1 e superior. Se você está usando um versão antiga da biblioteca do mysql client, PDO irá emular para você.

http://php.net/manual/pt_BR/ref.pdo-mysql.php

A utilização do PDO fornece uma camada de abstração em relação a conexão com o banco de dados visto que o PDO efetua a conexão com diversos bancos de dados da mesma maneira, modificando apenas a sua string de conexão.

Listagem 3: Conexão com o banco de dados com o PDO

```
$con = new PDO("mysql:host=localhost;dbname=exercicio", "root", "senha");
```

A classe PDO em sua instancia pede como parâmetro primeiro o banco que será utilizado, O caminho do banco de dados e o nome da base de dados. Após devemos inserir o login e a senha do banco de dados.

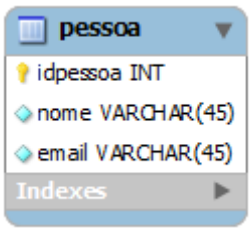
BANCO DE DADOS:host=CAMINHO BANCO;dbname=NOME BASE

Logo basta modificarmos essa string teremos conexão com qualquer outro banco de dados. Note que estamos apenas efetuando uma conexão simples com o banco de dados mysql, com a utilização do PDO poderíamos, se necessário, utilizar transações para as operações do banco, porem não é o objetivo deste artigo.

Para trabalhar com operações no banco, o PDO possui um método conhecido como prepare. Como o próprio nome diz, este método apenas prepara uma operação no banco de dados, logo se faz necessário a utilização de outros métodos como execute por exemplo, para realmente executar a operação.

Vejamos abaixo o exemplo de um inserção de dados no banco, imagem que temos uma tabela chamada de pessoa com os campos idpessoa, nome e email e iremos utilizar a biblioteca para inserir nesta tabela.

Lembrando, os métodos utilizados neste é exemplo são apenas alguns dos métodos existentes na biblioteca PDO, para o conhecimento de todos os métodos sugiro que acessem ao Manual do PHP (http://www.php.net/manual/pt_BR/book.pdo.php)



Listagem 4 - Criando a Tabela do Banco de dados

```
CREATE  TABLE IF NOT EXISTS pessoa (

  idpessoa INT NOT NULL AUTO_INCREMENT ,

  nome VARCHAR(45) NOT NULL ,

  email VARCHAR(45) NOT NULL ,

  PRIMARY KEY (idpessoa));
```

Listagem 5 - Exemplo de Inserção de Dados

```
$stmt = $con->prepare("INSERT INTO pessoa(nome, email) VALUES(?, ?)");

$stmt->bindParam(1,”Nome da Pessoa”);

$stmt->bindParam(2,”email@email.com”);

$stmt->execute();
```

Conforme podemos observar no código acima, primeiramente houve uma preparação no SQL que será enviado ao banco de dados. O método *prepare* ele apenas inicia uma query, esta possui diversos interrogações (?) que devemos substituir pelos valores reais adicionado.

Os valores deverão ser adicionados com o método *bindParam*, este método utiliza 2 ou 3 passagens de parâmetros. No exemplo criado temos apenas duas passagens de parâmetros, sendo a primeira referente a posição do interrogação que será substituído e o segundo parâmetro referente ao valor que será inserido no banco. Existe um 3 parâmetro que é referente ao tipo de valor que será enviado.

Com isso já temos uma inserção no banco de dados com o auxilio da biblioteca, com os dados já inseridos vamos conhecer como devemos fazer para buscar todos os valores da tabela pessoa.

Listagem 6 - Exemplo de Listagem de dados

```
$rs = $con->query(“SELEC idpessoa, nome, email FROM pessoa”);

while($row = $rs->fetch(PDO::FETCH_OBJ)){

  echo $row->idpessoa . “<br />”;
```

```
  echo $row->nome . “<br />”;

  echo $row->email . “<br />”;

}
```

No código acima temos uma consulta sem passagem de parâmetro de todos os dados armazenados na tabela pessoa. Utilizamos o método query para isso. O método query ira armazenar na variável $rs todos os dados referentes a consulta do banco.

Após a variável $rs ser populada, devemos utilizar um while para percorrer esses dados e trabalharmos com os valores. Dentro do while inserimos o método fetch do PDO que tem como função resgatar linha a linha do resultado da consulta. No método fetch utilizamos um parâmetro PDO::FETCH_OBJ, este parâmetro define a forma na qual os dados serão retornados e armazenados na variável $row, criada dentro do while.

O PDO::FETCH_OBJ trata cada linha da consulta como um objeto, transformando os campos que foram retornados em atributos do objeto $row. Acessamos estes atributos utilizando os caracteres -> (traço maior) e o nome do campo buscado.

Exemplo para imprimir o valor do campo nome da consulta teria que utilizar

Listagem 7 - Imprimindo Dados

```
$row->nome.
```

Se fossemos criar uma consulta com passagem de parâmetros seria utilizado o método *prepara*, pois o mesmo suporta inserção de elementos após a criação da query. Veja no exemplo abaixo se a consulta dos dados da tabela pessoa fosse realizada através de uma igualdade do nome.

Listagem 8 - Consulta com igualdade do nome

```
$rs = $con->prepare("SELECT idpessoa, nome, email FROM pessoa WHERE nome LIKE ?”);

$rs->bindParam(1, $nome . “%”);

if($rs->execute()){

if($rs->rowCount() > 0){

while($row = $rs->fetch(PDO::FETCH_OBJ)){
```

```
  echo $row->idpessoa . “<br />”;

  echo $row->nome . “<br />”;

  echo $row->email . “<br />”;

}

          }

       }
```

O método prepare foi utilizado no exemplo acima com a idéia apenas de iniciar a query e aguardar pela inclusão de valores posteriormente. O nome foi inserido em um segundo momento com o % (per cento), pois em SQL ele representa um coringa. Logo estaremos buscando por todas as pessoas armazenadas no banco de dados cujo seu nome inicie com a(s) letra(s) informada na variável $nome por exemplo.

Para efetuar as operações de atualização e deleção de dados no banco teremos um processo parecido com a inserção, seguem exemplos abaixo:

Listagem 9 - Deletando dados

```
$stmt = $con->prepare("DELETE FROM pessoa WHERE idpessoa = ?");

$stmt->bindParam(1, $id);

$stmt->execute();
```

Listagem 10 - Atualizando dados

```
$stmt = $con->prepare("UPDATE pessoa SET nome = ?, email = ? WHERE idpessoa = ?");

$stmt->bindParam(1,$nome );

$stmt->bindParam(2,$email);

$stmt->bindParam(3,$id);

$stmt->execute();
```