<

DEFINIÇÃO

Linguagem de Programação PHP. Linguagem PHP. Linguagem de script PHP. Linguagem de Programação Web PHP. Integração de PHP com banco de dados. Integração de PHP com MySQL. Integração de PHP com PostgreSQL. PDO. Classe PDO. Métodos da Classe PDO. Integração entre HTML, PHP e banco de dados.

PROPÓSITO

Apresentar as funcionalidades da linguagem de programação PHP para integração com banco de dados, fazendo uso da extensão PDO – PHP Data Objetcs.

PREPARAÇÃO

Para aplicação dos exemplos, serão necessários: um editor de texto com suporte à marcação PHP; um servidor web com suporte à linguagem PHP e com um SGBD instalado e configurado para ser utilizado com PHP; um aplicativo Cliente para acesso ao SGDB. Em relação ao editor, no Sistema Operacional Windows, é indicado o *Notepad*++. No Linux, o Nano Editor. Quanto ao servidor, recomenda-se o Apache. Sobre o SGDB deverá ser utilizado o MySQL ou o PostgreSQL. Por fim, quanto ao aplicativo Cliente para acesso ao BD, recomenda-se o DBeaver.

OBJETIVOS

MÓDULO 1

Definir a Classe PDO e sua utilização com MySQL e PostgreSQL

MÓDULO 2

Descrever os principais métodos da Classe PDO

MÓDULO 3

Construir uma aplicação contendo um formulário HTML, uma tabela HTML e scripts PHP para inserção e listagem de dados em/de um SGDB

INTRODUÇÃO

Ao longo deste tema, veremos como integrar o PHP com banco de dados fazendo uso da Classe PDO (PHP *Data Objects*), uma interface de acesso a *databases*.

Veremos desde a sua instalação, conceitos básicos, formas de utilização com MySQL e PostgreSQL, à utilização de dois dos seus principais métodos. Tudo isso apoiados em exemplos práticos.

Ao final deste tema, desenvolveremos um formulário HTML para a inclusão de informações em um banco de dados.

Vamos começar nossa jornada acessando os códigos-fontes originais propostos para o aprendizado de PHP com PDO. Baixe o **arquivo**, descompactando-o em seu dispositivo. Assim, você poderá utilizar os códigos como material de apoio ao longo do tema!

MÓDULO 1

- Definir a Classe PDO e sua utilização com MySQL e PostgreSQL

SISTEMAS GERENCIADORES DE BANCOS DE DADOS

Os bancos de dados são o ponto central de muitos sites, sistemas e aplicativos. Tais estruturas permitem o armazenamento e recuperação de qualquer tipo de informação, desde dados simples, como os posts de um blog pessoal, a dados altamente sensíveis, como os dados bancários ou financeiros, por exemplo. Nesse sentido, a linguagem de programação **PHP** oferece amplo suporte à integração com bancos de dados, sejam relacionais ou não, sejam suportados por diversos Sistemas Gerenciadores de Bancos de Dados (SGBD), como SQL Server, Oracle, MySQL e PostgreSQL – sendo esses dois últimos normalmente associados ao PHP.

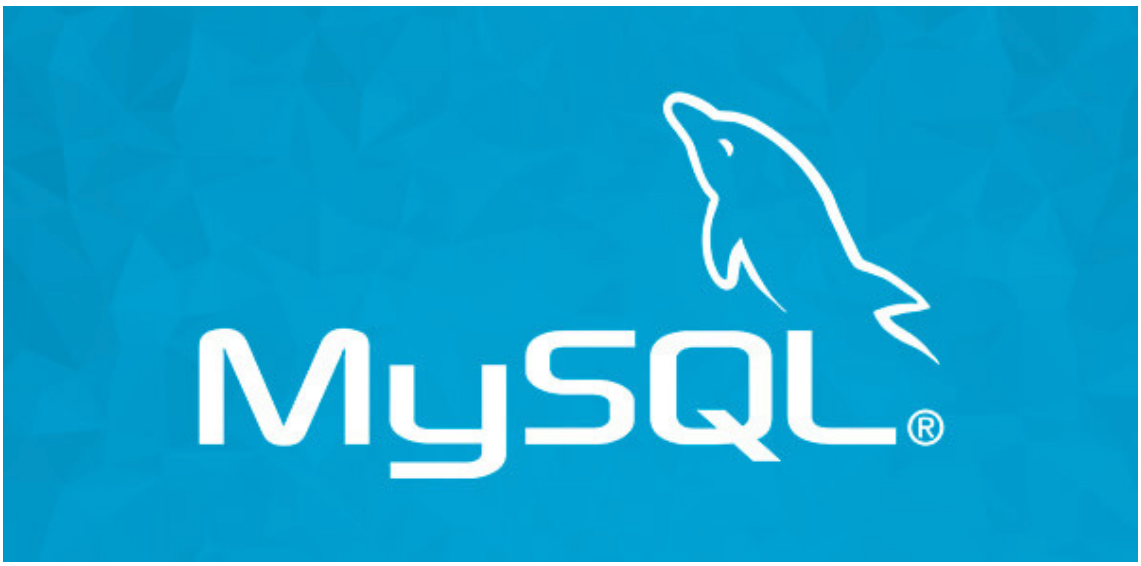
PHP

É uma linguagem interpretada livre, capaz de gerar conteúdo dinâmico na internet.

Inicialmente, antes de tratarmos da classe PDO e de sua utilização com dois dos principais SGBDs existentes, vamos observar um pouco mais esses sistemas e sua integração com PHP.

MYSQL

O MySQL é um sistema de gerenciamento de banco de dados criado em 1995. Inicialmente de código aberto, foi adquirido posteriormente pela Oracle, que, atualmente, mantém tanto uma versão GPL quanto uma versão comercial. Trata-se de um dos SGBDs mais utilizados na web, sendo, normalmente, associado ao PHP, dada a facilidade de conexão entre ambos.



Em PHP, estão disponíveis vários métodos e funções, através da extensão que leva o mesmo nome do SGBD, para conexão e execução de instruções SQL no MySQL.

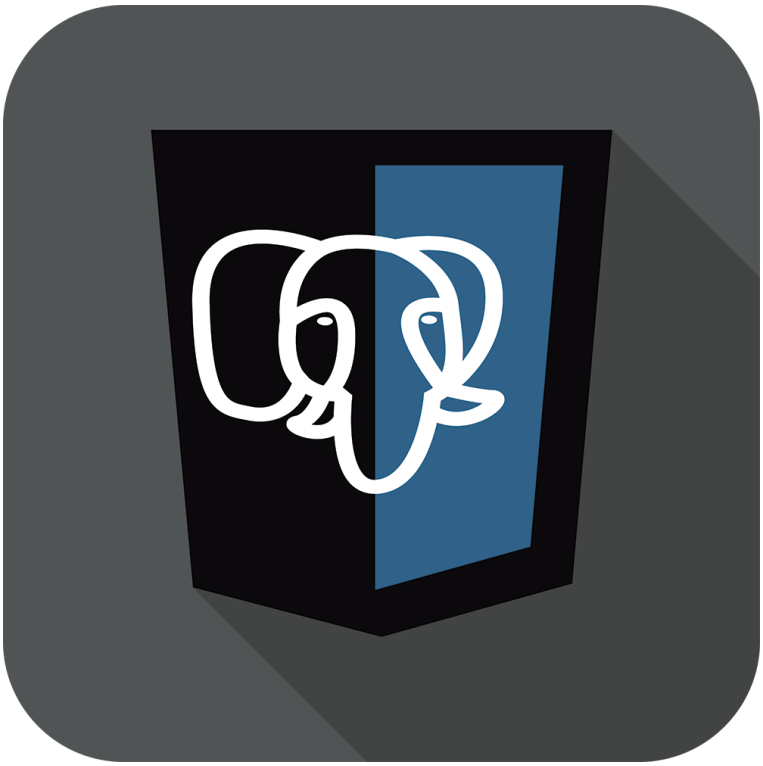
Para fins de comparação, o código abaixo demonstra a conexão com o MySQL e a execução de uma instrução de consulta SQL utilizando as funções PHP específicas para o SGBD em questão.

```
>?php
//Constantes para armazenamento das variáveis de conexão.
define('HOST', '127.0.0.1');
define('DBNAME', 'test');
define('USER', 'user');
define('PASSWORD', 'psswd');
//Conectando com o Servidor
$conn = mysqli_connect(HOST, USER, PASSWORD, DBNAME) or die( ' Não foi possível conectar.' );
//Realizando uma consulta no BD
$instrucaoSQL = "Select nome, cpf, telefone From Cliente";
```

```
$stmt = mysqli_prepare($conn, $instrucaoSQL);
mysqli_stmt_bind_result($stmt, $nome, $cpf, $tel);
mysqli_stmt_execute($stmt);
while (mysqli_stmt_fetch($stmt)) {
echo $nome . "\t";
echo $cpf . "\t";
echo $tel . "\n";
}
//Encerrando a conexão
mysqli_close($conn);
```

POSTGRESQL

O PostgreSQL é um SGBD de código aberto, atualmente disponível sob a licença BSD, criado em 1996. Enquanto o MySQL é um SGBD puramente relacional, o PostgreSQL é considerado um SGBD objeto-relacional. Na prática, essa diferença pode ser vista nas funcionalidades que o PostgreSQL possui, como a herança de tabelas e a sobreposição de funções. Outra diferença importante é que o PostgreSQL adere de forma mais próxima aos padrões SQL.



A linguagem PHP também possui uma série de métodos e funções, habilitados através da extensão pgsql, para conexão com o Postgres.

A exemplo do que fizemos com o MySQL, vejamos um código que utiliza funções PHP para conectar com o PostgresSQL e realizar uma instrução SQL de consulta de dados.

```
>?php
//Constantes para armazenamento das variáveis de conexão.
define('HOST', '127.0.0.1');
define('DBNAME', 'test');
define('USER', 'user');
define('PASSWORD', 'psswd');
//Conectando com o Banco de dados
$stringConn = "host=".HOST." dbname=".DBNAME." user=".USER." password=".PASSWORD;
$conn = pg_connect($stringConn) or die( ' Ocorreu um erro e não foi possível conectar ao
banco de dados.' );
//Realizando uma consulta no BD
$instrucaoSQL = "Select nome, cpf, telefone From Cliente";
$result = pg_query( $conn, $instrucaoSQL ) or die(' Ocorreu um erro na execução da instrução:
' . $instrucaoSQL );
//pg_query($dbcon, "SELECT id, nome FROM clientes");
//Imprimindo os dados da consulta
while ($row = pg_fetch_array( $result )){
echo $row['nome'] . "\t";
echo $row['cpf'] . "\t";
echo $row['telefone'] . "\n";
}
//Encerrando a conexão
pg_close($conn);
```

CONCEITOS BÁSICOS DA LINGUAGEM DE PROGRAMAÇÃO PHP

A linguagem PHP é uma linguagem **server side**, ou seja, atua no lado servidor, sobre um servidor web como o Apache, Nginx ou IIS. Trata-se de uma linguagem de script e que é gratuita, o que faz dela uma linguagem amplamente utilizada no Ambiente Web.

Em relação à sua sintaxe, é possível combinar código PHP com tags HTML, ou utilizar apenas código PHP em um script. Tal código deverá estar entre a tag inicial “<?php” e a tag final “?>” – sendo essa última não obrigatória, quando houver apenas código PHP em um script.

Tomando como base os exemplos de código acima, temos alguns recursos da linguagem sendo utilizados. A seguir, esses recursos serão revistos.

CONSTANTES

As constantes são identificadores ou nomes que contêm um valor único e imutável, ao contrário das variáveis que, conforme o nome indica, possuem valor variável ao longo de um programa.

No exemplo, foram declaradas constantes para armazenarem as credenciais de acesso ao SGBD. A sintaxe de utilização de constantes é composta pela palavra reservada *DEFINE*, seguida de parênteses, onde são incluídos o nome da constante e o seu valor.

VARIÁVEIS

As variáveis são objetos, ou espaços reservados na memória do computador, destinados a armazenarem e a exibirem os dados utilizados, e alterados, durante a execução de um programa.

Em PHP, as variáveis são representadas por um $ seguido pelo seu nome, que é *case-sensitive* - ou seja, uma letra maiúscula é diferente de uma letra minúscula ($var é diferente de $Var). Em relação à declaração e atribuição de valores, ambos os processos podem ocorrer ao mesmo tempo. Veja no último exemplo que a variável $stringConn foi declarada (inserida pela primeira vez no código) ao mesmo tempo que recebeu um valor (processo de atribuição). Por último, as variáveis em PHP não possuem tipo. Logo, não é necessário indicar o tipo de dado a ser armazenado por uma variável.

ESTRUTURAS DE DECISÃO E REPETIÇÃO

Nos exemplos, foi vista a estrutura de repetição *while*. Além dela, há outras estruturas de repetição disponíveis em PHP: *do-while*, *for* e *foreach*.

A respeito das estruturas de decisão, estão disponíveis em PHP: *if, else, elseif* e *switch*.

ARRAYS

Os arrays, ou vetores, são variáveis que guardam uma lista de itens relacionados. Tais variáveis são compostas pelo par índice/valor. A variável *$row*, nos exemplos, é um *array* associativo, ou seja, seus índices são compostos por strings – o nome de cada coluna selecionada do banco de dados. Há ainda os *arrays* numéricos e os híbridos (*arrays* com índices associativos e numéricos).

FUNÇÕES

As funções são pedaços de código que podem ser chamados a partir de qualquer parte do programa e que executam tarefas específicas. Além das funções definidas pelo usuário, em PHP estão disponíveis inúmeras funções nativas. Em nossos exemplos, temos as funções *define* e *echo*, além das relacionadas à conexão e ao manuseio de dados, como *mysql_connect* ou *pg_connect*, *mysql_query* ou *pg_query*, entre outras.

EXTENSÕES, BIBLIOTECAS E CLASSES

As extensões, bibliotecas e classes são um importante recurso presente na maioria das linguagens de programação. Através delas, é possível estender as funções básicas de uma linguagem, adicionando novos recursos para a execução de determinadas tarefas.

Em relação à definição, em PHP temos as extensões, que são arquivos binários (no S.O. Windows, as *.dll*, em S.O.s *unix-like*, os *.so*) que contêm bibliotecas. Já as bibliotecas são um conjunto de funções e/ou classes. As classes, então, dentro do paradigma de orientação a objetos, são um conjunto de códigos compostos por atributos e métodos. São exemplos de extensões os drivers para os SGBDs MySQL e PostgreSQL, que, quando habilitados, permitem a sua utilização com o PHP.

DICA

Consulte o Manual do PHP para uma lista completa de todas as suas extensões.

PDO

PDO, acrônimo para PHP Data Objects, ou Objetos de Dados PHP, e conforme definição vista anteriormente, é uma classe contida dentro de uma biblioteca e, por consequência, dentro de uma extensão. Ao referi-la, vamos chamá-la apenas de classe ao longo deste tema.

Trata-se de uma interface leve para acesso a bancos de dados em PHP. Nesse sentido, cabe a cada banco de dados implementar a interface PDO a fim de expor os seus recursos e funções específicos que, então, ficarão disponíveis para serem utilizados através do código PHP.

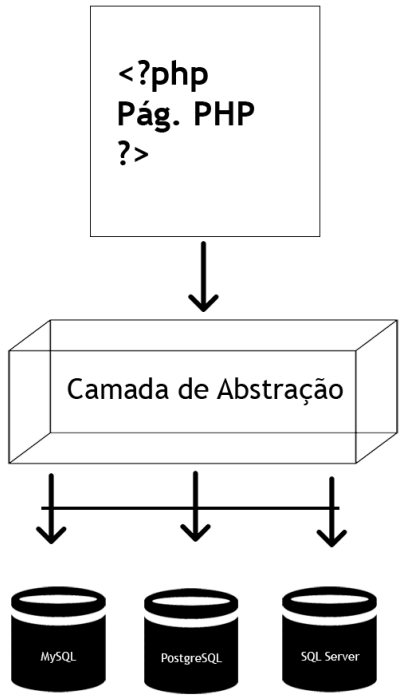


Figura 1: Funcionamento da Camada de Abstração

Para utilizarmos a classe PDO, é necessário habilitar sua extensão no servidor web, ou seja, habilitar a extensão que contém o driver específico para cada banco de dados com os quais se deseja trabalhar.

DICA

Em relação à instalação e configuração da extensão PDO e drivers mencionados, há diversos tutoriais disponíveis na internet. Lembre-se de que esse processo é diferente de acordo com o Sistema Operacional utilizado. A seção Explore + contém algumas referências base para esse processo.

Como mostrado na Figura 1, assim como nos códigos de exemplo relacionados à conexão e execução de instruções próprios do MySQL e do PostgreSQL, vistos anteriormente, a principal vantagem em se utilizar PDO no lugar das funções nativas do PHP para cada SGBD é o fato dessa extensão fornecer uma camada de abstração de acesso a dados. Em outras palavras, isso significa que:

É possível utilizar os métodos e funções PDO independente do SGBD.

Na prática, e voltando aos códigos PHP para integração com cada SGBD, perceba que, se fôssemos trocar o banco de dados utilizado, precisaríamos também mudar algumas linhas de

código: as de conexão com o banco de dados; as de execução de query, entre outras não vistas nos exemplos. Entretanto, fazendo uso do PDO, só precisaríamos mudar, nessa situação, o nome do driver a ser utilizado – conforme poderá ser visto adiante. No mais, todas as funções de execução de query, de tratamento de resultados, e demais, permaneceriam exatamente iguais.

DRIVERS PDO

Atualmente, estão disponíveis 12 drivers PDO, ou seja, é possível utilizá-lo com doze diferentes SGBDs. A tabela 1 ilustra os drivers e os bancos de dados suportados.

Driver	SGBD Suportados
PDO_CUBRID	Cubrid
PDO_DBLIB	FreeTDS / Microsoft SQL Server / Sybase
PDO_FIREBIRD	Firebird
PDO_IBM	IBM DB2
PDO_INFORMIX	IBM Informix Dynamic Server
PDO_MYSQL	MySQL 3.x/4.x/5.x
PDO_OCI	Oracle Call Interface
PDO_ODBC	ODBC v3 (IBM DB2, unixODBC e win32 ODBC)
PDO_PGSQL	PostgreSQL

PDO_SQLITE	SQLite 3 e SQLite 2
PDO_SQLSRV	Microsoft SQL Server / SQL Azure
PDO_4D	4D

Atenção! Para visualizaçãocompleta da tabela utilize a rolagem horizontal

CONEXÃO COM O SGBD UTILIZANDO PDO

A primeira ação necessária ao trabalharmos com banco de dados é realizar a conexão com o próprio. Para isso, utilizamos o construtor da classe PDO, que é responsável por criar uma instância de conexão. O fragmento abaixo demonstra a conexão com MySQL e com o PostgreSQL.

$dsn = new PDO("mysql:host=localhost;dbname=test", $user, $pass);

$dsn = new PDO("pgsql:host=localhost;dbname=test"; $user, $pass);

Como pode ser visto, o que muda entre as duas instruções é o nome do drive/SGBD. Na primeira, temos “mysql” e na segunda “pgsql”. Além disso, outras opções/parâmetros podem ser passadas no construtor, como a **porta**, por exemplo.

PORTA

Uma porta, em redes de computadores, tem a função de identificar aplicações e processos a fim de permitir que eles compartilhem uma única conexão com um determinado endereço IP.

Nesse sentido, a porta padrão do Mysql é a 3306 e a do PostgreSQL a 5432.

Reforçando o que foi dito, caso precisássemos alterar o SGBD usado em nosso projeto, só precisaríamos modificar essas informações acima e todo o restante do código, relacionado ao *database*, permaneceria inalterado.

TRATAMENTO DE EXCEÇÕES

Uma boa prática, em qualquer linguagem de programação, é tratar as possíveis exceções que possam ocorrer no código. Cada linguagem possui uma sintaxe própria, sendo muito comum na maioria a utilização da instrução “try/catch”.

A maioria das linguagens de programação conta com instruções específicas para o tratamento de exceções. Em linhas gerais, temos uma exceção que pode ser lançada e capturada, estando o respectivo código envolvido por um bloco *try*, cuja sintaxe deve contar com, ao menos, um bloco *catch* ou *finally*. Por fim, temos que o objeto lançado, ou seja, a exceção, precisa ser uma instância ou subclasse da classe *Exception*.

Em relação ao bloco *catch*, podemos ter vários relacionados a um bloco *try*, sendo cada um responsável pelo tratamento de um tipo de exceção. Já o conteúdo do bloco *finally* será sempre executado após o bloco *try* ou *catch*.

SAIBA MAIS

Consulte por **Exceções** no Manual do PHP.

O tratamento de exceções é uma prática recomendadíssima em qualquer código produzido. Além de informar ao usuário ou mesmo ao desenvolvedor que um erro ocorreu e que, com isso, o programa não funcionará de acordo com o esperado, o tratamento de exceções possibilita também que outras partes do programa permaneçam funcionais ou que o programa não seja encerrado de forma inesperada, mesmo que um erro tenha ocorrido.

A partir de um dos exemplos acima, poderíamos utilizar o seguinte código, com tratamento de possíveis exceções, para realizar a conexão com o MySQL, por exemplo:

```
<?php
//Constantes para armazenamento das variáveis de conexão.
define('HOST', '127.0.0.1');
define('PORT', '5432');
define('DBNAME', 'test');
define('USER', 'user');
define('PASSWORD', 'psswd');
try {
$dsn = new PDO("mysql:host=". HOST .
";port=".PORT.";dbname=" . DBNAME . ";user=" . USER .
```

";password=" . PASSWORD);
} catch (PDOException $e) {
echo 'A conexão falhou e retornou a seguinte mensagem de erro: ' . $e->getMessage();
}

ENCERRAMENTO DE CONEXÕES E CONEXÕES PERSISTENTES

Para encerrar uma conexão aberta através de PDO, basta atribuir *NULL* à variável que contém a instância da classe. O mesmo deve ser feito com as variáveis que armazenam o resultado da execução dos métodos que executam instruções SQL, como *Exec* e *Query* – que serão vistos mais adiante.

A linha a seguir demonstra o encerramento da conexão aberta no código anterior:

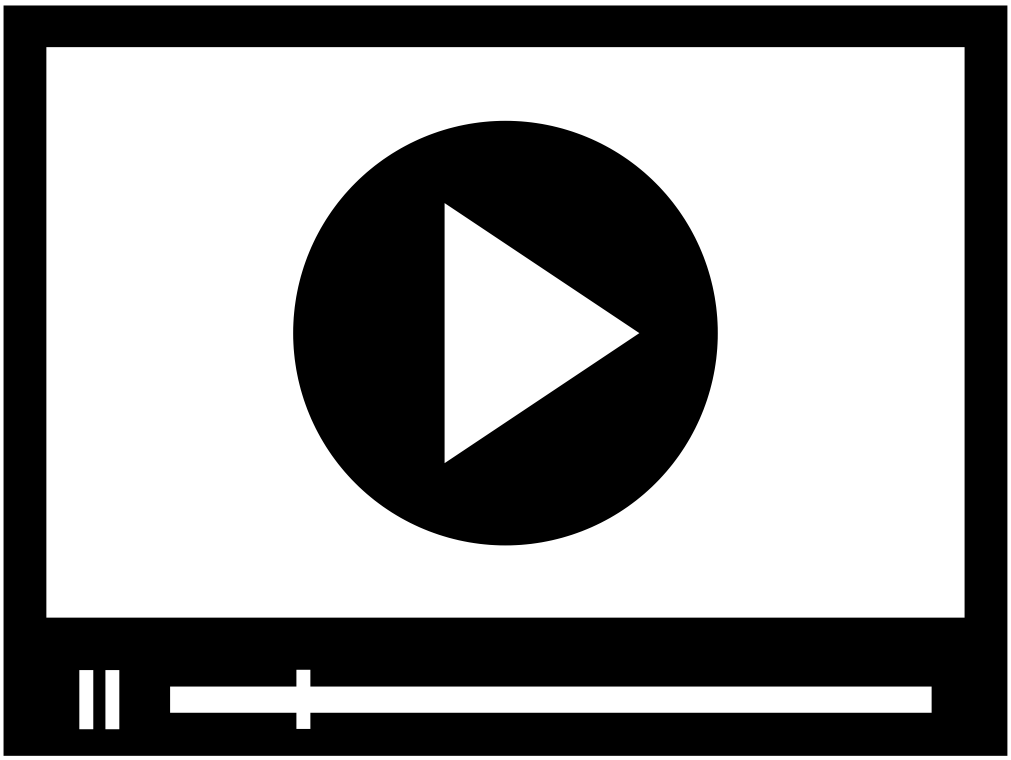
```
//...
$dsn = null;
```

PDO oferece suporte a conexões persistentes – que, em linhas gerais, consiste em não encerrar uma conexão aberta com o SGBD ao final de execução de um script. Com isso, é possível fazer cache da conexão e reutilizá-la quando outros scripts requisitarem uma conexão semelhante (com as mesmas credenciais) a essa que ficou aberta. Para usar conexões persistentes em PDO, é necessário incluir um parâmetro a mais no momento de criação da instância. Veja como ficaria o nosso exemplo de conexão com o PostgreSQL:

```
<?php
...
try {
$dsn = new PDO("pgsql:host=". HOST .
";port=".PORT.";dbname=" . DBNAME . ";user=" . USER .
";password=" . PASSWORD, array(PDO::ATTR_PERSISTENT => true));
} catch (PDOException $e) {
echo 'A conexão falhou e retornou a seguinte mensagem de erro: ' . $e->getMessage();
}
```

No vídeo a seguir, o especialista fará um resumo do módulo, descrevendo brevemente a classe PDO com ilustração da Figura 1 e demonstrando a conexão do PHP com MySQL e

PostgreSQL, usando recursos de captura de telas.



PDO E SUA CONEXÃO COM MYSQL E POSTGRESQL

Um resumo sobre PDO e sua conexão com MySQL e PostgreSQL

VERIFICANDO O APRENDIZADO

MÓDULO 2

- Descrever os principais métodos da Classe PDO

ORIENTAÇÃO A OBJETOS EM PHP

Como mencionado, uma classe, em linguagens de programação, é um conceito vinculado ao paradigma de orientação a objetos. Logo, antes de descrever a classe PDO e seus métodos e, para um melhor entendimento, serão descritos os conceitos básicos de OO em PHP.

O paradigma de orientação a objetos, na engenharia de software, representa uma mudança na forma de se construir programas: no lugar de um conjunto de procedimentos e variáveis agrupados por determinado contexto ou objetivo, muitas vezes, não organizados adequadamente, na O.O. temos uma visão mais próxima ao mundo real representada por objetos e estruturas (classes) com as quais temos maior familiaridade.

CLASSES E OBJETOS

Esses dois elementos são a base da programação orientada a objetos. As classes podem ser definidas como estruturas que definem um tipo de dados e podem ser compostas por atributos (variáveis) e por funções (métodos). Em alusão ao mundo real, são exemplos de classes: pessoas, produtos, veículos, imóveis etc. Assim como no mundo real, essas classes são compostas por características – os seus atributos. Por exemplo: uma pessoa possui nome, data de nascimento, CPF etc. Para acessar essas características, utilizamos métodos. Veja a representação dessa classe na figura abaixo:

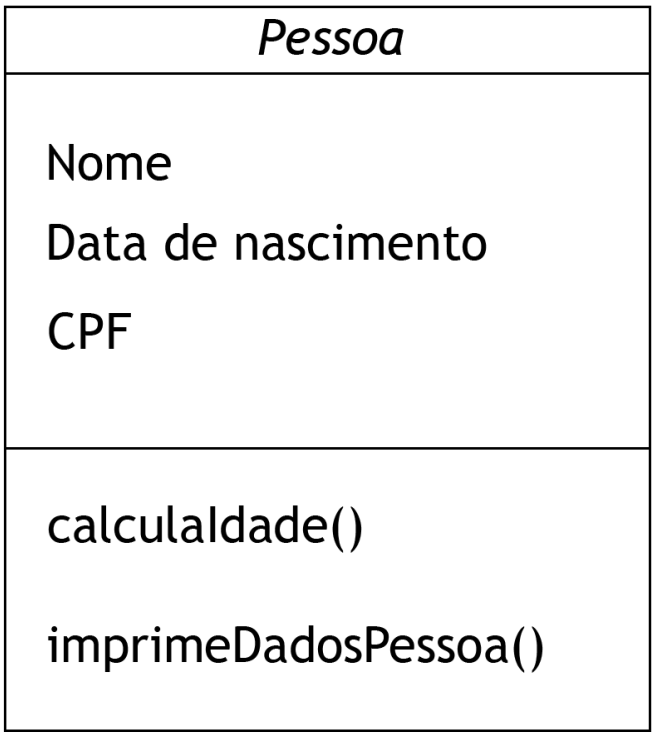


Figura 2: Diagrama de Classes – Classe Pessoa.

A partir dessa Classe, poderíamos construir novas classes que compartilhassem os mesmos atributos e funcionalidades que a Classe Pessoa, além de poderem possuir propriedades e funções próprias. Veja esse novo exemplo:

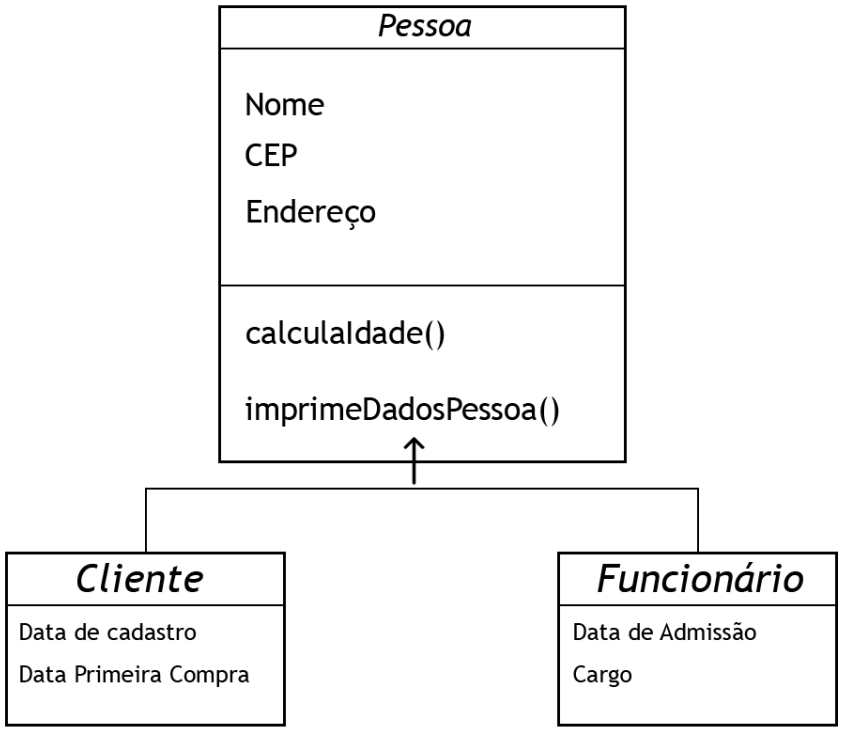


Figura 3: Diagrama de Classes – Exemplo de Herança entre Classes.

Esse compartilhamento de atributos e comportamentos, na orientação a objetos, recebe o nome de **herança**. Logo, dentro desse conceito, é possível criar classes, como as classes Cliente e Funcionário, que além de herdarem características de uma classe “pai”, possuem ainda suas próprias características distintas – data de cadastro e data da primeira compra, em Cliente e data de admissão e cargo, em Funcionário.

Há muito mais a ser visto sobre orientação a objetos. As considerações acima foram apresentadas de maneira resumida, a fim de iniciar esse assunto, ou mesmo revisá-lo, uma vez que utilizaremos alguns desses conceitos a seguir, quando abordarmos a Classe PDO, seus atributos e propriedades.

Por fim, abaixo são demonstrados exemplos práticos, em PHP, da criação de classes e instanciação de objetos de uma classe. Entre cada exemplo, estão presentes explicações adicionais. Note ainda os comentários inseridos no código, pois eles contêm algumas informações extras.

```
<?php
class Fruta
//Propriedades ou atributos da Classe
var $nome;
var $tipo;
//Construtor da Classe: Essa função é executada todas as vezes em que uma instância da classe é criada.
```

```
//Como abaixo, essa função pode ser vazia. Ou seja, não realiza nenhuma ação.
public function __construct(){ }
public function setNome($nome)
{
$this->nome = $nome;
}
public function getNome()
{
return $this->nome;
}
public function setTipo($tipo)
{
$this->tipo = $tipo;
}
public function getTipo()
{
return $this->tipo;
}
}
```

A partir dessa estrutura, a Classe “Fruta”, podemos declarar vários objetos “filhos” / provenientes dela mesma. Esses objetos também são chamados de instâncias e contêm valores distintos, uma vez que a estrutura é dinâmica. O próximo fragmento de exemplo demonstra a criação de alguns objetos oriundos da Classe “Fruta”.

```
<?php
include "fruta.class.php";

$fruta1 = new Fruta();
$fruta1->setNome("morango");
$fruta1->setTipo("vermelha");

$fruta2 = new Fruta();
$fruta2->setNome("laranja");
$fruta2->setTipo("cítrica");
```

Como visto no código acima, a criação de novas “frutas”, ou de objetos da classe “Frutas”, é feita com a criação de uma variável que recebe o operador “new” seguido do nome da Classe.

A partir desse ponto, essa variável em questão torna-se uma instância da classe. Na prática, essa instância possui os mesmos atributos e propriedades da classe da qual se originou.

A CLASSE PDO E SEUS MÉTODOS

A classe PDO é dividida em duas classes: a própria PDO e a PDOStatement. Enquanto a primeira contém métodos para conexão com os SGBDs, para o controle de transações e execução de queries; a segunda, PDOStatement, contém outros métodos relacionados às instruções executadas e ao seu retorno.

Os códigos a seguir mostram as estruturas dessas classes.

```
PDO {
public __construct ( string $dsn [, string $username [, string $passwd [, array $options ]]] )
public beginTransaction ( void ) : bool
public commit ( void ) : bool
public errorCode ( void ) : string
public errorInfo ( void ) : array
public exec ( string $statement ) : int
public getAttribute ( int $attribute ) : mixed
public static getAvailableDrivers ( void ) : array
public inTransaction ( void ) : bool
public lastInsertId ([ string $name = NULL ] ) : string
public prepare ( string $statement [, array $driver_options = array() ] ) : PDOStatement
public query ( string $statement ) : PDOStatement
public quote ( string $string [, int $parameter_type = PDO::PARAM_STR ] ) : string
public rollBack ( void ) : bool
public setAttribute ( int $attribute , mixed $value ) : bool
}
```

```
PDOStatement implements Traversable {
/* Propriedades */
readonly string $queryString;

/* Métodos */
public bindColumn ( mixed $column , mixed &$param [, int $type [, int $maxlen [, mixed
```

$driverdata]]]) : bool
public bindParam (mixed $parameter , mixed &$variable [, int $data_type = PDO::PARAM_STR
[, int $length [, mixed $driver_options]]]) : bool
public bindValue (mixed $parameter , mixed $value [, int $data_type = PDO::PARAM_STR]) :
bool
public closeCursor (void) : bool
public columnCount (void) : int
public debugDumpParams (void) : void
public errorCode (void) : string
public errorInfo (void) : array
public execute ([array $input_parameters = NULL]) : bool
public fetch ([int $fetch_style [, int $cursor_orientation = PDO::FETCH_ORI_NEXT [, int
$cursor_offset = 0]]]) : mixed
public fetchAll ([int $fetch_style [, mixed $fetch_argument [, array $ctor_args = array()]]]) :
array
public fetchColumn ([int $column_number = 0]) : mixed
public fetchObject ([string $class_name = "stdClass" [, array $ctor_args]]) : mixed
public getAttribute (int $attribute) : mixed
public getColumnMeta (int $column) : array
public nextRowset (void) : bool
public rowCount (void) : int
public setAttribute (int $attribute , mixed $value) : bool
public setFetchMode (int $mode) : bool
}

MÉTODO EXEC

Esse método pertencente à classe PDO, executa uma instrução SQL e retorna o número de linhas afetadas pela instrução. Sua sintaxe pode ser vista a seguir:

```
<?php
/*a variável $dsn, abaixo, corresponde à instação da classe PDO, inicializada na conexão com
o BD*/
$qtdeLinhasAfetadas = $dsn->exec("Delete From Cliente Where codigo_cliente = 1");
echo "Quantidade de linhas afetadas: " . $qtdeLinhasAfetadas
```

O código acima é funcional e complementar ao código anterior, uma vez que o método *Exec* deve ser invocado a partir da instância para a classe PDO (correspondente à variável $dsn em

nosso código).

Em relação às linhas afetadas, tal informação é útil para confirmarmos se a operação foi executada com sucesso. Logo, podemos utilizar uma estrutura de decisão, como *“if”*, para verificar o conteúdo da variável $qtdeLinhasAfetadas. Caso nenhuma linha tenha sido afetada, será retornado 0 (zero). Além disso, *Exec* pode retornar ainda o booleano *FALSE* ou então *“”* (vazio).

Por fim, cabe destacar que esse método não retorna à quantidade de linhas afetadas quando for executada uma instrução SELECT – o que pode ser feito através do próximo método que veremos, o *Query*.

MÉTODO QUERY

O método Query, também pertencente à classe PDO, tem função parecida com o método Exec. Entretanto, ele, além de executar uma instrução SQL, retorna – quando houver - um conjunto de resultados (*result set*) como um objeto PDOStatement. Em caso de falhas, é retornado o booleano *FALSE*. Vejamos um exemplo de sua utilização:

```
<?php
$instrucaoSQL = "Select nome, cpf, telefone From Cliente";
//a variável $dsn, abaixo, corresponde à instação da classe PDO, inicializada na conexão com o BD
$resultSet = $dsn->query($sql);
while ($row = $resultSet->fetch()) {
echo $row['nome'] . "\t";
echo $row['cpf'] . "\t";
echo $row['telefone'] . "\n";
}
```

Perceba que, como o método retorna um objeto, é possível acessar seus índices na forma de índices de *array*. Para isso, foi utilizado o método *fetch*, um laço *while* que percorreu o *result* set retornado, imprimindo os dados selecionados.

A respeito do método *fetch*, que retorna um *result* set no formato de array numérico e associativo, há ainda outros disponíveis: o *fetchAll, fetchColumn* e *fetchObject*. Além disso, esse método pode receber como parâmetro o tipo de retorno, ou seja, se deverá retornar um *array* associativo, numérico, associativo e numérico (que é o seu padrão por omissão), entre outros. Veja o exemplo de sua utilização retornando um *array* associativo:

```
<?php
//...
while ($row = $resultSet->fetch(PDO::FETCH_ASSOC)) {
//...
}
```

OUTROS MÉTODOS IMPORTANTES

Além desses dois métodos apresentados, as Classes PDO e PDOStatment possuem outros importantes métodos, como o *Prepare* e o *Execute*, por exemplo.

Considerando a sintaxe desses dois métodos, em comparação com o que vimos dos métodos *Exec* e *Query*, é boa prática fazer uso de ambos no lugar dos dois primeiros. Para embasar tal afirmação, veremos alguns conceitos extras, a começar pelo *SQL Injection*.

SQL *INJECTION*

O SQL *Injection*, ou Injeção de SQL, é um tipo de ataque baseado na manipulação e alteração de instruções SQL. Tal ataque é possível devido a vulnerabilidades encontradas em aplicações e sistemas que aceitam dados de entrada sem fazer o devido tratamento, além de executarem a conexão e instruções SQL utilizando usuários com privilégios altos.

Vejamos um exemplo prático, normalmente utilizado em formulários de login, onde poderia ser aplicada uma injeção de SQL.

```
<?php
//...
//Realizando uma consulta no BD através do login e senha recebidos por POST
$login = $_POST['login'];
$pswd = $_POST['pswd'];
$instrucaoSQL = "Select * From Usuario Where login = '$login' And password = '$pswd'";
$result = mysql_query( $instrucaoSQL ) or die(' Ocorreu um erro na execução da instrução: ' .
$instrucaoSQL . ' ' .
mysql_error() );
```

Perceba que, no código, o conteúdo das variáveis POST ‘login’ e ‘pswd’, equivalentes aos campos input de um formulário HTML para login, são recebidos no PHP e atribuídos a novas variáveis. A seguir, o conteúdo dessas variáveis é utilizado em uma instrução SQL. Veja o que

poderia acontecer se no formulário HTML fossem inseridos valores diferentes do previsto, como, por exemplo, os seguintes valores:

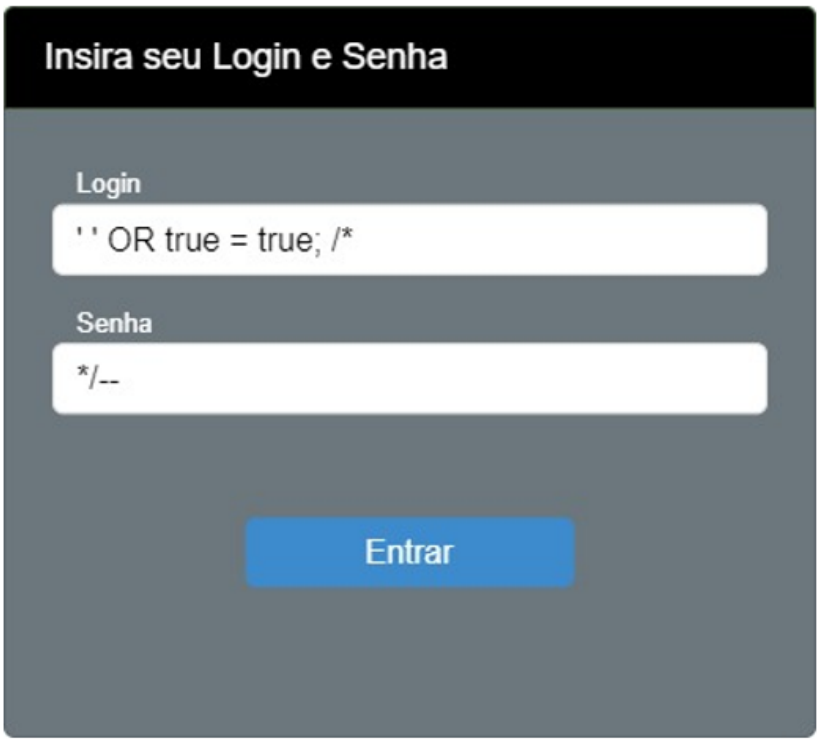


Figura 4: Exemplo de SQL *Injection* em um Formulário HTML.

Os valores inseridos no formulário mostrado na Figura 4 seriam recebidos no código PHP da seguinte maneira:

```
<?php
//...
//Realizando uma consulta no BD através do login e senha recebidos por POST
$login = $_POST['login'];
$pswd = $_POST['pswd'];
$instrucaoSQL = "Select * From Usuario Where login = '' OR true = true;/* And password = '*/--
'";
//...
```

Veja que, no lugar dos valores esperados – login e senha –, seriam recebidos comandos que modificariam o sentido original da instrução SQL, no código PHP, permitindo assim o acesso ao sistema em questão.

MÉTODOS *PREPARE* E *EXECUTE*

Para resolver os problemas acima, de SQL Injection, poderíamos escrever alguns códigos em PHP. Em linhas gerais, esses códigos conteriam instruções para tratar os dados recebidos antes de utilizá-los em instruções SQL. Esse trabalho poderia ser maçante, já que precisaria prever e tratar diversas formas de códigos maliciosos. Para facilitar e resolver tais questões, a classe PDO possui um método chamado *Prepare*.

- O método *Prepare*, como o nome indica, prepara uma instrução antes de ser executada, retornando um objeto do tipo *statement*, que será então executado através do método Execute (pertencente à classe PDOStatement).

- O método *Prepare* faz uso de um recurso chamado *bind*. Tal recurso vincula a variável ou valor a ser utilizado na instrução SQL a uma outra variável (também pode ser utilizado o sinal "?"). Durante esse processo de preparação da instrução e *bind* dos valores, a classe PDO realiza, de forma interna, ou seja, sem que precisemos nos preocupar com eles, uma série de tratamentos para evitar a SQL *Injection*.

Vejamos um exemplo prático de utilização do método *Prepare*:

```
<?php
//Constantes para armazenamento das variáveis de conexão.
define('HOST', '127.0.0.1');
define('PORT', '5432');
define('DBNAME', 'test');
define('USER', 'user');
define('PASSWORD', 'psswd');
try {
$dsn = new PDO("mysql:host=". HOST .
";port=".PORT.";dbname=" . DBNAME . ";user=" . USER .
";password=" . PASSWORD);
} catch (PDOException $e) {
echo 'A conexão falhou e retorno a seguinte mensagem de erro: ' . $e->getMessage();
}

//Realizando uma consulta no BD através do login e senha recebidos por POST
$login = $_POST['login'];
$pswd = $_POST['pswd'];

$stmt = $dsn->prepare("Select * From Usuario Where login = :login And password =
```

```
:password");
$stmt->execute([':login' => $login, ':password' => $pswd]);

/*Aqui entraria o código para tratar o resultado da instrução SQL acima*/

//Destruindo o objecto statement e fechando a conexão
$stmt = null;
$dsn = null;
```

DICA

Repare no código que o método *prepare* recebe como parâmetro a instrução SQL a ser executada e que, nos lugares onde seriam utilizadas as variáveis PHP com os valores provenientes do formulário HTML, são usadas outras variáveis, chamadas parâmetros nomeados (*named parameters*).

A seguir, o método *execute* faz o vínculo (*bind*) entre os *named parameters* e as variáveis PHP. O mesmo código acima poderia ser executado tendo o sinal de interrogação “?” no lugar dos parâmetros nomeados. Veja como ficaria esse fragmento de código:

```
<?php
//...

$stmt = $dsn->prepare("Select * From Usuario Where login = ? And password = ?");
$stmt->execute([$login, $pswd]);

//...
```

UM POUCO DE PRÁTICA

A fim de melhor assimilar o conteúdo apresentado até aqui, chegou a hora de praticar. Todos os códigos anteriores, usados como exemplos, são funcionais. Logo, a começar pelo código responsável pela conexão com o SGBD (e tomando o cuidado de alterar as credenciais, nas constantes PHP, para refletirem os dados de seu ambiente), copie e execute cada um deles

relacionados à classe PDO. Veja, por exemplo, as diferenças, na prática, entre os métodos *Exec* e *Query*; e entre esses dois métodos e os métodos *Prepare* e *Execute*. Tente realizar uma mesma consulta ou instrução SQL usando os dois e analise o resultado.

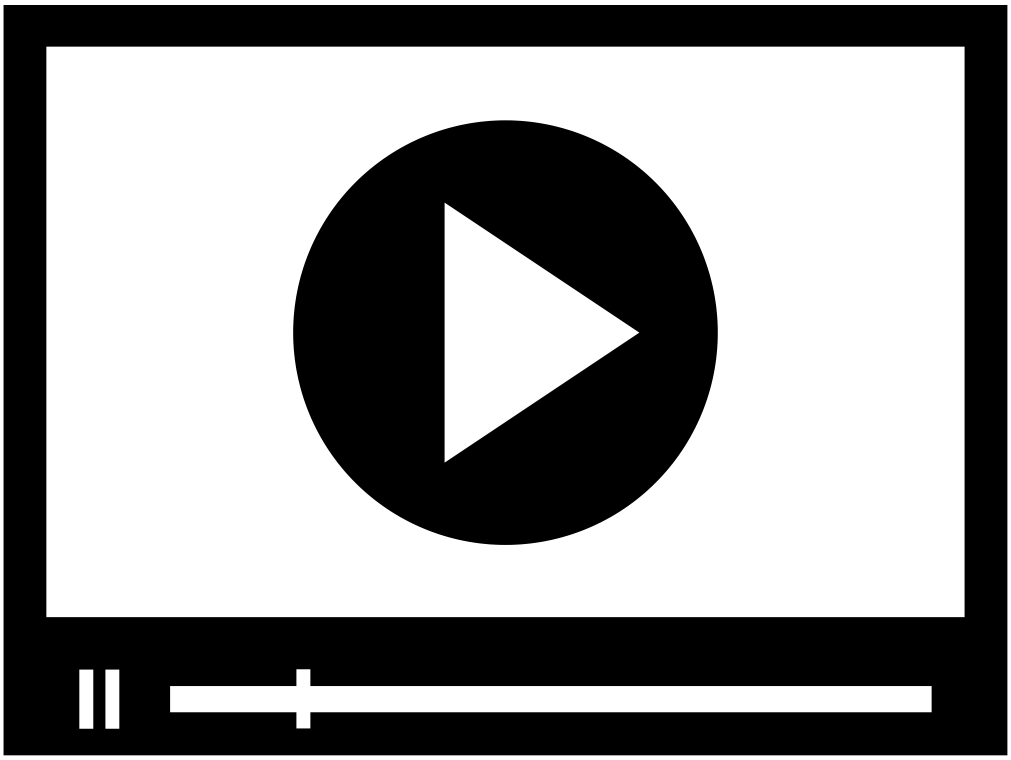
RESUMINDO

- Inclua, ao menos, uma instrução para cada tipo possível: *insert, update delete* e *select*;

- Faça o tratamento de erros e exceções;

- Veja onde é possível melhorar os códigos utilizados. Você pode, por exemplo, incluir algumas verificações extras etc.

A seguir o especialista demonstrará a execução de códigos apresentados no módulo, com consultas SELECT e manipulação de dados INSERT, UPDATE e DELETE. Assista ao vídeo.



UM POUCO DE PRÁTICA

VERIFICANDO O APRENDIZADO

MÓDULO 3

- Construir uma aplicação contendo um formulário HTML, uma tabela HTML e scripts PHP para inserção e listagem de dados em/de um SGDB

PREPARAÇÃO

Este módulo terá caráter prático. Logo, todos os conceitos vistos nos módulos anteriores serão utilizados na confecção de um Formulário HTML, que enviará dados para um script PHP que os inserirá em um SGBD – nesse caso, no PostgreSQL, e de uma página para listagem desses mesmos dados. Cada etapa desse processo será apresentada em detalhes, desde a criação das tabelas no SGBD, passando pela criação do formulário HTML e chegando nos scripts PHP que farão a conexão e inserção dos dados através das classes PDO e PDOStatement.

MOTIVAÇÃO

Deverá ser criado um formulário para armazenar os dados de clientes, contendo o seu nome completo (obrigatório), o seu CPF (obrigatório; apenas os números), um endereço de e-mail válido (obrigatório) e sua data de nascimento (obrigatório; dia, mês e ano). Além disso, deverá ser criada uma listagem para exibição dos clientes cadastrados.

ESQUEMATIZANDO O BANCO DE DADOS

Os dados dos clientes deverão ser armazenados em uma tabela. Tal tabela deverá possuir um campo identificador, autoincremental e único.

A instrução SQL abaixo contém um exemplo de como a tabela cliente pode ser criada.

```
CREATE TABLE cliente (
id_cliente serial NOT NULL,
nome_cliente varchar(255),
cpf_cliente varchar(11),
email_cliente varchar(255),
data_nascimento_cliente timestamp,
PRIMARY KEY (id_cliente)
);
```

DICA

A sintaxe completa da instrução “Create Table”, para o PostgreSQL, pode ser vista em seu próprio manual.

CONFECCIONANDO O FORMULÁRIO HTML

O formulário HTML deverá ser escrito utilizando a HTML5. Além disso, os campos deverão ser validados de acordo com seu tipo (quando possível), tamanho e obrigatoriedade de preenchimento. Poderá ser utilizado um framework CSS, como o Bootstrap, para uma melhor apresentação do cadastro.

O código do formulário poderá ser visto ao final deste módulo, junto aos demais exemplos de código. Tal código poderá ser usado como template para desenvolvimentos similares.

CONSIDERAÇÕES SOBRE O FORMULÁRIO HTML

O formulário HTML possui elementos de ligação com o script PHP para o qual será submetido. A tabela 1 apresenta esses elementos, seus atributos e uma breve descrição.

Elemento	Atributo	Obrigatório?	Descrição
form	*action*	Sim	Contém a URL ou nome do script, quando na mesma pasta que o arquivo HTML, que processará o documento.
form	*method*	Não. O padrão é POST	Define o método HTTP a ser utilizado para transmissão dos dados – GET ou POST.
input	*name*	Sim	Contém o nome do campo que será recebido no PHP como índice associativo no array $_GET, $_POST ou $_REQUEST.

button ou *input*	*type*	Sim	O valor “submit”, atribuído ao atributo *type* em um desses campos, cria o evento de envio do formulário.

Atenção! Para visualizaçãocompleta da tabela utilize a rolagem horizontal

Em termos de boas práticas, é recomendado validar os dados a serem submetidos tanto no lado cliente, ou seja, no código HTML diretamente, com a utilização dos recursos introduzidos pela HTML5, ou através de Javascript, assim como no lado servidor. Com isso, caso a validação falhe, por algum motivo, do lado cliente, teremos a garantia de validação do lado servidor. Veremos a seguir como realizar a validação de nosso formulário nos dois ambientes – cliente e servidor.

VALIDAÇÃO NO LADO CLIENTE – HTML5

A partir da HTML5, é possível marcar um elemento como de preenchimento obrigatório fazendo uso do atributo *required*. No código do formulário, visto ao final deste módulo, os campos “Nome”, “CPF”, “Endereço de E-mail” e “Data de Nascimento” receberam esse atributo. Além disso, foram incluídas outras regras de validação: o campo CPF precisa ter exatos 11 caracteres (definidos com o uso dos atributos *minlength* e *maxlength*) e o endereço de e-mail precisa ser válido – tal validação é feita diretamente pela HTML, ao definirmos a tag *<input>* sendo do tipo (*type*) “email”.

VALIDAÇÃO NO LADO CLIENTE – JAVASCRIPT

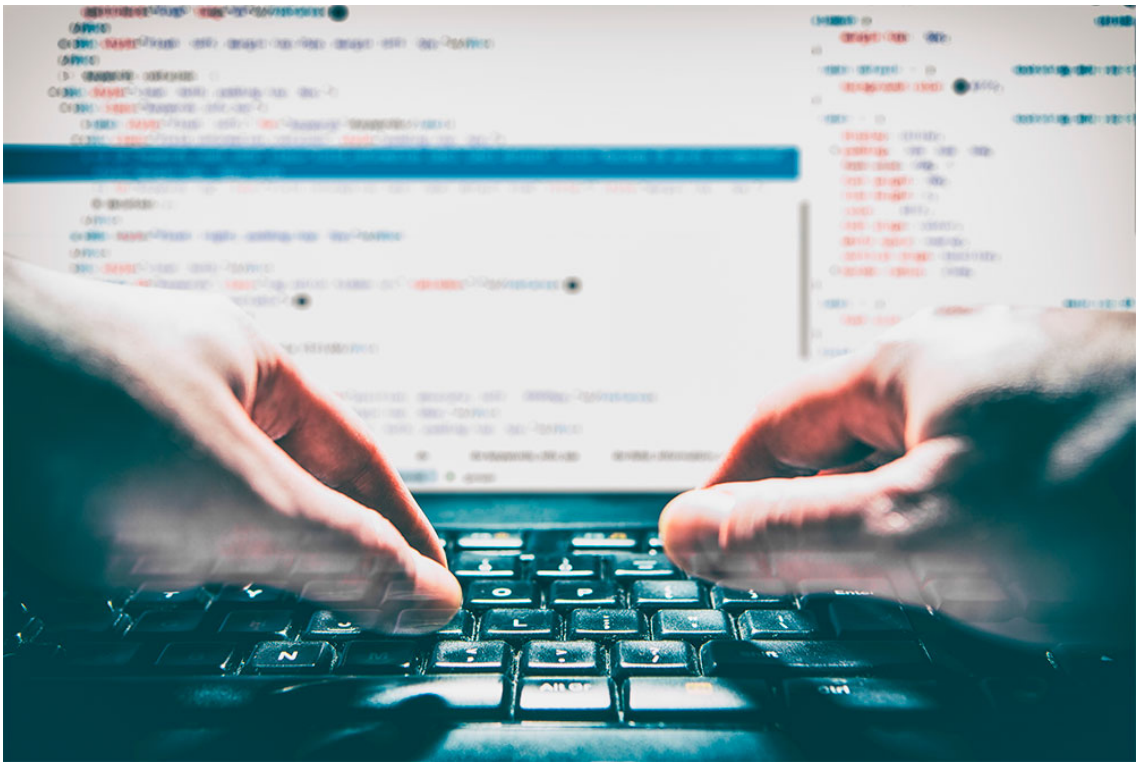
Para validar formulários, antes da HTML5, era necessário utilizar funções em códigos escritos na linguagem Javascript. Com isso, o formulário é verificado e, caso passe pelas validações, submetido ao servidor. Do contrário, a respectiva mensagem de falha é exibida, para as devidas correções. A validação do nosso formulário, utilizando Javascript, pode ser vista no código abaixo.

```
function validarFormulario(formulario){
if(formulario.nome_cliente.value === "" || formulario.nome_cliente.value === null) {
alert("O campo Nome não pode ficar vazio.");
formulario.nome_cliente.focus();
return false;
}
if(formulario.cpf_cliente.value.length != 11) {
alert("O campo CPF precisa ter 11 caracteres.");
```

```
formulario.cpf_cliente.focus();
return false;
}
//o campo e-mail precisa ser válido, ou seja, deve : "@" e "."
if(formulario.email_cliente.value.indexOf("@") == -1 || formulario.email_cliente.value.indexOf(".")
== -1) {
alert("O campo E-mail não é válido.");
formulario.email_cliente.focus();
return false;
}
if(formulario.data_nascimento_cliente.value === "" || formulario.data_nascimento_cliente.value
=== null) {
alert("O campo Data de Nascimento não pode ficar vazio.");
formulario.data_nascimento_cliente.focus();
return false;
}
}
```

Para utilizar a função Javascript acima, é necessário mudar um pouco o código do arquivo HTML que contém o formulário, removendo os atributos da HTML5, modificando o DocType para uma versão anterior da HTML, entre outros ajustes. Para facilitar, o código completo desse arquivo HTML foi adicionado ao final, junto aos demais códigos.

CODIFICANDO O SCRIPT PHP QUE PROCESSARÁ O FORMULÁRIO



O script PHP definido no atributo *“action”* da tag *“form”*, no arquivo HTML, é o responsável por receber os dados do formulário, validar os dados recebidos, conectar com o banco de dados e inserir as informações.

O atributo *method*, do formulário HTML, define o método HTTP utilizado para a transmissão dos dados – POST ou GET, sendo POST o valor padrão, caso esse atributo seja omitido no formulário. Para tratar os dados transmitidos por cada um desses métodos, há uma variável global pré-definida em PHP: $_POST e $_GET. Além disso, há também a variável $_REQUEST, que recebe os dados transmitidos por ambos os métodos.

Essas variáveis pré-definidas são, na verdade, um *array* associativo, cujos índices equivalem ao valor definido para o atributo *name*, em cada campo do formulário. Logo, se um *input* no formulário HTML for definido com *“name*=nome_cliente”, em PHP ele poderá ser lido dessa forma: $_REQUEST[‘nome_cliente’] - ou através de $_POST[‘nome_cliente’] ou $_GET[‘nome_cliente’], dependendo do método utilizado.

DICA

Normalmente, em cenários reais, são utilizados padrões de projeto, como o MVC (*Model-View-Controller*), e paradigmas como a Orientação a Objetos, para realizar a integração entre a HTML, o PHP e o Banco de Dados. Tais questões foram deixadas de lado em nosso cenário,

por estarem fora do escopo deste tema. Entretanto, é recomendado estudar sobre esses padrões e paradigmas a fim de aplicá-los, garantindo assim maior qualidade ao código gerado, além de outros benefícios.

O script PHP que processa o formulário pode ser visto ao final deste módulo. Tal script realiza a conexão com o SGBD PostgreSQL, prepara a instrução SQL com o método *Prepare* e os insere no banco de dados através do método *Execute*. Mais adiante, é realizada uma verificação e, em caso de sucesso ou erro, exibida uma mensagem correspondente. Por fim, tanto a instância de conexão PDO quanto o objeto PDOStatement são encerrados.

LISTANDO OS DADOS CADASTRADOS

Embora o objetivo principal fosse o formulário de inserção de dados no SGBD, é interessante vermos também a recuperação de informações e exibição em uma página HTML. Como dito, embora funcional, o código utilizado para listagem das informações tem caráter apenas de estudo. Em situações reais, uma série de cuidados, tanto estéticos como de padrões de codificação, devem ser utilizados.

Nosso código de exemplo, que pode ser visto ao final do módulo, é composto por uma única página PHP contendo, tanto códigos de programação, quanto elementos HTML. Veja o código base no exemplo.

CÓDIGOS DE EXEMPLO

CÓDIGO DO FORMULÁRIO HTML PARA INSERÇÃO DE CLIENTES

```
<!DOCTYPE html>
<html>
<head>
<title>Formulário HTML - Cadastro de Clientes
<meta charset="utf-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
<!-- Bootstrap -->
<link href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css"
rel="stylesheet" />
</head>
<body>
<div class="container">
<div class="row">
<div class="col-md-12">
<form action="processa_cliente.php" method="post">
<div class="row">
<div class="col-md-8">
<div class="card">
<div class="card-header">
<h3>Cadastro de Clientes</h3>
</div>
<div class="card-body">
<div class="form-group">
<label for="nome_cliente">Nome
<input type="text" class="form-control" id="nome_cliente" name="nome_cliente"
placeholder="Seu nome" required>
</div>
<div class="form-group">
<label for="cpf_cliente">CPF
<input type="text" class="form-control" minlength="11" maxlength="11" id="cpf_cliente"
name="cpf_cliente" placeholder="Seu CPF" title="Digite apenas números" required>
</div>
<div class="form-group">
<label for="email_cliente">Endereço de E-mail</label>
<input type="email" class="form-control" id="email_cliente" name="email_cliente" aria-
describedby="emailHelp" placeholder="Seu e-mail" required>
<small id="emailHelp" class="form-text text-muted">Nunca vamos compartilhar seu email, com
ninguém.
</div>
<div class="form-group">
<label for="data_nascimento_cliente">Data de Nascimento <input type="date" class="form-
control" id="data_nascimento_cliente" name="data_nascimento_cliente" placeholder="Sua data
```

```
de nascimento" required>
</div>
<div class="form-group text-center">
<button type="submit" class="btn btn-primary">Enviar
</div>
</div>
</div>
</div>
</div>
</form>
</div>
</div>
</div>
<script src="https://code.jquery.com/jquery-3.5.1.min.js">>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/js/bootstrap.min.js">>
</body>
</html>
```

CÓDIGO DO FORMULÁRIO COM VALIDAÇÃO EM JAVASCRIPT

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Formulário HTML - Cadastro de Clientes
<meta charset="utf-8">
<link href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css"
rel="stylesheet" />
<script type="text/javascript">
function validarFormulario(formulario){
//o código da função está no tópico “Validação no lado Cliente – Javascript”
}
</script>
</head>
<body>
<div class="container">
```

```
<div class="row">
<div class="col-md-12">
<form action="processa_cliente.php" name="form_clientes" method="post" onsubmit="return
validarFormulario(this);">
<div class="row">
<div class="col-md-8">
<div class="card">
<div class="card-header">
<h3>Cadastro de Clientes</h3>
</div>
<div class="card-body">
<div class="form-group">
<label for="nome_cliente">Nome</label>
<input type="text" class="form-control" id="nome_cliente" name="nome_cliente"
placeholder="Seu nome" >
</div>
<div class="form-group">
<label for="cpf_cliente">CPF</label>
<input type="text" class="form-control" id="cpf_cliente" name="cpf_cliente" placeholder="Seu
CPF" title="Digite apenas números" >
</div>
<div class="form-group">
<label for="email_cliente">Endereço de E-mail</label>
<input type="text" class="form-control" id="email_cliente" name="email_cliente"
placeholder="Seu e-mail" >
</div>
<div class="form-group">
<label for="data_nascimento_cliente">Data de Nascimento</label>
<input type="text" class="form-control" id="data_nascimento_cliente"
name="data_nascimento_cliente" placeholder="Sua data de nascimento" >
</div>
<div class="form-group text-center">
<button type="submit" class="btn btn-primary">Enviar</button>
</div>
</div>
</div>
```

```
</div>
</div>
</form>
</div>
</div>
</div>
<script src="https://code.jquery.com/jquery-3.5.1.min.js">>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/js/bootstrap.min.js">>
</body>
</html>
```

SCRIPT PHP PARA PROCESSAMENTO DO FORMULÁRIO HTML

```
<?php
/***VALIDAÇÃO DOS DADOS RECEBIDOS DO FORMULÁRIO ***/
if($_REQUEST['nome_cliente'] == ""){
echo "O campo Nome não pode ficar vazio.";
exit;
}
if(strlen($_REQUEST['cpf_cliente']) != 11){
echo "O campo CPF precisa ter 11 caracteres.";
exit;
}
if(!stripos($_REQUEST['email_cliente'], "@") || !stripos($_REQUEST['email_cliente'], ".")){
echo "O campo E-mail não é válido.";
exit;
}
if($_REQUEST['data_nascimento_cliente'] == ""){
echo "O campo Data de Nascimento não pode ficar vazio.";
exit;
}
/***FIM DA VALIDAÇÃO DOS DADOS RECEBIDOS DO FORMULÁRIO ***/

/***CONEXÃO COM O BD ***/
//Constantes para armazenamento das variáveis de conexão.
define('HOST', '192.168.52.128');
```

```
define('PORT', '5432');
define('DBNAME', 'minimundo');
define('USER', 'postgres');
define('PASSWORD', '159753');
try {
$dsn = new PDO("pgsql:host=". HOST . ";port=".PORT.";dbname=" . DBNAME . ";user=" .
USER . ";password=" . PASSWORD);
} catch (PDOException $e) {
echo 'A conexão falhou e retorno a seguinte mensagem de erro: ' . $e->getMessage();
}
/*** FIM DOS CÓDIGOS DE CONEXÃO COM O BD ***/

/***PREPARAÇÃO E INSERÇÃO NO BANCO DE DADOS ***/
$stmt = $dsn->prepare("INSERT INTO cliente(nome_cliente, cpf_cliente, email_cliente,
data_nascimento_cliente) VALUES (?, ?, ?, ?) ");
$resultSet = $stmt->execute([$_REQUEST['nome_cliente'],
$_REQUEST['cpf_cliente'], $_REQUEST['email_cliente'],
$_REQUEST['data_nascimento_cliente']]);
if($resultSet){
echo "Os dados foram inseridos com sucesso.";
}else{
echo "Ocorreu um erro e não foi possível inserir os dados.";
}
//Destruindo o objecto statement e fechando a conexão
$stmt = null;
$dsn = null;
```

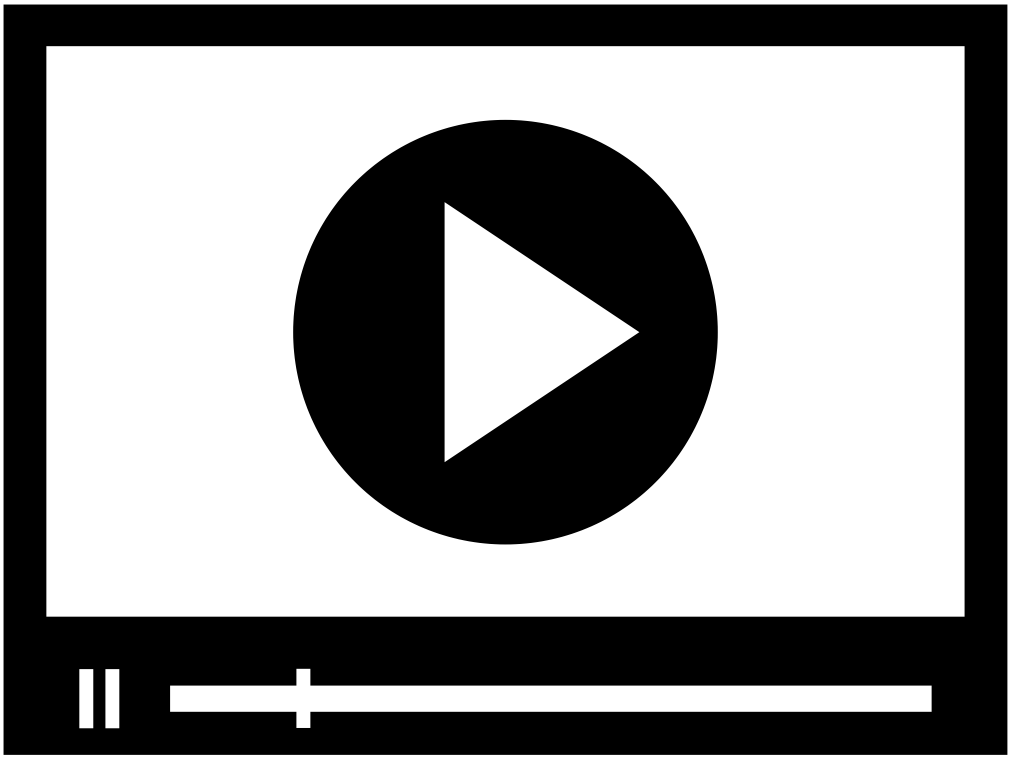
SCRIPT PHP PARA LISTAGEM DE CLIENTES

```
<?php
/***CONEXÃO COM O BD ***/
//O código de validação server side pode ser visto no exemplo de código 3.
//Constantes para armazenamento das variáveis de conexão.
define('HOST', '192.168.52.128');
define('PORT', '5432');
define('DBNAME', 'minimundo');
define('USER', 'postgres');
```

```
define('PASSWORD', '159753');
try {
$dsn = new PDO("pgsql:host=". HOST . ";port=".PORT.";dbname=" . DBNAME . ";user=" .
USER . ";password=" . PASSWORD);
} catch (PDOException $e) {
echo 'A conexão falhou e retorno a seguinte mensagem de erro: ' . $e->getMessage();
}
/***PREPARAÇÃO E INSERÇÃO NO BANCO DE DADOS ***/
$instrucaoSQL = "Select id_cliente, nome_cliente, cpf_cliente, email_cliente,
data_nascimento_cliente From cliente";
$resultSet = $dsn->query($instrucaoSQL);
?>
<!DOCTYPE html>
<html>
<head>
<title>Formulário HTML - Cadastro de Clientes</title>
<meta charset="utf-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<!-- Bootstrap -->
<link href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css"
rel="stylesheet" />
</head>
<body>
<div class="container">
<div class="row">
<div class="col-md-12">
<div class="row">
<div class="col-md-8">
<div class="card">
<div class="card-header">
<h3>Listagem de Clientes</h3>
</div>
<div class="card-body">
<table class="table">
<thead class="thead-dark">
```

```
<tr>
<th scope="col">#
<th scope="col">Nome
<th scope="col">CPF
<th scope="col">E-mail
<th scope="col">Data de Nascimento
</tr>
</thead>
<tbody>
<?php
<while ($row = $resultSet->fetch(PDO::FETCH_ASSOC)):
?>>
<tr>
<th scope="row"><?php echo $row['id_cliente']; ?></th>
<td><?php echo $row['nome_cliente']; ?></td>
<td><?php echo preg_replace("/(\d{3})(\d{3})(\d{3})(\d{2})/", "\$1.\$2.\$3-\$4",
$row['cpf_cliente']); ?></td>
<td><?php echo $row['email_cliente']; ?></td>
<td><?php echo date('d/m/Y', strtotime($row['data_nascimento_cliente'])); ?></td>
</tr>
<?php>
endwhile;
?>
</tbody>
</table>
</div>
</div>
</div>
</div>
</div>
</div>
</div>
<script src="https://code.jquery.com/jquery-3.5.1.min.js">>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/js/bootstrap.min.js">>
</body>
</html>
```

Como este módulo é essencialmente prático, o especialista demonstrará, no vídeo a seguir, a execução dos códigos apresentados ao longo do módulo.



RESUMO DO MÓDULO

VERIFICANDO O APRENDIZADO

CONCLUSÃO

CONSIDERAÇÕES FINAIS

Ao longo deste tema, cujo propósito foi apresentar as funcionalidades da linguagem de programação PHP para integração com bancos de dados, demonstramos alguns recursos da linguagem. Alguns diretamente voltados para esse propósito, como as funções de conexão específicas com os SGBDs MySQL e PostgreSQL, além da classe PDO e alguns de seus métodos, outros relacionados à linguagem em si, como as constantes, as funções, as extensões, o tratamento de exceções, e o paradigma de orientação a objetos. Ao final, cada um desses conceitos foi aplicado na construção de um Formulário e de uma Listagem HTML que insere e recupera informações de uma base de dados.

⚠ PODCAST

REFERÊNCIAS

PHP. **Manual do PHP.** *In*: PHP. Publicado em: 5 ago. 2020.

EXPLORE+

Para saber mais sobre os assuntos explorados neste tema:

Pesquise:

PHP. Mysql. *In*: Manual do PHP.

PHP. PostgreSQL. *In*: Manual do PHP.

PHP. Data Objects. *In*: Manual do PHP.

O PHP possui algumas funções específicas para vários SGBDs, através das quais é possível realizar a conexão com o banco de dados e executar instruções SQL. Além disso, há funções próprias para a recuperação de dados, como as que transformam o conjunto de resultados em *arrays* numéricos ou associativos. O Manual do PHP contém a listagem dessas funções, assim como exemplos de sua utilização. Cabe ressaltar que, no lugar da utilização dessas funções específicas, deve-se dar preferência ao uso da Classe PDO como camada abstração para esse fim. Neste mesmo Manual, não deixe de ler também maiores sobre o método *fetch* e equivalentes.

CONTEUDISTA

Alexandre de Oliveira Paixão

🔗 CURRÍCULO LATTES