
INSTITUTO
VIANNA JUNIOR

Técnicas Avançadas em Desenvolvimento de Software

Prof. Ms. Gildo de Almeida Leonel

capítulo 8

Integração de PHP e MySQL

Neste capítulo, apresentaremos como se dá a utilização do PHP em conjunto com o Sistema Gerenciador de Banco de Dados (SGBD), mais especificamente com o MySQL. Serão abordados os procedimentos necessários para permitir o gerenciamento e a consulta das informações armazenadas no exemplo de sistema de compras online trabalhado em todos os capítulos.

Objetivos de aprendizagem

» Realizar os procedimentos necessários para a estruturação das bases de dados utilizadas em sistemas Web.

» Gerenciar as informações armazenadas nas bases de dados.

» Realizar e tratar os resultados de consultas realizadas sobre bases de dados MySQL.

» IMPORTANTE

O MySQL também possibilita a aquisição de licenças específicas para uso comercial do SGBD.

» Introdução

o PHP apresenta compatibilidade com um conjunto bastante significativo de Sistemas Gerenciadores de Bancos de Dados (SGBD). Dentre os sistemas relacionais suportados, destaca-se o **MySQL**, distribuído sob licença GPL (*GNU General Public License*), e de ampla utilização e compatibilidade com distintas linguagem de programação e plataformas de operação.

» CURIOSIDADE

O MySQL foi disponibilizado, em sua primeira versão, em 1996, apesar de já ter ocorrido um lançamento interno do sistema em maio de 1995. É considerado o segundo SGBD mais utilizado no mundo, de acordo com dados de outubro de 2013, da pesquisa da DB-Engines.

PHP – Banco de Dados

- Banco de Dados
- Necessário para armazenar dados, e fazer aplicações web dinâmicas.

- Mysql
- Postgresql
- MSSQL Server
- Firebird
- Oracle

Para atuar em conjunto com o SGBD MySQL, a linguagem PHP apresenta um amplo conjunto de funções. Apresentaremos, no decorrer deste capítulo, as fundamentais para habilitá-lo a implementar um sistema Web que possibilite o gerenciamento e a consulta de informações armazenadas em um banco de dados.

A fim de que seja possível realizar testes e implementar os exemplos que serão apresentados, é fundamental que você tenha o MySQL instalado e configurado em um servidor ou em seu computador pessoal.

Inicialmente, abordaremos os aspectos fundamentais para que se estabeleça a conexão com um SGBD MySQL. Na sequência, apresentaremos os procedimentos para o gerenciamento e seleção de bases de dados. A interação com uma base de dados é apresentada no tópico seguinte, em que será explorada a realização de consultas SQL. Por fim, chegará a vez das funções e dos procedimentos para o tratamento das consultas realizadas e das possibilidades de recursos para utilização do MySQL com orientação a objetos.

PHP – Banco de Dados

- Existem 5 fases na utilização de um banco de dados
  - 1 - Fazer a conexão, abrir um link de comunicação entre a aplicação e o SGDB
  - 2 - Abrir uma transação
  - 3 - Executar Comandos SQL, Selects, Inserts, Updates, Deletes
  - 4 - Finalizar a transação (commit, Rollback)
  - 5 - Fechar a Conexão

- Em alguns SGDB as fases de Abrir transação e finalizar podem ser omitidas

Conexão com o banco de dados

Após o MySQL estar devidamente configurado no servidor, o primeiro procedimento necessário a ser implementado em um script que realiza a interação com o banco de dados é estabelecer uma conexão com o mesmo. Para isso, deve-se fazer uso de uma função específica do PHP, denominada **`mysql_connect()`**.

A sintaxe de `mysql_connect()` é:

```
resource mysql_connect ([ string $server [, string
$username [, string $password [, bool $new_link [,
int $client_flags ]]]]])
```

```
resource mysql_connect ([ string $server [, string
$username [, string $password [, bool $new_link [,
int $client_flags ]]]]])
```

Para estabelecer uma conexão com o servidor, deve-se especificar:

- `$server`: o endereço do servidor que oferece o serviço de banco de dados. Pode ser informada também a porta específica utilizada adicionando ":" e o número ao final da informação do servidor.

- `$user`: o usuário que será conectado ao MySQL.

- `$password`: a senha do respectivo usuário.

- `$new_link`: parâmetro usado apenas nos casos em que se desejar estabelecer uma nova conexão em um mesmo script, utilizando os mesmos dados de servidor e usuário.

- `$client_flags`: critério que possibilita o uso de uma série de constantes na conexão a ser estabelecida.

Como resultado da execução da função `mysql_connect()`, há o retorno de um identificador dessa conexão em caso de sucesso no procedimento. Caso ocorra alguma falha no estabelecimento da conexão, teremos o retorno do valor `false`.

Veja a seguir o exemplo do script que apresenta o código utilizado para o estabelecimento de conexão com um servidor localhost (que também pode ser identificado pelo IP 127.0.0.1) do usuário "administrador" e senha "minhasenha".

» EXEMPLO

```
<?php
  $con = mysql_connect('localhost', 'administrador', 'minhasenha');
  if (!$con)
    exit('Não foi possível conectar: ' . mysql_error());
  // insira aqui o restante do código para utilização da conexão...
  mysql_close($con); // encerramento da conexão
?>
```

Observe, no exemplo, que é feito um teste após a tentativa de estabelecimento de conexão, para verificar se o procedimento foi realizado com sucesso. Caso tenha ocorrido algum problema, `$con` receberá o valor `false` e será utilizada a função `mysql_error()` para apresentar informações complementares sobre a falha no estabelecimento, encerrando a execução do script com a utilização da construção de linguagem `exit()`.

» CURIOSIDADE

O procedimento de fazer uso da última conexão estabelecida em caso de supressão do identificador é geralmente adotado nas demais funções relacionadas ao MySQL que não apresentam esse parâmetro como obrigatório.

» IMPORTANTE

O PHP disponibiliza também a função `mysql_pconnect()` para o estabelecimento de conexão com o MySQL, diferenciando-se da `mysql_connect()` por estabelecer uma conexão persistente com o banco, não sendo encerrada ao final da execução do script.

Após a inclusão do código adicional que fará uso da conexão recém-estabelecida, deve-se inserir uma chamada à função `mysql_close()`, a qual recebe como parâmetro o identificador da conexão para proceder com seu encerramento. Caso não seja passado um identificador como parâmetro, a última conexão estabelecida será encerrada. Se não for feito uso dessa função, a conexão será encerrada tão logo a execução do script terminar.

Observe que o procedimento de estabelecimento de conexão apresentado no exemplo da seção "Conexão com o banco de dados" foi disponibilizado dentro do arquivo conectamysql.php, o qual foi incluído no script na primeira linha. Esse procedimento é recomendado para evitar que informações referentes ao usuário e à senha sejam disponibilizadas em todos os scripts que interajam com o banco de dados, gerando problemas de segurança e dificuldade na manutenção do sistema em caso de alterações nas configurações do servidor MySQL.

Seleção de bases de dados

Uma vez estabelecida uma conexão com o MySQL e partindo-se de uma base de dados já existente, deve-se passar à seleção dessa base de dados, para que seja possível o gerenciamento das informações nela armazenadas. É a função `mysql_select_db()` que possibilita essa seleção. Para realizar a seleção de uma base de dados, é necessário especificar o nome da base que se deseja selecionar e, opcionalmente, o identificador de conexão adotado. Em caso de sucesso na seleção, aparece o retorno do valor `true`, e, para casos de insucesso, o retorno `false`.

A sintaxe da função `mysql_select_db()` é:

```
bool mysql_select_db (string $database_name [,
resource $ link_identifier ])
```

A criação de uma base de dados pode ser feita com a função especifica do PHP `mysql_create_db()`. No entanto, o uso dessa função tem se tornado obsoleto, sendo preferencial o uso da consulta, em SQL, CREATE DATABASE, em conjunto com a função `mysql_query()`. Veremos os aspectos relacionados ao uso desta função de execução de comandos SQL no decorrer deste capítulo.

Veja o exemplo que apresenta um trecho de código do procedimento de seleção para interação de uma base de dados já existente denominada "sitedecompras".

» EXEMPLO

```
<?php
  include 'conectamysql.php'; // arquivo que estabelece a conexão
  $db_selecionado = mysql_select_db('sitedecompras', $con);
  if(!$db_selecionado)
    exit('Não foi possível selecionar a base de dados: '. mysql_error());
  // insira aqui seu código para uso da base de dados...
  mysql_close($con); // encerramento da conexão
?>
```

No código apresentado no exemplo anterior, foi estabelecida uma conexão com o banco de dados do servidor e, em seguida, realizada a seleção da base de dados "sitedecompras". Caso tenha ocorrido algum problema na seleção da base, é apresentada uma mensagem ao usuário especificando o problema.

PHP 7.0 – Banco de Dados

- No Mysql
  - trabalhamos sem as fases de transação
- A partir da versão 7 os métodos de conexão foram alterados:

```
$db = new mysqli('localhost', 'user', 'pass', 'database');
      if($db->connect_errno > 0){
 die('Unable to connect to database [' . $db->connect_error . ']'); }
```

Realização de consultas SQL

A partir do estabelecimento de conexão com o banco de dados e da seleção da base de dados adotada, é possível passar à interação com esta última. Esse processo se dará principalmente por meio da realização de consultas SQL, possibilitada pelo uso da função `mysql_query()`.

A sintaxe para `mysql_query()` é:

```
resource mysql_query (string $query [, resource
$link_identifier ])
```

Ao fazer uso dessa função, deve-se passar como primeiro parâmetro a consulta SQL que se deseja realizar (`$query`) e, opcionalmente, o identificador de uma conexão já estabelecida com o banco de dados. O valor retornado pela execução com sucesso da função dependerá do tipo de comando SQL utilizado: SELECT, SHOW, DESCRIBE ou EXPLAIN retornam um `resource` que poderá ser tratado no decorrer do script; já as demais consultas SQL (como UPDATE ou DELETE) retornam `true`. Para qualquer SQL realizado, o valor `false` será retornado sempre que ocorrerem falhas.

» EXEMPLO

```
<?php
  include 'conectamysql.php'; // arquivo que estabelece a conexão
  $basecriada=mysql_query('CREATE DATABASE sitedecompras', $con);
  if(!$basecriada)
    exit ('Erro ao criar a base de dados: '. mysql_error());
  mysql_close($con);
?>
```

Há casos em que o identificador de conexão `$con` para a função `mysql_query()` pode ser omitido, sendo adotada automaticamente a última conexão estabelecida com o servidor (o próprio `$con`). No entanto, esse tipo de procedimento pode gerar resultado inesperado caso outra conexão distinta seja estabelecida previamente à realização da consulta SQL.

PHP 7.0 – Banco de Dados

- Executar uma ação no banco de dados

```
$sql = 'SQL
        SELECT *
        FROM `users`
       WHERE `live` = 1‘;
if(!$result = $db->query($sql)){
    die('There was an error running the query [' . $db->error . ']');
}
```

PHP – Banco de Dados

- quando usado com selects ele apresenta em seu resultado o resultado da consulta, nos outros casos trás apenas True
- para acessar o resultado de uma consulta usamos
  - mysql_fetch_assoc(Resultado da pesquisa)
  - para cada interação vai retornar uma linha do resultado
- o resultado será um array associativo no qual cada elemento será uma coluna na tabela, conforme especificado no sql

```
while($row = $result->fetch_assoc()){
    echo $row['username'] . '<br />';
}
```

PHP – Banco de Dados

- Quantidade de linhas no resultado:

```php
<?php
    echo 'Total results: ' . $result->num_rows;
?>
```

PHP – Banco de Dados

- Após utilizar a conexão com o banco é necessário fecharmos, existem dois caminhos:

  1 - podemos esperar a execução do código e o término do script (mais usado 😕)

  2 - podemos usar a função

```
$db->close();
```

Trabalhando com resultados - Mysql

Exemplo de resultados POO - Mysqli fetch_row

mysqli_result::fetch_row -- mysqli_fetch_row

- Obtém uma linha do resultado como uma matriz numerada

```php
<?php
$query = "SELECT Name, CountryCode FROM City ORDER by ID";
if ($result = $mysqli->query($query)) {
    while ($row = $result->fetch_row()) {
        printf ("%s (%s)\n", $row[0], $row[1]);
    }
    $result->close();
}
$mysqli->close();
?>
```

Exemplo de resultados POO - Mysqli fetch_assoc

mysqli_result::fetch_assoc -- mysqli_fetch_assoc

Obtem uma linha do conjunto de resultados como uma matriz associativa

```php
<?php
$query = "SELECT Name, CountryCode FROM City ORDER by ID";

if ($result = $mysqli->query($query)) {

    /* Array associativo */
    while ($row = $result->fetch_assoc()) {
        printf ("%s (%s)\n", $row["Name"], $row["CountryCode"]);
    }
    $result->close();
}

$mysqli->close();
?>
```

Exemplo de resultados POO – Mysqli fetch_array

mysqli_fetch_array -- result->fetch_array

- Obtem uma linha do resultado como uma matriz associativa, numérica, ou ambas

```php
<?php
$query = "SELECT Name, CountryCode FROM City ORDER by ID LIMIT 3";
$result = $mysqli->query($query);

/* Array numérico */
$row = $result->fetch_array(MYSQLI_NUM);
printf ("%s (%s)\n", $row[0], $row[1]);

/* Array associativo */
$row = $result->fetch_array(MYSQLI_ASSOC);
printf ("%s (%s)\n", $row["Name"], $row["CountryCode"]);

/* Array numérico e associativo */
$row = $result->fetch_array(MYSQLI_BOTH);
printf ("%s (%s)\n", $row[0], $row["CountryCode"]);

$result->close();
$mysqli->close();
?>
```

Exemplo de resultados POO - Mysqli fetch_object

mysqli_fetch_object -- result->fetch_object

Retorna a linha atual do conjunto de resultados como um objeto

```php
<?php
$query = "SELECT Name, CountryCode FROM City ORDER by ID";

if ($result = $mysqli->query($query)) {

    while ($obj = $result->fetch_object()) {
        printf ("%s (%s)\n", $obj->Name, $obj->CountryCode);
    }
    $result->close();
}
$mysqli->close();
?>
```

Exemplo de resultados POO - Mysqli Mapeando objetos automaticamente

```php
<?php
class Aluno {
    public $aluno;
    public $nome;
    public $data_nasc;

    public function info()
    {
        return '<hr>#'.$this->aluno.': '.$this->nome.' '.$this->data_nasc;
    }
}


$mysqli->select_db('aluno');

$sql ="SELECT * FROM aluno";

if($result = $mysqli->query($sql)) {
    while ($obj =  $result->fetch_object('Aluno')) {
        echo $obj->info();
    }
    $result->close();
} else {
    echo 'Falha na seleção<br />';
}
$mysqli->close();

?>
```

Exercício – PHP Mysql

Criar um Mural de Recados.

Os dados a serem solicitados são:
Nome:
E-mail:
Cidade:
Recado:

Todos os dados deverão ficar salvos em um banco de dados a ser criado.

Haverá uma página para o usuário fazer a inserção do recado e na mesma página fazer a leitura de todos os recados.

Também deverão ser implementadas as opções para alteração dos recados e exclusão dos recados.

O que é SQL Injection e como evitar

SQL Injection

Evitar injeção de SQL é uma prática de segurança básica

Suponhamos que, em parte do nosso código PHP, haja a seguinte consulta no MySQL:

```
SELECT * FROM usuarios WHERE nome = ‘pedro' AND sobrenome = ‘pedreira'
```

Vamos considerar, ainda, que o nome e o sobrenome venham a partir de variáveis obtidas pelos métodos GET ou POST do PHP.

SQL Injection

O usuário preencheu os campos do formulário aonde digita seu nome e sobrenome da seguinte forma:

NOME: `pedro'; DROP TABLE usuarios ;`

SOBRENOME: `pedreir`

SQL Injection - Mysqli

No modelo de código:

```php
<?php

//Obtém as variáveis pelo formulário, através do método POST.
$nome = $_POST['nome'];
$sobrenome = $_POST['sobrenome'];

//Abre uma conexão com o MySQL, através do mysqli.
$mysqli = new mysqli('localhost', 'root', 'senha', 'banco');

//Query.
$sql = "SELECT * FROM usuarios WHERE nome = '$nome' AND sobrenome = '$sobrenome'";

?>
```

A última linha ficaria assim:

```
$sql = "SELECT * FROM usuarios WHERE nome = 'pedro';  DROP TABLE usuarios
;
      --' AND sobrenome = 'pedreira';"
```

SQL Injection

User name	Password	SQL Query
tom	tom	SELECT * FROM users WHERE name='tom' and password='tom'
tom	' or '1'='1	SELECT * FROM users WHERE name='tom' and password='' or '1'='1'
tom	' or 1='1	SELECT * FROM users WHERE name='tom' and password='' or 1='1'
tom	1' or 1=1 -- -	SELECT * FROM users WHERE name='tom' and password='' or 1=1-- -'
' or '1'='1	' or '1'='1	SELECT * FROM users WHERE name='' or '1'='1' and password='' or '1'='1'
' or ' 1=1	' or ' 1=1	SELECT * FROM users WHERE name='' or ' 1=1' and password='' or ' 1=1'
1' or 1=1 -- -	blah	SELECT * FROM users WHERE name='1' or 1=1 -- -' and password='blah'

SQL Injection

- **Função 'addslashes' do PHP:**

Ao usar a função addslashes($string), o PHP adicionará uma barra invertida antes das aspas na query.

Esse é um método que ajuda a proteger de injections, mas ele não é totalmente eficiente pois existem caracteres que não são reconhecidos e passam direto pela função addslashes($string).

- **Função 'mysql_*real*_escape_string' do PHP**

A função mysql_real_escape_string é um pouco mais segura. A função mysql_real_escape_string() se difere da addslashes() por saber o charset da conexão.

A proteção é feita para o charset que o servidor está esperando. Mas alguns caracteres podem também burlar a função

SQL Injection - Como se proteger?

- **Usando MySQLi:**

```php
$stmt = $dbConnection->prepare('SELECT * FROM employees WHERE name = ?');
$stmt->bind_param('s', $name);

$stmt->execute();

$result = $stmt->get_result();
while ($row = $result->fetch_assoc()) {
    ...
}
```

mysqli_prepare

- Primeiro nós **PREPARAMOS** uma consulta com um local para receber um valor variável

Estilo orientado à objeto

```
mysqli::prepare ( string $query )
```

Estilo procedural

```
mysqli_prepare ( mysqli $link , string $query )
```

```
$dbConnection->prepare('SELECT * FROM employees WHERE name = ?');
```

Na posição onde serão inseridos os dados das variáveis colocamos o sinal de ? Ou \g

mysqli_bind_param

- Depois nós dizemos que o local reservado receberá um conteúdo específico.

O método passa variáveis para marcadores de parâmetros no comando SQL que foi passado para mysqli_prepare().

Modo orientado a objeto (método):

```
bool mysqli_stmt::bind_param ( string $types , mixed &$var1 [, mixed &$... ] )
```

Modo procedural:

```
bool mysqli_stmt_bind_param ( mysqli_stmt $stmt , string $types , mixed &$var1
 [, mixed &$... ] )
```

mysqli_bind_param

```
bool mysqli_stmt::bind_param ( string $types , mixed &$var1 [, mixed &$... ] )
```

stmt

Somente no estilo procedural: Um recurso statement retornado por mysqli_stmt_init().

types

A string que contém um ou mais caracteres que especifica os tipos para as correspondente variáveis passadas:

Caracteres de especificação de tipo

Caractere	Descrição
i	corresponde a uma variável de tipo inteiro
d	corresponde a uma variável de tipo double
s	corresponde a uma variável de tipo string
b	corresponde a uma variável que contém dados para um blob e enviará em pacotes

var1

O número de variáveis e tamanho da string **types** precisa combinar com os parâmetros no comando.

mysqli_stmt_execute

- Por último executamos a query preparada e trabalhamos com o resultado

```php
$stmt->execute();

$result = $stmt->get_result();
while ($row = $result->fetch_assoc()) {
    ...
}
```

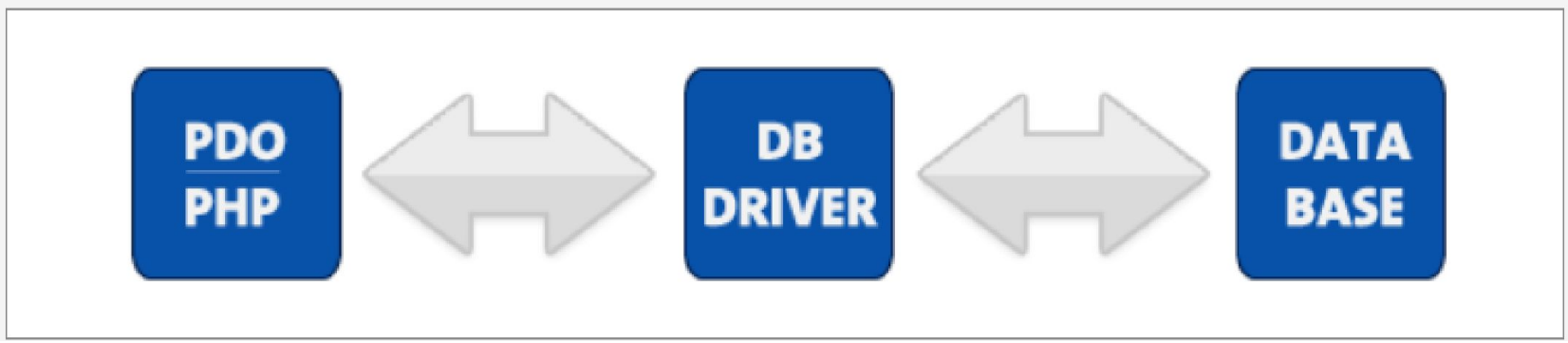
Exercício – PHP Mysql

Mural de Recados.

Modifique o mural de recados para utilizar prepared statements

PDO:

- PDO – PHP Data Objects – é uma camada de acesso a banco de dados fornecendo um método uniforme de acesso a vários bancos de dados desde o PHP 5.1.
- Independentemente de qual banco de dados você está usando, você usa as mesmas funções para realizar consultas e buscar dados.



Uma maneira rápida de descobrir quais drivers você tem:

- print_r(PDO::getAvailableDrivers());
  - Phpinfo()

Conexão

```
                    Database Type
$DBH = new PDO("mysql:host=$host;dbname=$dbname", $user, $pass);
Database Handle          Database Specific Connection String
```

MySQL

```
$dsn = 'mysql:host=localhost;port=3366;dbname=custorders';
$username = 'root';
$password = 'aluno';
$db = new PDO($dsn, $username, $password);
```

Ou:

```
$db = new PDO('mysql:host=localhost;dbname=pdodb', 'root', 'aluno');
```

Execução de instruções SQL com PDO

- SQL Select statement:
  - PDO::query — Executa uma instrução SQL Select, retornando um conjunto de resultados como um objeto **PDOStatement**
- SQL Insert, Delete e Update:
  - PDO::exec — Executa uma instrução SQL e **retornar o número de linhas afetadas**
- Parameterized (Prepared) statement:
  - PDO::prepare — Prepara um comando para execução e retorna um objeto **PDOstatement object**

Seleção de dados com uma consulta PDO

- PDO::query — Executa uma instrução SQL Select, retornando um conjunto de resultados como um objeto PDOStatement

- Um recurso interessante do PDO :: query () é que ele permite você iterar sobre o conjunto de linhas retornado por uma instrução SELECT executada com êxito.

Usando o método PDOStatement fetch ()

- A query retorna um objeto PDOStatement, podemos usar método fech() PDOStatement para recuperar uma linha do conjunto de resultados.

- **Cada linha é representada como uma matriz indexada por tanto nome da coluna e número.**

- O valor de retorno desta função em caso de sucesso depende do tipo de busca. Em todos os casos, é retornado FALSE em caso de falha.

A classe PDOStatement

Representa uma declaração preparada e, após a instrução é executada com um conjunto de resultados associados

- **PDOStatement::fetch** — Obtém a próxima linha de um conjunto de resultados
- **PDOStatement::fetchAll** — Retorna um array contendo todas as linhas do conjunto de resultados
- **PDOStatement::rowCount** — Retorna o número de linhas afetadas pela última instrução SQL

Usando while com fetch () para iterar sobre o conjunto de resultados

```php
<?php
$db = new PDO('mysql:host=localhost;dbname=pdodb', 'root', '');
$query = "select * from customers";
$customers = $db->query($query);

echo $customers->rowCount() . " rows.";
echo "<table border=1><tr>" . "<th>CID</th>" . "<th>Nome</th>" .
    "<th>Cidade</th>" . "<th>Avaliação</th></tr>";

while ($customer = $customers->fetch()) {
    $cid = $customer["cid"];
    $Cname = $customer["cname"];
    $City = $customer["city"];
    $Rating = $customer["rating"];
    echo "<tr><td>$cid</td>" . "<td>$Cname</td>" . "<td>$City</td>" .
        "<td>$Rating</td></tr>";
}

echo "</table>";
?>
```

PDOStatement::rowCount()

- PDOStatement::rowCount() retorna o número de linhas afetadas pela última DELETE, INSERT ou UPDATE executada pelo objeto PDOStatement correspondente.

- Se a última instrução SQL executado pelo PDOStatement associada foi uma instrução SELECT, alguns bancos de dados podem informar o número de linhas retornados por essa declaração.

- No entanto, esse comportamento não é garantida para todos os bancos de dados e não deve ser invocado para todas aplicações.

Exemplo de Uso do método de consulta e iterar sobre o conjunto de resultados usando a instrução foreach

```php
<?php
$db = new PDO('mysql:host=localhost;dbname=pdodb', 'root', '');
$query = "select * from customers";
$customers = $db->query($query);   //$customers resultdo do PDOStatement

echo "<table border=1><tr>" . "<th>CID</th>" . "<th>Nome</th>" .
    "<th>Cidade</th>" . "<th>Avaliação</th></tr>";

foreach ($customers as $customer) {
    $cid = $customer["cid"];
    $Cname = $customer["cname"];
    $City = $customer["city"];
    $Rating = $customer["rating"];
    echo "<tr><td>$cid</td>" . "<td>$Cname</td>" . "<td>$City</td>" .
        "<td>$Rating</td></tr>";
}

echo "</table>";
?>
```

Exemplo fetchAll()

```php
<?php
$db = new PDO('mysql:host=localhost;dbname=pdodb', 'root', '');
$query = "select * from customers";
$customers = $db->query($query);
$AllCustomers = $customers->fetchAll();

echo "<table border=1><tr>" . "<th>CID</th>" . "<th>Nome</th>" .
    "<th>Cidade</th>" . "<th>Avaliação</th></tr>";

foreach ($AllCustomers as $customer) {
    $cid = $customer["cid"];
    $Cname = $customer[1];
    $City = $customer[2];
    $Rating = $customer[3];
    echo "<tr><td>$cid</td>" . "<td>$Cname</td>" . "<td>$City</td>" .
        "<td>$Rating</td></tr>";
}

echo "</table>";

?>
```

Recuperar registro com base em uma chave:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>PDO4</title>
</head>
<body>
<form name="getData" method="post" action="pdo4.php">
    Entre o CID: <input type="text" name="CID" value=""/><br><br>
    <input type="submit" value="Buscar valor" name="btnSubmit" />
</form>
</body>

</html>
```

```php
<?php
$db = new PDO('mysql:host=localhost;dbname=pdodb', 'root', '');
$cid = $_POST['CID'];
$query = "SELECT * FROM customers WHERE CID= '$cid'";
$customers = $db->query($query);
if ($customers->rowCount() == 0) echo "Registro não existe!";
else {
    foreach ($customers as $customer) {
        $cid = $customer['cid'];
        $Cname = $customer['cname'];
        $City = $customer['city'];
        $Rating = $customer['rating'];
        echo "<p>CID: $cid </P>";
        echo "<p>Nome: $Cname </P>";
        echo "<p>Cidade: $City </P>";
        echo "<p>Avaliação: $Rating </P>";
    }
} ?>
```

Nota: o loop foreach é usado mesmo que a instrução retorne no máximo um registro

Usando fetch()

```php
<?php
$db = new PDO('mysql:host=localhost;dbname=pdodb', 'root', '');
$cid = $_POST['CID'];
$query = "SELECT * FROM customers WHERE CID= '$cid'";
$customers = $db->query($query);

if ($customers->rowCount() == 0) echo "Registro não existe!";
else {
    $customer = $customers->fetch();
    $cid = $customer['cid'];
    $Cname = $customer['cname'];
    $City = $customer['city'];
    $Rating = $customer['rating'];
    echo "<p>CID: $cid </P>";
    echo "<p>Nome: $Cname </P>";
    echo "<p>Cidade: $City </P>";
    echo "<p>Avaliação: $Rating </P>";
}

?>
```

Usando uma página PHP para entrada e saída

```php
<?php
if (!empty($_POST)) {
    $db = new PDO('mysql:host=localhost;dbname=pdodb' , 'root', '');
    $cid = $_POST['CID'];
    $query = "SELECT * FROM customers WHERE CID= ' $cid'";
    $customers = $db->query($query);
    if ($customers->rowCount() == 0) echo "Registro não existe!";
    else {
        foreach ($customers as $customer) {
            $cid = $customer['cid'];
            $Cname = $customer['cname'];
            $City = $customer['city'];
            $Rating = $customer['rating'];
            echo "<p>CID: $cid </P>";
            echo "<p>Nome: $Cname </P>";
            echo "<p>Cidade: $City </P>";
            echo "<p>Avaliação: $Rating </P>";
        }
    }
}
?>
<form name="getData" method="post" action="pdo6.php">
    CID: <input type="text" name="CID" value=""/><br><br>
    <input type="submit" value="Buscar" name="btnSubmit"/>
</form>
```

setFetchModemethod

Antes de chamar o fetch, é necessário dizer a PDO como você gostaria que os dados a fossem obtidos. Temos as seguintes opções:

- PDO::FETCH_ASSOC: retorna um array indexado pelo nome da coluna
- PDO::FETCH_BOTH (**default**): retorna uma matriz indexada por tanto nome da coluna e número
- PDO::FETCH_CLASS: Atribui os valores de suas colunas para propriedades da classe nomeada. Ele vai criar as propriedades se não existirem propriedades correspondentes
- PDO::FETCH_INTO: Atualiza uma instância existente da classe chamada
- PDO::FETCH_NUM: retorna uma matriz indexada pelo número da coluna
- PDO::FETCH_OBJ: retorna um objeto anônimo com nomes de propriedade que correspondem aos nomes de coluna

Especificando um modo de busca, por exemplo, PDO::FETCH_ASSOC

```php
<?php
$db = new PDO('mysql:host=localhost;dbname=pdodb', 'root', '');
$query = "select * from customers";
$customers = $db->query($query);
$customers->setFetchMode(PDO::FETCH_ASSOC);
echo "<table border=1><tr>" . "<th>CID</th>" . "<th>Nome</th>" .
    "<th>Cidade</th>" . "<th>Avaliação</th></tr>";
while ($customer = $customers->fetch()) {
    $cid = $customer["cid"];  // case sensitive
    $Cname = $customer["cname"];
    $City = $customer["city"];
    $Rating = $customer["rating"];
    echo "<tr><td>$cid</td>" . "<td>$Cname</td>" . "<td>$City</td>" .
        "<td>$Rating</td></tr>";
}
echo "</table>";
?>
```

Use PDO :: exec para executar SQL Insert/Delete/Update Statements

- Executa um Statement SQL Insert/Delete/Update e retorna o número de linhas afetadas.

Formulário de entrada de dados para novos clientes

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>PDO8</title>
</head>
<body>
<form name="newCustomerForm" action="pdo8.php" method="POST">
    CID: <input type="text" name="CID" value=""/><br><br>
    Nome: <input type="text" name="Cname" value=""/><br><br>
    Cidade: <input type="text" name="City" value=""/><br><br>
    Avaliação: <input type="text" name="Rating" value=""/><br><br>
    <input type="submit" value="Adicionar" name="btnSubmit"/>
</form>
</body>

</html>
```

Código para adicionar um novo cliente

```php
<?php
$db = new PDO('mysql:host=localhost;dbname=pdodb' , 'root', '');
$cid = $_POST["CID"];
$cname = $_POST["Cname"];
$city = $_POST["City"];
$rating = $_POST["Rating"];
$queryINS = "insert into customers value(' $cid','$cname','$city','$rating')";

if ($db->exec($queryINS) == 1)
    echo "Registro adicionado";
else
    echo "Registro não adicionado";
?>
```

Usando um programa PHP para adicionar um novo cliente

```php
<?php
if (!empty($_POST)) {

    $db = new PDO('mysql:host=localhost;dbname=pdodb', 'root', '');
    $cid = $_POST["CID"];
    $cname = $_POST["Cname"];
    $city = $_POST["City"];
    $rating = $_POST["Rating"];
    $queryINS = "insert into customers value('$cid','$cname','$city','$rating')";
    if ($db->exec($queryINS) == 1)
        echo "Registro Adicionado";
    else
        echo "Registro não adicionado";
}
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>PDO8</title>
</head>
<body>
<form name="newCustomerForm" action="pdo9.php" method="POST">
    CID: <input type="text" name="CID" value=""/><br><br>
    Nome: <input type="text" name="Cname" value=""/><br><br>
    Cidade: <input type="text" name="City" value=""/><br><br>
    Avaliação: <input type="text" name="Rating" value=""/><br><br>
    <input type="submit" value="Adicionar" name="btnSubmit"/>
</form>
</body>

</html>
```

Formulário de exclusão

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>PDO 10</title>
</head>
<body>
<form name="deleteForm" action="pdo10.php" method="POST">
    CID a ser excluido: <input type="text" name="CID" value=""/><br>
    <input type="submit" value="Deletar" name="btnSubmit"/>
</form>
</body>

</html>
```

Código para excluir um cliente

```php
<?php

$db = new PDO('mysql:host=localhost;dbname=pdodb', 'root', '');
$cid = $_POST["CID"];
$queryDEL = "delete from customers where cid='$cid'";
if ($db->exec($queryDEL) == 1)
    echo "Exclusão bem sucedida ";
else
    echo " Exclusão não realizada";

?>
```

Exemplo: Atualizando campo de avaliação de um cliente

- Um programa em php para criar um formulário com uma caixa de listagem de clientes e três botões de opção para selecionar nova classificação.

- Um segundo programa de php é chamado para atualizar a classificação.

Criação do formulário

```php
<?php
$db = new PDO('mysql:host=localhost;dbname=pdodb' , 'root', '');
$query = "select * from customers";
$customers = $db->query($query);
echo "<form name='updateRating' method='post' action='pdo12.php'>" ;
echo "CID: <br>";
echo "<select name = 'cid'>";
foreach ($customers as $customer) {
    $cid = $customer["cid"];
    $cname = $customer["cname"];
    $rating = $customer["rating"];
    $cidnamerating = $cid . ": " . $cname . ": " . $rating;
    echo "<option value= $cid >$cidnamerating</option>";
}
echo "</select><br><br>";
echo "Nova avaliação:<br>";
echo "<input type='radio' name='newRating' value='A' checked='checked' /> A<br>" ;
echo "<input type='radio' name='newRating' value='B' /> B<br>" ;
echo "<input type='radio' name='newRating' value='C' /> C<br><br>" ;
echo "<input type='submit' value='Atualizar avaliação' name='btnSubmit'/>
</form>";
?>
```

PHP programa para atualizar a avaliação

```php
<?php
$db = new PDO('mysql:host=localhost;dbname=pdodb', 'root', '');
$cid = $_POST["cid"];
$newRating = $_POST["newRating"];
$queryUpd = "update customers set rating = '" . $newRating .
    "' where cid = '" . $cid . "'";
if ($db->exec($queryUpd) == 1)
    echo "Atualização bem sucedida";
else
    echo "Atualização não realizada";
?>
```

PDO - Usando Prepared Statement

- A prepared statement ou parameterized statement é uma instrução SQL pré-compilada que pode ser executada várias vezes, enviando apenas os dados para o servidor.

- Ela é utilizada para executar a mesma instrução repetidamente com alta eficiência.

- Você usa uma declaração preparada por incluindo espaços reservados (placeholders) no seu SQL.

Benefícios do Uso de declarações preparadas

- A consulta apenas necessita de ser analisada (ou preparadas) uma vez, mas pode ser executada várias vezes com o mesmo ou diferentes parâmetros.

- Os parâmetros para instruções preparadas não precisa ter aspas; o driver lida com isso automaticamente. Portanto, nenhuma injeção SQL irá ocorrer.

- Exemplo:

```
$stmt = $db->prepare("SELECT * FROM Customers where CID = ?");
```

Usando PDO::prepare para criar uma declaração preparada e retornar um objeto PDOstatement

- PDO::prepare
- Exemplo :
  - $stmt=$db->prepare($queryINS);

Métodos de classe PDOStatement

- PDOStatement::bindParam — - Passa um parâmetro para o nome da variável especificada
- PDOStatement::bindValue — Passa um valor a um parâmetro
- PDOStatement::execute — - Executa uma declaração preparada. Retorna TRUE em caso de sucesso ou FALSE em caso de falhas.

Placeholders sem nome, ?

Exemplo :

```
$queryINS = $db->("INSERT INTO customers values (?, ?, ?, ?)”);
```

Usando marcadores Sem nome :

bindParam - Passa um parâmetro para o nome da variável especificada

```
$stmt->bindParam(1, $cid);
$stmt->bindParam(2, $cname);
$stmt->bindParam(3, $city);
$stmt->bindParam(4, $rating);
```

Execute(): Retorna TRUE em caso de sucesso .

```
if($stmt->execute())
    echo “Adicionado";
else
    echo “Não Adicionado";
}
```

prepared statement

```php
<?php
$db = new PDO('mysql:host=localhost;dbname=pdodb', 'root', '');
$queryINS = "insert into customers value(?,?,?,?)";
$cid = $_POST["CID"];
$cname = $_POST["Cname"];
$city = $_POST["City"];
$rating = $_POST["Rating"];
$stmt = $db->prepare($queryINS);
$stmt->bindParam(1, $cid);
$stmt->bindParam(2, $cname);
$stmt->bindParam(3, $city);
$stmt->bindParam(4, $rating);

if ($stmt->execute())
    echo "Adicionado";
else
    echo "Não adicionado";
?>
```

prepared statement

```php
<?php
$db = new PDO('mysql:host=localhost;dbname=pdodb', 'root', '');
$query = "SELECT * FROM customers WHERE CID= ?";
$stmt = $db->prepare($query);
$cid = $_POST["CID"];
$stmt->bindParam(1, $cid);
echo "<table border=1><tr>" . "<th>CID</th>" . "<th>Nome</th>" .
    "<th>Cidade</th>" . "<th>Avaliação</th></tr>";
if ($stmt->execute()) {
    while ($row = $stmt->fetch()) {
        $cid = $row["cid"];
        $Cname = $row["cname"];
        $City = $row["city"];
        $Rating = $row["rating"];
        echo "<tr><td>$cid</td>" . "<td>$Cname</td>" . "<td>$City</td>" .
            "<td>$Rating</td></tr>";
    }
}
echo "</table>";
?>
```

Se os dados são armazenados em uma matriz

```
$data = array($cid, $cname, $city, $rating);
$stmt=$db->prepare($queryINS);

if($stmt->execute($data))
    echo “Adicionado";
else
    echo “Não adicionado";
}
```

```php
<?php

$db = new PDO('mysql:host=localhost;dbname=pdodb', 'root', '');
$queryINS = "insert into customers value(?,?,?,?)";
$cid = $_POST["CID"];
$cname = $_POST["Cname"];
$city = $_POST["City"];
$rating = $_POST["Rating"];
$data = array($cid, $cname, $city, $rating);
$stmt = $db->prepare($queryINS);
if ($stmt->execute($data))
    echo "Adicionado";
else
    echo "Não adicionado";
?>
```

Placeholders nomeados

```
$queryINS = "INSERT INTO customers values (:cid,:cname,:city,:rating)";
```

```php
<?php

$db = new PDO('mysql:host=localhost;dbname=pdodb', 'root', '');
$queryINS = "INSERT INTO customers values (:cid,:cname,:city,:rating)";
$cid = $_POST["CID"];
$cname = $_POST["Cname"];
$city = $_POST["City"];
$rating = $_POST["Rating"];
$stmt = $db->prepare($queryINS);
$stmt->bindParam(':cid', $cid);
$stmt->bindParam(':cname', $cname);
$stmt->bindParam(':city', $city);
$stmt->bindParam(':rating', $rating);
if ($stmt->execute())
    echo "Adicionado";
else
    echo "Não adicionado";
?>
```

Mysqli X PDO

MySQLi:

Vantagens:

- API Orientada a objetos e procedural;
- Performace elevada;
- Sintaxe relativamente mais simples (e similar a antiga API mysql_*);

Desvantagens:

- Só funciona com bancos MySQL;
- Não possui parâmetros nomeados;
- Não possui prepared statements do lado cliente;

PDO:

Vantagens:

- Funciona com 12 drivers de bancos de dados;
- API Orientada a objetos;
- Possui parâmetros nomeados;
- Possui *prepared statements* do lado cliente

Desvantagens:

- Não tão veloz quanto MySQLi;
- Por padrão, ele **simula** prepared statements