

PHP E MYSQL

Olá,

Começaremos a Aula 06 dando continuidade à disciplina Desenvolvimento Voltado à Web I. Na aula anterior, algumas funções disponíveis no PHP também encontradas na documentação da linguagem foram apresentadas e exemplificadas.

Nesta aula, veremos o banco de dados MySQL e como realizar a conexão com o PHP para recuperar dados.

Leia atentamente esta aula e, se tiver alguma dúvida, use os recursos que estão na sua área do aluno.

⚡ Bons estudos!

Objetivos de aprendizagem

Ao término desta aula, vocês serão capazes de:

- utilizar um banco de dados de exemplo MySQL;
- conectar o PHP ao MySQL;
- executar comandos no MySQL pelo PHP.

Seções de estudo

- 1 – O que é o MySQL
- 2 – Conectando ao Banco de Dados
- 3 – CRUD
- 4 – Construindo uma API

1- O que é o MySQL

MySQL

O MySQL é um SGBD, publicado em 1995 pela empresa MySQL AB, adquirida pela Sun Microsystems, que foi incorporada pela Oracle. Esta última mantém uma versão gratuita que pode ser utilizada pela comunidade em geral.

Antes de iniciar a conexão do MySQL com o PHP, é preciso que tenha o MySQL devidamente instalado. Na disciplina de Desenvolvimento voltado à *Web I* foi demonstrada a instalação do Apache HTTP Server (Xampp), que instala além do PHP, o MySQL e o PHPMyAdmin. Caso você não possua o Xampp instalado, serão disponibilizados na plataforma vídeos e materiais que podem te ajudar.

Além disso, é importante ter um banco de dados criado para que seja possível fazer os testes, como veremos mais adiante. Nos materiais de aula serão disponibilizados também um banco de dados de exemplo.

PHPMyAdmin

O PHPMyAdmin é uma ferramenta gratuita escrita em PHP, para gerenciar o MySQL através da Web. Tem por características principais:

- Interface Web Intuitiva;
- Suporte para a maioria dos recursos do MySQL;
- Gerenciamento contas de usuários e privilégios;
- Gerenciamento de procedures e triggers;
- Exportar dados;
- Criar layout do banco de dados.

2- Conectando ao Banco de Dados

Para conectar o banco de dados MySQL ao PHP, podemos utilizar a extensão MySQLi e o PDO (PHP Data Object), ambos possuem vantagens, são elas:

- O PDO funciona com 12 sistemas de banco de dados diferentes;
- O MySQLi funciona apenas com bancos de dados MySQL;
- Ambos são orientados a objetos, mas o MySQLi oferece uma API de procedimentos;
- Ambos suportam Prepared Statements que protegem de SQL Injection.

Nos exemplos que serão apresentados nas subseções seguintes, utilizaremos a extensão MySQLi em sua versão orientada a objetos.

Subseção A – Abrindo a conexão

A classe MySQLi representa uma conexão entre o PHP e um banco de dados MySQL. Para realizar a conexão, criamos um novo objeto do tipo MySQLi com a palavra reservada `new`, passando por parâmetros o nome do servidor, o usuário, a senha, e também o nome do banco de dados a que se quer conectar. O objeto resultado dessa operação é a conexão com o banco de dados. Métodos e atributos desse objeto podem ser acessados para manipular essa classe.

No exemplo abaixo, após a conexão com o banco de dados, é verificado pelo atributo `connect_error` se foi possível ou não realizar a conexão, encerrando o programa em caso negativo.

```
<?php
$server = "localhost";
$user = "root";
$pass = ""; $mydb = "vendas";
$conn = new mysqli($server, $user,
$pass, $mydb);
if ($conn->connect_error) {
    die("Conexão Falhou: " . $conn-
>connect_error);
}
?>
```

Para fechar a conexão com o banco de dados, utilizamos o método `close()`, depois de efetuarmos todas as operações sobre o banco de dados.

```
<?php
$conn->close();
?>
```

Subseção B – Inserindo registros

O método `query` do objeto conexão executa uma query SQL no banco de dados. Uma query é uma consulta ao banco de dados. A resposta a uma consulta pode ser um conjunto de registros.

Na inserção a query não retorna registros, mas é possível saber se a query foi realmente executada.

Inserido um registro

```
<?php
$sql = "INSERT INTO vendedor
(NOMEVEND, SALARIO, CODSETOR) VALUES
('Felipe Perez', 500, 1)";
if ($conn->query($sql) === TRUE) {
    echo "Novo registro inserido";
}else{
    echo "Error: " . $sql . "<br>" .
$conn->error;
}
?>
```

Inserido vários registros

Para inserir mais de um registro ao mesmo tempo, devemos concatenar as várias queries, uma após a outra,

separadas por ponto e vírgula (;) e para que todas sejam executadas o método `multi_query()` é utilizado.

```
<?php
$sql = "INSERT INTO vendedor
(NOMEVEND, SALARIO, CODSETOR) VALUES
('Felipe Perez',500,1);";
$sql .= "INSERT INTO vendedor
(NOMEVEND, SALARIO, CODSETOR) VALUES
('Perez Felipe',250,1);";
if($conn->multi_query($sql) ===
TRUE) {
    echo "Novos registros
inseridos";
}
?>
```

Prepared Statements

O prepared statements é um recurso utilizado para executar as mesmas instruções SQL repetidamente com alta eficiência. Para que possa ser implementado deve seguir algumas etapas:

1. Preparar: Criar um modelo de instrução SQL, sem especificar os valores. Esse modelo é enviado ao banco de dados (prepare).
2. O banco de dados analisa, compila e executa otimização no modelo e armazena o resultado sem executá-lo.
3. Executar: Vincula os valores aos parâmetros e o banco de dados executa as instruções, podendo executar quantas vezes quiser (`bind_param` e `execute`).

É utilizado quando é necessário o envio de muitas instruções repetidas, o que proporciona diversas vantagens, dentre elas estão:

- Redução no tempo de análise, pois a preparação é feita apenas uma vez apesar da instrução poder ser executada várias vezes.
- Minimização do tráfego de banda para o servidor, pois a cada execução somente são enviados os parâmetros.
- Útil contra injeções SQL, os valores dos parâmetros são transmitidos posteriormente usando um protocolo diferente.

O método `prepare()`, do objeto conexão, é responsável pela criação do modelo. No exemplo abaixo está sendo preparada a inserção de registros na tabela `vendedor`, os valores que irão variar a cada inserção são representados pelo ponto de interrogação (?). Nesse caso, temos três pontos de interrogação (?, ?, ?), para as colunas `NOMEVEND`, `SALARIO` e `CODSETOR`.

Depois de preparar o modelo, informa-se ao PHP quais são as variáveis que serão vinculadas aos parâmetros dos prepared statements. É passado para através do método `bind_param()` uma string que representa os tipos de parâmetros que serão enviados (i - Integer, d - Double, s - String e b - Blob) na ordem em que os pontos de interrogação foram declarados no método `prepare`. Também na mesma ordem são definidas

as variáveis.

Com o modelo preparado, e os parâmetros definidos resta executar cada uma das inserções. No exemplo abaixo são inseridos três registros através do método `execute()`.

```
<?php
$stmt = $conn->prepare("INSERT INTO
vendedor (NOMEVEND, SALARIO, CODSETOR)
VALUES (?, ?, ?)");
$stmt->bind_param("sii", $nomeVend,
$salario, $codSetor);
$nomeVend = "Felipe";
$salario = 500;
$codSetor = 1;
$stmt->execute();

$nomeVend = "Felipe Pereira Perez";
$salario = 750;
$codSetor = 1;
$stmt->execute();

$nomeVend = "Perez Felipe";
$salario = 250;
$codSetor = 1;
$stmt->execute();

?>
```

Lendo registros

Além de inserir dados em nosso banco de dados, podemos recuperá-los. Ainda utilizamos o método `query()` no objeto da conexão para realizar o `SELECT`. No exemplo queremos selecionar todos os registros da tabela `vendedor`.

Executada a query, o banco de dados pode retornar 0 ou mais registros. Para isso, deve ser feita a verificação se no retorno da execução da query o atributo `num_rows` é maior que 0. Caso seja menor que 0 nada será executado nesse caso. Caso seja maior que 0, o PHP percorre todos os registros retornados com o `while`. A cada iteração um registro é colocado no array `$row`. O nome das colunas da tabela são as chaves para pegar o conteúdo de cada registro.

```
<?php
$sql = "SELECT * FROM vendedor";
$result = $conn->query($sql);
if($result->num_rows > 0) {
    while($row = $result->fetch_assoc())
    {
        echo "Nomevend: " . $row["NOMEVEND"] .
"<br>";
    }
}
?>
```

3- CRUD

Create, Read, Update e Delete, a partir das letras iniciais de cada uma dessas operações temos o acrônimo **CRUD**. Ele trata das quatro operações básicas de SQL, Insert, Select, Update e Delete, respectivamente.



Figura 1 – CRUD. Fonte: Acervo pessoal.

O paradigma CRUD é muito utilizado na construção de aplicações web, já que permite a manipulação completa do banco de dados. Por exemplo, para construir um dicionário online, o que cada operação do CRUD deveria fazer:

Create

Responsável pela inserção de novas palavras, sua definição e classe gramatical. Além disso, essa nova palavra terá uma identificação única que poderá ser usada para acessar esse recurso posteriormente. Criar a palavra Abacaxi.

INSERT INTO palavras VALUES ('Abacaxi', 'Planta originária da América tropical.', 'Substantivo de dois gêneros');

Read

Tem por função retornar as palavras que já foram inseridas no banco de dados, podendo ainda retornar um conjunto específico de acordo com alguma condição, seja por seu id ou pelo seu nome. Mostrar o significado da palavra Abacaxi.

SELECT significado FROM palavras WHERE nome='Abacaxi';

Update

Atualiza um ou mais atributos de uma ou um conjunto de palavras a partir de uma condição. Atualizar a classe gramatical da palavra Abacaxi.

UPDATE palavra SET classe='Adjetivo de dois gêneros' WHERE nome='Abacaxi';

Delete

Remove palavras do dicionário de acordo com uma condição. Delete a palavra Abacaxi.

DELETE FROM palavra WHERE nome='Abacaxi';

Para cada uma das ações deve-se retornar o que foi exigido ou se foi possível ou não realizar o comando.

4- Construindo uma API

Pires (2017) explica que uma API (Application Programming Interface) é uma coleção de rotinas e padrões, desenvolvidos e documentados que podem ser utilizados por outras aplicações, sem que estas precisem conhecer a implementação detalhadamente, “APIs permitem uma interoperabilidade entre aplicações”.

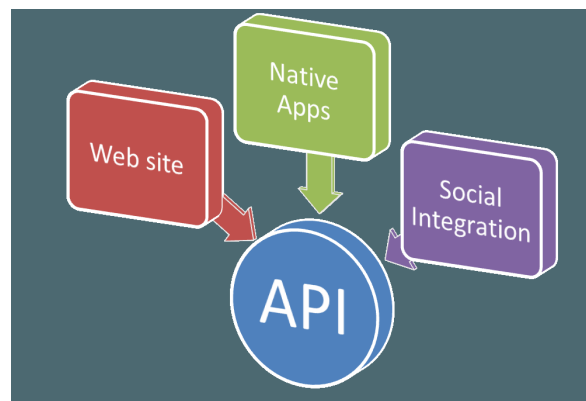


Figura 2 – API. Fonte: PIRES (2017).

O mais importante da API é que ela contenha padrões, devido a sua interoperabilidade, todas as aplicações devem saber como chamar a API, e o que esperar da resposta dela. A resposta da API pode ser dada em formatos diferentes, pode ser em XML, JSON, YAML. Enquanto o primeiro demanda de mais escrita, o segundo e terceiro é muito mais leve e parecido com a forma que escrevemos diariamente.

XML

```
<dicionario>
  <palavra>
    Abacaxi
  </palavra>
  <significado>
    Planta originária da América tropical
  </significado>
  <classegramatical>
    Substantivo de dois gêneros
  </classegramatical>
</dicionario>
```

JSON

```
{
  dicionário: {
    palavra: Abacaxi
    significado: Planta originária da América tropical
    classegramatical: Substantivo de dois gêneros
  }
}
```

YAML

```
dicionário:
  palavra: Abacaxi
  significado: Planta originária da América tropical
  classegramatical: Substantivo de dois gêneros
```

Algumas APIs permitem o retorno em várias representações, sendo passado também por parâmetro a representação desejada. Por exemplo: a API VIACEP, que possibilita a consulta gratuita de CEPs Brasileiros, seja pelo número do CEP ou pelo endereço, como pode ser visto na documentação em <https://viacep.com.br/>. Exemplo

de pesquisa com CEP em JSON ou XML: viacep.com.br/ws/01001000/json/ ou viacep.com.br/ws/01001000/xml/.

```
{
  cep: "01001-000",
  logradouro: "Praça da Sé",
  complemento: "lado ímpar",
  bairro: "Sé",
  localidade: "São Paulo",
  uf: "SP",
  unidade: "",
  ibge: "3550308",
  gia: "1004"
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xmlcep>
  <cep>01001-000</cep>
  <logradouro>Praça da Sé</logradouro>
  <complemento>lado ímpar</complemento>
  <bairro>Sé</bairro>
  <localidade>São Paulo</localidade>
  <uf>SP</uf>
  <unidade></unidade>
  <ibge>3550308</ibge>
  <gia>1004</gia>
</xmlcep>
```

Figura 3 – JSON x XML. Fonte: Adaptado de viacep.com.br/.

Vamos então começar a construção da nossa API. Nela utilizaremos apenas o retorno no formato JSON. Será utilizado um banco de dados de exemplo, que tem como estrutura o diagrama da Figura 6. Ele possui dados inseridos em todas as tabelas, possibilitando uma melhor visualização dos resultados.

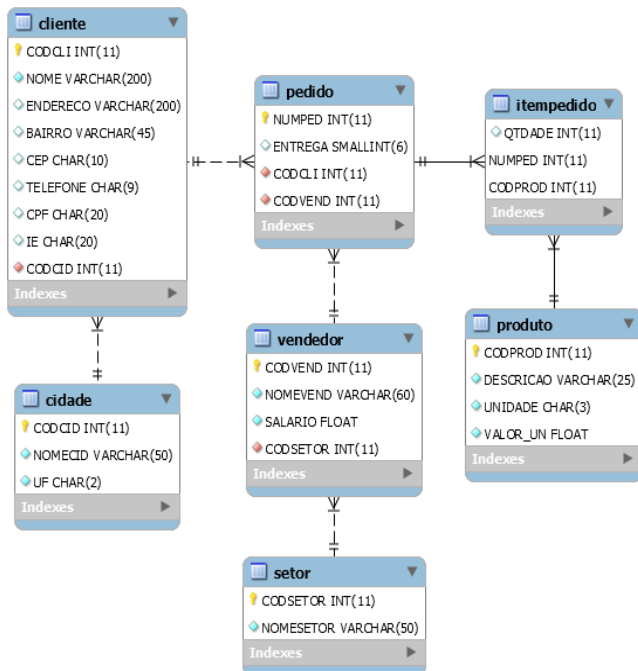


Figura 4 – Estrutura BD. Fonte: Acervo pessoal.

O arquivo principal para nossa api será o `api.php`. A primeira coisa a se fazer é fazer a conexão ao banco de dados. Como essa conexão pode ser feita repetidas vezes, pode ser interessante criar um arquivo separado para a conexão, que ao ser incluído no começo das páginas que manipulam o banco de dados evitará escrita de código sem necessidade.

```
<?php
$server = "localhost";
$user = "root";
$pass = ""; $mydb = "vendas";
$conn = new mysqli($server, $user,
$pass, $mydb);
if($conn->connect_error){
```

```
die("Conexão Falhou: ".$conn-
>connect_error);
}
```

Com a conexão devidamente aberta, as informações enviadas pelo usuário serão tratadas e devidamente retornadas. O usuário enviará por GET (na URL). Os parâmetros são: tabela, coluna e valor. Os dois últimos só serão enviados caso o usuário queira especificar que valores ele deseja na consulta.

A URL para chamar a API atenderá ao seguinte padrão: `http://localhost/api.php?tabela=cliente&coluna=nome&valor=Felipe`

Para a URL acima, a api responde o conteúdo da tabela cliente, que na coluna nome contenha o valor 'Felipe'.

No trecho abaixo, a api trata de armazenar nas variáveis `$tabela`, `$coluna` e `$valor` os respectivos valores recebidos por GET na URL.

Operador ternário

O operador ternário é uma versão compacta da estrutura de controle if. Sua sintaxe é simples e segue o padrão "condição ? valorSeVerdadeiro : valorSeFalso" (SOARES, 2017).

Exemplo prático:

```
if($idade>=18){
    $x = "Maior de Idade";
}else{
    $x = "Menor de Idade";
}
```

```
$x $idade>=18 ? "Maior de Idade" :
"Menor de Idade";
```

Ambos os códigos tem a mesma função, colocar na variável `$x` a string referente à maioridade ou não. Porém, observa-se que o operador ternário simplifica a codificação.

```
if ( $_SERVER['REQUEST_
METHOD'] === 'GET' ) {
    $tabela= filter_input(INPUT_GET,
'tabela' , FILTER_SANITIZE_SPECIAL_
CHARS);
    $coluna= isset($_GET['coluna'])
? filter_input(INPUT_GET, 'coluna' ,
FILTER_SANITIZE_SPECIAL_CHARS) : '';
    $valor= isset($_GET['valor']) ?
filter_input(INPUT_GET, 'valor' , FILTER_
SANITIZE_SPECIAL_CHARS) : '';
}
```

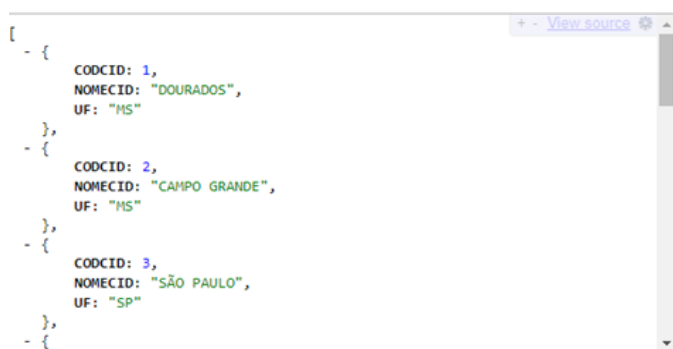
Em posse das informações passadas pelo usuário, a api efetua então a query no banco de dados, processa essa informação armazenando em um array todo o conteúdo recebido e, por fim, converte o array para uma string JSON que pode ser enviada para o usuário.

```
$sql = "SELECT * FROM $tabela ";
$sql .= $coluna!='' ? "WHERE
$coluna=$valor" : "";
$result = $conn->query($sql);
```



```
if($result->num_rows > 0){  
    while($row = $result->fetch_assoc())  
    {  
        echo json_encode($row);  
    }  
}
```


Caso o usuário não envie os parâmetros coluna e valor para a api apenas o nome da tabela cidade (<http://localhost/api.php?tabela=cidade>), o usuário receberá todas as cidades que estão na referida tabela, como na Figura 5.



```
[  
  {  
    "CODCID": 1,  
    "NOMECID": "DOURADOS",  
    "UF": "MS"  
  },  
  {  
    "CODCID": 2,  
    "NOMECID": "CAMPO GRANDE",  
    "UF": "MS"  
  },  
  {  
    "CODCID": 3,  
    "NOMECID": "SÃO PAULO",  
    "UF": "SP"  
  },  
]
```

Figura 5 – Select. Fonte: Acervo pessoal.

Caso o usuário envie também os parâmetros coluna e valor para a api (<http://localhost/api.php?tabela=cidade&coluna=codcid&valor=6>) o retorno para o usuário será os registros que se adequem à comparação que foi definida como na Figura 6.



```
[  
  {  
    "CODCID": 6,  
    "NOMECID": "SÃO SEBASTIÃO",  
    "UF": "SC"  
  }  
]
```

Figura 6 – Select Where. Fonte: Acervo pessoal.

Retomando a aula

Chegamos ao final da sexta aula. Vamos recordar?

1 – O que é o MySQL

Na primeira seção, o MySQL e o PHPMyAdmin foram apresentados, enquanto o primeiro é um banco de dados, o segundo é uma ferramenta web para gerenciar o banco de dados.

2 – Conectando ao Banco de Dados

Nesta seção, foi detalhada a conexão com o banco de dados, além de serem exemplificadas as operações de inserção e leitura de registros.

3 – CRUD

Em seguida, o CRUD (create, read, update e delete) foi definido demonstrando qual operação no banco de dados é representada por cada uma das letras do CRUD.

4 – Construindo uma API

Por fim, a API que é uma interface que pode integrar diversas plataformas, foi apresentada e codificada para a tabela de exemplo da seção.

Vale a pena

Vale a pena ler,

SOARES, W. *Programação Web com PHP 5*, Érica, 2014.
BEIGHLEY, L; MORRISON, M. *Use a Cabeça! PHP & MySQL*. Alta Books, 2010.
SILVA, J. M. C. *PHP na prática 200 Exercícios resolvidos*. Elsevier, 2014.
BENTO, E. J. *Desenvolvimento web com PHP e MySQL*. Editora Casa do Código, 2014.

Vale a pena acessar,

PIRES, J. In: O que é API? REST e RESTful? Conheça as definições e diferenças, 2017. Disponível em: <https://becode.com.br/o-que-e-api-rest-e-restful/>. Acesso em 26 out. 2019.

SOARES, E. In: O PHP: If/else e o operador ternário, 2017. Disponível em: <https://www.devmedia.com.br/php-if-else-e-o-operador-ternario/38219>. Acesso em 26 out. 2019.

Minhas anotações