12. PHP com Banco de Dados MySQL

Nesta unidade vamos abordar em detalhes o acesso a banco de dados com a linguagem PHP, fundamental para criarmos sites dinâmicos que facilitem a atualização de conteúdo. O PHP possibilita acesso a diversos bancos de dados. Nesta unidade vamos apresentar a integração com o banco de dados MySql, mas de uma forma que fica fácil a utilização de outros bancos de dados.

Iniciaremos vendo as principais funções para acesso ao banco de dados por meio da linguagem PHP, em seguida veremos como manter os dados, com exemplos de inclusão, alteração e exclusão de dados. Por fim será passado um exercício englobando todos esses conceitos.

Existem duas maneiras de se conectar com um banco de dados MySQL usando o PHP: MySQLi e PDO**.** Para se conectar e fazer consultas a um Banco de Dados MySQL, primeiramente é necessário criar um banco de dados MySQL que pode ser criado no phpMyAdmin (ver tópico 12.3) instalado junto ao xampp. A figura 7 mostra as etapas da conexão do PHP com MySQL.

Figura 7 – Etapas da conexão do PHP com MySQL

12.1. MySQLi

MySQLi significa **MySQL Improved**. Trata-se de uma extensão exclusiva para o MySQL que adiciona novos recursos para uma interface de *database* MySQL. O MySQLi é tanto

processual quanto orientada por objeto – este último atributo foi herdado da versão mais antiga do MySQL. O quadro 57 mostra as funções da API MySQLi.

Quadro 57

Função	**Para que serve**
int **mysqli_connect**(host, username, password, database)	Estabelece uma conexão ao host onde se encontra o banco de dados
Int **mysqli_select_db**(database[, int link_identifier])	Selecionar o banco de dados
Int **mysqli_query**(query [, int link_identifier])	Executar um comando no banco de dados
mysqli_close($conexao)	Fechar a conexão
array **mysqli_fetch_array**(int result[, int result_type])	armazenar a linha atual em um array associativo
mysqli_fetch_object()	retornar uma linha como um objeto
array **mysqli_fetch_row**(int result)	armazenar a linha atual em um array
int **mysqli_insert_id**([int link_identifier])	retornar o número de linhas após execução de um comando select
Int **mysqli_affected_rows**([int link_identifier])	retornar o nº de linhas que foram afetados em uma tabela por comandos update, insert e delete
mysqli_num_rows()	retornar o número de linhas de uma consulta
mysqli_num_fields()	retornar o número de colunas de uma consulta
mysqli_field_name()	retornar o nome de uma coluna em uma consulta
mysqli_result()	retornar uma coluna do resultado
mysqli_error()	Retornar uma mensagem de erro

Conectando ao Banco de Dados

O exemplo do código no quadro 58, mostra como usar o MySQLi para conectar um script PHP para o MySQL usando um estilo procedural, já o código do quadro 59 mostra como usar o MySQLi para conectar ao banco no estilo de programação orientado a objeto.

```
<?php
$servername = "localhost";
$database = "databasename";
$username = "username";
$password = "password";
// Cria conexão
$conn = mysqli_connect($servername, $username, $password, $database);
// Checa conexão
if (!$conn) {
   die("Falha na Conexão: " . mysqli_connect_error());
}
echo "Conectado com Sucesso!";
mysqli_close($conn);
?>
```

O principal método usado neste script é o **mysqli_connect()**. Essa é uma função PHP interna para estabelecer uma nova conexão com um servidor MySQL.

No início do código, foi necessário a declaração e atribuição de quatro variáveis para estabelecer uma conexão de banco de dados apropriada: **$servername, $database, $username**, e **$password**.

Se a conexão não foi bem-sucedida, então a função **die()** é executada. Isso basicamente mata o nosso script e nos apresenta o erro de mensagem de conexão que definimos. Por padrão, o erro de conexão MySQL vai dizer **Conexão falhou** seguido de uma mensagem de erro exata descrevendo o problema.

Por outro lado, se a conexão MySQL foi bem-sucedida, o código vai imprimir **Conectado com sucesso** ao invés disso.

A última parte do código é **mysqli_close**, que simplesmente vai fechar a conexão com o banco de dados de modo manual. Se isso não for especificado, as conexões MySQL vão fechar por conta própria quando o script for encerrado.

Quadro 59

```
<?php
$servername = "localhost";
$database = "databasename";
$username = "username";
$password = "password";
// Cria conexão
$conn = new mysqli_connect($servername, $username, $password, $database);
// Checa conexão
if ($conn -> connect_errno) {
  echo "Falha na conexão com MySQL: " . $conn -> connect_error;
  exit();
}
echo "Conectado com Sucesso!";
$conn -> close();
?>
```

Inserindo dados no Banco de Dados

O primeiro procedimento que deve ser feito é estabelecer uma conexão com o banco de dados. Depois dessa etapa, podemos prosseguir com o **INSERT** do *query* do MySQL. O quadro 60 mostra um exemplo de código completo com os métodos básicos de conexão e inserção no estilo procedural.

Quadro 60

```
<?php
$servername = "localhost";
$database = "academico";
$username = "root";
```

```
$password = "";
// Cria conexão
$conn = mysqli_connect($servername, $username, $password, $database);
// Checa conexão
if (!$conn) {
    die("Conexão com falha: " . mysqli_connect_error());
}
echo "Conectado com Sucesso!";
$sql = "INSERT INTO alunos (nome, sobrenome, email) VALUES ('Pedro', 'II',
'pedro@gmail.com')";
if (mysqli_query($conn, $sql)) {
    echo "Novo registro criado com sucesso";
} else {
    echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}
mysqli_close($conn);
?>
```

A linha 13 é a linha onde será instanciado o comando para inserir dados no MySQL através do PHP. O **INSERT INTO** é uma instrução que adiciona dados em banco de dados específicos da tabela. Neste exemplo, estamos adicionando dados à tabela **alunos**.

Seguindo adiante, entre os parenteses, temos as colunas de tabela para as quais queremos adicionar valores (**nome**, **sobrenome**, **email**). Os dados serão preenchidos na ordem especificada.

A próxima parte é sobre a declaração VALUES, cujo os valores seguem a ordem das colunas determinadas anteriormente. Dessa maneira, cada coluna representa um valor específico. No exemplo, seria: nome = ‘Pedro’, sobrenome = ‘II’, email = ‘pedro@gmail.com’**.**

Vale a pena lembrar, que as SQL Queries devem ser posicionados entre as aspas. No exemplo do quadro 60, tudo o que está entre aspas e depois de $sql = é uma SQL Query.

Atualizando dados no Banco de Dados

A estrutura do código segue a mesma da inserção, o que muda é o comando SQL. O quadro 61 mostra o código para atualização das informações na tabela alunos.

Quadro 61

```
<?php
$servername = "localhost";
$database = "academico";
$username = "root";
$password = "";
// Cria conexão
$conn = mysqli_connect($servername, $username, $password, $database);
// Checa conexão
if (!$conn) {
    die("Conexão com falha: " . mysqli_connect_error());
}
echo "Conectado com Sucesso!";
$sql = "UPDATE alunos set email='pedro2@gmail.com' where nome='Pedro' and
sobrenome='II'";
if (mysqli_query($conn, $sql)) {
    echo "Registro atualizado com sucesso";
} else {
    echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}
mysqli_close($conn);
?>
```

Repare que tanto o exemplo de inserção como de atualização, usa valores fixos no código, isto é, não usam variáveis, portanto sempre vão inserir Pedro para o nome, II para o sobrenome e pedro@gmail.com para o email. E a atualização sempre vai ser para o email pedro2@gmail.com. Para fazer algo mais flexível, como por exemplo, inserir

valores que venha de um formulário, será necessário o uso de variáveis. O quadro 62 mostra um exemplo de código com os valores passados via post pelo formulário.

Quadro 62 – Inserindo dados no banco de dados a partir de um formulário

```
<?php
$servername = "localhost";
$database = "academico";
$username = "root";
$password = "";

$nome = $_POST[‘nome’];
$sobrenome = $_POST[‘sobrenome’];
$email = $_POST[‘email’];

// Cria conexão
$conn = mysqli_connect($servername, $username, $password, $database);
// Checa conexão
if (!$conn) {
    die("Conexão com falha: " . mysqli_connect_error());
}
echo "Conectado com Sucesso!";
$sql = "INSERT INTO alunos (nome, sobrenome, email) VALUES
(‘“.$nome.”’,’”.$sobrenome.”’,’”.$email.”’)";
if (mysqli_query($conn, $sql)) {
    echo "Novo registro criado com sucesso";
} else {
    echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}
mysqli_close($conn);
?>
```

Em caso de dúvida sobre os métodos POST e GET, consulte a unidade 7.

Removendo Dados do Banco de Dados

Assim como a inserção e a atualização, a remoção dos dados seguem a mesma estrutura. O quadro 63 mostra um exemplo de código de remoção com dados oriundos de um formulário.

Quadro 63

```
<?php
$servername = "localhost";
$database = "academico";
$username = "root";
$password = "";

$nome = $_POST[‘nome’];
$sobrenome = $_POST[‘sobrenome’];
$email = $_POST[‘email’];

// Cria conexão
$conn = mysqli_connect($servername, $username, $password, $database);
// Checa conexão
if (!$conn) {
    die("Conexão com falha: " . mysqli_connect_error());
}
echo "Conectado com Sucesso!";
$sql = "DELETE from alunos where nome=’”.$nome.”’ and
sobrenome=’”.$sobrenome.”’";
if (mysqli_query($conn, $sql)) {
    echo "Registro removido com sucesso";
} else {
    echo "Error: " . $sql . "<br>" . mysqli_error($conn);
```

```
}
mysqli_close($conn);
?>
```

Repare a concatenação na variável $sql, o primeiro trecho do texto **DELETE from alunos where nome=’** termina com o apostrofe pelo motivo do nome ser uma string (e string fica entre aspas ou apostrofe). Na segunda parte encontra-se a variável **.$nome.**, nesse momento o valor da variável (supondo que seja Pedro) será concatenado com o primeiro trecho, ficando então, **DELETE from alunos where nome=’Pedro** . Na terceira parte do texto será concatenado o trecho **’ and sobrenome=’** , repare novamente os apostrofes tanto no início como no fim, no início significa a apostrofe após o nome **Pedro’** e no fim o início da próxima condição **‘ and sobrenome=’**. A mesma lógica se repete para adicionar o conteúdo do sobrenome **.**