



# Programação para Internet

---

## Módulo 6

Websites Dinâmicos com Acesso a Banco de Dados (MySQL)

Prof. Dr. Daniel A. Furtado - FACOM/UFU

Conteúdo protegido por direito autoral, nos termos da Lei nº 9 610/98

A cópia, reprodução ou apropriação deste material, total ou parcialmente, é proibida pelo autor

# Conteúdo do Módulo

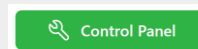
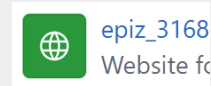
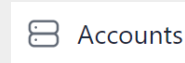
- Criação de um banco de dados gratuito no [infinityfree.net](http://infinityfree.net)
- Formas de comunicação com o MySQL: MySQLi x PDO
- Conexão com o MySQL
- Execução de Declarações SQL: métodos [fetch](#), [fetchAll](#), [query](#) e [exec](#)
- Injeção de SQL e declarações preparadas ([prepared statements](#))
- Transações

# Criando um Banco de Dados de Teste

---

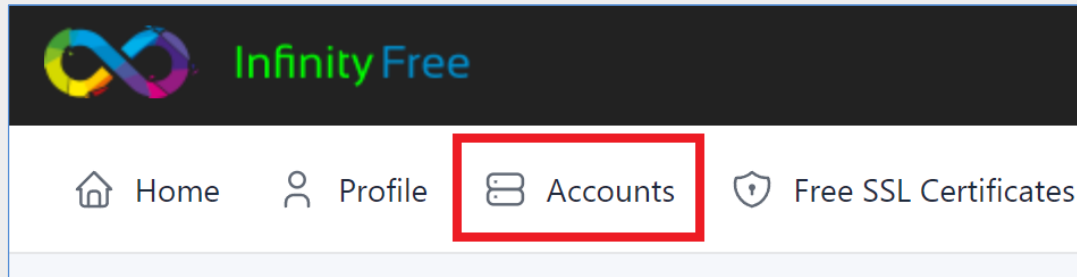
# Criando um Banco de Dados no infinityfree

1. Clique em **Accounts** no menu superior
2. Selecione a conta criada anteriormente
3. Acesse o **Painel de Controle**, clicando em **Control Panel**
4. Dentro do grupo **Databases**, escolha **MySQL Databases**
5. Preencha o campo para criar um novo banco de dados

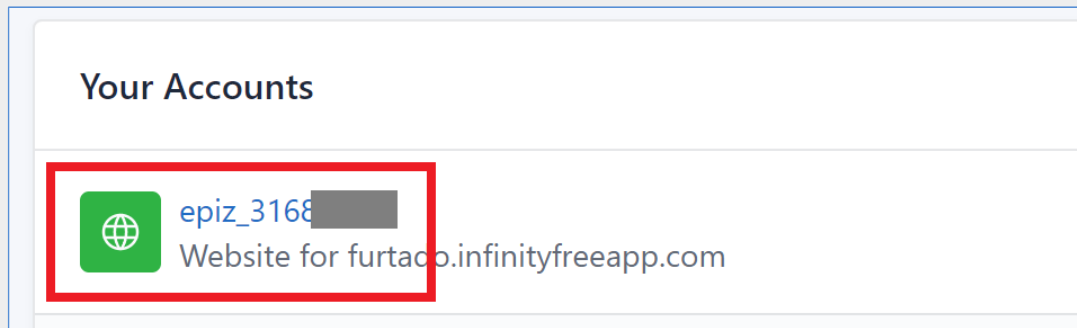


# Criando um Banco de Dados no infinityfree

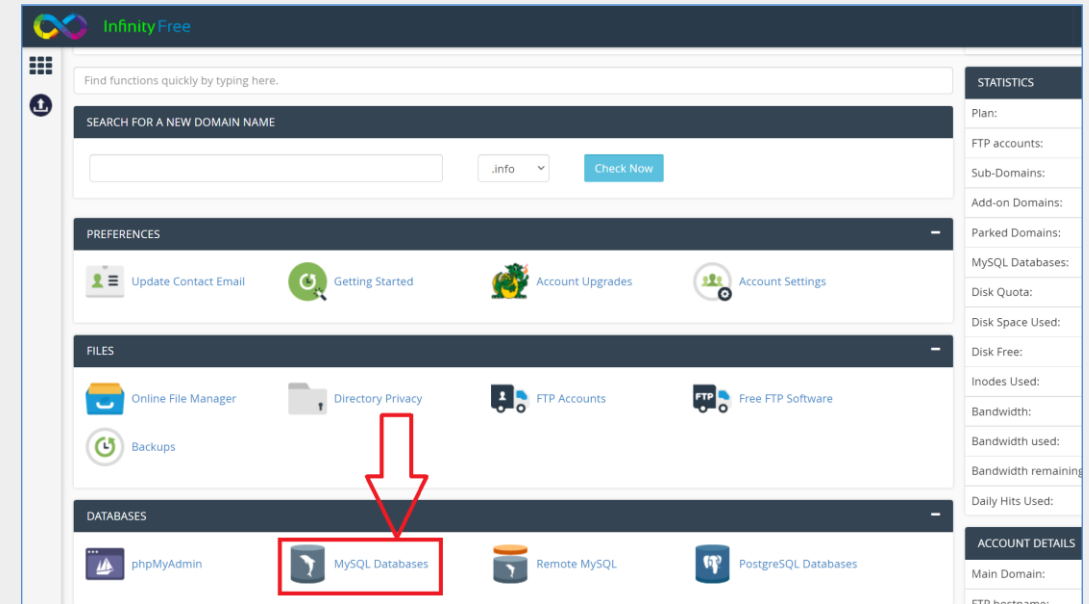
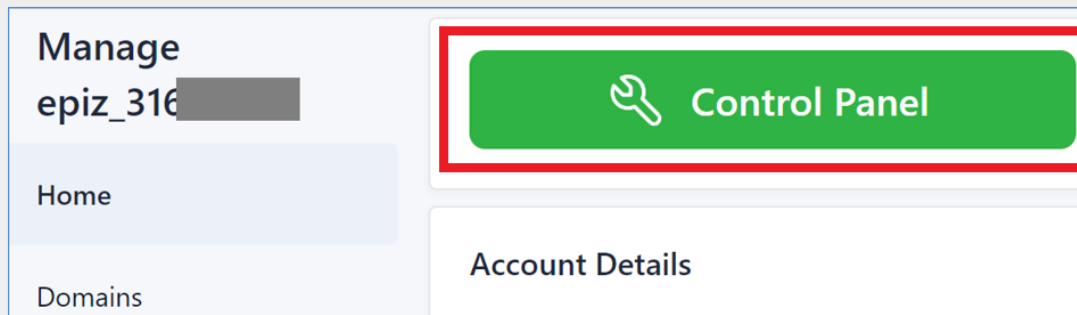
1



2



3



4

## Create New Database

Currently using 1 of 400 available databases.

### New Database:

epiz\_31681102\_ ppi

Create Database

5

# Acessando os Detalhes do Banco de Dados Criado

Manage epiz\_3168

Home

Domains

FTP Details

MySQL Details

Account Settings

MySQL Details for epiz\_3168

MYSQL USERNAME  
epiz\_3168

MYSQL PASSWORD  
\*\*\*\*\* Show/Hide

MYSQL HOSTNAME  
sql106.epizy.com

MYSQL PORT (OPTIONAL)  
3306

MYSQL DATABASE NAME  
epiz\_3168\_XXX  
(create this in the control panel)

How to use MySQL

Esses dados serão necessários no código PHP para efetuar a conexão com o MySQL

# Criando Tabela de Teste com o phpMyAdmin

1. Acesse novamente o **Control Panel** 
2. Clique em **phpMyAdmin** no grupo **Databases** 
3. Clique no botão **Connect Now**
4. Clique na aba **SQL**, digite o código a seguir e tecle **ctrl-enter** para executar:

```
CREATE TABLE aluno
(
  nome varchar(50),
  telefone varchar(50)
) ENGINE=InnoDB;
```

```
INSERT INTO aluno VALUES ("Fulano", "123");
INSERT INTO aluno VALUES ("Ciclano", "456");
```

```
SELECT * FROM aluno;
```

# Conectando ao MySQL com PHP

---



# Interfaces do PHP para Comunicação com o MySQL

## MySQLi Extension (MySQL Improved)

- Interface específica para o MySQL
- Desempenho otimizado
- Suporta alguns recursos específicos do MySQL

## PHP Data Objects (PDO) Extension

- Provê interface única e consistente para vários SGBDs
- Incluindo MySQL, PostgreSQL, Firebird, IBM DB2, etc.

**OBS:** Neste material utilizaremos apenas o PHP Data Objects (PDO) para conectar ao MySQL

# Exemplo de Função para Conexão com o MySQL

```
function mysqlConnect() {  
    $host = "host do mysql";  
    $username = "usuario no mysql";  
    $password = "senha do usuario mysql";  
    $dbname = "nome do banco de dados";  
  
    $options = [  
        PDO::ATTR_EMULATE_PREPARES => false,    // desativa a execução emulada de prepared statements  
        PDO::ATTR_ERRMODE          => PDO::ERRMODE_EXCEPTION, // ativa o modo de erros para lançar exceções  
        PDO::ATTR_PERSISTENT        => true,      // ativa o uso de conexões persistentes para maior eficiência  
        PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC, // altera o modo de busca padrão para FETCH_ASSOC  
    ];  
  
    try {  
        // O objeto $pdo será utilizado nas operações com o BD  
        $pdo = new PDO("mysql:host=$host; dbname=$dbname; charset=utf8mb4", $username, $password, $options);  
        return $pdo;  
    } catch (Exception $e) {  
        exit('Falha na conexão com o MySQL: ' . $e->getMessage());  
    }  
}
```

# Executando Operações SQL Após Conexão

---

# Executando Declarações SQL Após Conexão

- Após conexão com o MySQL, as operações no banco de dados podem ser executadas de três formas, dependendo do tipo de operação e das circunstâncias da execução:
  - Com o método **query** do objeto `$pdo`
  - Com o método **exec** do objeto `$pdo`
  - Utilizando **prepared statements**
    - Com os métodos **prepare** e **execute**

# Formas de Executar Declarações SQL

```
$pdo->query("Código SQL");
```

Utilize quando **não há** a possibilidade de injeção de SQL e o código SQL **retorna** um conteúdo a ser processado

```
$pdo->exec("Código SQL");
```

Utilize quando **não há** a possibilidade de injeção de SQL e o código SQL **não** retorna conteúdo (ex. INSERT, DELETE, UPDATE)

```
$pdo->prepare("Código SQL");  
$pdo->execute(...);
```

Utilize quando o código SQL inclui dados fornecidos pelo usuário (vindos de formulários, da URL, etc.)

**OBS:** O conceito de **injeção de SQL** será apresentado ao longo deste material

# Método query

- O método **query** prepara e executa uma declaração SQL sem parâmetros
- O método retorna um objeto do tipo **PDOStatement**, que permite acessar os dados retornados pela declaração SQL
- O objeto retornado disponibiliza o método **fetch**, o qual retorna a próxima linha do resultado (ou falso quando não há mais linhas)

```
$sql = <<<SQL
    SELECT nome, telefone as tel
    FROM aluno
SQL;

$stmt = $pdo->query($sql);

while ($row = $stmt->fetch()) {
    echo $row['nome'];
    echo $row['tel'];
}
```

Por padrão, o método **fetch** retorna a próxima linha do resultado na forma de um **array associativo**, onde os dados são acessados pelo **nome da coluna** na tabela (ou o nome produzido na consulta SQL)

# Hello World – Listagem dos Dados na Tabela aluno

```
1 <?php
2
3 require "../conexaoMysql.php";
4 $pdo = mysqlConnect();
5
6 try {
7     $sql = <<<SQL
8         SELECT nome, telefone
9         FROM aluno
10    SQL;
11
12    $stmt = $pdo->query($sql);
13 }
14 catch (Exception $e) {
15     exit('Ocorreu uma falha: ' . $e->getMessage());
16 }
17
18 ?>
19 <!doctype html>
20 <html lang="pt-BR">
```

```
<table class="table table-striped table-hover">
    <tr>
        <th>Nome</th>
        <th>Telefone</th>
    </tr>
    <?php
    while ($row = $stmt->fetch())
    {
        $nome = htmlspecialchars($row['nome']);
        $telefone = htmlspecialchars($row['telefone']);

        echo <<<HTML
        <tr>
            <td>$nome</td>
            <td>$telefone</td>
        </tr>
        HTML;
    }
    ?>
</table>
```

# Outras Formas de Uso do Método fetch

```
// retorna array indexado por nome da coluna
$stmt->fetch(PDO::FETCH_ASSOC);

// retorna array indexado por número da coluna
$stmt->fetch(PDO::FETCH_NUM);

// retorna array indexado por nome da coluna e número
$stmt->fetch(PDO::FETCH_BOTH);

// retorna objeto com propriedades corresp. às colunas
$stmt->fetch(PDO::FETCH_OBJ);
```



# Outras Formas de Uso do Método fetch

- Se a consulta retorna um único escalar, é possível resgatá-lo de forma simples e prática com `fetch` e `PDO::FETCH_COLUMN`
- O mesmo efeito pode ser obtido com `$stmt->fetchColumn()`

```
$sql = <<<SQL
SELECT COUNT(*)
FROM paciente
SQL;

$stmt = $pdo->query($sql);
$numPacientes = $stmt->fetch(PDO::FETCH_COLUMN);
```

# Método fetchAll

- Retorna, de uma vez, todas as linhas restantes do resultado na forma de um array de arrays
- O uso inadequado **pode sobrecarregar** o servidor ou a rede
- Retorna todas as linhas de uma única coluna quando utilizado em conjunto com **PDO::FETCH\_COLUMN**

*\$especialidades*  
será um array  
simples, indexado  
por números

```
$sql = <<<SQL
    SELECT especialidade
    FROM especilidades_medicas
SQL;

$stmt = $pdo->query($sql);
$especialidades = $stmt->fetchAll(PDO::FETCH_COLUMN);

foreach ($especialidades as $especialidade)
    echo $especialidade;
```

**fetchAll** conta com  
um 2ª parâmetro  
opcional para indicar  
o número da coluna  
(começando em 0)

# Método exec

- O método `exec` executa uma declaração SQL e retorna o número de linhas afetadas
- Deve ser executado quando não há risco de injeção de SQL e quando a declaração SQL não retorna um resultado a ser processado

```
$sql = <<<SQL
UPDATE funcionario
SET salario = salario * 1.2
WHERE cargo = 'Gerente'
SQL;

$numLinhasAfetadas = $pdo->exec($sql);
```

# Injeção de SQL e Prepared Statements

---

# Injeção de SQL (*SQL Injection*)

- Técnica utilizada por usuários maliciosos
- Injetam código SQL dentro de uma instrução SQL lícita
- Geralmente utiliza campos de formulário ou a URL
- Pode comprometer a segurança da aplicação Web
  - Existe a possibilidade do usuário malicioso realizar consultas, atualizações e exclusões no BD, **sem qualquer autorização**

# Exemplo de Código Vulnerável a Injeção de SQL

Formulário de login

Usuário	Senha
<input type="text" value="tolo ' or '='"/>	<input type="password" value="....."/>
<input type="button" value="Entrar"/>	

Exemplo de código PHP vulnerável para validar o login

```
$usuario = $_POST["user"];
$senha   = $_POST["password"];

$sql = <<<SQL
    SELECT count(*) FROM usuarios
    WHERE user = '$usuario' AND senha = '$senha'
SQL;

$stmt = $pdo->query($sql);
if ($stmt->fetchColumn() > 0) // login com sucesso
```

Não faça isso!

Ao inserir o texto `tolo ' or '='` nos campos do formulário o usuário conseguiria burlar a validação do login injetando condições na consulta SQL que resultariam sempre em verdadeiro e mudaria o propósito da consulta original

Expressão SQL resultante após avaliação do PHP

```
SELECT * FROM usuarios
WHERE user = 'tolo' or '=' AND senha = 'tolo' or '='
```

# Exemplo de Código Vulnerável a Injeção de SQL

Formulário de cadastro

Nome	Endereço
<input type="text" value="Tolo"/>	<input type="text" value="tolo'); DELETE FROM aluno; -- comment"/>
<input type="button" value="Cadastrar Aluno"/>	

Exemplo de código PHP vulnerável para validar o login

```
$nome = $_POST["nome"];  
$endereco = $_POST["endereco"];  
$sql = <<<SQL  
    INSERT INTO aluno (nome, endereco)  
    VALUES ('$nome', '$endereco')  
SQL;  
$pdo->exec($sql);
```

Não faça isso!

Neste exemplo um usuário malicioso conseguiria injetar, pelo campo endereço, um código SQL para excluir todo o conteúdo da tabela aluno



Expressão SQL resultante após avaliação do PHP

```
INSERT INTO aluno (nome, endereco)  
VALUES ('tolo', 'tolo'); DELETE FROM aluno; -- comment')
```

# Prepared Statements

- Técnica que permite executar, de forma mais segura, operações SQL que trazem risco de injeção de SQL
- Indicada quando a operação SQL inclui **parâmetros** (placeholders) com possíveis **dados produzidos pelo usuário** (seja através de campos de formulário ou da própria URL)
- Técnica suportada por diversos SGBDs



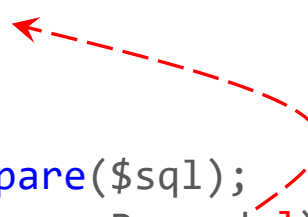
# Prepared Statements

- Com **prepared statements**, os dados dos parâmetros são passados separadamente da declaração SQL
- Dessa forma, não é possível alterar a estrutura em si da declaração SQL por meio dos parâmetros (como nos exemplos de injeção anteriores)
- Elimina a necessidade de utilizar aspas nos parâmetros
- Outra vantagem é o ganho em eficiência quando se deseja executar a mesma declaração múltiplas vezes, alterando apenas os dados
  - O código SQL pode ser preparado uma única vez pelo SGBD

# Prepared Statements - Exemplo de Uso - **select**

**Ponto de Interrogação**  
Usado para indicar um parâmetro em aberto, cujo valor será fornecido posteriormente, no momento da execução da operação.

```
$marcaBusca = $_GET['marca'] ?? "";  
$sql = <<<SQL  
    SELECT descricao, preco  
    FROM produto  
    WHERE marca = ?  
SQL;  
  
$stmt = $pdo->prepare($sql);  
$stmt->execute([$marcaBuscada]);  
  
while ($row = $stmt->fetch()) {  
  
    echo $row['descricao'];  
    echo $row['preco'];  
  
}
```



**Método *prepare***  
Prepara a declaração SQL e retorna um objeto do tipo *PDOStatement*

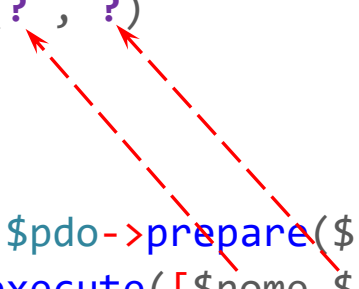
**Método *execute***  
Executa a declaração preparada associando valores aos parâmetros em aberto.

# Prepared Statements - Exemplo de Uso - insert

```
$nome = $_POST['nome'] ?? "";
$idade = $_POST['idade'] ?? "";

$sql = <<<SQL
    INSERT INTO cliente (nome, idade)
    VALUES (?, ?)
SQL;

try {
    $stmt = $pdo->prepare($sql);
    $stmt->execute([$nome,$idade]);
}
catch (Exception $e) {
    exit('Falha inesperada: ' . $e->getMessage());
}
```



# Prepared Statements - Exemplo de Uso - **insert**

```
$sql = <<<SQL
    INSERT INTO cliente (nome, idade)
    VALUES (?, ?)
SQL;

try {
    $stmt = $pdo->prepare($sql);
    $stmt->execute(['Fulano1',20]);
    $stmt->execute(['Fulano2',30]);
    $stmt->execute(['Fulano3',40]);
}
catch (Exception $e) {
    exit('Falha inesperada: ' . $e->getMessage());
}
```

*Neste exemplo a operação INSERT é avaliada uma única vez pelo MySQL (com o método **prepare**), mas executada múltiplas vezes com o método **execute** (mudando apenas os dados de inserção).*

# Prepared Statements - Exemplo com **bindParam**

Neste exemplo os dados dos parâmetros não são fornecidos na chamada do método **execute**. Como alternativa, são vinculadas as variáveis **\$nome** e **\$idade** aos parâmetros. Dessa forma, o valor **corrente** das variáveis será utilizado no momento da chamada do **execute**

```
$nome = null; $idade = null;

$sql = <<<SQL
    INSERT INTO cliente (nome, idade)
    VALUES (?, ?)
SQL;

$stmt = $pdo->prepare($sql);

// Vincula as variáveis aos parâmetros
$stmt->bindParam(1, $nome);
$stmt->bindParam(2, $idade);

// Insere uma linha
$nome = 'Pedro';
$idade = 30;
$stmt->execute();

// Insere outra linha com valores diferentes
$nome = 'Maria';
$idade = 40;
$stmt->execute();
```

O método **bindParam** permite vincular nomes de variáveis aos parâmetros. No momento da execução os valores correntes das variáveis serão utilizados.

Um bloco **try-catch** não foi utilizado aqui para não tornar o exemplo muito extenso.

# Prepared Statements - Exemplo com **bindParam**

```
$nome = null; $idade = null;

$sql = <<<SQL
    INSERT INTO cliente (nome, idade)
    VALUES (:nome, :idade)
SQL;

$stmt = $pdo->prepare($sql);

// Vincula as variáveis aos parâmetros
$stmt->bindParam(':nome', $nome);
$stmt->bindParam(':idade', $idade);

// Insere uma linha
$nome = "Pedro";
$idade = 30;
$stmt->execute();

// Insere outra linha com valores diferentes
$nome = "Maria";
$idade = 40;
$stmt->execute();
```

Neste exemplo os parâmetros em aberto são nomeados utilizando o caracter **dois-pontos** seguido de um **identificador**. Esta notação fornece **maior clareza**, porém é menos prática do que a notação que utiliza apenas o ponto-de-interrogação.

# Número de Linhas Afetadas

```
$nroLinhas = $stmt->rowCount();
```

- **rowCount()**, do objeto **PDOStatement**, retorna o número de linha afetadas pela última operação SQL
- Exemplos
  - Número de linhas afetadas após INSERT, DELETE ou UPDATE
  - Número de linhas retornadas pela operação SELECT em alguns SGBDs (por ex. no MySQL no modo buffer)

# Transações

---



# Transações

- Sequência de operações "indivisíveis"
  - Executa-se **todas** ou **nenhuma**
- O início da transação é normalmente definido com a operação **begin transaction**
- O término da transação é marcado pela operação **commit** para **efetivar** todas as operações realizadas desde o início da transação
- Em caso de falha durante a transação, pode-se executar a operação **rollback** para desfazer todas as operações iniciadas após o **begin transaction**

# Exemplo de Uso de Transações

- Cadastro de cliente com inserção em duas tabelas
  - Dados pessoais na tabela `cliente`
  - Dados do endereço na tabela `endereco_cliente`
- Não se deve permitir o cadastro parcial
  - Dados pessoais do cliente, sem os dados do endereço
- Portanto, as operações de inserção dos dados do cliente podem ser tratadas como uma transação

# Exemplo de Uso de Transações

```
try {  
    // início da transação  
    $pdo->beginTransaction();  
  
    $stmt = $pdo->prepare('INSERT INTO Cliente VALUES ...');  
    if (! $stmt->execute([dados do cliente]))  
        throw new Exception('Falha na operação 1');  
  
    $stmt = $pdo->prepare('INSERT INTO EnderecoCliente VALUES ...');  
    if (! $stmt->execute([dados do endereco do cliente]))  
        throw new Exception('Falha na operação 2');  
  
    // se nenhuma exceção foi lançada, efetiva as operações  
    $pdo->commit();  
}  
catch (Exception $e)  
{  
    // desfaz as operações em caso de erro (exceção lançada)  
    $pdo->rollback();  
    exit('Falha na transação: ' . $e->getMessage());  
}
```

# Último ID Inserido

```
$ultimoIdInserido = $pdo->lastInsertId();
```

- Comumente utilizado em colunas do tipo `auto_increment`
- Retorna o último código/id inserido automaticamente na linha
- O id pode ser usado, por exemplo, para inserção em campo do tipo chave estrangeira em tabela vinculada

# Códigos de Exemplo

<http://www.furtado.prof.ufu.br/site/teaching/PPI/Exemplos-Mysql.zip>

# Referências

- <https://www.php.net/docs.php>
- NIXON, R. *Learning PHP, MySQL & JavaScript: With jQuery, CSS & HTML5*. 5. ed. O'Reilly Media, 2018