Desenvolvimento Web II

Módulo 3

Acesso a Banco de Dados: Aspectos de Segurança e Transações

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

Conteúdo do Módulo

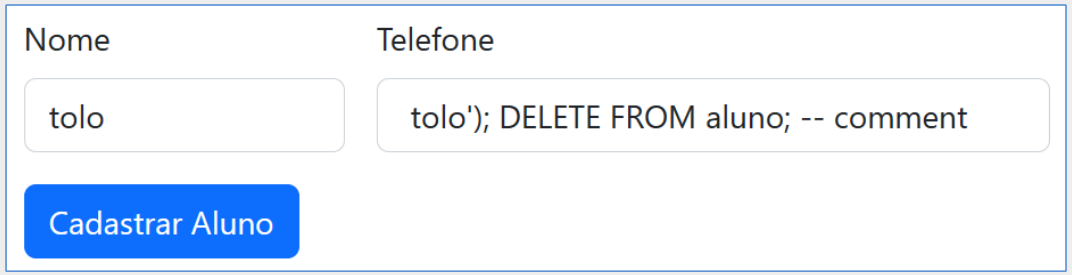
- Aspectos de segurança no acesso à banco de dados em sistemas web
- Ataques de injeção de SQL
- Conceito de declarações preparadas (prepared statements)
- Transações

Ataques de Injeção de SQL (*SQL Injection*)

- Injeção de SQL é um tipo de ataque utilizado por usuários maliciosos para injetar código SQL ilícito dentro de uma instrução SQL lícita, podendo causar danos no sistema Web.
- 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 até mesmo exclusões no banco de dados relacional, **sem qualquer autorização**.

Exemplo de Código PHP Vulnerável a Injeção de SQL

Formulário de cadastro preenchido com código malicioso (injeção de SQL)



Exemplo de código PHP vulnerável para receber o formulário e inserir no banco de dados

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

Não faça isso!

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

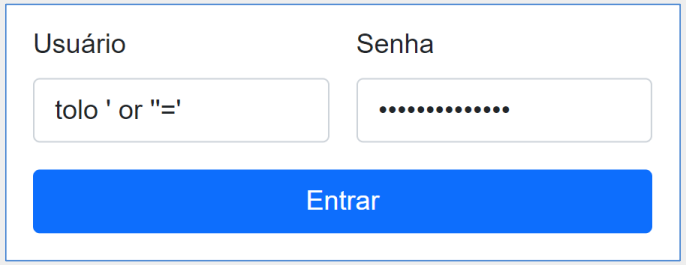
Expressão SQL resultante após avaliação do PHP

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

Quando o PHP avalia a string $sql e substitui os nomes das variáveis $nome e $telefone pelos respectivos conteúdos (fornecidos pelo usuário), uma nova string é obtida, a qual passa a ter dois comandos SQL válidos seguidos de um comentário em SQL. Através do método exec, essa string é então repassada ao servidor do MySQL, que irá executá-la sem "saber" que ela foi alterada indevidamente no código PHP. Repare que o usuário acaba conseguindo introduzir seu próprio comando SQL (DELETE) dentro da string SQL do desenvolvedor, sem produzir qualquer erro de sintaxe.

Exemplo de Código Vulnerável a Injeção de SQL

Formulário de login



Exemplo de código PHP vulnerável para validar o login

```php
$usuario = $_POST["usuario"];
$senha   = $_POST["senha"];
$sql = <<<SQL
  SELECT count(*) FROM usuarios
  WHERE usuario = '$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 de login injetando condições na consulta SQL que resultariam sempre em verdadeiro e mudaria o propósito da consulta original

OBS: senhas não devem ser armazenadas de forma clara no BD, como apresentado neste exemplo vulnerável. Utilize funções de hash.

Expressão SQL resultante após avaliação do PHP

```sql
SELECT count(*) FROM usuarios
WHERE usuario = 'tolo' or ''='' AND senha = 'tolo' or ''=''
```

Prepared Statements

- Técnica que permite executar, de forma 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, como nos exemplos anteriores, ou passados pela própria URL).
- A técnica é suportada por diversos SGBDs e linguagens de programação.

Prepared Statements

- Com prepared statements, a declaração SQL em si, sem os dados do usuário, é passada ao servidor de banco de dados em uma 1ª etapa, denominada etapa da preparação;
- Nesse momento o servidor de banco de dados poderá planejar a execução da declaração SQL e utilizar mecanismos de segurança internos para garantir que a estrutura lógica da declaração **não seja mais alterada**;
- Porém, para poder executar de fato a declaração iniciada, o servidor de banco de dados ainda precisa dos dados que não foram fornecidos, que devem então ser passados de forma isolada em uma 2ª etapa, denominada etapa da execução.

Exemplo de Prepared Statements no PHP - **insert**

```php
$nome = $_POST['nome'] ?? "";
$telefone = $_POST[telefone'] ?? "";

$sql = <<<SQL
  INSERT INTO aluno (nome, telefone)
  VALUES (? , ?)
  SQL;

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

Ponto de Interrogação
*É um **placeholder** usado para indicar um parâmetro em aberto, cujo valor será fornecido posteriormente, no momento da execução da declaração preparada.*

*Método **prepare***
*Prepara a declaração SQL utilizando o "template" da declaração, sem a presença dos dados do usuário. No lugar de cada dado coloca-se um **placeholder** (caractere interrogação).*
*O método **prepare** retorna um objeto do tipo PDOStatement*

*Método **execute***
*Executa a declaração preparada anteriormente fornecendo valores aos **placeholders** (pontos de interrogação).*

Os valores dos **placeholders** (pontos de interrogação) devem ser passados em um *array*, no momento da chamada do método **execute**, na respectiva ordem. Há também uma sintaxe alternativa, que será apresentada posteriormente.

Exemplo de Prepared Statements no PHP - **select**

```php
$marcaBuscada = $_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'];
}
```

Prepared Statements

- Outro benefício que a técnica oferece é o ganho em eficiência quando se deseja executar a mesma declaração SQL múltiplas vezes, mas alterando apenas os dados;
- Neste caso, o SGBD pode fazer o planejamento da declaração uma única vez (na etapa da preparação), mas executá-la várias vezes com múltiplas chamadas do método execute, alterando apenas os valores dos parâmetros (exemplo no próximo slide).

Repetindo uma Declaração SQL com Prepared Statements

```php
$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 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.*

```php
$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();
```

Repare que os dados dos parâmetros não são passados na chamada do método execute. Os parâmetros são vinculados às variáveis $nome e $idade utilizando o método bindParam. Assim, quando o método execute é chamado, os valores das variáveis são repassados aos parâmetros vinculados.

Prepared Statements – Sintaxe Alternativa

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

// Prepara a declaração
$stmt = $pdo->prepare($sql);


// Dados a serem inseridos
$nome = 'Pedro';
$idade = 30;

// Executa a declaração passando os dados
$stmt->execute([
    ':nome' => $nome,
    ':idade' => $idade
]);
```

Transações

- Uma transação é uma **sequência de operações** que deve ser tratada de forma indivisível, ou seja:
  - Executa-se **todas** ou não se executa **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 realizadas desde o início da transação.

Exemplo de Transação

- Cadastro de cliente com inserção em duas tabelas:
  - Dados pessoais na tabela Cliente
  - Dados do endereço na tabela EnderecoCliente
- 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 nas duas tabelas podem ser tratadas como uma transação.

Tabelas Correlacionadas (relacionamento 1 x n)

```
CREATE TABLE Cliente
(
    id int PRIMARY KEY auto_increment,
    nome varchar(50),
    cpf char(14) UNIQUE,
) ENGINE=InnoDB;

CREATE TABLE EnderecoCliente
(
    id int PRIMARY KEY auto_increment,
    cep char(10),
    cidade varchar(50),
    id_cliente int not null,
    FOREIGN KEY (id_cliente) REFERENCES Cliente(id) ON DELETE CASCADE
) ENGINE=InnoDB;
```

Exemplo Completo de Inserção em Tabelas Correlacionadas

```php
<?php
require "conexaoMysql.php";
$pdo = mysqlConnect();

// dados do cliente
$nome = $_POST["nome"] ?? "";
$cpf  = $_POST["cpf"] ?? "";

// dados do endereço
$cep = $_POST["cep"] ?? "";
$cidade = $_POST["cidade"] ?? "";
```

```php
try {
  $pdo->beginTransaction(); // Inicia a transação

  // Prepara e executa a primeira inserção
  // Uma exceção será lançada em caso de falha
  $stmt1 = $pdo->prepare(
    <<<SQL
    INSERT INTO Cliente (nome, cpf) VALUES (?, ?)
    SQL
  );
  $stmt1->execute([$nome, $cpf]);

  // Prepara e executa a segunda inserção
  // Uma exceção será lançada em caso de falha
  $idNovoCliente = $pdo->lastInsertId();
  $stmt2 = $pdo->prepare(
    <<<SQL
    INSERT INTO EnderecoCliente (cep, cidade, id_cliente)
    VALUES (?, ?, ?)
    SQL
  );
  $stmt2->execute([$cep, $cidade, $idNovoCliente]);

  $pdo->commit(); // Efetiva as operações caso nenhuma exceção seja lançada
}
catch (Exception $e) {
  $pdo->rollBack(); // Desfaz as operações caso ocorra falha em alguma inserção
  echo "Falha na transação: " . $e->getMessage();
}
```

Repare que o campo **id** do cliente é omitido, pois será gerado automaticamente (**auto_increment**)

O método **lastInsertId**() retorna o último id gerado pelo MySQL para inserção em coluna do tipo **auto_increment**. Precisamos desse id para vincular os dados do endereço na tabela correlacionada

Outras 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 (ex. SELECT).

```
$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).

Método query

```php
// String com operação SQL a ser executada.
// Não deve conter parâmetros do usuário (injeção de SQL).
$sql = <<<SQL
  SELECT nome, telefone as tel
  FROM aluno
  SQL;

// Executa a operação SQL e retorna objeto do tipo
// PDOStatement, que dá acesso ao resultado
$stmt = $pdo->query($sql);

// Acessa linha a linha os resultados da operação.
while ($row = $stmt->fetch()) {
  echo $row['nome'];
  echo $row['tel'];
}
```

- O método **query** prepara e executa uma declaração SQL sem parâmetros.
- **query** retorna um objeto do tipo PDOStatement, que dá acesso ao resultado.
- O método **fetch** do objeto PDOStatement retorna a próxima linha do resultado (ou falso, quando não há mais linhas a serem processadas).

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 pelo *alias* produzido na consulta SQL – *tel* neste exemplo). Porém o formato pode ser alterado com parâmetros. Por exemplo, `$stmt->fetch(PDO::FETCH_OBJ)` retornará a próxima linha do resultado como um objeto PHP (exemplo completo no próximo slide).

Método query

```php
$stmt = $pdo->query(
  <<<SQL
  SELECT nome, telefone as tel
  FROM aluno
  SQL;
);

while ($objeto = $stmt->fetch(PDO::FETCH_OBJ)) {
  echo $objeto->nome;
  echo $objeto->tel;
}
```

Neste exemplo o método **fetch** retorna a próxima linha do resultado na forma de um objeto PHP, pois é passado o parâmetro PDO::FETCH_OBJ na chamada do método. O formato do objeto é definido automaticamente com base nos nomes das colunas do resultado. Neste exemplo, o objeto possui os campos nome e tel.

Método fetchAll

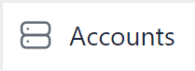
- Retorna, de uma vez, todas as linhas restantes do resultado na forma de um array de arrays associativos (ou array de objetos)
- Deve ser utilizado com cautela, pois o uso inadequado pode sobrecarregar o servidor ou a rede

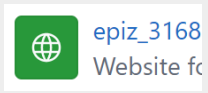
```php
$stmt = $pdo->query(
  <<<SQL
  SELECT nome, telefone
  FROM aluno
  LIMIT 30
  SQL;
);
$arrayAlunos = $stmt->fetchAll(PDO::FETCH_OBJ);

foreach ($arrayAlunos as $aluno)
  echo $aluno->nome, $aluno->telefone;
```

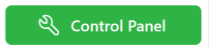
OBS: Ao invés de resgatar todos os dados e processá-los no PHP, considere utilizar o próprio servidor de banco de dados para filtrar/selecionar os dados (por exemplo utilizando WHERE, HAVING etc.)

Criando um Banco de Dados de Teste

Criando um Banco de Dados no infinityfree

1. Clique em Accounts no menu superior 

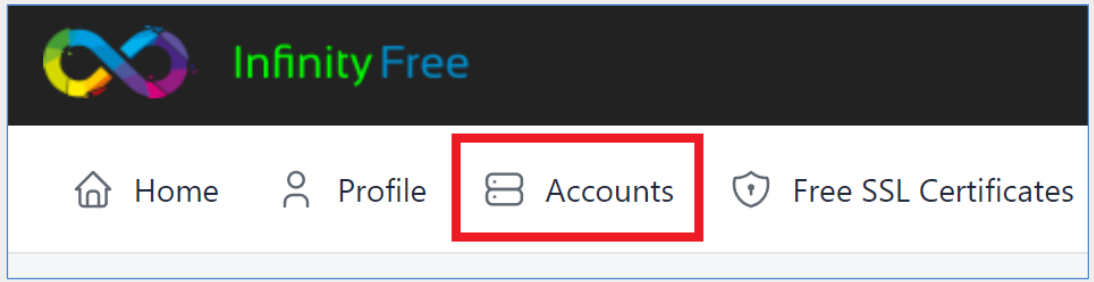
2. Selecione a conta criada anteriormente 

3. Acesse o Painel de Controle clicando em Control Panel 

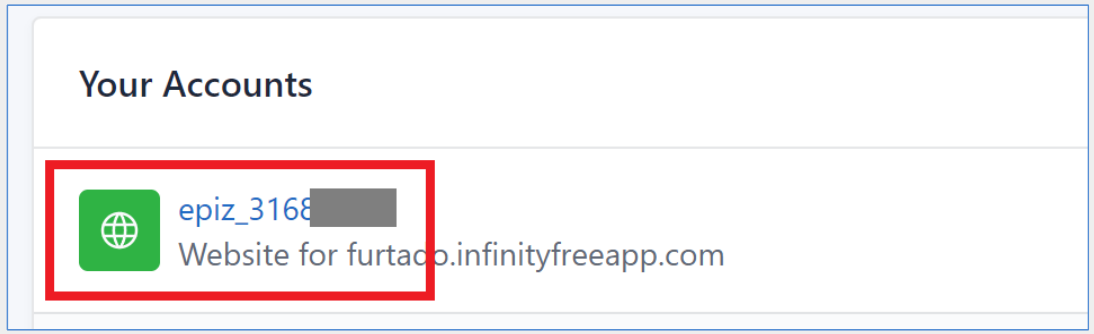
4. Dentro do grupo Databases, escolha MySQL Databases
5. Preencha o campo para criar um novo banco de dados

Criando um Banco de Dados no infinityfree

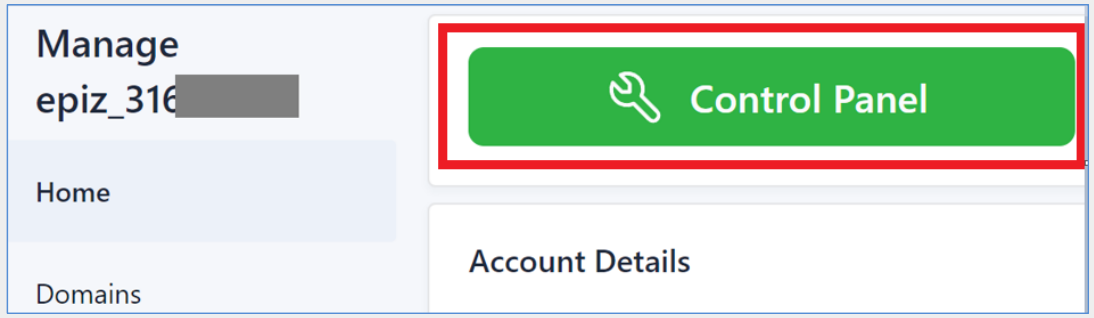
1



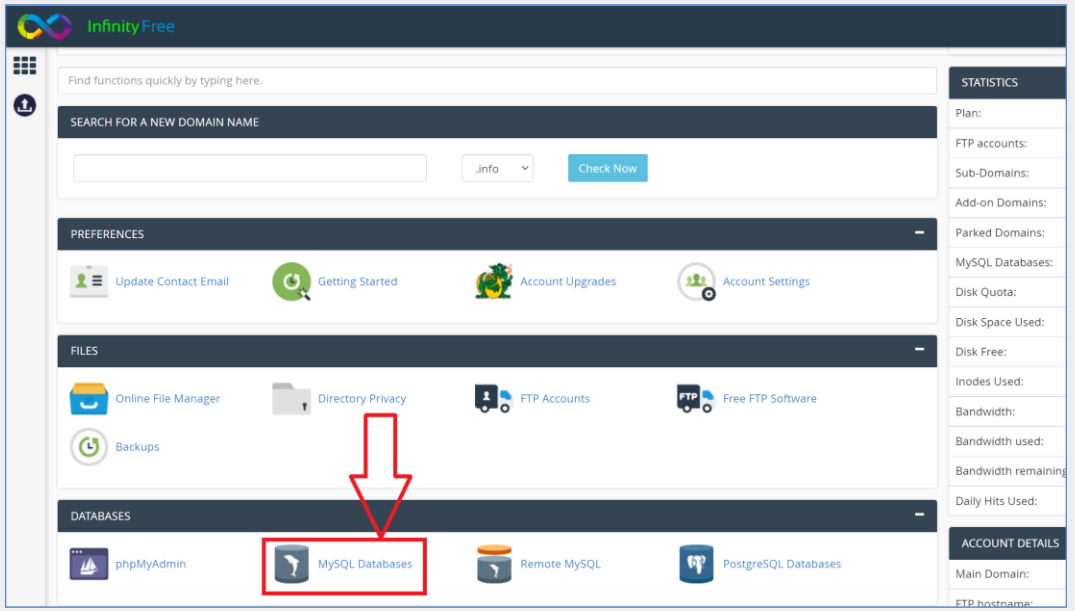
2



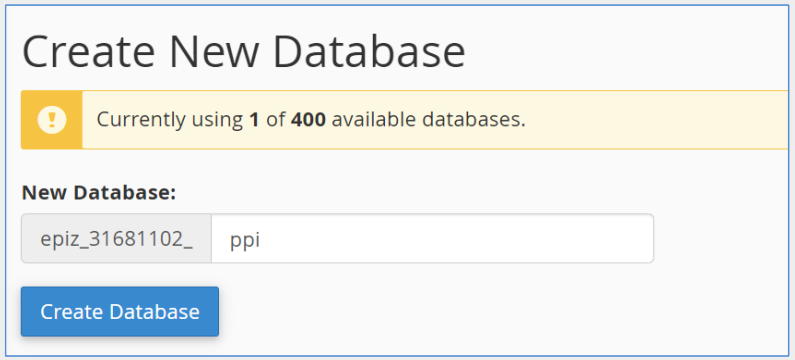
3



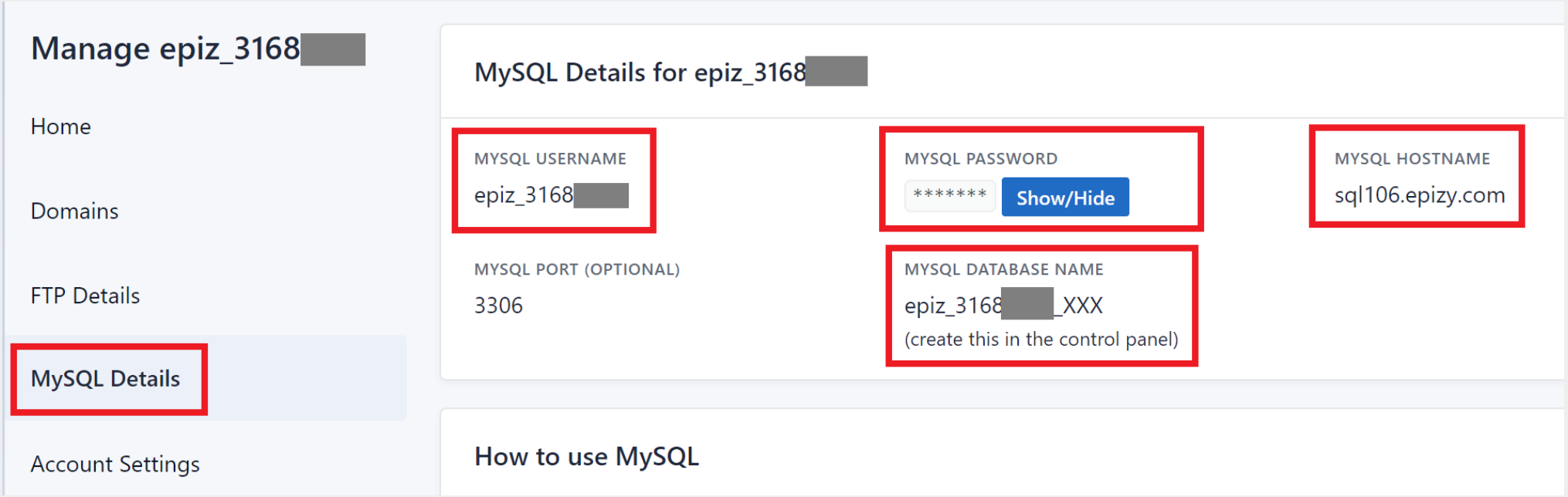
4



5

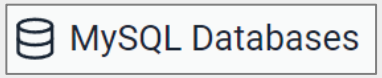


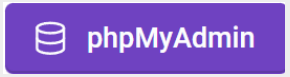
Acessando os Detalhes do Banco de Dados Criado



Esses dados serão necessários no código PHP para efetuar a conexão com o MySQL
Atenção: no nome do banco de dados, substitua o XXX por "ppi" (parte final do nome do banco de dados fornecido no momento da criação do banco de dados)

Criando Tabela de Teste com o phpMyAdmin

1. Acesse novamente Accounts → epiz_xxxx
2. No painel esquerdo, clique em 

3. E clique no botão 

na frente do nome do banco de dados
4. Na nova página, clique na aba SQL, digite o código a seguir e tecle Ctrl+Enter para executá-lo:

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

5. Insira linhas na tabela e visualize os dados utilizando o código a seguir:

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

```
SELECT * FROM aluno;
```

Exemplo de Função PHP para Conexão com o MySQL

```php
function mysqlConnect() {
  $host = "host do mysql"; // veja dados de conexão no slide 6
  $username = "usuario no mysql"; // veja dados de conexão no slide 6
  $password = "senha do usuario mysql"; // veja dados de conexão no slide 6
  $dbname = "nome do banco de dados"; // veja dados de conexão no slide 6

  $options = [
    PDO::ATTR_EMULATE_PREPARES   => false,  // desativa a execução emulada de prepared statements
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC, // altera o modo de busca padrão para FETCH_ASSOC
    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION // para que exceções sejam lançadas (padrão no PHP > 8.0)
  ];

  try {
    // Efetua a conexão com o MySQL. O objeto $pdo será utilizado posteriormente nas operações.
    $pdo = new PDO("mysql:host=$host; dbname=$dbname; charset=utf8mb4", $username, $password, $options);
    return $pdo;
  } catch (Exception $e) {
    exit('Falha na conexão com o MySQL: ' . $e->getMessage());
  }
}
```

Códigos de Exemplo

https://furtado.prof.ufu.br/site/teaching/PPI/exemplos/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