Programação para Internet

Módulo 7

Websites Dinâmicos com Acesso a Banco de Dados (MySQL)

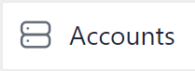
Prof. Dr. Daniel A. Furtado - FACOM/UFU

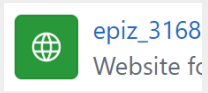
Conteúdo do Módulo

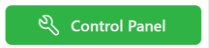
- Criação de um banco de dados gratuito no infinityfree.net
- Formas de comunicação do PHP com o MySQL: MySQLi x PDO
- Conexão com o MySQL
- Execução de Declarações SQL: métodos fetch, fetchAll, query e exec
- Injeção de SQL e declarações preparadas (prepared statements)
- Transações

Criando um Banco de Dados de Teste

Criando um Banco de Dados no infinityfree

1. Clique em Accounts no menu superior 

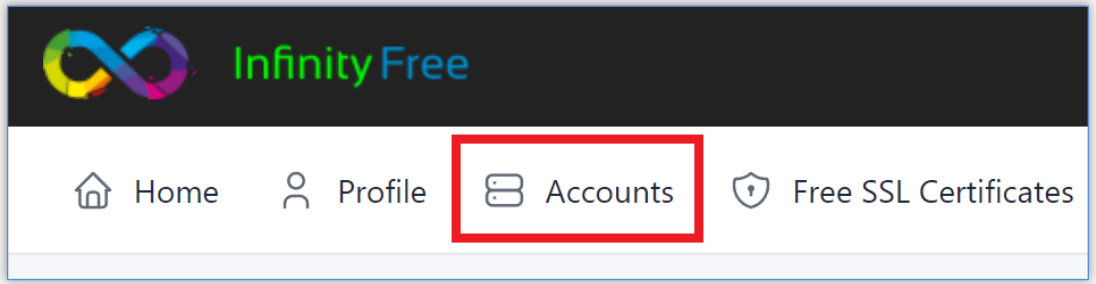
2. Selecione a conta criada anteriormente 

3. Acesse o Painel de Controle clicando em Control Panel 

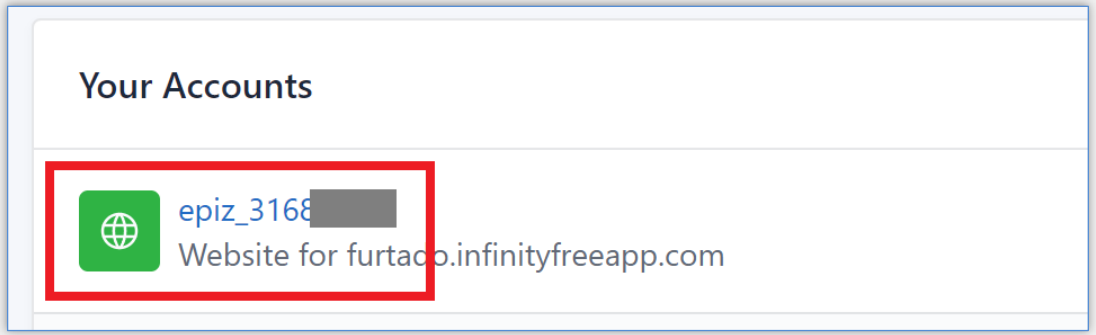
4. Dentro do grupo Databases, escolha MySQL Databases
5. Preencha o campo para criar um novo banco de dados

Criando um Banco de Dados no infinityfree

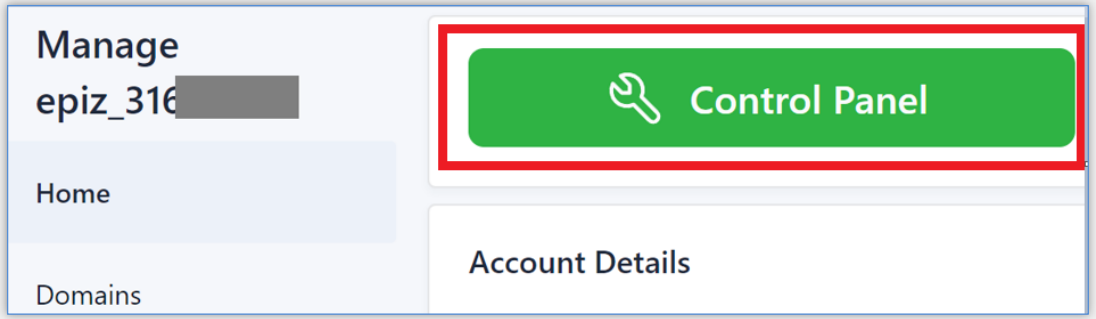
1



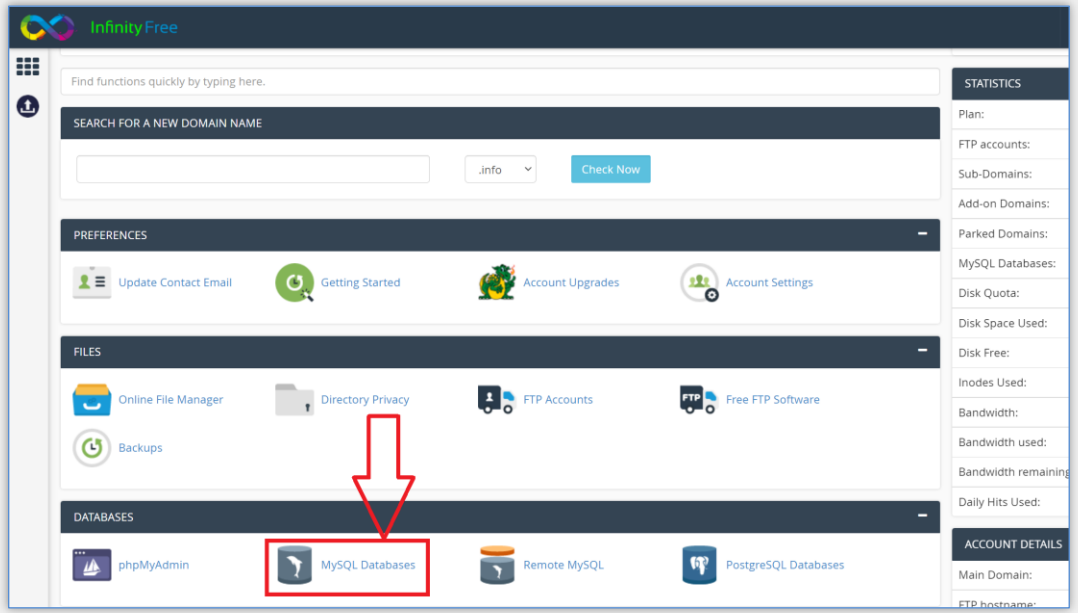
2



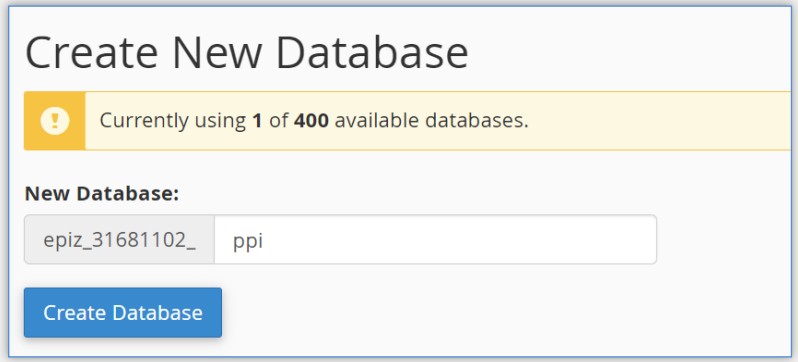
3



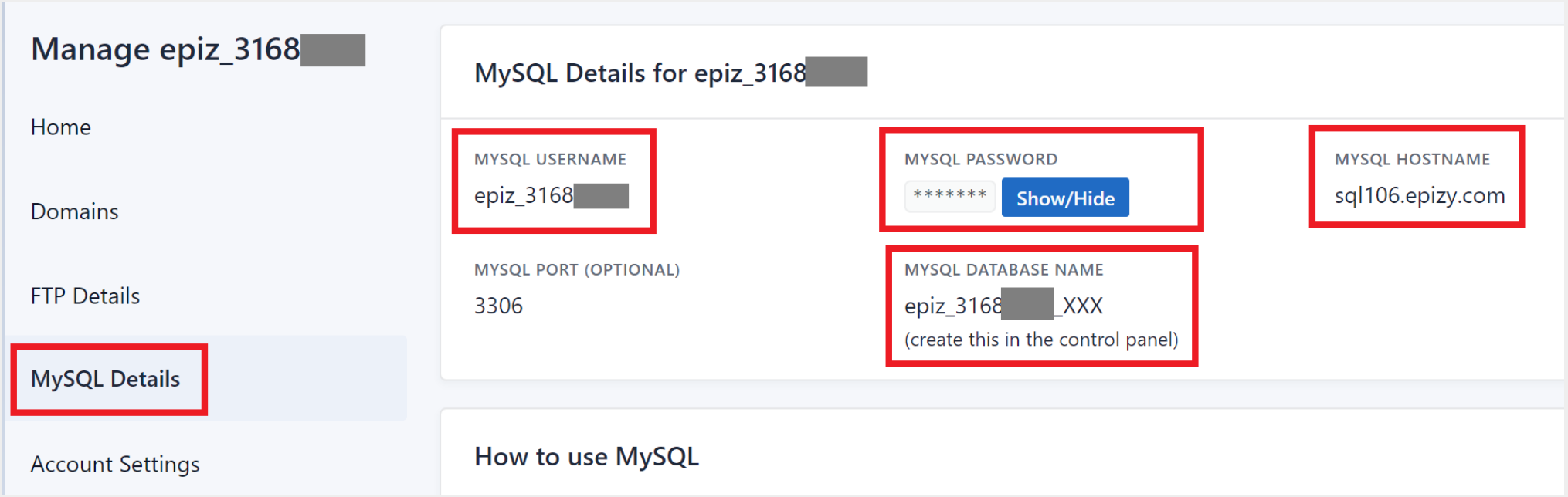
4



5

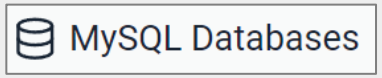


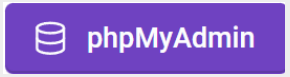
Acessando os Detalhes do Banco de Dados Criado



Esses dados serão necessários no código PHP para efetuar a conexão com o MySQL
OBS: 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;
```

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
- Exemplos: 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

```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 {
    // 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

```php
$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)

```php
$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)

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

Utilize quando **há** a possiblidade de injeção de SQL, ou seja, quando o código SQL inclui parâmetros com dados produzidos pelo usuário (vindos de campos 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, o qual disponibiliza métodos para resgatar os dados retornados pela declaração SQL
- O método **fetch** do objeto PDOStatement retorna a próxima linha do resultado (ou falso, quando não há mais linhas a serem processadas)

```
$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 pelo *alias* produzido na consulta SQL). 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.

Método query

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

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

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

O formato dos dados retornado pelo método fetch pode ser alterado com parâmetros. Por exemplo, `$stmt->fetch(PDO::FETCH_OBJ)` retornará a próxima linha do resultado como um objeto PHP.

Exemplo Simples de Listagem dos Dados da Tabela Aluno

```
<html>
<body>
  <table>
    <tr><th>Nome</th><th>Telefone</th></tr>
    <?php
    require "conexaoMysql.php";
    $pdo = mysqlConnect();
    $stmt = $pdo->query("SELECT nome, telefone FROM aluno");
    while ($row = $stmt->fetch()) {
      $nome = htmlspecialchars($row['nome']);
      $telefone = htmlspecialchars($row['telefone']);
      echo <<<HTML
        <tr>
          <td>$nome</td>
          <td>$telefone</td>
        </tr>
      HTML;
    }
    ?>
  </table>
</body>
</html>
```

Este exemplo simplificado não faz qualquer tratamento de erros. Apenas gera uma página HTML dinâmica simples listando os dados da tabela aluno em uma tabela HTML.

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
$sql = <<<SQL
  SELECT nome, telefone
  FROM aluno
  LIMIT 30
SQL;

$stmt = $pdo->query($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 ou melhor selecionar os dados (por exemplo utilizando WHERE, HAVING, etc.)

Método fetchColumn

- Para resgatar uma **única coluna da linha**, na forma de uma escalar, pode-se utilizar o método fetchColumn

```
$sql = <<<SQL
  SELECT COUNT(*) as numAlunos
  FROM aluno
  SQL;

$stmt = $pdo->query($sql);
$numAlunos = $stmt->fetchColumn();
```

```
$sql = <<<SQL
  SELECT nome
  FROM aluno
  SQL;

$stmt = $pdo->query($sql);
while ($nome = $stmt->fetchColumn())
  echo $nome;
```

OBS 1: É possível passar um índice numérico na chamada de fetchColumn para indicar o número da coluna desejada. Quando não informado, a função retorna o valor da 1ª coluna.

OBS 2: fetchColumn não deve ser usado para recuperar colunas com valores booleanos, pois não seria possível distinguir o valor lógico da coluna do valor false que é retornado quando não há mais linhas a serem processadas.

Conteúdo Complementar

Método fetchAll

- O método fetchAll, quando utilizado em conjunto com PDO::FETCH_COLUMN, retorna todas as linhas da coluna na forma de um array simples

$arrayEspecialidades será um array simples, indexado por números

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

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

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

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

Conteúdo Complementar

Método exec

- O método exec do objeto $pdo executa uma declaração SQL e retorna o número de linhas afetadas
- Pode ser usado 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);
```

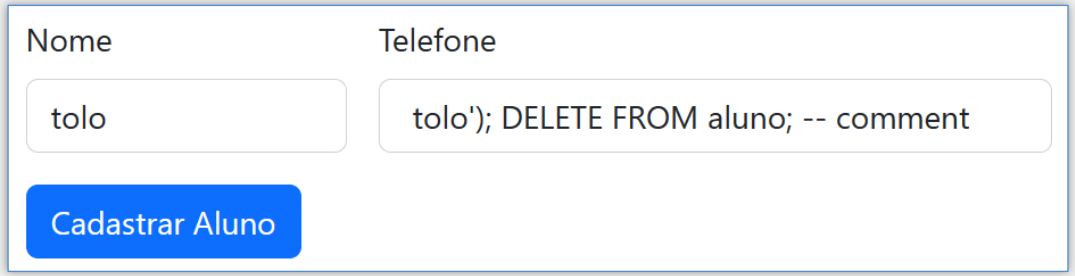
Injeção de SQL e Prepared Statements

Injeção de SQL (*SQL Injection*)

- Técnica utilizada por usuários maliciosos para injetar código SQL ilícito 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 até mesmo exclusões no banco de dados, **sem qualquer autorização**

Exemplo de Código Vulnerável a Injeção de SQL

Formulário de cadastro



Exemplo de código PHP vulnerável em inserção de dados

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

Não faça isso!

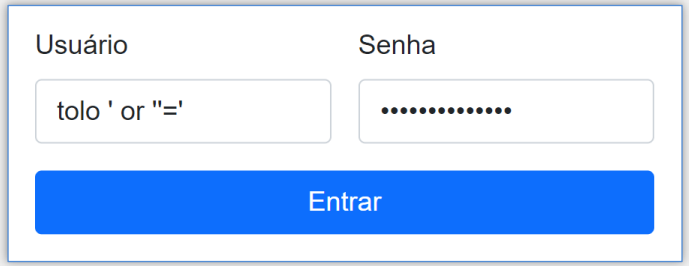
Neste exemplo um usuário malicioso conseguiria injetar, pelo campo telefone, um código SQL para excluir todo o conteúdo da tabela aluno

Expressão SQL resultante após avaliação do PHP

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

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 ''=''
```

Conteúdo Complementar

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 ou da própria URL)
- A técnica é suportada por diversos SGBDs e linguagens de programação
  - Não é um recurso específico do PHP

Prepared Statements

- Com prepared statements, os dados dos parâmetros são passados **separadamente** da declaração SQL (isolamento declaração/dados)
- 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
  - A declaração SQL pode ser planejada uma única vez pelo SGBD

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

```php
$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'];

}
```

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

*Método **prepare***
Prepara a declaração SQL utilizando o "template" da declaração, sem a presença dos dados do usuário. Observe que nesse momento há apenas parâmetros/placeholders (?) para os dados. Retorna um objeto do tipo PDOStatement

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

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

```php
$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**

```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.

Conteúdo Complementar

Número de Linhas Afetadas

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

- O método 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)

Conteúdo Complementar

Transações

Transações

- Sequência de operações "indivisíveis"
  - 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 iniciadas após begin transaction

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

```
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 nas 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"] ?? "";

$sql1 = <<<SQL
  INSERT INTO Cliente (nome, cpf)
  VALUES (?, ?)
  SQL;

$sql2 = <<<SQL
  INSERT INTO EnderecoCliente (cep, cidade, id_cliente)
  VALUES (?, ?, ?)
  SQL;
```

```php
try {
  $pdo->beginTransaction();

  // Prepara e executa a primeira inserção
  // Uma exceção será lançada em caso de falha
  $stmt1 = $pdo->prepare($sql1);
  $stmt1->execute([$nome, $cpf]);

  // Prepara e executa a segunda inserção
  // Uma exceção será lançada em caso de falha
  $idNovoCliente = $pdo->lastInsertId();
  $stmt2 = $pdo->prepare($sql2);
  $stmt2->execute([$cep, $cidade, $idNovoCliente]);

  $pdo->commit();
}
catch (Exception $e) {
  $pdo->rollBack();
}
```

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

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