



Programação para Internet

Módulo 6

Websites com Bancos de Dados - MySQL e PHP

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

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

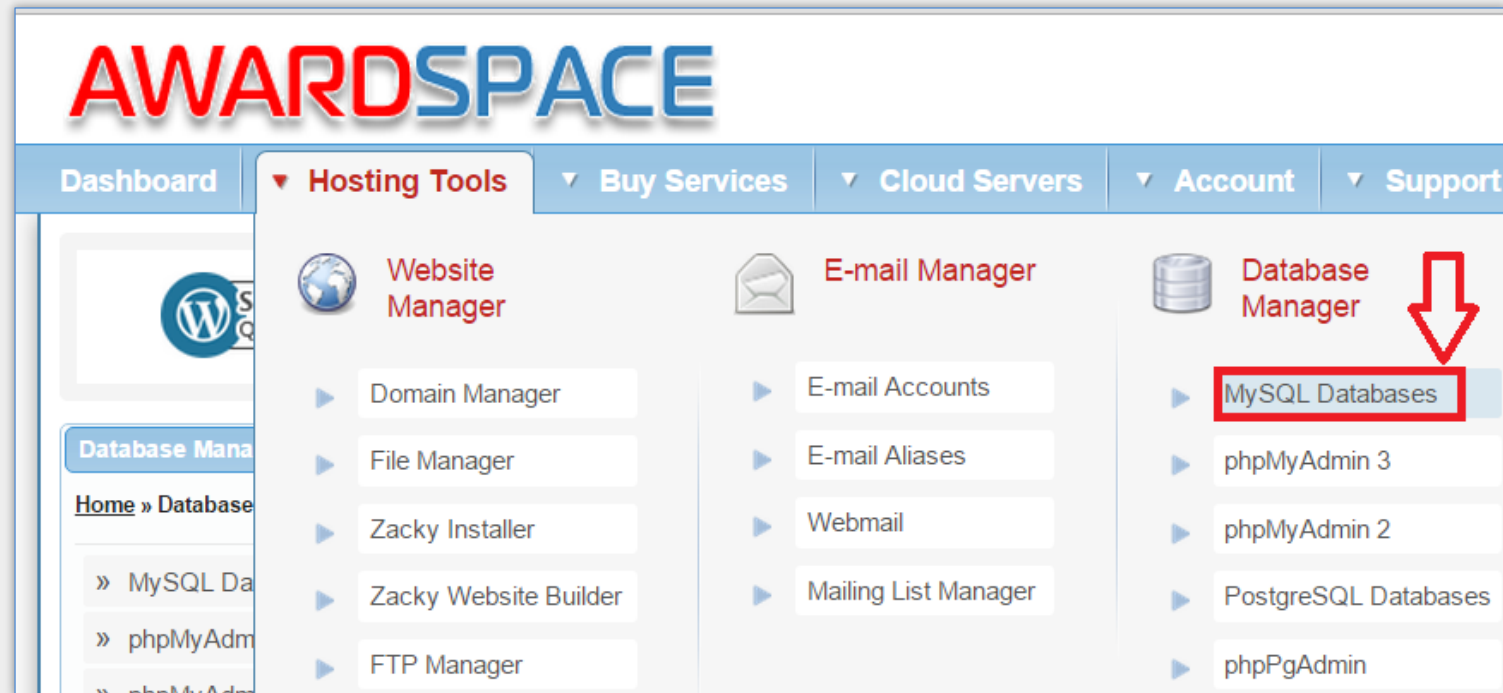
A cópia, reprodução ou apropriação deste material, total ou parcialmente, é proibida pelo autor

Conteúdo da Aula

1. Criação de um banco de dados gratuito no *awardspace.com*
2. Comunicação com o MySQL: *MySQLi* x *PDO*
3. Conexão utilizando o *PDO*
4. *SQL Injection* e declaração preparadas com *PDO*
5. Métodos *fetch*, *fetchAll*, *query* e *exec*
6. Transações com *PDO*

Criando um Banco de Dados de Teste no *awardspace.com*

Criando um Banco de Dados no *Awardspace*



Criando um Banco de Dados no *Awardspace*

The screenshot shows the 'Create MySQL Database' form in the Awardspace control panel. The interface includes a sidebar with 'Database' and 'Section Information' tabs. Under 'Database', there are two buttons: 'Create MySQL Database' (highlighted with a red box and a red arrow pointing to it) and 'Create PostgreSQL Database'. The main form area is titled 'Create MySQL Database' and contains several input fields: 'Database Version' (set to 5.5), 'Database Name' (containing '1875689_' and highlighted with a red box), 'Database Password' (with a placeholder '(8-32 alphanumeric chars)' and a red arrow pointing to it), 'Confirm Database Password' (with a placeholder '(8-32 alphanumeric chars)'), and 'Storage Engine' (set to MyISAM). A 'Create Database' button is at the bottom right. A text box on the left contains a note in Portuguese: 'O nome do banco e a senha são necessários no momento de realizar a conexão com o MySQL utilizando o PHP'.

Database Section Information

Create MySQL Database

Create PostgreSQL Database

Create MySQL Database

Database Version 5.5

Database Name 1875689_

Database Password (8-32 alphanumeric chars)

Please enter a password.

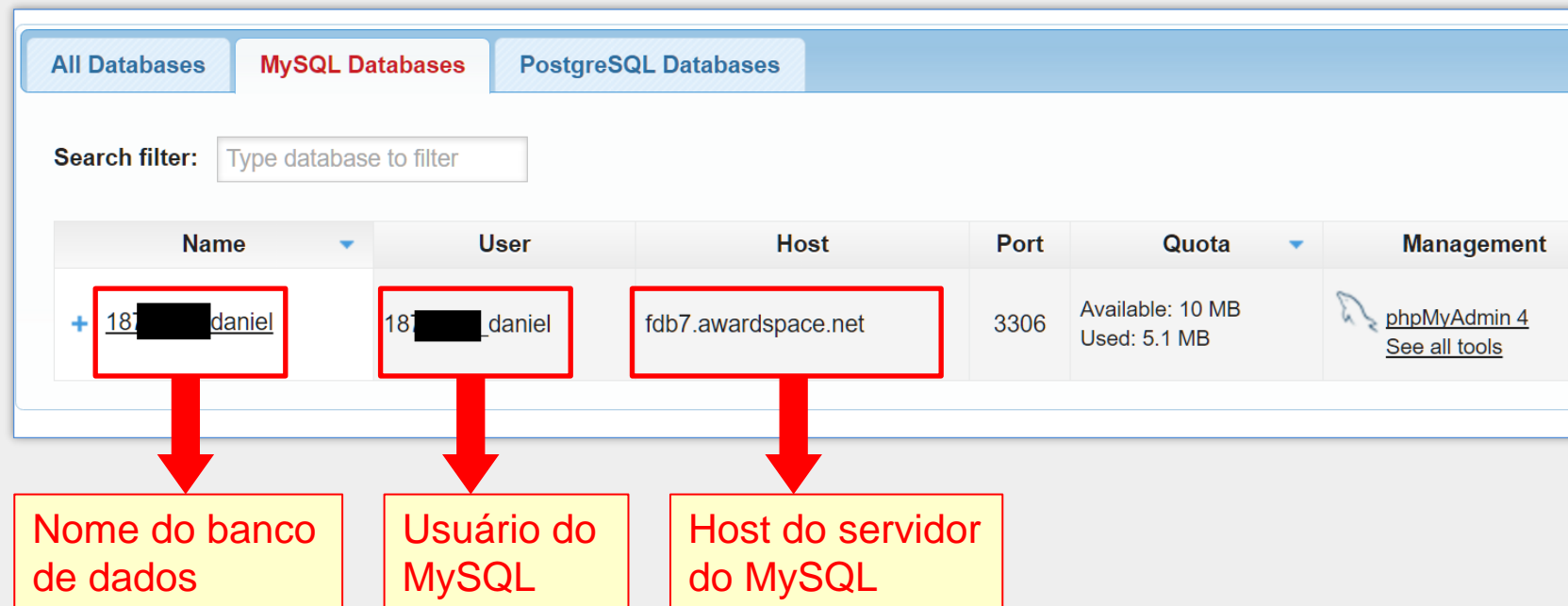
Confirm Database Password (8-32 alphanumeric chars)

Storage Engine MyISAM

Create Database


O nome do banco e a senha são necessários no momento de realizar a conexão com o MySQL utilizando o PHP

Criando um Banco de Dados no *Awardspace*




The screenshot shows the 'MySQL Databases' tab in the Awardspace control panel. A search filter is present above a table of databases. The first row of the table is highlighted with red boxes around the 'Name', 'User', and 'Host' columns. Red arrows point from these boxes to three yellow boxes below the table, which contain the following text:

- Nome do banco de dados
- Usuário do MySQL
- Host do servidor do MySQL

Name	User	Host	Port	Quota	Management
+ 18[redacted]daniel	18[redacted]daniel	fdb7.awardspace.net	3306	Available: 10 MB Used: 5.1 MB	 phpMyAdmin 4 See all tools

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

Criando um Banco de Dados no *Awardspace*

<div>All Databases MySQL Databases PostgreSQL Databases</div>					
Search filter: <input type="text" value="Type database to filter"/>					
Name	User	Host	Port	Quota	Management
+ 187[REDACTED]daniel	187[REDACTED]daniel	fdb7.awardspace.net	3306	Available: 10 MB Used: 5.1 MB	 phpMyAdmin 4 See all tools

Clique em **phpMyAdmin4** para acessar o banco de dados a partir do navegador (para criação de tabelas, realizar testes, etc.)

Criando um Banco de Dados no *Awardspace*

The screenshot shows the phpMyAdmin interface. On the left, the 'Recent' tab is active, showing the database '1875689_daniel'. A red arrow points to this database name with the instruction '1. Clique aqui para ativar o banco de dados'. On the right, the 'SQL' tab is active, showing a text area for entering SQL queries. The instruction '2. Coloque o código SQL aqui' is written in red text in the text area. The top of the interface shows the server 'fdb7.awardspace.net' and the database '1875689_daniel'.

1. Clique aqui para ativar o banco de dados

2. Coloque o código SQL aqui

MySQL com PHP

Interfaces 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.

PHP Data Objects - PDO

Conexão

```
$db_host = "nome_host_mysql";
$db_username = "seu_usuario_no_mysql";
$db_password = "senha_do_usuario";
$db_name = "nome_do_banco_de_dados";

$dsn = "mysql : host=$db_host; dbname=$db_name; charset=utf8mb4";

$options = [
    PDO::ATTR_EMULATE_PREPARES => false,
    PDO::ATTR_ERRMODE           => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
];

try {
    $pdo = new PDO($dsn, $db_username, $db_password, $options);
}
catch (Exception $e) {
    exit('Falha na conexão com o MySQL: ' . $e->getMessage());
}
```

Formas de Executar Código SQL

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

Utilize quando o código SQL inclui dados fornecidos pelo usuário (formulário, URL, etc.)

```
$pdo->exec("Código SQL");
```

Utilize quando não há possibilidade de SQL Injection* e o código SQL **não** retorna conteúdo

```
$pdo->query("Código SQL");
```

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

*veja sobre **SQL Injection** no próximo slide

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 executar consultas, atualizações e exclusões, **sem qualquer autorização**

Exemplo de SQL Injection

Exemplo de 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"];  
$sql = <<<SQL  
    SELECT * FROM usuarios  
    WHERE user = '$usuario' AND senha = '$senha'  
SQL;  
$pdo->exec($sql);
```

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

String SQL resultante após avaliação do PHP

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

Prepared Statements

- Técnica que permite executar operações SQL com risco de injeção de forma mais segura
- Indicada quando a operação SQL inclui parâmetros produzidos pelo usuário, como em campos de formulário ou parâmetros da URL
- Técnica suportada por diversos SGBDs

Prepared Statements

- Os dados são passados separadamente da declaração SQL
- Dessa forma, não é possível alterar a estrutura em si da operação SQL por meio dos parâmetros (como no exemplo anterior)
- Elimina a necessidade de utilizar aspas nos parâmetros
- Também é mais eficiente quando se deseja executar a mesma operação múltiplas vezes, variando apenas os dados
 - O código SQL pode ser preparado uma única vez pelo SGBD

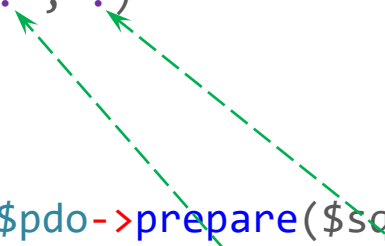
Prepared Statements - Exemplo de Uso - insert

Ponto de Interrogação

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

```
$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());
}
```



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.

As variáveis *\$nome* e *\$idade* podem ser iniciadas com valores de campos de formulário, por exemplo

Prepared Statements - Exemplo com **bindParam**

```
$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();
```

*Variáveis podem ser vinculadas aos parâmetros em aberto (?, ?) com o método **bindParam**, possibilitando múltiplas inserções de forma prática e mais eficiente.*

*Um bloco **try-catch** não foi utilizado para simplificar o exemplo.*

Prepared Statements - Exemplo com `bindParam`

```
$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 anterior que utiliza o marcador '?'

Prepared Statements - **select** e **fetch**

```
$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 fetch

*retorna a próxima linha do resultado na forma de um array associativo (ou **falso** quando não há mais linhas)*

Prepared Statements - **fetch**

É possível indicar outras formas de retorno 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);

// retorna apenas um escalar
$stmt->fetch(PDO::FETCH_COLUMN);
```

PDO::FETCH_COLUMN - Exemplo

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

```
$sql = <<<SQL
    SELECT COUNT(*)
    FROM paciente
    WHERE altura > ?
SQL;

$stmt = $pdo->prepare($sql);
$stmt->execute([190]);

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

Método exec

- Deve ser usado quando:
 1. não há a possibilidade de injeção de SQL e
 2. a declaração SQL **não** retorna dados a serem processados
- **exec** retorna apenas o número de linhas afetadas

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

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


Método query

- Deve ser usado quando:
 1. não há a possibilidade de injeção de SQL e
 2. a declaração SQL **retorna** dados a serem processados
- `query` retorna um objeto do tipo PDOStatement

```
$sql = <<<SQL
      SELECT nome, duracao
      FROM curso
      SQL;

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

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

Método **fetchAll** com **PDO::FETCH_COLUMN**

Busca de uma única coluna na forma de um array

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

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

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

\$especialidades será um array simples, indexado por números

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 (apenas em alguns SGBDs)

Transações

- Sequência de operações "indivisíveis"
- Ou executa todas ou não executa nenhuma
- **rollback**: desfaz as operações iniciadas
- **commit**: efetiva as operações

Transações - Exemplo 1

- Operação para mover dados de uma tabela para outra
- Envolve uma sequência de ações "indivisíveis":
 1. Inserir o dado na nova tabela
 2. Excluir o dado da tabela antiga
- Não é permitido que o dado permaneça nas duas tabelas

Transações - Exemplo 2

- Cadastro de cliente com inserção em duas tabelas
 - Dados pessoais na tabela `cliente`
 - Dados do endereço na tabela `endereco_cliente`
- Não se deve, jamais, permite o cadastro parcial
 - Dados pessoais do cliente, sem os dados do endereço

Transações

```
try {  
    // início da transação  
    $pdo->beginTransaction();  
  
    $stmt = $pdo->prepare('INSERT INTO B VALUES (?)');  
    if (! $stmt->execute([100]))  
        throw new Exception('Falha na operação 1');  
  
    $stmt = $pdo->prepare('DELETE FROM A WHERE ID = ?');  
    if (! $stmt->execute([100]))  
        throw new Exception('Falha na operação 2');  
  
    // se nenhuma excecao foi lancada, efetiva as operacoes  
    $pdo->commit();  
}  
catch (Exception $e)  
{  
    // desfaz as operacoes em caso de erro (exceção lançada)  
    $pdo->rollBack();  
    exit('Falha na transação: ' . $e->getMessage());  
}
```

Último ID Inserido

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

- Comumemente utilizado em colunas do tipo `auto_increment`
- Retorna o último código/id inserido automaticamente na linha
- Comumente usado ao inserir valor de chave estrangeira em tabela vinculada

Referências

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