Universidade Federal de Uberlândia
Faculdade de Computação

Programação para Internet

Módulo 8

Web Dinâmica com PHP e MySQL

Prof. Dr. Daniel A. Furtado

PHP e MySQL

Dependendo da versão do PHP são disponibilizadas duas ou três APIs (módulos ou extensões) para comunicação com o MySQL:

- MySQL *Extension* (obsoleta, retirada do PHP 7)
  - Suportada apenas por versões anteriores do PHP. Não utilize para novos projetos!
- MySQLi *Extension* (MySQL ***Improved***)
  - Provê acesso às funcionalidades do MySQL 4.1 e superior
  - Disponibiliza uma coleção de métodos para comunicação especificamente com o MySQL
- PHP *Data Objects* (PDO) *Extension*
  - Provê uma interface "única" e consistente para acessar diversos SGBDs (MySQL, PostgreSQL, Oracle, Firebird, etc) ;
  - PDO suporta 12 sistemas de gerenciamento de banco de dados

MySQLi vs PDO

- **Ambas as tecnologias:**
  - Possuem interface de comunicação orientada a objetos;
  - Suportam ***prepared statements***. *Prepared Statements* protegem contra SQL *injection*, e são muito importantes para a segurança da aplicação;
  - Suportam o conceito de ***transações.***
- **PDO**
  - Possui a vantagem de utilizar basicamente a mesma sintaxe para comunicação com os diversos SGBDs suportados;
  - Assim, caso seja necessário adaptar o projeto para utilizar um SGBD diferente, a alteração é mais rápida. Com MySQLi pode ser necessário reescrever uma parte considerável do código;
- **MySQLi**
  - Possui a vantagem de ser otimizado especificamente para o MySQL, oferecendo um melhor desempenho quando comparado ao PDO.

PHP e Bancos de Dados

- O acesso a bancos de dados utilizando o PHP geralmente envolve três etapas:
  1. Conexão com o servidor de banco de dados utilizando um nome de usuário e uma senha (e seleção de um banco de dados);
  2. Execução das operações de dados necessárias, como inserções, atualizações e consultas no banco de dados;

     Para o caso de uma operação de consulta, deve-se processar os resultados.
  3. Encerramento da conexão.

Acessando o **MySQL com MySQLi**

1. Conexão com o servidor de banco de dados MySQL

```php
<?php
$servidor = "localhost";
$usuario  = "usuario";
$senha    = "senha";
$nomeBD   = "nomeBancoDeDados";

// Inicia uma nova conexão com o servidor MySQL.
// Em caso de sucesso na conexão, a variável $conn será
// ser utilizada posteriormente para manipulação do banco
// de dados através dessa conexão
$conn = new mysqli($servidor, $usuario, $senha, $nomeBD);

// Verifica se ocorreu alguma falha durante a conexão
if ($conn->connect_error)
   die("Falha na conexão com o MySQL: " . $conn->connect_error);
else
   echo "Conectado ao MySQL";
?>
```

Acessando o **MySQL com MySQLi**

2. Execução das operações de dados necessárias utilizando SQL

```php
<?php
// Define o código SQL referente à operação
// a ser executada
$sql = "SELECT * FROM..." ou "INSERT INTO...", etc.

// Executa a operação e verifica se
// ocorreu alguma falha
$resultado = $conn->query($sql)
if ($resultado)
   echo "Operacao realizada com sucesso!";
else
   echo "Erro ao executar: "  . $conn->error;
?>
```

O método ***query*** retorna:

- **FALSE**, caso ocorra algum erro na operação;
- **TRUE**, no caso de sucesso em operações que não retornam um resultado (como INSERT e UPDATE);
- Ou um ***objeto*** que dá acesso ao resultado da operação (caso da operação *SELECT,* por exemplo)

Acessando o **MySQL com MySQLi**

2.1. Processando os resultados de consultas

```php
<?php

// Verifica se a consulta retornou algum resultado
if ($resultado->num_rows > 0)
{
    // Navega pelo resultado da consulta, linha a linha.
    // O metodo fetch_assoc constroi um array associativo
    // para a linha corrente do resultado.
    while ($row = $resultado->fetch_assoc())
    {
        // processar cada linha do resultado
        // echo $row["NomeColuna"]...
    }
}
else
    echo "Nenhum dado encontrado...";
?>
```

Nota: No lugar de **fetch_assoc**, também é possível utilizar ***fetch_row***. Neste caso, um array convencional é retornado e os campos individuais da tupla podem ser acessados por um índice: $row[0], $row[1], etc.

Acessando o **MySQL com MySQLi**

3. Encerrando a conexão com o servidor do MySQL

```php
<?php

// Encerra a conexao com o MySQL
$conn->close();

?>
```

Acessando o **MySQL com MySQLi**

Alguns cuidados que devem ser tomados ao executar operações SQL com PHP:

- A declaração SQL deve ser criada como uma string, entre aspas;
- Strings dentro da declaração SQL devem aparecer entre aspas simples;
- Não utilize aspas para valores numéricos;
- Não utilize aspas para a palavra NULL;
- Strings contendo **datas** devem aparecer entre aspas;

Acessando o **MySQL com *MySQLi*** - Exemplo

```php
<?php

// Dados de conexão com o MySQL
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Cria uma conexão com o MySQL
$conn = new mysqli($servername, $username, $password, $dbname);
if ($conn->connect_error)
    die("Connection failed: " . $conn->connect_error);

// Define a operação SQL que deve ser executada
$sql = "
    INSERT INTO Cliente(Nome, Email, Idade)
    VALUES ('Paulo', 'paulo@mail.com', 20)
";

// Executa a operação
if ($conn->query($sql)
    echo "Dados inseridos com sucesso!";
else
    echo "Erro na operação: " . $sql . "<br>" . $conn->error;

// Encerra a conexão com o MySQL
$conn->close();

?>
```

Acessando o **MySQL com PDO** - Exemplo

```php
<?php

// Dados de conexão com o MySQL
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

try {
   // Estabelece a conexão com o MySQL e
   // define o modo de tratamento de erros do PDO para “lançar exceções”
   $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
   $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

   // Define a operação SQL que deve ser executada
   $sql = "
      INSERT INTO Cliente(Nome, Email, Idade)
      VALUES ('Paulo', 'paulo@mail.com', 20)
   ";

   // Executa a operação SQL com exec()
   $conn->exec($sql);
   echo "Operação realizada com sucesso";
}
catch (PDOException $e) {
   echo "A operação não pode ser executada: " . $e->getMessage();
}

$conn = null;

?>
```

Dados de Conexão em Arquivo Separado

Quando vários arquivos PHP acessam o banco de dados, pode-se colocar os dados de conexão separadamente em outro arquivo PHP e fazer inclusão do mesmo utilizando a declaração ***include*** ou ***require***.

```php
<?php  // arquivo conexaoMysql.php

define("HOST", "IP FORNECIDO PELO PROFESSOR");
define("USER", "ppi");
define("PASSWORD", "ppi");
define("DATABASE", "ppi");

function conectaAoMySQL()
{
  $conn = new mysqli(HOST, USER, PASSWORD, DATABASE);
  if ($conn->connect_error)
    throw new Exception('Falha na conexão com o MySQL: ' . $conn->connect_error);

  return $conn;
}

?>
```

```php
<?php
  require "conexaoMysql.php";
  $conn = conectaAoMySQL();
?>
```

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

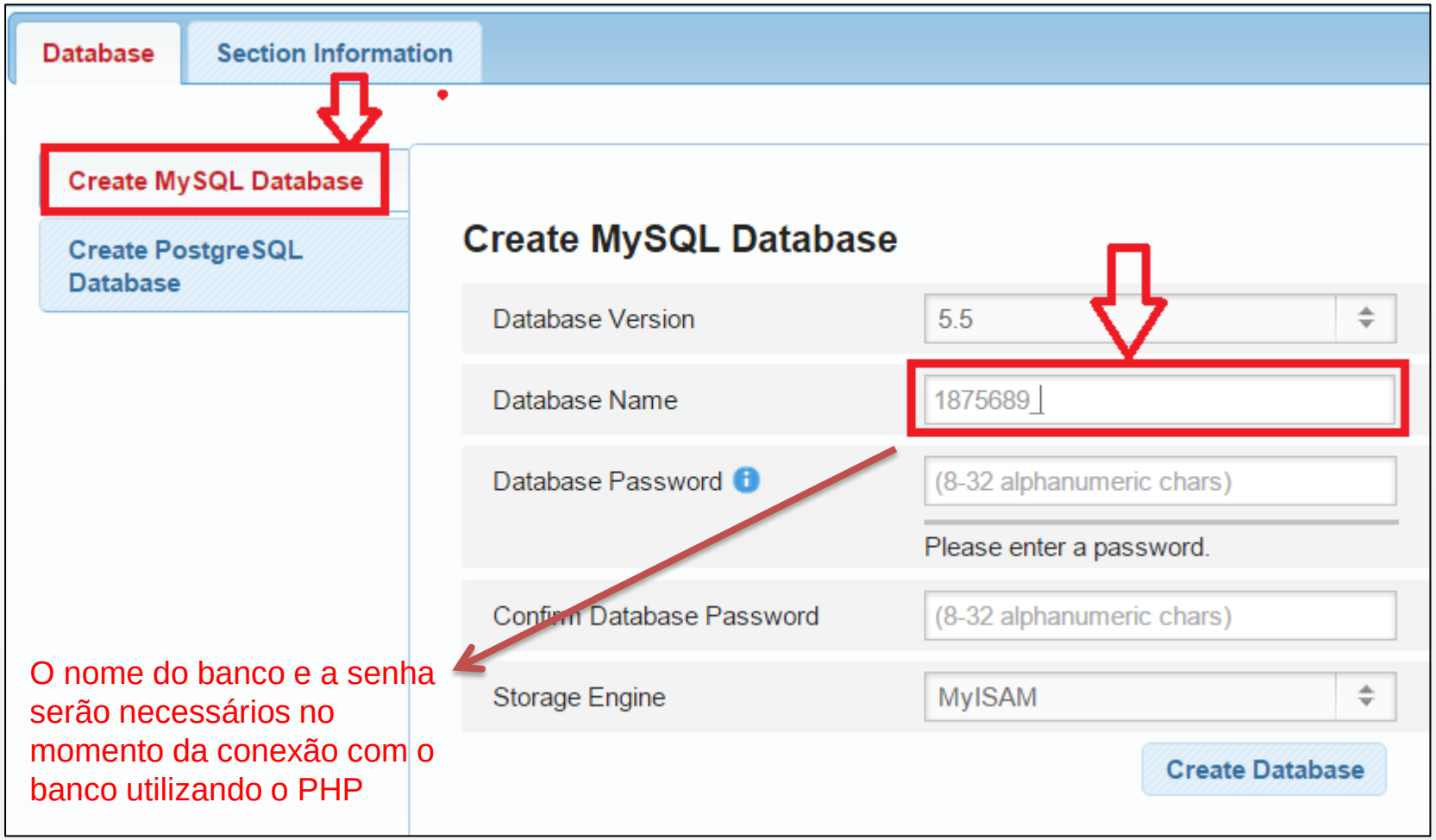
Utilizando o phpMyAdmin

Criando um Banco de Dados no *Awardspace.com*

- Faça *login* e acesse ***MySQL Databases***

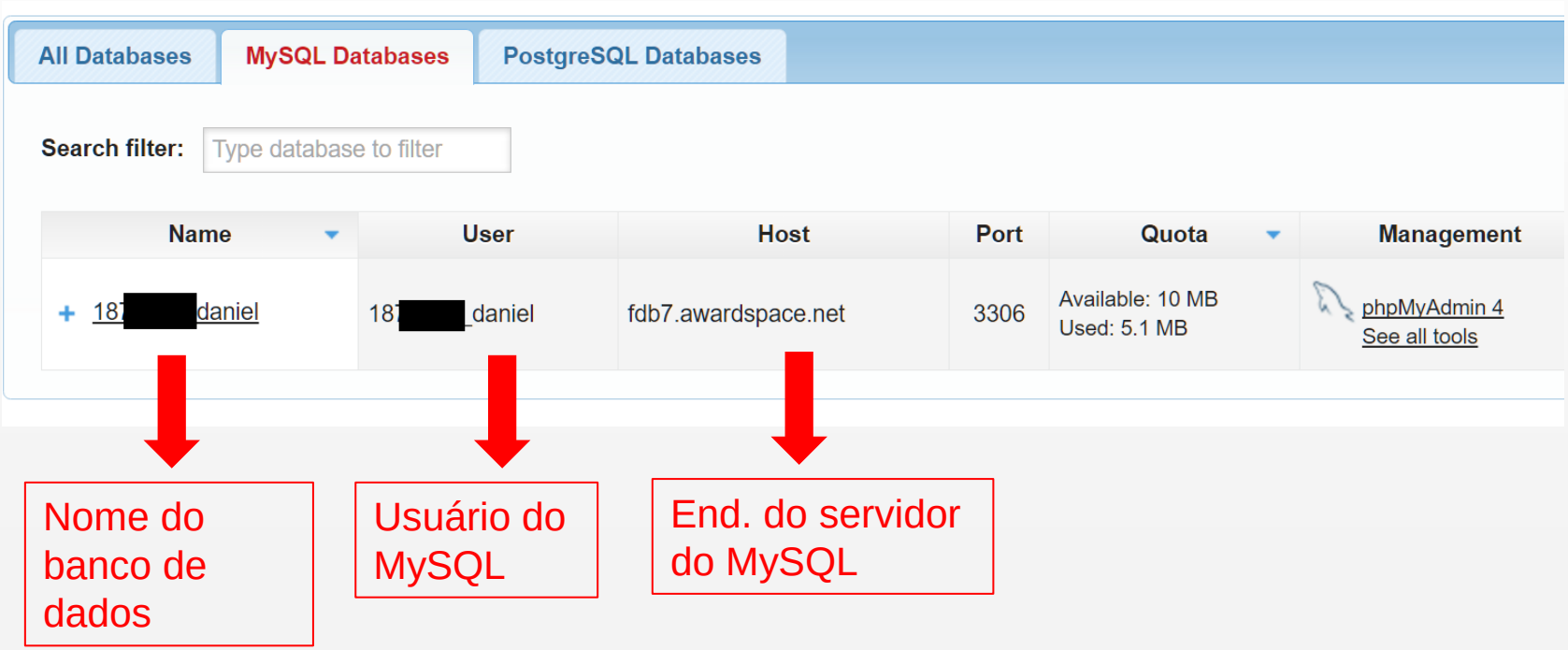
Criando um Banco de Dados no *Awardspace.com*

- Forneça um nome para o BD e uma senha de acesso:



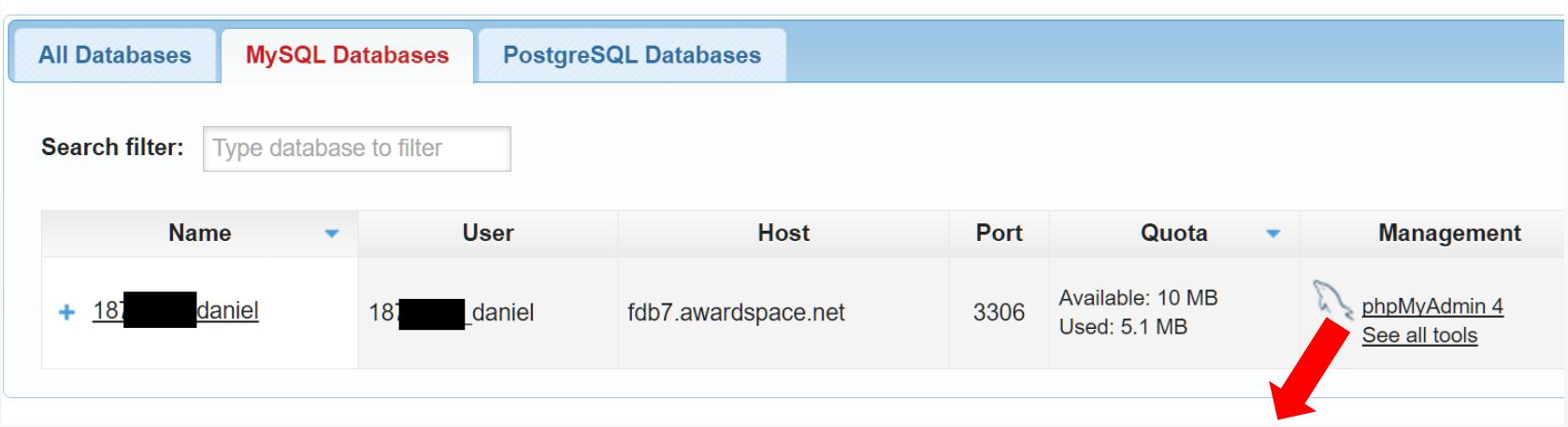
Criando um Banco de Dados no *Awardspace.com*

- Confira os detalhes do banco de dados criado;
- Esses dados serão necessários para realizar a conexão utilizando os *scripts* PHP;



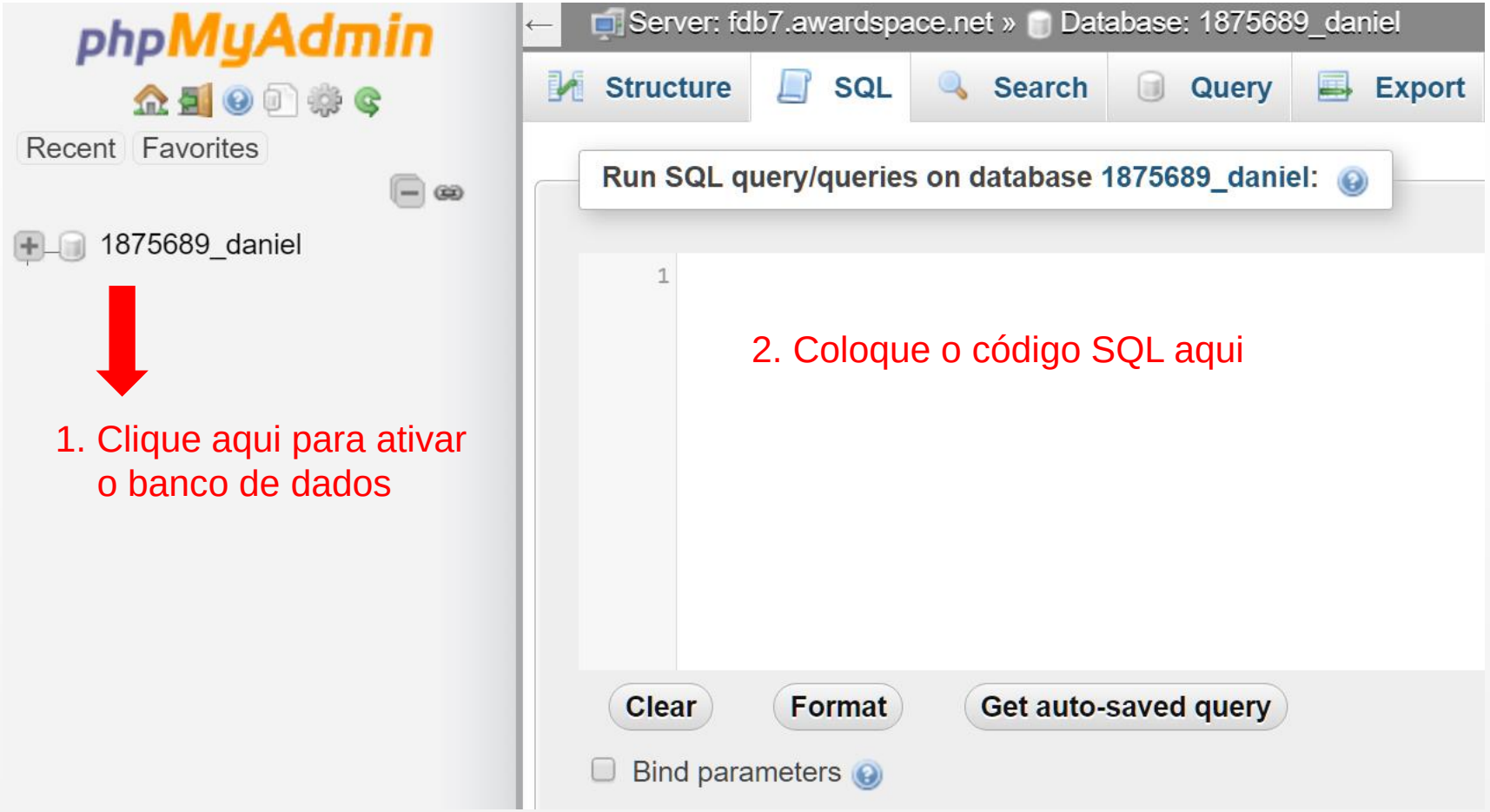
Criando um Banco de Dados no *Awardspace.com*

- Para manipular o banco de dados utilizando diretamente um programa navegador como o Google Chrome, acesse o **phpMyAdmin**



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.com*



MySQL no Awardspace

- É importante notar que a versão do MySQL disponibilizada no Awardspace opera com o motor de armazenamento ***MyISAM***, que, diferente do motor **InnoDB,** não suporta checagem de chave estrangeira, transações, *on delete cascade,* etc.;
- Para utilizar plenamente todas essas funcionalidades no MySQL recomenda-se a contratação de soluções de hospedagem baseada em VPS (*virtual private server*), onde é possível realizar a configuração completa do servidor, incluindo a definição do **InnoDB** no MySQL.

Informação sobre InnoDb no awardspace:
https://www.awardspace.com/kb/create-innodb-database/

Exercício 1

1. Crie uma tabela **Cliente** no seu espaço do Awardspace:

```
CREATE TABLE Cliente
(
  id int PRIMARY KEY auto_increment,
  nome varchar(50),
  email varchar(50),
  estadoCivil varchar(30),
  diaNascimento int
)
```

2. Modifique o arquivo anexo **conexaoMysql.php** inserindo os dados de acordo com sua conta e conexão no Awardspace;
3. Conecte ao Awardspace utilizando o WinSCP e transfira os arquivos anexos *conexaoMysql.php, ex01-cadastraCliente.php, ex01-cliente.php, ex01-mostraClientes.php, ex01-menu.php;*
4. Testar os arquivos acessando os mesmos pelo navegador;
5. Analisar e entender o código fonte dos arquivos.

Exercício 2

1. Crie uma tabela Aluno (matricula, nome, sexo) nesse banco de dados;
2. Crie um script ***novoAluno.php*** para apresentar um formulário de cadastro de aluno e realizar a devida inserção dos dados na tabela Aluno;
3. Crie um script PHP para listar os dados dos alunos cadastrados em uma tabela (***mostraAlunos.php***) (Crie uma função para buscar os dados na tabela e montar um array de objetos. Em seguida, monte a tabela HTML utilizando os dados desse *array).*

Passando argumentos pela URL - Exemplo

```
...
$sql = "SELECT cpf, nome FROM Cliente";
$resultado = $conn->query($sql);

if ($resultado->num_rows > 0)
{
    echo "
        <h1>Clientes Cadastrados</h1>
        <table>
        <thead><th>CPF</th><th>Nome</th><th></th></thead>";

    while ($row = $resultado->fetch_assoc())
    {
        $cpf = $row["cpf"];
        $nome = $row["nome"];
        echo "
            <tr>
              <td>$cpf</td><td>$nome</td>
              <td><a href=modificaCliente.php?cpf=$cpf>Modificar</a></td>
            <tr>
        ";
    }
}
...
```

Inicializando Campos de Formulário para Edição - Exemplo

```php
<?php
    $cpf = $_GET['cpf'];

    $sql = "SELECT nome , nroFilhos
             FROM Cliente
             WHERE cpf = '$cpf'";

    $resultado = $conn->query($sql);

    $row = $resultado->fetch_assoc();
    $nome = $row["nome"];
    $nroFilhos = $row["nroFilhos"];
?>

<!DOCTYPE html>
<html>
<body>

<form>
    Nome: <input type="text" name="nome" value="<?php echo $nome; ?>">
    Filhos: <input type="text" name="filhos" value="<?php echo $nroFilhos; ?>">
</form>

</body>
</html>
```

PHP, MySQL e Transações

Transações no MySQL - Introdução

- No contexto de banco de dados, uma transação é basicamente uma sequência de operações que devem ser executadas na totalidade: **não se permite a execução parcial de tais operações** *(executa-se todas elas ou nenhuma).*
- Uma transação deve ter um **início** e um **fim**;
- Assim, se ocorrer uma falha no 'meio' de uma transação, deve haver um mecanismo para **desfazer** as operações que já tenham sido executadas do início até o ponto da falha;
- A operação ***commit*** é normalmente utilizada no final da transação para efetivar todas as operações efetuadas;
- A operação ***rollback*** desfaz as operações da transação já efetuadas, voltando o banco de dados para o estado anterior ao início da transação.

Transações no MySQL - Introdução

- Por exemplo, o conceito de transação poderia ser utilizado para inserir dados correlacionados no banco de dados;
- Em um cadastro de cliente, por exemplo, pode ser necessário inserir os ***dados pessoais do cliente*** em uma tabela do banco de dados e o seu ***endereço***, em outra tabela. Neste caso, utilizando o conceito de transação, podemos garantir que o cadastro seja sempre realizado na totalidade (ou a inserção é realizada nas duas tabelas, ou em nenhuma).

Exemplo de Transação no MySQL

(*relacionamento 1-N*)

```
CREATE TABLE Cliente
(
    codigo int PRIMARY KEY auto_increment,
    nome varchar(50)
);

CREATE TABLE Endereco
(
    rua varchar(30),
    numero int,
    codCliente int,
    FOREIGN KEY (codCliente) REFERENCES Cliente(codigo)
);

begin; /* inicio da transação */

INSERT INTO Cliente VALUES (null, 'Fulano');
INSERT INTO Endereco VALUES ('Rua tal', 100, LAST_INSERT_ID());

/* LAST_INSERT_ID é uma função do MySQL que retorna o último ID
inserido para um campo do tipo 'auto_increment' */

commit; /* efetiva todas as operações da transação */
```

Exemplo de Transação no PHP com MySQLi

```
...

try
{
   // inicio da transacao
   $conn->begin_transaction();

   if (! $conn->query("insert into Cliente values (null, 'Beltrano')"))
      throw new Exception('Erro ao inserir na tabela cliente');

   if (! $conn->query("insert into Endereco values ('Rua Abc', 100, LAST_INSERT_ID())"))
      throw new Exception('Erro ao inserir na tabela Endereco');

   // se nenhuma excecao foi lancada, efetiva as operacoes
   $conn->commit();

   echo "Transacao executada com sucesso";
}
catch (Exception $e)
{
   // desfaz as operacoes caso algum erro tenha ocorrido (e uma exceção lançada)
   $conn->rollback();

   echo "Ocorreu um erro na transacao: " . $e->getMessage();
}

...
```

Aspectos de Segurança

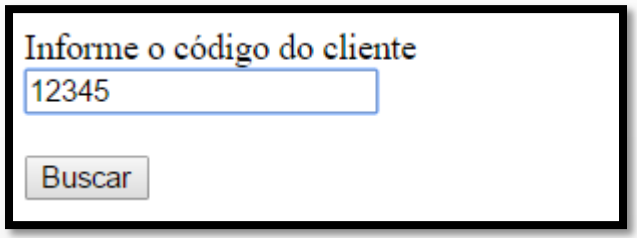
SQL Injection

- Técnica que usuários maliciosos utilizam para “injetar” **código SQL** dentro de uma instrução SQL lícita, utilizando campos de formulário Web, a URL, ou outros meios de entrada;
- Pode comprometer a segurança da aplicação Web;
- Pode possibilitar que operações de consultas, atualizações e exclusões, **sem autorização**, sejam realizadas no banco de dados.

Exemplo de SQL Injection – *Expressão 1=1*

Considere o formulário HTML a seguir e o respectivo código PHP. Eles poderiam ser utilizados, eventualmente, para buscar um cliente no banco de dados a partir do seu código;

Formulário HTML




```
<form action="buscaCliente.php" method="post">
<input type="text" name="codCliente">
...
```

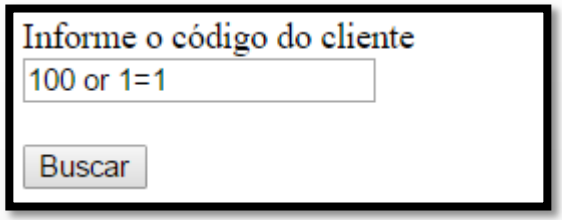
Código no script ***buscaCliente.php***

```
$codCliente = $_POST["codCliente"];
SQL = "select * from Cliente where codigo = $codCliente";
```

Continuação no próximo slide...

Exemplo de SQL Injection – *Expressão 1=1*

O que aconteceria se um usuário mal intencionado informasse um código de cliente qualquer seguido da expressão “or 1=1” ?



Se nenhum tratamento do dado for feito, o código em *buscaCliente.php*:

```
$SQL = "select * from Cliente where codigo = $codCliente";
```

será avaliado como:

```
$SQL = "select * from Cliente where codigo = 100 or 1=1";
```

Observe que a expressão SQL acima continua sendo válida, porém o acréscimo de “or 1=1” faz com que a condição de seleção na cláusula ***where*** seja sempre verdadeira. Logo, a consulta retornaria os dados de **TODOS os clientes cadastrados na tabela** (e não de um cliente específico).

Exemplo de SQL Injection – *Expressão 1=1*

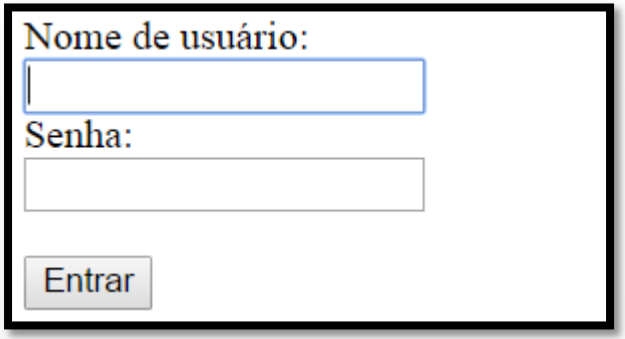
Exercício. Testar o caso de *SQL Injection* ilustrado anteriormente.

1. Crie uma tabela no servidor contendo dois campos: (codCliente (int) e nomeCliente (varchar(50));
2. Utilize a SQL para inserir três registros na tabela;
3. Crie o formulário HTML;
4. Crie um script em PHP que receba os dados do formulário e faça a devida consulta no banco de dados. O script deve montar uma página HTML e exibir os dados retornados pela consulta (considere a possibilidade de ter mais de uma linha no resultado).

Exemplo de SQL Injection – *Expressão '' = ''*

Considere o formulário HTML a seguir e o respectivo código PHP. Eles poderiam ser utilizados, eventualmente, para realizar a autenticação dos usuários que terão acesso a um sistema;

Formulário




```
<form action="login.php" method="post">
<input type="text" name="usuario">
<input type="password" name="senha">
...
```

Código no script *login.php*

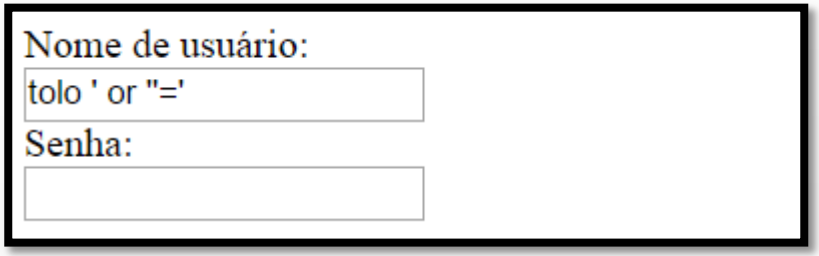
```
$usuario = $_POST["usuario"];
$senha   = $_POST["senha"];

// Verifica se usuario e a respectiva senha existem no banco de dados
$sql = "SELECT * FROM Usuarios WHERE username = '$usuario' AND password = '$senha'";
```

Continuação no próximo slide...

Exemplo de SQL Injection – *Expressão '' = ''*

O que aconteceria se um usuário mal intencionado informasse um nome de usuário qualquer e uma senha qualquer, ambos seguidos da expressão ' or ''=' ?

Exemplo de SQL Injection



Assim, o código no script *login.php:*

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

$sql = "SELECT * FROM Usuarios
        WHERE username = '$usuario' AND password = '$senha'";
```

seria avaliado pelo PHP como:

```
$sql = "SELECT * FROM Usuarios
        WHERE username = 'tolo' or ''='' AND password = 'tolo' or ''='' ";
```

Exemplo de SQL Injection – *Expressão '' = ''*

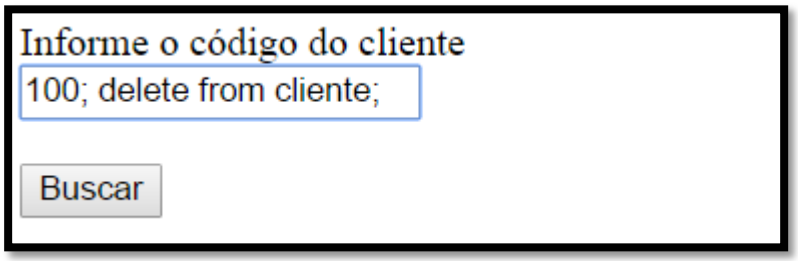
Exercício. Testar o caso de SQL Injection ilustrado anteriormente:

1. Crie uma tabela no BD de nome Usuario contendo dois campos: (**username**, do tipo varchar(50); e **password**, do tipo char(8));
2. Utilize a SQL para alguns registros na tabela;
3. Crie o formulário HTML apresentado no slide anterior;
4. Crie um script em PHP que receba os dados do formulário e faça a autenticação utilizando os dados disponíveis na tabela **Usuarios**. O script deve mostrar a mensagem "Login efetuado com sucesso!", caso os dados existam na tabela; ou a mensagem "Dados inválidos", caso contrário;
5. Experimente inserir os dados ilustrados no slide anterior para confirmar o resultado da *SQL Injection.*

SQL Injection – Múltiplos comandos SQL

No formulário a seguir, o que aconteceria se um usuário mal intencionado informasse um código qualquer para o cliente seguido da expressão "DELETE FROM CLIENTE" ?

Exemplo de SQL Injection:



Possível código PHP que resultaria na **exclusão não planejada de todos os clientes**

```
$codCliente = $_POST["codCliente"];
$sql = "SELECT *
        FROM Cliente
        WHERE codigo = $codCliente";

$resultado = $conn->multi_query($sql);
if (!$resultado)
    die("Ocorreu um erro ao executar a consulta: " . $sql . $conn->error);
```

Evitando SQL Injection

- Recomenda-se o pré-processamento (validação) dos dados preenchidos em campos de formulário (ou da URL) para evitar ataques de SQL Injection;
- A função PHP **htmlspecialchar**, por exemplo, remove alguns caracteres especiais da *string*, como aspas, trocando tais caracteres pelos códigos correspondentes do HTML;

```php
<?php

function filtraEntrada($dado)
{
  // remove espaços no inicio e
  // no final da string
  $dado = trim($dado);

  // remove contra barras:
  // "cobra d\'agua" vira "cobra d'agua"
  $dado = stripslashes($dado);

  $dado = htmlspecialchars($dado);
  return $dado;
}

$nome   = filtraEntrada($_POST["nome"]);
$codigo = filtraEntrada($_GET["codigo"]);
...
?>
```

Evitando SQL Injection

- Além de realizar uma validação dos dados vindos de campos de formulários ou da URL, também é recomendado não inserir o nome de variáveis PHP diretamente na *string* SQL;
- Ao invés disso, utilize o conceito de ***prepared statements*** (apresentado a seguir).

Evitando SQL *Injection* com *Prepared Statements*

- ***Prepared Statement*** *(declaração preparada)* é um recurso que permite executar uma mesma instrução SQL repetidas vezes, com maior eficiência e de maneira mais segura.
- Principais Vantagens:
  - Maior segurança contra ataques do tipo *SQL Injection*.
  - Pode reduzir o tempo de execução de múltiplas consultas SQL no servidor, uma vez que sua preparação pode ser realizada uma única vez (embora a consulta possa ser executada repetidas vezes);
  - Pode reduzir o tráfego de dados com o servidor, uma vez que a instrução SQL não precisa ser reenviada por inteira repetidas vezes;

Evitando SQL *Injection* com *Prepared Statements*

Funcionamento básico:

- **Preparação**: Um *template* SQL é criado e enviado ao SGBD. Porém, determinado valores são deixados em aberto (denominados parâmetros). Exemplo: *INSERT INTO Clientes VALUES (**?**, **?**, **?**)*
- O SGBD pré-processa a declaração SQL (faz checagem da sintaxe, entre outros), mas não a executa;
- **Execução**: mais tarde, a aplicação fornece valores aos parâmetros e executa a declaração;

Evitando SQL *Injection* com *Prepared Statements*

Exemplo 1: Operação de **inserção** utilizando *prepared statements* com MySQLi

```php
// Estágio 1: Preparação
$stmt = $mysqli->prepare("INSERT INTO Teste(id) VALUES (?)");


// Estágio 2: Associação dos parâmetros (bind)
$id = 1;
$stmt->bind_param("i", $id);


// Estágio 3: execução
$stmt->execute();


// Execução repetida: somente os dados são transferidos
// do cliente para o servidor
for ($id = 2; $id < 100; $id++)
{
   // Insere na tabela o valor corrente da variável $id
   if (!$stmt->execute())
      echo "Falha: (" . $stmt->errno . ") " . $stmt->error;
}
```

Evitando SQL *Injection* com *Prepared Statements*

Exemplo 2: Operação de **consulta** utilizando *prepared statements* com MySQLi

```php
// prepara a declaração SQL (stmt é um abreviação de statement)
$stmt = $conn->prepare("SELECT Codigo, Nome FROM Cliente WHERE Codigo = ? ");

// bind_param faz a ligação dos parâmetros em aberto com os valores.
// Utilize i para var. inteira, s para string, d para double ou float
// Exemplo de uso com várias variáveis: $stmt->bind_param("iss", $codigo, $nome, $email);
$stmt->bind_param("i", $codCliente);

// Executa a declaração SQL previamente preparada
$stmt->execute();

// store_result é opcional. Busca todo o resultado da consulta, armazenando em um buffer.
// Em alguns casos otimiza o processamento dos resultados da consulta.
$stmt->store_result();

// Indica as variáveis PHP que receberão os resultados
$stmt->bind_result($cod, $nome);

// Navega pelas linhas do resultado
while ($stmt->fetch())
{
   echo $cod, $nome;
}
```

Resumo das Principais Operações Utilizando o MySQLi

- Estabelece uma conexão com o MySQL
```
$conn = new mysqli(servername, username, password, dbname);
```
- Verifica a ocorrência de um eventual erro durante a conexão
```
$conn->connect_error
```
- Executa um comando SQL (insert, update, select, etc.)
```
$resultado = $conn->query($sql)
```
- Retorna o número de linhas do resultado
```
$resultado->num_rows
```
- Resgata a próxima linha do resultado na forma de um *array* associativo
```
$row = $resultado->fetch_assoc()
```
- Encerra a conexão com o servidor do MySQL
```
$conn->close()
```
- Prepara uma declaração SQL
```
$stmt = $conn->prepare("SQL")
```
- Faz a associação dos parâmetros com as variáveis
```
$stmt->bind_param
```
- Faz a indicação das variáveis que receberão os resultados de uma consulta
```
$stmt->bind_result
```
- Executa a declaração previamente preparada
```
$stmt->execute()
```

Registro de Domínio

- br.**godaddy**.com
- www.ehost.com
- www.web.com

Referências

- www.w3schools.com
- www.mysql.com
- www.php.net
- phpbestpractices.org