Aula Teste

Acesso ao banco de dados com PHP

Objetivos da Aula

- Criar banco de dados e tabelas no MySQL.
- Entender como conectar PHP a bancos de dados.
- Diferenciar MySQLi e PDO.
- Usar prepared statements para segurança.
- Realizar operações CRUD: SELECT, INSERT, UPDATE, DELETE.
- Aplicar boas práticas.

Configuração do Ambiente

- Servidor web com PHP (ex: XAMPP, WAMP).
- Banco MySQL instalado e em execução.
- Ferramenta para gerenciar o banco (ex: phpMyAdmin).

Ferramentas para acesso ao MySQL

- Linha de comando
- phpMyAdmin
- MySQL Workbench
- Dbeaver
- HeidiSQL

Criando o banco de dados I

```
CREATE DATABASE sistema_usuarios
    CHARACTER SET utf8mb4
    COLLATE utf8mb4_unicode_ci;

    COLLATE utf8mb4_unicode_ci;

USE sistema_usuarios;

CREATE TABLE usuarios (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(100) NOT NULL,
    email VARCHAR(150) NOT NULL UNIQUE,
    senha VARCHAR(255) NOT NULL,
    criado_em TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Criando o banco de dados II

```sql
DESCRIBE usuarios;

INSERT INTO usuarios (nome, email, senha)
VALUES ('João Silva', 'joao@email.com', 'senha123');
```

MySQLi x PDO (PHP Data Objects)

- PHP possui extensões e bibliotecas para facilitar a tarefa de acesso a banco de dados.

- Duas abordagens principais para bancos relacionais como MySQL/MariaDB:

  - MySQLi → específica para MySQL/MariaDB.

  - PDO (PHP Data Objects) → mais flexível e segura.

Conexão com PDO

```php
try {
    $pdo = new PDO("mysql:host=localhost;dbname=sistema_usuarios", "usuario", "senha");
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    echo "Conectado com sucesso!";
} catch (PDOException $e) {
    echo "Erro: " . $e->getMessage();
}
```

Select com PDO

```php
$sql = "SELECT id, nome FROM usuarios";
$stmt = $pdo->prepare($sql);
$stmt->execute();
```

```php

$stmt = $pdo->query("SELECT id, nome FROM usuarios");
while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
    echo "ID: " . $row['id'] . " - Nome: " . $row['nome'] . "<br>";
}
```

Insert com PDO e Prepared Statement

```php
$sql = "INSERT INTO usuarios (nome, email, senha) VALUES (:nome, :email, :senha)";
$stmt = $pdo->prepare($sql);

$dados = [
    'nome' => 'Ana',
    'email' => 'ana@email.com',
    'senha' => password_hash('senha123', PASSWORD_DEFAULT)
];

$stmt->execute($dados);
echo "Usuário inserido!";
```

Update com PDO

```php
$sql = "UPDATE usuarios SET email = :email WHERE id = :id";
$stmt = $pdo->prepare($sql);

$stmt->execute([
    'email' => 'novoemail@email.com',
    'id' => 1
]);

echo "Usuário atualizado.";
```

Delete com PDO

```php
$sql = "DELETE FROM usuarios WHERE id = :id";
$stmt = $pdo->prepare($sql);
$stmt->execute(['id' => 3]);

echo "Usuário excluído.";
```

Importância do Prepared Statement

- Segurança contra SQL Injection
- Desempenho e eficiência
- Organização e clareza do código
- Prevenção de erros de sintaxe
- Portabilidade entre bancos

Boas Práticas

- Sempre usar prepared statements.
- Não expor mensagens de erro em produção.
- Armazenar senhas criptografadas.
- Centralizar configurações de conexão.
- Usar charset utf8mb4.