**PHP com PDO e MySQL**

**Introdução ao PDO**

**PHP Data Objects** – conjunto de objetos que auxiliam no trabalho de banco de dados.

Objetivo: **prover** uma **padronização** da forma com que o **PHP se comunica com** os **bancos** de dados;

Esses objetos são **agregados ao PHP** no formato de extensão, portanto, **são habilitados ou desabilitados através do** arquivo **php.ini** conforme a necessidade.

O PHP é conhecido como uma **poderosa linguagem de programação em termos de compatibilidade com os mais diversos bancos de dados**.

**Criando uma conexão entre o PHP e o MySql com PDO**

- Dar start nos serviços Apache e MySql;

- Ir para diretório público e criar nova pasta: **php\_com\_pdo**;

Criando instância de PDO:

$dsn = 'mysql:host=localhost;dbname=php\_pdo'; //data source name

$user = 'root';

$senha = '';

$conexao = new PDO($dsn, $user, $senha);

Precisam ser passados alguns parâmetros para configuração;

1º parâmetro – Data Source Name (DSN): nome da fonte de dados;

**Tratando exceptions (PDOException)**

try {

    $conexao = new PDO($dsn, $user, $senha);

} catch (PDOException $e) {

    echo 'Erro: ' . $e->getCode() . ' Mensagem: ' . $e->getMessage();

    //registrar erro

**try** – tente **/ catch** – caso haja erros dentro de try

'Erro: ' . $e->getCode() . ' Mensagem: ' . $e->getMessage();

Recuperando código e mensagem do erro.

//registrar erro - importante, pois com a correção de erros naturalmente a aplicação se torna mais consistente

No front-end, podemos redirecionar o usuário para uma outra tela, ou informar que naquele momento a aplicação não está disponível ou fornecer links ou entretenimento para que usuário não se fruste.

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente com confiança média

Bancos de dados no php.ini, caso queira utilizar outro banco de dados, basta descomentar a extensão e reiniciar o Apache.

2º parâmetro – host: local

- Nome do banco de dados (dbname=<nomedobanco>);

- Usuário, senha (‘’,’’)

**Executando instruções SQL (Exec->método)**

- Montar query a ser utilizada;

- A partir da query conseguimos realizar uma criação de tabela dentro do banco de dados.

Ao invés de create table <nometabela> (<colunas>), foi utilizado  **(<colunas>)**;

Com isso, em futuras execuções do script, caso a tabela já exista no banco de dados, logo não haverá uma tentativa de recriá-la, evitando erro de tabela já existente no banco (**~~already exists~~**).

$query = '

    create table if not exists tb\_usuarios(

        id not null primary key auto\_increment,

        nome varchar(50) not null,

        email varchar(100) not null,

        senha varchar(32) not null

    )';

$conexao->exec($query); //método espera especificamente a query

    //retorno esperado - 0

Através da conexão que contém a instância do método PDO, executamos o **método exec**(que **espera** justamente a **query** **que é passada por parâmetro**);

**PDO** **processa informação** **junto ao banco** estabelecido a conexão;

**Exec retorna** apenas o **nº de linhas modificadas ou removidas** pela instrução SQL encaminhada.

Quando trabalhamos com querys **DDL** (Data Definition Language – instruções que irão definir a estrutura dos dados), teremos sempre o **retorno de 0**;

**Retorno** de **0**, pois nesse processo de definição dos dados de criação das estruturas para receber dados, eles **não estão sendo de fato** **afetados**;

Um **SELECT** também **retornaria 0**, pois não está **nem modificando e nem removendo** registros;

Instruções como **UPDATE**, **DELETE** e **INSERT**, **retornam** quantidade de **linhas afetadas** no processo.

Em operações de CRUD exec não será utilizado.

C – Create (criar);

R- Read (ler);

U – Update (atualizar);

D – Delete (deletar).

$query = '

        insert into tb\_usuarios(

            nome, email, senha

        ) values (

            "Ana Almeida", "aninhaalme@teste.com", "123456"

        )';

    $retorno = $conexao->exec($query);

    echo $retorno;

$query = '

        delete from tb\_usuarios

    ';

Removendo registros de tb\_usuarios

**PDOStatement Object (Query) com fetchAll**

**PDOStatement Object** – objeto que contém declaração da query, que permite por exemplo recuperar informações contidas no BD.

$query = '

        insert into tb\_usuarios( nome, email, senha)

        values ( "Ana Almeida", "aninha@teste.com", "123456")

    ';

Inserindo registro.

$query = 'select \* from tb\_usuarios';

$stmt = $conexao->query($query); //stmt - statement //query - retorna um pdo statement

$lista = $stmt->fetchAll(); //retorna todos os registros retornados da consulta

print\_r($lista);

Array encapsula registros e registro encapsulam dados dentro de array;

Há índices associativos da tabela(nome da coluna) /// índices numéricos.

Texto, Carta

Descrição gerada automaticamente

Acessando item de índice 2 da lista:

    echo $lista[1]['nome'];

    echo $lista[1][1];

**FetchALL – Trabalhando os tipos de retorno**

Acessando apenas índice associativo:

    $lista = $stmt->fetchAll(PDO::FETCH\_ASSOC); //fetchAll -> retorna todos os registros retornados da consulta

Acessando apenas índice numérico:

    $lista = $stmt->fetchAll(PDO::FETCH\_NUM); //retorna todos os registros retornados da consulta

Para manter padrão de retornar tudo:

    $lista = $stmt->fetchAll(PDO::FETCH\_BOTH); //retorna todos os registros retornados da consulta

Não passa nem um parâmetro ou PDO::FETCH\_BOTH.

Recuperar Array de objetos ao invés de Array de Array’s:

    $lista = $stmt->fetchAll(PDO::FETCH\_OBJ); //retorna todos os registros retornados da consulta

**Conforme** o **retorno muda,** deve-se **modificar também** as **formas de recuperação** **de** **valores**. No caso do Array, valor é recuperado utilizando os índices.

Retorno de valor dentro de objeto:

    echo $lista[1]->nome;

**FETCH\_ASSOC**, **FETCH\_NUM**, **FETCH\_BOTH** e **FETCH\_OBJ** são as opções **mais utilizadas** para retornos específicos. Porém, não são os únicos, podemos acessá-los em: *php.net* > *pdo* > [*pdostatement*](https://www.php.net/manual/pt_BR/class.pdostatement.php).

**Fetch – Retornando apenas um registro**

Diferente do fetchAll, o fetch recupera apenas um registro.

Acessando apenas um objeto, com id = 9:

     $query = '

        select \* from tb\_usuarios where id = 9

    ';

$usuario = $stmt->fetch(PDO::FETCH\_OBJ); //retorna todos os registros retornados da consulta

Texto

Descrição gerada automaticamente

Acessando registro de id = 9 pelo FETCH\_ASSOC

$usuario = $stmt->fetch(PDO::FETCH\_ASSOC); //retorna todos os registros retornados da consulta

    echo '<pre>';

    print\_r($usuario);

    echo '</pre>';

    echo $usuario['nome'];

Texto, Carta

Descrição gerada automaticamente

Recuperando 1 registro (limit 1) de uma consulta de todos os registros (select \* from tb\_usuarios), ordenado pela coluna nome (order by nome) de forma decrescente (desc).

Texto, Carta

Descrição gerada automaticamente select\*from tb\_usuarios order by nome desc limit 1

**Listando registros com Foreach**

Percorrer cada uma daas informações e fazer a impressão de acordo com a necessidade.

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente recuperando nomes.

Uma imagem contendo Tabela

Descrição gerada automaticamente

Retornando índices associativos e numéricos:

$query = '

        select \* from tb\_usuarios

    ';

    foreach($conexao->query($query) as $chave => $valor){

        echo '<pre>';

        print\_r($valor);

        echo '<hr>';

        echo '</pre>';

    }

Recupera apenas um valor de chave associativa:

 print\_r($valor['nome']);

**SQL Injection**

Injeção de instruções SQL seja na URL ou no body do request de informações que serão utilizadas para compor as querys.

<?php if (!empty($\_POST['usuario']) && !empty($\_POST['senha'])) { //só entra na lógica se índices 'usuario' e 'senha' da super-global POST não estiverem vazos

            //query

            $query = "select \* from tb\_usuarios where ";

            $query .= " email = '{$\_POST['usuario']}'"; //atribuindo e concatenando com o que já existe na variável query

            $query .= " AND senha = '{$\_POST['senha']}'"; //utilizando chaves para que não haja conflitos por conta das aspas

            echo $query;

            $stmt = $conexao->query($query); //instância do método PDO, executando método query, passando nossa query

            $usuario = $stmt->fetch(); //recupera usuario através de statement executando o método fetch

SQL Injection é uma **vulnerabilidade**, **quando** do lado do back-end as **instruções SQL não são previamente tratadas**.

**Prepare Statement**

- Ao invés de executar uma query, prepararemos um statement;

- Incluindo registros no banco de dados.

try {

            $conexao = new PDO($dsn, $usuario, $senha);

            $query = "insert into tb\_usuarios(nome, email, senha)values('Ana Almeida', 'aninhaa@teste.com', '1234');";

            $conexao->query($query);

Método prepare():

            $stmt = $conexao->prepare($query);

***//como o método query retorna um PDOStatement, porém ele não executa a query diretamente, fica aguardando que ordem seja dada***

Método bindValue:

            $stmt->bindValue('', $\_POST['usuario']); //bindValue recebe: ('variável de ligação(bind)' + valor da ligação)

            $stmt->bindValue(':senha', $\_POST['senha']);

Configs do método bindValue:

$query = "select \* from tb\_usuarios where ";

            $query .= " email = :usuario ";

        $query .= " AND senha = :senha";

**bindValue pode trabalhar também com um 3° parâmetro:**

- **Tipo de dado** dentro do parâmetro informado, que **deve ser considerado** caso haja um SQLInjection.

- Todo valor passado é interpretado como string, texto;

- Esse parâmetro verifica se há um valor de determinado tipo dentro de bindValue, como 2° parâmetro (value);

- Essa informação pode ser extraída e utilizada dentro da consulta.

            $stmt->bindValue(':senha', $\_POST['senha'], ***PDO::PARAM\_INT***);

**Verificando se há valores inteiros dentro de value.**