**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

    ';