

REDES DE COMUNICAÇÃO

MÓDULO 7A - Acesso às bases de dados via Web

1 PDO

Para criar aplicações com acesso ou armazenamento de dados numa base de dados, é necessário estabelecer uma **conexão** entre o PHP e o Sistema de Gestão de Base de Dados (SGBD), que no nosso caso, iremos explorar o **MySQL** (mais tarde poderemos explorar a ligação ao servidor SQLServer).

O PHP oferece duas maneiras diferentes de se conectar ao servidor MySQL: pelas extensões **MySQLi** (*Improved MySQL*) e **PDO** (*PHP Data Objects*).

Para entender como funciona a conexão utilizando o PDO, observe a **Figura 1**:

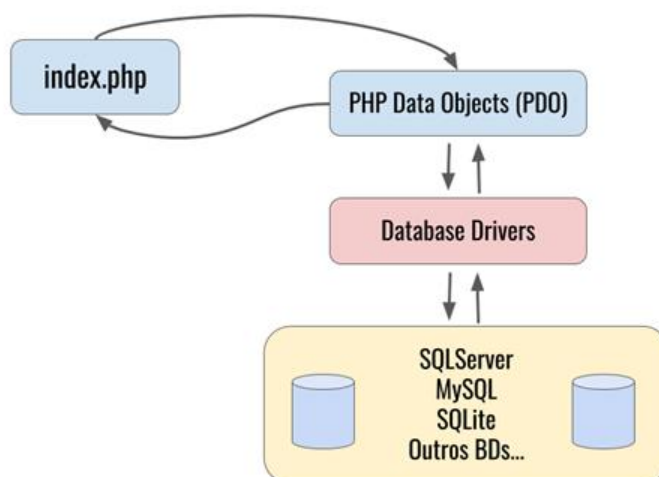


Figura 1. Conexão utilizando PDO

A **extensão PDO** define uma interface leve e consistente (camada de abstração) para **acesso à base de dados em PHP**. Como se pode observar na figura, são necessários drivers para acesso às Bases de Dados alojadas em SGBD MySQL.

Uma listagem dos drivers PDO podem ser encontrados em:

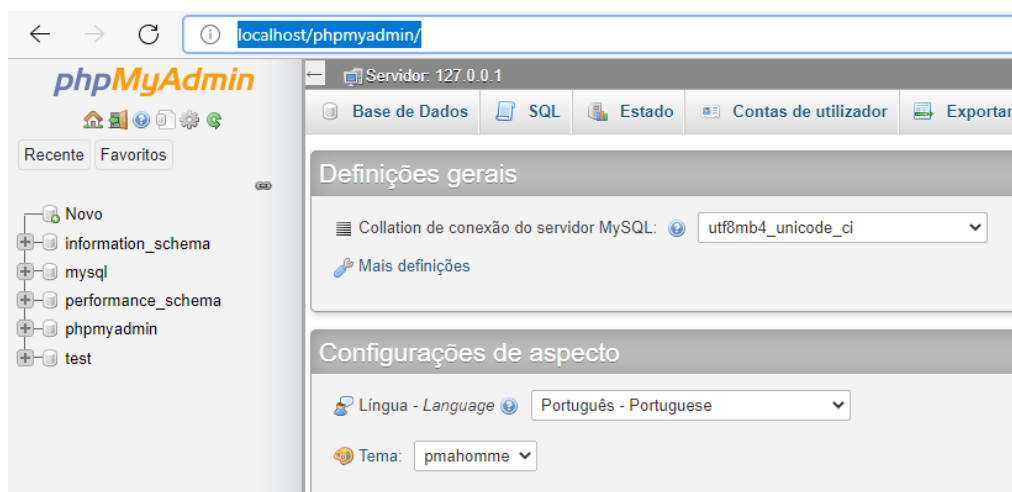
<https://www.php.net/manual/en/pdo.drivers.php>

Driver name	Supported databases
PDO_CUBRID	Cubrid
PDO_DBLIB	FreeTDS / Microsoft SQL Server / Sybase
PDO_FIREBIRD	Firebird
PDO_IBM	IBM DB2
PDO_INFORMIX	IBM Informix Dynamic Server
PDO_MYSQL	MySQL 3.x/4.x/5.x
PDO_OCI	Oracle Call Interface
PDO_ODBC	ODBC v3 (IBM DB2, unixODBC and win32 ODBC)
PDO_PGSQL	PostgreSQL
PDO_SQLITE	SQLite 3 and SQLite 2
PDO_SQLSRV	Microsoft SQL Server / SQL Azure

Figura 2 – drivers suportados pelo PDO

2 Configuração de acesso ao servidor MySQL e à base de dados



As bases de dados podem ser geridas recorrendo à programação ou à utilização de ferramentas, como o *phpmyadmin* (abre no browser: <http://localhost/phpmyadmin/>).



2.1 Criar a base de dados e as tabelas

1. Crie uma base de dados (BD) com o nome **company**.

Base de Dados

 Criar base de dados 

Criar

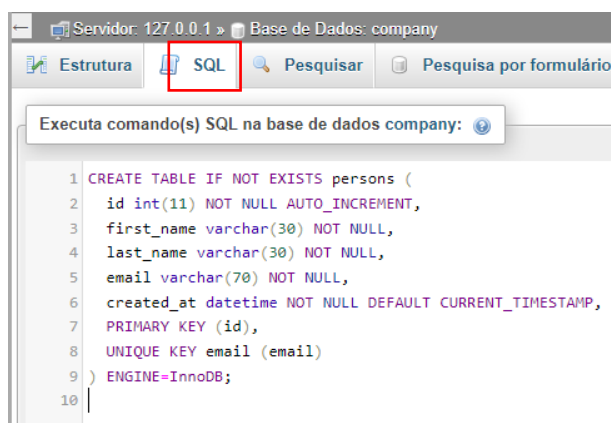
2. Crie a tabela **persons** com a seguinte estrutura:

Campo	Tipo de dados
id	Int
first_name	Varchar (30)
last_name	Varchar (30)
email	Varchar (70)
created_at	Datetime

Para criar esta tabela no MySQL, pode criar no modo gráfico, na ferramenta phpmyadmin ou executar o seguinte código no separador SQL da mesma ferramenta.

```
CREATE TABLE IF NOT EXISTS persons (  
  id int(11) NOT NULL AUTO_INCREMENT,  
  first_name varchar(30) NOT NULL,  
  last_name varchar(30) NOT NULL,  
  email varchar(70) NOT NULL,  
  created_at datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,  
  PRIMARY KEY (id),  
  UNIQUE KEY email (email)  
) ENGINE=InnoDB;
```

Com a base de dados selecionada, digite o comando na janela SQL:



Clique no botão Executar.

Crie a tabela **categories** que permita relacionar esta tabela com a tabela persons:

Para criar a relação entre as duas tabelas, crie primeiro a chave externa na tabela persons e, de seguida, clique no botão **Visão de relação** e preencha os campos.

Campo	Tipo de dados
idCategory	Int
assignment	Varchar (30)
created_at	Datetime

2.2 Criar a conexão ao servidor MySQL

Vamos **criar um ficheiro de configuração** que permita efetuar a conexão com a BD. Estando a trabalhar num servidor, normalmente é necessário **criar uma conta de utilizador e a respetiva password**. Cada utilizador terá acesso a determinadas BD com determinados privilégios.

1. **Crie uma conta no servidor MySQL, guarda o nome de utilizador e a respetiva password.** Selecione a opção privilégios globais.

Servidor: 127.0.0.1

Base de Dados SQL Estado Contas de utilizador

Adicionar conta de utilizador

Informação de Login

Nome de utilizador: Usar campo de texto ▼ adelino

⚠ Uma conta já existe com o mesmo nome de utilizador, mas possivelmente com um

Nome do Host: Local ▼ localhost

Palavra-passe: Usar campo de texto ▼ Strength:

Confirma:

Plugin de autenticação: Autenticação nativa do MySQL ▼

Gerar palavra-passe: Gerar

Banco de dados para a conta de utilizador

☐ Criar banco de dados com o mesmo nome e conceder todos os privilégios.

☐ Conceder todos os privilégios no nome curinga (nome_do_utilizador_%).

Privilégios Globais ☒ Marcar todos

Clique no botão **Executar**.

Neste momento estamos munidos de todos os dados para construir o ficheiro de configuração. Na prática, este ficheiro fica escondido numa pasta do servidor, no nosso caso, vamos guardá-lo na pasta do projeto. Nesta pasta vamos guardar **toda** a informação relacionada com o projeto.

2. Crie um ficheiro com o nome **config.php**, digita o seguinte conteúdo e altere o nome de utilizador e respetiva senha com os seus dados.

```
<?php
// definição das constantes a utilizar na classe PDO
define('DB_SERVER', 'localhost');
define('DB_USERNAME', 'nome_utilizador');
define('DB_PASSWORD', '123456');
define('DB_NAME', 'company');
// Cria a conexão com o servidor MySQL
try{
    $db = "mysql:host=" . DB_SERVER . ";dbname=" . DB_NAME . ";charset=utf8";

    $link = new PDO($db, DB_USERNAME, DB_PASSWORD);

    $link->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

} catch(PDOException $e){
    // Um erro é disparado se houver erro com a conexão com o MySQL
    die("ERRO: Não foi possível estabelecer a ligação. " . $e->getMessage());
}
?>
```

Comentários:

O primeiro passo consistiu em estabelecer uma ligação com o servidor MySQL que, neste caso, se encontra a correr na máquina local. Nesta ligação define-se as credenciais de acesso ao servidor, a base de dados de trabalho e o seu tipo de caracteres.

```
$pdo = new PDO("mysql:host=hostname;dbname=database;charset=utf8", "username",
"password");
```

O atributo PDO::ATTR_ERRMODE permite apresentar o relatório de erros, lançando a exceção PDO::ERRMODE_EXCEPTION sempre que ocorrer um erro com a BD.

```
$link->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

Estas instruções ficam dentro de uma estrutura try {} catch() para detetar as exceções automaticamente.

Após estabelecida a conexão com a BD, qualquer ação pode ser executada através da linguagem PHP.

Outras informações importantes

Para obter informação da ligação ao servidor utilizaríamos a instrução:

```
echo "Ligação realizada com sucesso. Informação do Host: " .
```

```
$link->getAttribute(constant("PDO::ATTR_CONNECTION_STATUS"));
```

Para fechar a ligação à BD recorreremos à instrução:

```
$link = null;
```

também poderíamos usar a instrução

```
unset($link)
```

No entanto, a conexão com o servidor de BD MySQL será encerrada automaticamente assim que a execução do script terminar.

Se ocorrer algum erro, existem métodos que os exploram, tais como os métodos: getCode(), getFile(), getLine(), getTrace(), getTraceAsString(), getMessage()

```
die("ERRO: Não foi possível estabelecer a ligação. " . $e->getMessage());
```

O PDO oferece 3 alternativas para manipulação de erros.

PDO::ERRMODE_SILENT - Este é o tipo padrão utilizado pelo PDO. O PDO atribui internamente um código de um determinado erro, este pode ser obtido através dos métodos PDO::errorCode() e PDO::errorInfo().

PDO::ERRMODE_WARNING - Além de armazenar o código do erro, este tipo de manipulação de erro irá enviar uma mensagem E_WARNING, sendo este muito utilizado durante a depuração e/ou teste da aplicação.

PDO::ERRMODE_EXCEPTION - Além de armazenar o código de erro, este tipo de manipulação de erro irá lançar uma exceção PDOException, esta alternativa é recomendada, deixando o código mais limpo e legível.

3 Inserção de Registos

3.1 Inserir dados numa tabela

A tabela da BD está vazia, vamos começar a inserir registos.

Crie um ficheiro com o nome **insert_data_table.php** e digite o seguinte conteúdo:

```
<?php
    require_once "config.php";

    try{
        // definição do comando SQL para inserir um registo
        $sql = "INSERT INTO persons (first_name, last_name, email)
                VALUES ('Peter', 'Parker', 'peterparker@mail.com')";
        // Executa o comando SQL INSERT
        $link->exec($sql);
        echo "Dados inseridos com sucesso.";
    } catch(PDOException $e){
        die("ERRO: Não foi possível executar o comando. " . $e->getMessage());
    }
    // fecha a ligação ao servidor
    unset($link);
?>
```

Recorde que o campo **id** foi definido como **AUTO_INCREMENT** na estrutura da tabela, querendo dizer que o seu valor será incrementado automaticamente em 1 unidade. Deste modo não é preciso a sua especificação no comando SQL. O mesmo acontece com o campo **created_at** que assume o valor da data corrente.

Tarefa: insere mais 2 registos na tabela. Agora insere um novo registo com um email já registado anteriormente. O que acontece?

3.2 Inserção de Múltiplos Registos

Embora menos frequente, podemos inserir vários registos na tabela de uma só vez. Interprete o comando SQL.

Cria um ficheiro com o nome **insert_multiples_data_table.php** e digita o seguinte conteúdo:

```
<?php
    require_once "config.php";

    try{
        // define o comando SQL que permite inserir vários registos
        $sql = "INSERT INTO persons (first_name, last_name, email) VALUES
            ('John', 'Rambo', 'johnrambo@mail.com'),
            ('Clark', 'Kent', 'clarkkent@mail.com'),
            ('John', 'Carter', 'johncarter@mail.com'),
            ('Harry', 'Potter', 'harrypotter@mail.com')";
        $link->exec($sql);
        echo "Dados inseridos com sucesso.";
    } catch(PDOException $e){
        die("ERRO: Não foi possível executar o comando. " . $e->getMessage());
    }
    // fecha a ligação ao servidor
    unset($link);
?>
```

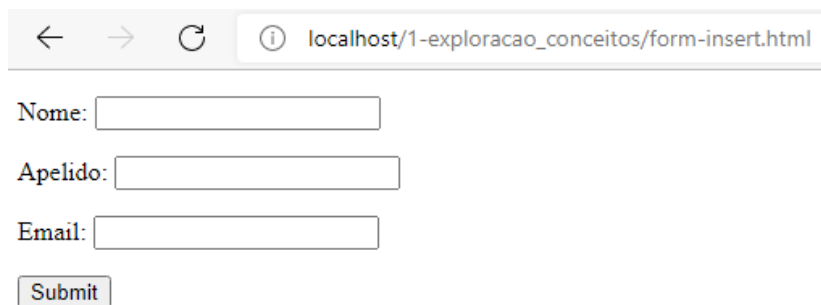
Observação muito importante:

O método **exec()** permite executar comandos SQL do tipo **insert, update e delete**.

3.3 Inserção de dados na BD recorrendo a um formulário

Uma operação mais comum consiste na inserção de dados na tabela recorrendo aos dados introduzidos num formulário.

Construiremos um formulário que ajudará na inserção de dados pelo utilizador.



← → ↻ ⓘ localhost/1-exploracao_conceitos/form-insert.html

Nome:

Apelido:

Email:

Atribui o nome **form-insert.html** ao ficheiro. Digita o seguinte conteúdo:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Inserção de Registos</title>
</head>
<body>
  <form action="insert.php" method="post">
    <p>
      <label for="firstName">Nome:</label>
      <input type="text" name="first_name" id="firstName">
    </p>
    <p>
      <label for="lastName">Apelido:</label>
      <input type="text" name="last_name" id="lastName">
    </p>
    <p>
      <label for="emailAddress">Email:</label>
      <input type="text" name="email" id="emailAddress">
    </p>
    <input type="submit" value="Submit">
  </form>
</body>
</html>
```

Observação:

Um processo de realizar as tarefas anteriores com mais segurança, consiste em evitar ataques do tipo *SQL Injection* (pesquisa na Internet o seu significado).

O processo envolve a aplicação de **duas fases** (*prepared statements*) : preparação e execução.

A fase de preparação consiste na definição de **parâmetros** cujos valores serão enviados ao servidor de BD separadamente. A fase de execução consiste em sincronizar a engrenagem definida (explicação simplificada).

A implementação deste processo passa pelo código que se apresenta a seguir e que deve constar num ficheiro com o nome **insert.php**

```
<?php
    require_once "config.php";

    try{
        // Define o comando SQL com 3 parâmetros
        $sql = "INSERT INTO persons (first_name, last_name, email)
                VALUES (:first_name, :last_name, :email)";

        // Prepara o comando SQL
        $stmt = $link->prepare($sql);

        // Ligação dos parâmetros aos campos da tabela
        $stmt->bindParam(':first_name', $firstname, PDO::PARAM_STR);
        $stmt->bindParam(':last_name', $lastname, PDO::PARAM_STR);
        $stmt->bindParam(':email', $email, PDO::PARAM_STR);

        $firstname = $_POST['first_name'];
        $lastname = $_POST['last_name'];
        $email = $_POST['email'];

        // executa a query preparada
        $stmt->execute();

        $last_id = $link->lastInsertId();
        echo "Registo $last_id foi inserido com sucesso.";
    } catch(PDOException $e){
        die("ERRO: Não foi possível executar o comando. " . $e->getMessage());
    }

    unset($link);
```

Observação:

Este é um exemplo muito básico de inserção de dados na tabela da BD recorrendo a um formulário. Pode otimizar o código, proceda à validação das entradas do utilizador antes de inserir os dados na tabela da base de dados.

Explicação pormenorizada do código:

```
$sql = "INSERT INTO persons (first_name, last_name, email)
VALUES (:first_name, :last_name, :email)";
```

Campos da tabela persons

Parâmetros que representam cada campo da tabela, segue a mesma ordem anterior.

Começa a fase de preparação do comando SQL

```
$stmt = $link->prepare($sql);
```

```
// Ligação dos parâmetros aos campos da tabela
```

```
$stmt->bindParam(':first_name', $firstname, PDO::PARAM_STR);
```

```
$stmt->bindParam(':last_name', $lastname, PDO::PARAM_STR);
```

```
$stmt->bindParam(':email', $email, PDO::PARAM_STR);
```

':email', \$_POST['email'] liga o parâmetro :email ao valor que existe em \$_POST['email']

Depois de preparada a engrenagem, chegou a altura de a executar.

```
$stmt->execute();
```

Os *prepared statements* oferecem dois ótimos benefícios:

- A query é preparada apenas uma vez, mas pode ser executada várias vezes.
- Não se aplicam funções de *escapes special caracteres* aos parâmetros, o driver trata deste processo automaticamente.

Estes benefícios significam duas coisas, agilidade e segurança.

No processo de ligação dos parâmetros aos campos da tabela pode-se utilizar dois métodos: **bindValue()** ou o **bindParam()**. O exemplo anterior utilizou o método **bindParam()**, agora tens que explorar o método **bindValue()**. Fica aqui um exemplo: **\$stmt->bindValue(':email', \$email);**

Também poderíamos utilizar *placeholders*:

```
INSERT INTO persons (first_name, last_name, email) VALUES (?, ?, ?);
```

Explora este conceito.

Ao inserir um novo registo na tabela, o campo definido como AUTO_INCREMENT gera um valor associado à chave primária. Existem situações em que a reutilização deste valor numa segunda tabela é muito importante (por exemplo, numa tabela relacionada). Para estas situações, o PHP disponibiliza o método **lastInsertId()** que devolve o número do último registo inserido.

Constantes predefinidas a utilizar no método bindParam()

PDO::PARAM_BOOL	Representa um tipo boolean
PDO::PARAM_INT	Representa um tipo int
PDO::PARAM_STR	Representa um tipo string, cha, varchar

Mais constantes podem ser encontradas em: <https://www.php.net/manual/en/pdo.constants.php>

Uma adaptação ao código apresentado no exemplo anterior, pode ser escrita na seguinte forma:

```
<?php
    require_once "config.php";

    $first_name = $_POST['first_name'];
    $last_name  = $_POST['last_name'];
    $email      = $_POST['email'];

    $data = [
        ':first_name' => $first_name,
        ':last_name'  => $last_name,
        ':email'      => $email,
    ];

    try{
        $sql = "INSERT INTO persons (first_name, last_name, email)
                VALUES (:first_name, :last_name, :email)";
        $stmt = $link->prepare($sql);
        $stmt->execute($data);

        $last_id = $link->lastInsertId();
        echo "Registo $last_id foi inserido com sucesso.";
    } catch(PDOException $e){
        die("ERRO: Não foi possível executar o comando. " . $e->getMessage());
    }
    unset($stmt);
    unset($link);
```

O array do exemplo anterior também poderia ser escrito da seguinte forma:

```
$data = [  
    'first_name' => $first_name,  
    'last_name' => $last_name,  
    'email' => $email,  
];
```

Tarefa:

1. Juntar num só ficheiro o formulário (form-insert.html) e o procedimento de inserção do registo na base de dados (insert.php).
2. Validar os dados inseridos nos campos do formulário, considerando que os campos first_name e email são de preenchimento obrigatório e válidos.
3. Enviar mensagens de erro caso os campos do formulário não estejam preenchidos ou tenham valores inválidos.
4. Em caso de erro na submissão do formulário, os campos preenchidos corretamente têm que manter os mesmos dados iniciais. Os campos com valores incorretos ficam em branco.

Vejamos a inserção de Múltiplos registos recorrendo a uma só fase de preparação. Neste caso, sem utilizar formulários, vamos explorar este conceito.

Crie um ficheiro com o nome **insert_multiple_data_parameters.php** e digita o seguinte conteúdo:

```
<?php  
require_once "config.php";  
try{  
    $sql = "INSERT INTO persons (first_name, last_name, email)  
          VALUES (:first_name, :last_name, :email)";  
    $stmt = $link->prepare($sql);  
  
    $stmt->bindParam(':first_name', $first_name, PDO::PARAM_STR);  
    $stmt->bindParam(':last_name', $last_name, PDO::PARAM_STR);  
    $stmt->bindParam(':email', $email, PDO::PARAM_STR);  
  
    /* Atribuição de valores nos parâmetro de cada registo */  
    $first_name = "Hermione";  
    $last_name = "Granger";  
    $email = "hermionegranger@mail.com";  
    $stmt->execute();
```

```

    $first_name = "Ron";
    $last_name = "Weasley";
    $email = "ronweasley@mail.com";
    $stmt->execute();
    echo "Registos inseridos com sucesso.";
} catch(PDOException $e){
die("ERRO: A query não foi preparada ou executada: $sql. " . $e->getMessage());
}
unset($stmt);
unset($link);

```

4 Listagem de registos

4.1 Selecionar dados de uma tabela

Pretende-se executar consultas (*queries*) de seleção. Repare que agora recorremos ao método **query** para executar o comando SQL **SELECT**.

Para obter os resultados do comando select podemos utilizar métodos diferentes. Analisa a seguinte tabela.

Método	Objetivo
fetch()	Retorna a próxima linha do resultado.
fetchAll()	Retorna um array com todos os resultados.
fetchObject()	Retorna a próxima linha do resultado como objeto.
fetchColumn()	Retorna uma coluna da próxima linha do resultado.

O output do ficheiro index.php

←
→
↻
localhost/1-exploracao_conceitos/index.php

Detalhes do Empregados

Exemplificação do CRUD com empregados [Adicionar Empregado](#)

#	Nome	Apelido	Email	Ação
1	Adelino	Amaral	aa@sapo.pt	Ver Registo Atualizar Registo Apagar Registo
2	Raul	Gouveia	rg@gmail.com	Ver Registo Atualizar Registo Apagar Registo

Total de empregados: 2

Crie um ficheiro com o nome **index.php** e digita o seguinte conteúdo:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Dashboard</title>
</head>
<body>
    <h3>Detalhes do Empregados</h3>
    <p>Exemplificação do CRUD com empregados
        <a href="create.php">Adicionar Empregado</a>
    </p>

    <?php
        require_once "config.php";

        $sql = "SELECT * FROM persons";
        if ($result = $link->query($sql))
        {
            // verifica se existem registos
            if ($result->rowCount() > 0) {
                echo "<table>";
                echo "<thead>";
                echo "<tr>";
                echo "<th>#</th>";
                echo "<th>Nome</th>";
                echo "<th>Apelido</th>";
                echo "<th>Email</th>";
                echo "<th>Ação</th>";
                echo "</tr>";
                echo "</thead>";
                echo "<tbody>";

                while ($row = $result->fetch()) {
                    echo "<tr>";
                    echo "<td>" . $row['id'] . "</td>";
                    echo "<td>" . $row['first_name'] . "</td>";

                    echo "<td>" . $row['last_name'] . "</td>";
```

```

        echo "<td>" . $row['email'] . "</td>";
        echo "<td>";
        echo "<a href='read.php?id=" . $row['id'] . "' title='Ver
Registo'>Ver Registo </a> &nbsp;&nbsp;&nbsp;| &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;";
        echo "<a href='update.php?id=" . $row['id'] . "'
title='Atualizar Registo'>Atualizar Registo </a> &nbsp;&nbsp;&nbsp;|&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;";
        echo "<a href='delete.php?id=" . $row['id'] . "'
title='Eliminar Registo'> Apagar Registo</a>";
        echo "</td>";
        echo "</tr>";
    }
    echo "</tbody>";
    echo "</table>";

    // determina o total dos salários
    $sql = "SELECT COUNT(email) as Total FROM persons";
    if ($result = $link->query($sql)){
        $row = $result->fetch();
        echo '<div><br><br>';
        echo "Total de empregados: " . $row['Total'];
        echo '</div>';
    }
    // Liberta os registos poupando memória
    unset($result);
} else {
    echo "<p><em>Não há empregados registados.</em></p>";
}
} else {
    echo "ERRO: $sql. " . $mysqli->error;
}
// Fecha a ligação ao servidor
unset($link);
?>
</body>
</html>

```

Observa: o método **fetch()** retira um registo de cada vez no conjunto de registos guardados em \$result, o registo está no formato array. O método **rowCount()** devolve o número de registos da query.

Quando utilizado outro tipo de BD que não BD em MySQL, o método `rowCount()` pode não devolver o número de registos resultantes do comando `SELECT`. Neste caso, recorra à função `COUNT(*)` do SQL e ao método `fetchColumn()` para obter o número correto de registos. No entanto, o método **`rowCount()`** devolve o número de registos afetados pela execução dos comandos `DELETE`, `INSERT`, o `UPDATE`.

Vejamos mais um exemplo:

```
<?php
$sql = "SELECT COUNT(*) FROM persons WHERE id > 10";
$res = $link->query($sql);
$count = $res->fetchColumn();

print "Existem " . $count . " registos.";
```

Para obter os dados de um determinado registo, poderemos passar o **id** da tabela *persons* pela URL. O ficheiro que permite apresentar a informação de um registo individual irá chamar-se `read.php`, como veremos mais à frente.

```
echo "<a href='read.php?id=" . $row['id'] . "' title='Ver Registo'>Ver Registo </a>
```

As outras operações, de eliminar e atualizar seguem o mesmo raciocínio.

Exemplos de consultas:

- Lista todos os registos cujo campo `first_name` toma o valor 'john'

```
$sql = "SELECT * FROM persons WHERE first_name='john'"
```

- Lista os registos de 2 a 4

```
SELECT * FROM persons LIMIT 1, 3;
```

- Lista os primeiros 3 registos

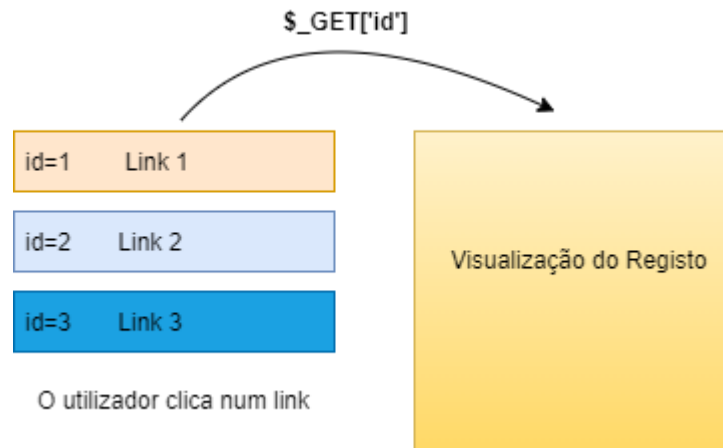
```
SELECT * FROM persons LIMIT 3
```

- Ordena os registos da tabela pelo campo `first_name`

```
SELECT * FROM persons ORDER BY first_name
```

4.2 Listagem individual

Como vimos no código da listagem anterior, por vezes há necessidade de passar variáveis de uma página para outra, como foi o caso de passar o 'id' da página `index.php` para a página `read.php`. Por norma, as variáveis existentes numa página não são visíveis noutras páginas, a não ser que sejam variáveis de sessão ou cookies. O valor da variável 'id' foi passada pela URL. Na página `read.php` podemos obter o valor da variável `id` através do array associativo `$_GET[]`.



É importante considerar que podemos aplicar os *prepared statements* ao comando SELECT. Analise o seguinte exemplo.

Crie um ficheiro com o nome **read.php** e digite o seguinte conteúdo:

```
<?php

// Obtém o id pela URL
if(isset($_GET["id"]) && !empty(trim($_GET["id"])))
{
    require_once "config.php";

    $sql = "SELECT * FROM persons WHERE id = :id";
    if($stmt = $link->prepare($sql))
    {
        // Liga o placeholder à variável
        $stmt->bindParam(":id", $param_id);
        $param_id = trim($_GET["id"]);
        if($stmt->execute())
        {
            if($stmt->rowCount() == 1)
            {
                /* Obtém um registo como array associativo. */
                $row = $stmt->fetch(PDO::FETCH_ASSOC);
                $first_name = $row["first_name"];
                $last_name = $row["last_name"];
                $email = $row["email"];
            }
        }
    }
}
```

```

    } else{

        // o registo não foi encontrado
        $erro = "O registo não foi encontrado";

    }
} else{
    $erro = "Ocorreu um erro. Tente mais tarde.";
}
}
unset($stmt);
unset($link);
} else{
    // O parâmetro id não veio na URL. Redirecionar para uma página de erro.
    $erro = "Ocorreu um erro. Tente mais tarde.";
}
?>

```

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Ver Registo</title>
</head>
<body>
    <?php
        if(isset($erro))
        {
            echo '<div>';
            echo $erro;
            echo '</div>';
        }
    ?>
    <div>
        <table class="table">
            <thead>
                <tr>
                    <th scope="col">#</th>
                    <th scope="col">Nome</th>
                    <th scope="col">Apelido</th>
                    <th scope="col">Email</th>

```

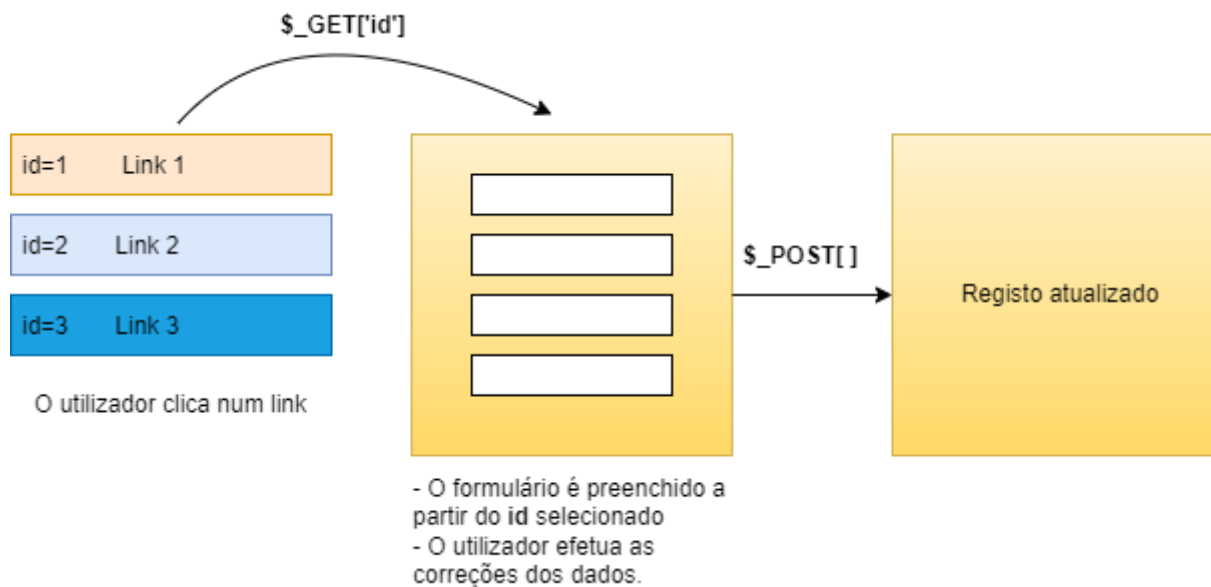
```

        </tr>
    </thead>
    <tbody>
        <tr>
            <td scope="row"><?php echo $row["id"]; ?></td>
            <td><?php echo $row["first_name"]; ?></td>
            <td><?php echo $row["last_name"]; ?></td>
            <td><?php echo $row["email"]; ?></td>
        </tr>
    </tbody>
</table>
<p><a href="index.php">Voltar</a></p>
</div>
</body>
</html>

```

5 Atualizar dados na tabela

Vejamos como iremos atualizar um registo na tabela de uma base de dados.



O exemplo seguinte permite atualizar um registo da tabela.

Crie um ficheiro com o nome **update.php** e digite o seguinte conteúdo:

```
<?php

require_once "config.php";

// Inicializa variáveis
$first_name = $last_name = $email = "";
$first_name_err = $last_name_err = $email_err = "";

// Deteta se o formulário foi submetido para fazer a atualização.
if(isset($_POST["id"]) && !empty($_POST["id"]))
{
    // Obtém o valor do id
    $id = $_POST["id"];

    // Validação do nome, aceita apenas letras e espaço
    $input_first_name = trim($_POST["first_name"]);
    if(empty($input_first_name)){
        $name_err = "Insira o nome.";
    } elseif(!filter_var($input_first_name, FILTER_VALIDATE_REGEXP,
array("options"=>array("regexp"=>"/^[a-zA-Z\s]+$/")))){
        $first_name_err = "Insira um nome válido.";
    } else{
        $first_name = $input_first_name;
    }

    // Validação do apelido, aceita apenas letras
    $input_last_name = trim($_POST["last_name"]);
    if(empty($input_last_name)){
        $last_name_err = "Insira o apelido.";
    } elseif(!filter_var($input_last_name, FILTER_VALIDATE_REGEXP,
array("options"=>array("regexp"=>"/^[a-zA-Z\s]+$/")))){
        $last_name_err = "Insira um apelido válido.";
    } else{
        $last_name = $input_last_name;
    }

    $input_email = trim($_POST["email"]);
    if(empty($input_email)){
```

```

        $email_err = "Insira o email.";
    } elseif(!filter_var($input_email, FILTER_VALIDATE_EMAIL)){
        $email_err = "Insira um email válido.";
    } else{
        $email = $input_email;
    }

// Verifica se existem erros
if(empty($first_name_err) && empty($last_name_err) && empty($email_err))
{
    // Prepara a sentença/comando
    $sql = "UPDATE persons SET first_name=:first_name,
last_name=:last_name, email=:email WHERE id=:id";

    if($stmt = $link->prepare($sql)){
        $stmt->bindParam(":first_name", $param_first_name);
        $stmt->bindParam(":last_name", $param_last_name);
        $stmt->bindParam(":email", $param_email);
        $stmt->bindParam(":id", $param_id);

        $param_first_name = $first_name;
        $param_last_name = $last_name;
        $param_email = $email;
        $param_id = $id;

        if($stmt->execute())
        {
            // Records updated successfully. Redirect to landing page
            header("location: index.php");
            exit();
        } else{
            echo "Algo aconteceu, tente mais tarde.";
        }
    }

    unset($stmt);
}

unset($link);

```

```

} else{

    // Verifica se o ID foi passado pela URL
    if(isset($_GET["id"]) && !empty(trim($_GET["id"])))
    {
        // Obtém o parâmetro da URL
        $id = trim($_GET["id"]);

        // Obtém os resultados para mostrar nas input's
        $sql = "SELECT * FROM persons WHERE id = :id";
        if($stmt = $link->prepare($sql))
        {
            $stmt->bindParam(":id", $param_id);
            $param_id = $id;

            if($stmt->execute())
            {
                if($stmt->rowCount() == 1){
                    $row = $stmt->fetch(PDO::FETCH_ASSOC);
                    $first_name = $row["first_name"];
                    $last_name = $row["last_name"];
                    $email = $row["email"];
                } else{
                    // O ID é inválido
                    header("location: error.php");
                    exit();
                }
            } else{
                echo "Oops! algo de errado aconteceu...";
            }
        }
        unset($stmt);
        unset($link);
    } else{
        header("location: error.php");
        exit();
    }
}
?>

```

```

<!DOCTYPE html>
<html lang="pt-pt">
<head>
    <meta charset="UTF-8">
    <title>Atualizar Registo</title>
</head>
<body>
    <h2>Atualize os seus dados</h2> <hr>
    <p>Altere os campos que pretende modificar.</p>
    <form action="<?php echo
htmlspecialchars(dirname($_SERVER['REQUEST_URI'])); ?>" method="post">
        <div>
            <label>Nome</label>
            <input type="text" name="first_name" value="<?php echo $first_name; ?>">
            <span><?php echo $first_name_err; ?></span>
        </div>
        <div>
            <label>Apelido</label>
            <input type="text" name="last_name" value="<?php echo $last_name; ?>">
            <span><?php echo $last_name_err; ?></span>
        </div>
        <div>
            <label>Email</label>
            <input type="text" name="email" value="<?php echo $email; ?>">
            <span><?php echo $email_err; ?></span>
        </div>

        <!-- Guarda o ID para posteriormente atualizar o registo correto -->
        <input type="hidden" name="id" value="<?php echo $id; ?>"/>

        <input type="submit" class="btn btn-primary" value="Gravar">
        <a href="index.php" class="btn btn-default">Cancelar</a>
    </form>
</body>
</html>

```

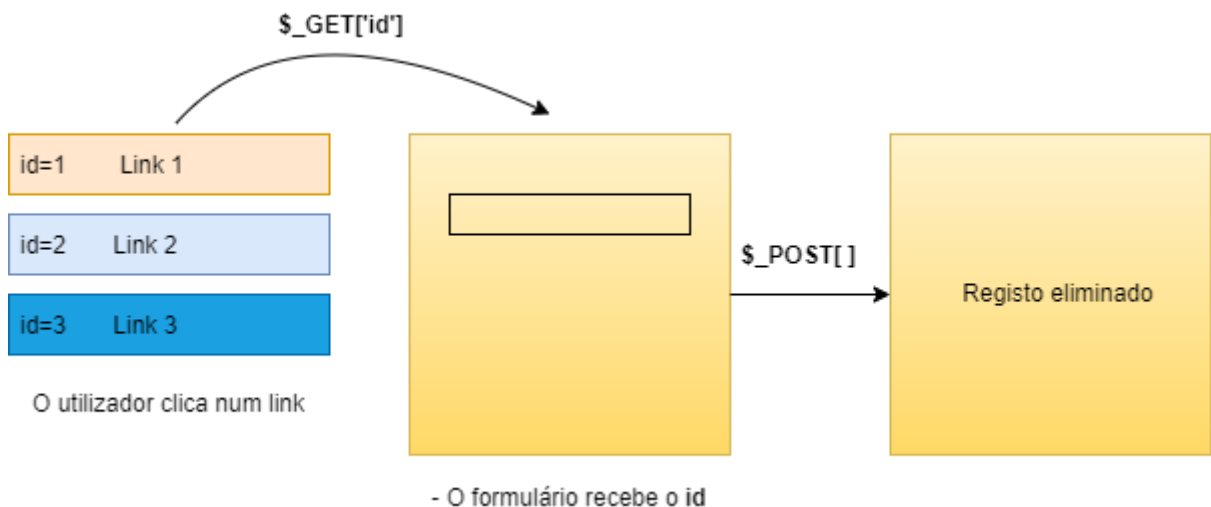

Exemplos de comandos:

```
$sql = "UPDATE persons SET email='peterparker_new@mail.com' WHERE id=1";
```

A clausura WHERE no comando UPDATE especifica o registo ou os registos a serem atualizados. Se omitir a clausura WHERE, todos os registos serão atualizados.

6 Apagar dados na tabela

Por fim, pretende-se eliminar registos da tabela.



Interpretar o código do exemplo seguinte.

Crie um ficheiro com o nome **delete.php** e digite o seguinte conteúdo:

```
<?php
// submissão do formulário
if(isset($_POST["id"]) && !empty($_POST["id"])){
    // Include config file
    require_once "config.php";

    // Prepare a delete statement
    $sql = "DELETE FROM persons WHERE id = :id";

    if($stmt = $link->prepare($sql)){
        $stmt->bindParam(":id", $param_id);
        $param_id = trim($_POST["id"]);
        if($stmt->execute()){
            // Registo eliminado com sucesso.
```

```

        header("location: index.php");
        exit();
    } else{
        echo "Oops! Algo de errado aconteceu. Tente mais tarde.";
    }
}
unset($stmt);

unset($link);
} else{
    // verifica se o parâmetro não foi passado pela URL
    if(empty(trim($_GET["id"]))) {
        // URL doesn't contain id parameter. Redirect to error page
        header("location: error.php");
        exit();
    }
}
?>

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>View Record</title>
</head>
<body>
    <div>
        <h1>Apagar Registo</h1>
    </div>
    <form action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]); ?>"
method="post">
        <div>
            <input type="hidden" name="id" value="<?php echo trim($_GET["id"]);
?>"/>

            <p>Tem a certeza de que quer apagar o registo?</p><br>
            <p>
                <input type="submit" value="Yes">
                <a href="index.php">No</a>
            </p>

```

```

        </div>
    </form>
</body>
</html>

```

A deteção da submissão do formulário deve ser melhorada tendo em atenção à matéria lecionada.

A clausura WHERE no comando DELETE especifica o registo ou os registos a serem apagados. Se omitir a clausura WHERE, todos os registos serão apagados.

Recapitulando:

Após abrir uma conexão podemos interagir com o banco utilizando 3 métodos da classe PDO:

Método	Retorno	Objetivo
exec	int	Utilizado no insert, update e delete.
query	PDOStatement	Utilizado em resultados tabulares, comando select.
Prepare	PDOStatement	Cria um <i>prepared statement</i> , utilizado em dados variáveis.

Conteúdo

1	PDO.....	1
2	Configuração de acesso ao servidor MySQL e à base de dados.....	2
2.1	Criar a base de dados e as tabelas	3
2.2	Criar a conexão ao servidor MySQL	4
3	Inserção de Registos.....	7
3.1	Inserir dados numa tabela.....	7
3.2	Inserção de Múltiplos Registos	7
3.3	Inserção de dados na BD recorrendo a um formulário.....	8
4	Listagem de registos.....	14
4.1	Selecionar dados de uma tabela	14
4.2	Listagem individual.....	17
5	Atualizar dados na tabela.....	20
6	Apagar dados na tabela.....	25