



CAPÍTULO

BASES DE DADOS

JPSantos
2023

6. BASES de DADOS

6.1. Conceitos

De uma forma simplista poderíamos dizer que uma base de dados mais não é que um repositório de dados. Estes dados podem ser organizados num conjunto de tabelas, e as tabelas podem estar relacionadas entre si. Para gerir e armazenar estas tabelas e em disco podem ser utilizados programas conhecidos por “gestores de base de dados”.

Sistema de gestão de bases de dados - SGBD

Existem vários gestores de bases de dados comerciais, ou em inglês *DBMS - Database Management System*, cada um deles armazena os dados em disco de uma forma específica, mas em geral todos eles conhecem e aceitam pedidos realizados por outros programas, desde que estes pedidos sejam escritos de acordo com a linguagem *SQL (Structured Query Language)*.

Os programas que iremos desenvolver em VBasic também podem fazer pedidos a estes gestores de bases de dados e enviar-lhes mensagens de texto escritas de acordo com a linguagem “SQL”.

6.2. Structured Query Language – SQL

A linguagem *SQL* permite consultar uma base de dados, inserir dados ou alterar os dados nela armazenados. Esta linguagem foi desenvolvida nos anos 70 pela IBM para aceder e gerir bases de dados relacionais. Foi posteriormente normalizada em 1986 pelo instituto *ANSI (American National Standards Institute)* sob a designação ANSI-92 SQL.

Um exemplo de uma pergunta/comando SQL é “*Select * from Motor*”. O gestor de bases de dados que receber este comando devolve todos os registos/linhas e campos/colunas da tabela *Motor*. Em Visual Basic existem diversos objetos, com diversos métodos que permitem enviar os comandos SQL para os gestores de bases de dados e guardar as respostas a essas perguntas.

Learn SQL in 1 Hour - SQL Basics for Beginners
<https://www.youtube.com/watch?v=9Pzj7Aj25lw>

Nas linhas seguintes são apresentados cinco tipos de comandos SQL:

“**CREATE TABLE** customer(CustID INTEGER PRIMARY KEY, LastName CHARACTER VARYING (25), FirstName CHARACTER VARYING (20), Address addr_typ, Phone CHARACTER VARYING (15) ARRAY [3]) “

“**SELECT** OrderID, CustomerID **FROM** Orders”

“**SELECT** * **FROM** Table1 **ORDER BY** field1 **ASC** or **SELECT** * **FROM** Table1 **ORDER BY** field1 **DESC**”

“**INSERT INTO** table_name(column_1, column_2, ..., column_n) **VALUES** (value_1, value_2, ..., value_n)”

“**INSERT INTO** Customers(CustomerID, CompanyName) **VALUES**('NWIND', 'Northwind Traders') **WHERE** id=1”

“**INSERT INTO** Motor(VelMotorOrdem, VelMotorLeitura) **VALUES** (?, ?)”

“**UPDATE** table_name **SET** column_1 = expression_1, column_2 = expression_2,..., column_n = expression_n [**WHERE** predicates]”

“**UPDATE** members **SET** first_name='Jose' **WHERE** id='1'”

“**UPDATE** COMP **SET** Pay = Pay + 100”

“**DELETE FROM** Customer **WHERE** FirstName = ‘David’ **AND** LastName = ‘Taylor’”

“**DELETE FROM** members **WHERE** first_name='Jose' ”

Modelo relacional

Uma base de dados relacional não é mais que um conjunto de relações (tabelas) relacionadas entre si. Cada relação (tabela, entidade) pode ser armazenada em disco num ficheiro. Não confundir relação com relacionamento entre relações. Cada relacionamento entre relações pode dar origem a uma nova tabela, um novo ficheiro, com atributos de ambas as tabelas que se pretende relacionar.

Cada linha da tabela pode ser designada por *registo*, *record* ou *tuplo*.

Cada coluna da tabela contém *atributos*, *campos* ou *fields* dos registos.

Ao número de atributos de uma relação dá-se o nome de *grau da relação* e

Ao número de registo de uma relação dá-se o nome de *cardinalidade* da relação.

A *chave primária* de cada relação de uma base de dados relacional consiste num atributo ou num conjunto de atributos dessa relação que permite identificar de forma biunívoca um só registo. Por exemplo o número do B.I, ou o número mecanográfico, são atributos únicos, não existem dois registos, relativos a dois alunos que tenham o mesmo número mecanográfico. O número mecanográfico é por isso uma boa chave primária de uma relação.

Projeto de uma base de dados normalizada

Uma base de dados deve ser projetada de forma a satisfazer um conjunto de requisitos, a base de dados:

- 1- Deve ter a capacidade necessária para armazenar todos os dados, atributos, necessários à empresa;
- 2- Pode ter dados duplicados mas não dados redundantes;
- 3- Deve conter o menor número possível de relações;
- 4- Deve ter todas as relações normalizadas para evitar problemas futuros na atualização, remoção ou atualização de dados.

Um dado pode estar duplicado em duas relações mas anda assim não ser redundante. Por exemplo, se esse dado, atributo, permitir o relacionamento entre duas relações ele pode estar presente em duas relações mas ainda assim ser indispensável. Contudo, os dados duplicados e redundantes têm de ser evitados.

Dados duplicados mas não redundantes

Os nomes João e Silva aparecem duplicados na tabela seguinte, mas esses dados são necessários.

NºEmpegado	NomeSupervisor
125	João
138	Silva
195	Silva
200	João

Tabela com dados duplicados mas não redundantes

Pois se fossem apagados (tabela seguinte) perdia-se informação útil e deixava de se saber quem eram os supervisores dos empregados nº 195 e nº200

NºEmpegado	NomeSupervisor
125	João
138	Silva
195	-
200	-

Falta de informação nesta tabela

Dados redundantes

Os telefones dos supervisores João e Silva estão duplicados na tabela seguinte e são também redundantes.

NºEmpegado	NomeSupervisor	TelSup
125	João	3051
138	Silva	2222
195	Silva	2222
200	João	3051

Tabela com dados redundantes

Podemos observar na tabela seguinte que se os números de telefones duplicados na tabela fossem eliminados continuava a ser possível saber quais eram os números de telefone dos supervisores. Contudo, a tabela passava a ter campos vazios o que levantaria problemas na consulta e tratamento dos dados.

NºEmpegado	NomeSupervisor	TelSup
125	João	3051
138	Silva	2222
195	Silva	-
200	João	-

Tabela sem dados redundantes mas com campos nulos

Dados duplicados mas não redundantes

A solução para não existirem campos redundantes na tabela passa pela sua subdivisão em duas novas tabelas. As duas tabelas seguintes contêm toda a informação da anterior sem dados redundantes ou campos nulos.

NºEmpegado	NomeSupervisor
125	João
138	Silva
195	Silva
200	João

NomeSupervisor	TelSup
João	3051
Silva	2222

Ao invés de usar uma só relação com todos os dados relativos a uma empresa é preferível subdividir essa *relação universal* em várias relações, cada uma com menor cardinalidade e grau pois é mais fácil de gerir e evita problemas futuros na gestão da base de dados. A esse processo de subdivisão uma tabela universal em várias tabelas, de acordo com um conjunto de regras que iremos abordar nas secções seguintes, dá-se o nome de *normalização da base de dados*.

Uma relação não Universal

NºAluno	NomeAluno	NºQuarto	TelQuarto	Cadeira	Semestre	Nota
3215	João	120	2136	MAT122	SET84	6.4
				CIE120	SET84	9.6
				FIS230	JUN85	8.4
3462	Silva	238	2344	MAT122	JUN85	9.2
				MAT122	JUN84	9.2
				MAT123	JUN85	14
3567	Oscar	120	2136	PSI220	JUN85	14.8
				CIE239	JUN84	13.2
				EGR171	SET84	14
4756	Alex	345	3321	FIS141	SET84	7.2
				MUS389	SET83	16

Uma relação Universal

NºAluno	NomeAluno	NºQuarto	TelQuarto	Cadeira	Semestre	Nota
3215	João	120	2136	MAT122	SET84	6.4
3215	João	120	2136	CIE120	SET84	9.6
3215	João	120	2136	FIS230	JUN85	8.4
3215	João	120	2136	MAT122	JUN85	9.2
3462	Silva	238	2344	MAT122	JUN84	9.2
3462	Silva	238	2344	MAT123	JUN85	14
3462	Silva	238	2344	PSI220	JUN85	14.8
3567	Oscar	120	2136	CIE239	JUN84	13.2
3567	Oscar	120	2136	EGR171	SET84	14
3567	Oscar	120	2136	FIS141	SET84	7.2
4756	Alex	345	3321	MUS389	SET83	16

Normalização da base de dados

As tabelas/relações de uma base de dados podem estar na primeira forma normal (1FN), segunda forma normal (2FN), na terceira forma normal (3FN) ou na Forma Normal de Boyce Codd.

1FN: Atomicidade, cada atributo tem valores indivisíveis/atômicos. Não há grupos de valores repetidos.

2FN: cada tabela da BD, além de estar na 1FN, sem dependências parciais. Todos os atributos de cada tabela devem estar dependentes de parte dos atributos da chave primária. Elimina dependências parciais dos atributos a partes da chave primária.

3FN: Todos os atributos de uma tabela dependem inequivocamente da chave primária, sem dependências transitivas. Podem ter duas ou mais chaves candidatas, atributos em comum nas chaves candidatas.

Dependência transitiva: quando um atributo da tabela não depende da chave da tabela mas de outro da mesma tabela que não é parte da chave primária.

Dependência multivalorada:

FNBC : os únicos determinantes da tabela são a chave primária. Todos os atributos da tabela dependem da sua chave primária.

Diagramas para representar a base de dados e auxiliar na sua normalização

Existem vários diagramas, símbolos e regras para representar uma base de dados. Por exemplo, [Diagramas de Dependências funcionais \(DFD\)](#), [Diagramas de Entidade Relacionamento \(DER\)](#), e mais recentemente os [diagramas de classes da Unified Modelling Language \(UML - Diagrama de classes\)](#)

6.3. Base de dados MySQL

O servidor de base de dados MySQL pode ser instalado isoladamente, ou instalando o XAMPP.

O XAMPP contém vários programas:

- Um servidor WEB (APACHE)
- Um servidor/gestor de base de dados (MariaDB MySQL)
- Interpretador de páginas PHP
- ...

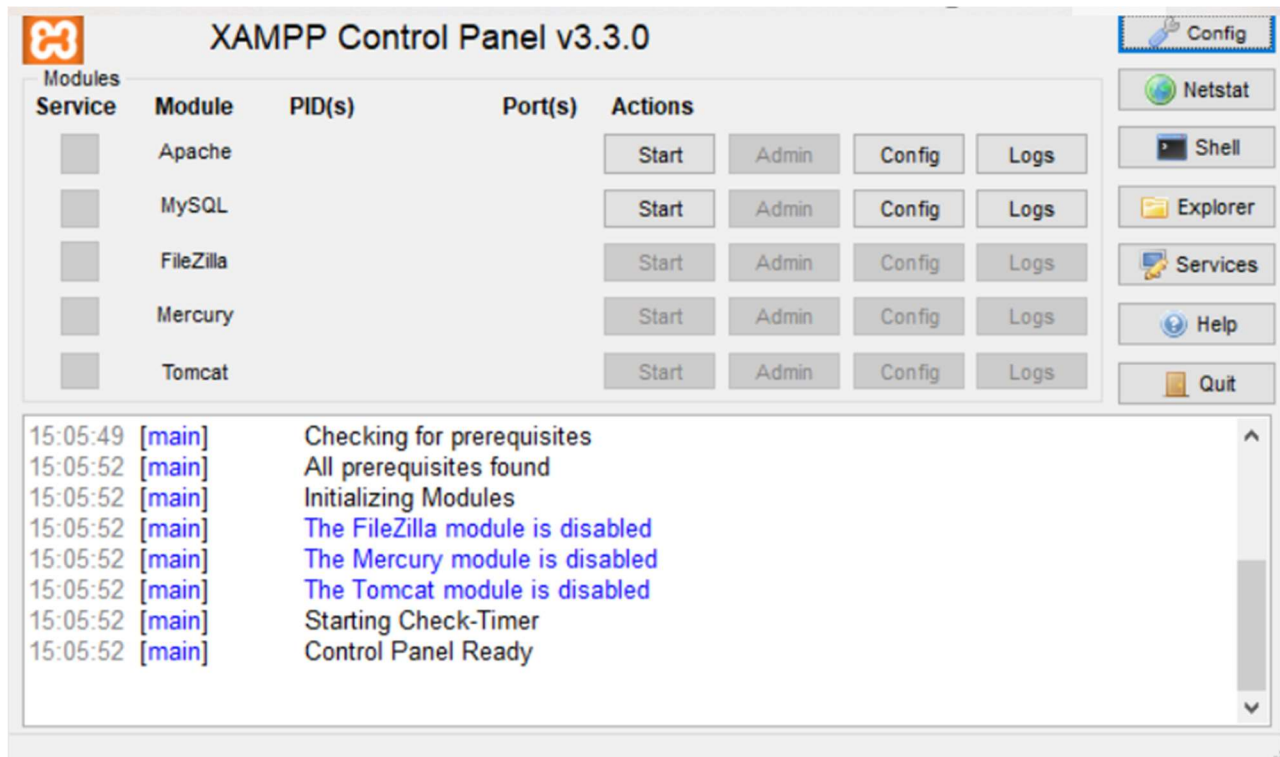
Deve executar o instalador do XAMPP como administrador. Anote a password que lhe será pedida quando o XAMPP instalar o MySQL/MariaDB.

Como o Apache fica à escuta da porta 80 e o MySQL na porta 3306, a firewall do windows deve permitir receber ligações nessas portas TCP/IP.

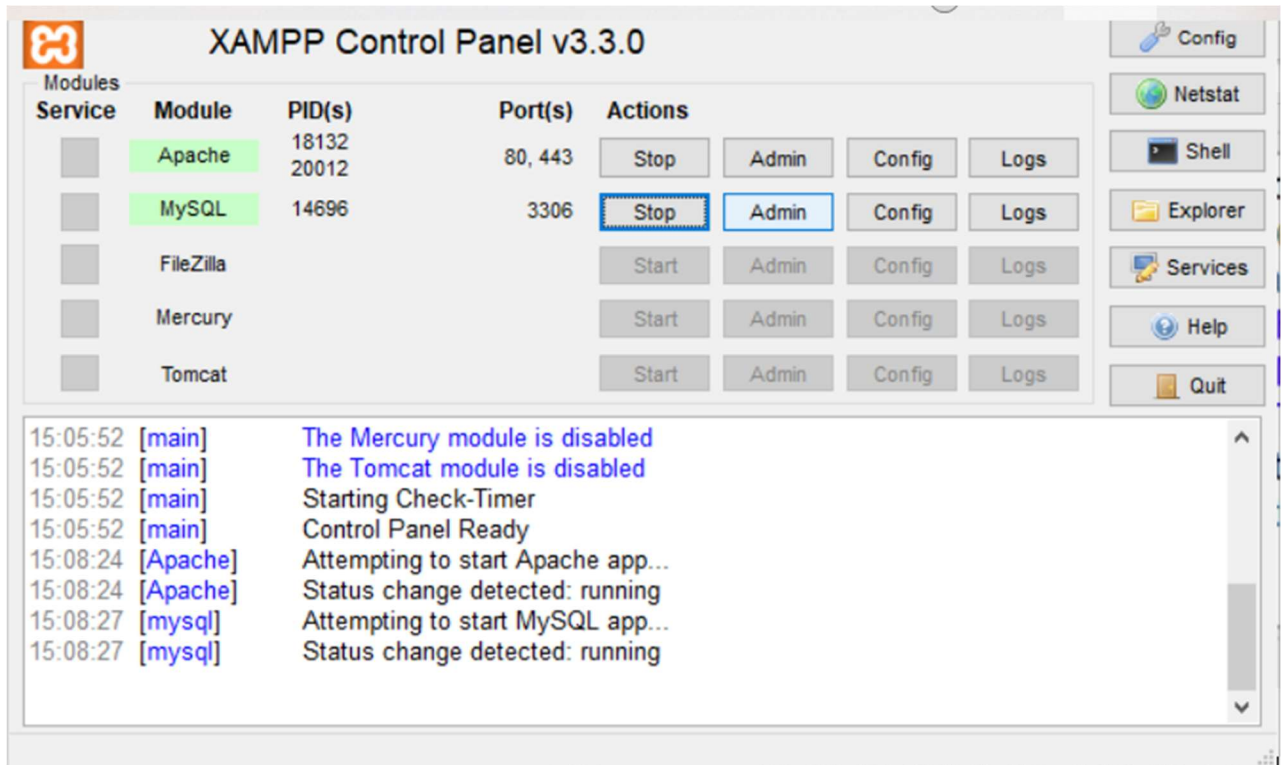
Para instalar o XAMPP aceda a:

<https://www.apachefriends.org/download.html>

Depois de instalado, deve executar o XAMPP panel Control:



Faça “**Start**” do Apache e do MySQL.
O FileZilla, o Mercury e o Tomcat não serão necessários.

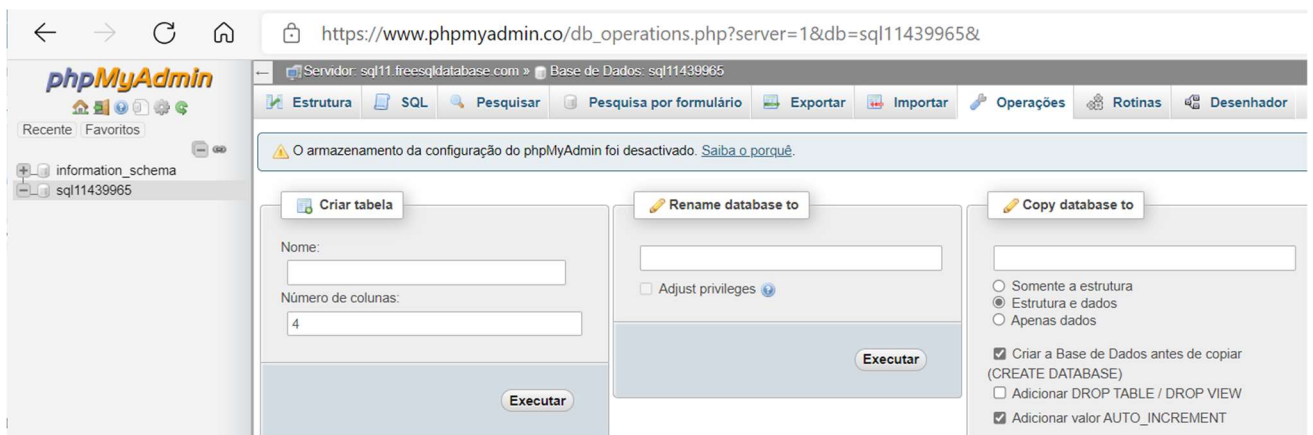


6.3.1. PhpMyAdmin

Para ativar o PhpMySQL, no painel de controlo do XAMPP deve seleccionar o botão “Admin” do MySQL.

Use o PhpMyAdmin para criar uma tabela na base de dados:

- No lado esquerdo da figura, selecione com o rato o nome da base de dados “sql11439965”.
- Para “Criar tabela”, escreva o “nome” da tabela, neste exemplo “ESP_FluxoLuminoso”, o “número de colunas” e selecione o botão “Executar”.



Na janela seguinte, escreva o nome das várias colunas da tabela, escolha qual o tipo de dados que serão guardados em cada uma delas, depois selecione o botão “Guardar”.

Cada linha da tabela deve ter um número único, que não se repita, que possa ser a chave primária da tabela (relação). Esse número, nessa coluna, pode ser auto-incrementado “AI” pela base de dados se selecionar a checkbox AI correspondente. Existem muitos tipos de dados à escolha para cada coluna

CAPÍTULO

PHP

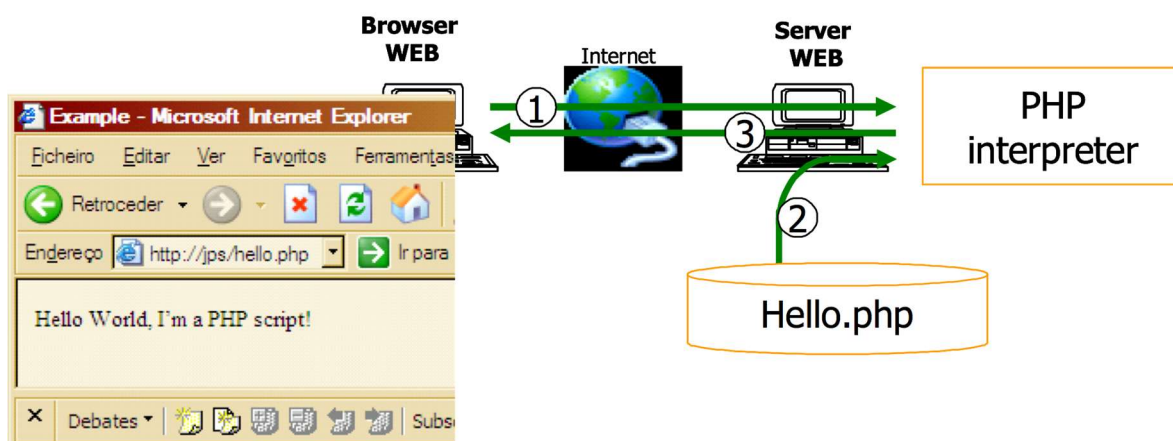
JPSantos
2023

7. PHP

O *PHP (Hypertext Preprocessor)* é uma linguagem simples, gratuita e aberta que permite elaborar páginas dinâmicas para a WEB. É possível escrever e gravar no disco, de um *ServidorWEB*, simples ficheiros de texto utilizando a linguagem PHP. Os ficheiros de texto escritos nesta linguagem não são interpretados pelo servidor WEB mas sim por outro programa por ele designado, neste caso o *PHP_interpreter*. Os programas escritos nesta linguagem podem ser inseridos dentro do código HTML.

O exemplo seguinte apresenta um ficheiro de texto do tipo *php* intitulado “Hello.php” guardado no disco duro do servidor. Quando este ficheiro é pedido pelo *BrowserWEB*, através do protocolo HTTP, o servidor WEB abre esse ficheiro e envia-o para o “*PHP_interpreter*” (figura 7.1). O “*PHP_interpreter*” que também está instalado no computador servidor, interpreta o ficheiro e converte para HTML as instruções PHP. Neste caso, o resultado da conversão consiste no texto “*Hello World, I’m a PHP script!*”, o servidor, por sua vez, reenvia esse texto para o *BrowserWEB* numa mensagem HTTP de resposta.

Para que as instruções PHP sejam processadas pelo “*PHP_interpreter*” e convertidas em HTML devem estar contidas na etiqueta `<?php ?>` e o ficheiro tem de ter a extensão `.php`



```
// ficheiro C:\inetpub\wwwroot\Hello.php

<html>
<head>
  <title>Example</title>
</head>
<body>

  <?php
  echo "Hello World, I'm a PHP script!";
  ?>
</body>
</html>
```

figura 7.1: Página PHP – Hello World

O programa que interpreta as páginas/programas PHP pode ser instalado em várias **plataformas** (Windows, Unix, Linux) e pode interagir com vários **servidores WEB** (IIS, Apache, Caudium, Netscape, OmniHTTPd, Sambar, Xitami,...)

Nos exemplos seguintes os programas PHP estão gravados no “home directory” do *ServidorWEB*. No caso de um servidor IIS/Microsoft o “home directory” é o “C:\inetpub\wwwroot”. No caso de um servidor Apache2 o “home directory” é por defeito o “C:\Programas\Apache Group\Apache2\htdocs”. No caso do XAMPP a “home directory” é o diretório “C:\Programas\XAMPP\htdocs”.

Os browsers WEB não possuem um interpretador de PHP. Para que o documento PHP seja convertido em HTML e o Browser o possa apresentar ao utilizador, o browser tem de aceder a um servidor WEB com um interpretador de PHP instalado:

<http://localhost/hello.php>

O documento tem de ter uma extensão *.php e pode conter apenas etiquetas HTML , apenas etiquetas PHP ou ambas, de forma alternada.

```
<HTML>
<BODY>
<!-- Comentário em HTML -->
Zona HTML<BR>

<?php
/* Os comentários em PHP podem ocupar várias linhas
   segunda linha de comentário */
echo "Zona PHP <BR>"; // Isto é um comentário: a instrução echo converte para HTML o que estiver entre
aspas
?>

<i>Outra zona HTML <BR></i>

<?PHP
echo "Outra Zona PHP <BR>";
?>

</BODY>
</HTML>
```

7.1. Tipos de variáveis

O nome das variáveis em PHP começa sempre por um cifrão \$ seguido de um conjunto de caracteres alfanuméricos. Existe apenas uma limitação, depois do cifrão, o primeiro carácter alfanumérico não pode ser um número. O interpretador de PHP distingue as letras minúsculas e maiúsculas no nome de uma variável.

```
<?php
$a = 1.234;           // cria uma variável
$b = 1.2e3;
$c = 7E-10;

// a instrução "echo" gera uma string HTML
echo $a . "<BR>" . $b . "<BR>" . $c . "<BR>"; // neste caso igual: 1.234<BR>1200<BR>7.0E-10<BR>

// a instrução "array" cria um array, neste caso um array com dois elementos, o primeiro elemento contém a
// string "bar" e o segundo elemento contém o valor "true". Os índices deste array em vez do habitual 1,2,3..
// são [foo] e [12]
$sarr = array("foo" => "bar", 12 => true);

echo $sarr["foo"];    // o elemento $sarr["foo"] contém o valor "bar"
echo "<BR>";          // gera o texto <BR> que provoca a mudança de linha
echo $sarr[12];       // o elemento $sarr[12] contém o valor 1
echo "<BR>";

$sarr = array("somearray" => array(6 => 5, 13 => 9, "a" => 42));

echo $sarr["somearray"][6]; // 5
echo "<BR>";
echo $sarr["somearray"][13]; // 9
echo "<BR>";
echo $sarr["somearray"]["a"]; // 42
?>
```

7.2. Tipos de operadores

7.2.1. Operadores aritméticos

Em PHP os operadores aritméticos são: +, -, *, /, %

O exemplo seguinte ilustra a sintaxe e finalidade dos operadores aritméticos.

```
<?PHP
$a = 10;
$b = 5;
echo $a + $b;        // retorna 15
echo "<BR>";          // muda de linha
echo $a - $b;        // retorna 5
echo "<BR>";
echo $a * $b;        // retorna 50
echo "<BR>";
echo $a / $b;        // retorna 2
echo "<BR>";
$b = 3;
echo $a / $b;        // retorna 3,3333
echo "<BR>";
echo $a % $b;        // retorna 1
echo $a % $b;        // retorna 1
?>
```

7.2.2. Operadores relacionais/comparação

Os operadores booleanos comparam dois valores e com base na comparação realizada retornam um valor booleano “Verdadeiro” ou “Falso”.

Maior que	>
Maior ou igual	>=
Menor que	<
Menor ou igual	<=
Igual	=
Diferente	!= ou <>

<?PHP

```
$A=10;
$B=15;
$C=20;

echo $A < $B; // O resultado da comparação é TRUE, 10 é menor que 15
echo "<BR>";
echo $B > $A; // O resultado da comparação é TRUE, 15 é maior que 10
echo "<BR>";
echo $A != $B; // O resultado da comparação é TRUE, 10 é diferente de 10
echo "<BR>";
echo $A == $A; // O resultado da comparação é TRUE, 10 é igual a 10
```

?>

7.2.3. Operadores lógicos

Intercepção de conjuntos	(AND)	and ou &&
Reunião de conjuntos	(OR)	or ou
Negação	(NOT)	!
Ou exclusivo	(XOR)	xor

<?PHP

```
$A=TRUE;
$B=FALSE;

echo ($A and $B); // O resultado é FALSE
echo ($B or $A); // O resultado é TRUE
echo (!$B); // O resultado é TRUE
echo ($A xor $A); // O resultado FALSE
```

?>

7.2.4. Operadores bit a bit

AND bit a bit	&
OR bit a bit	
Complemento bit a bit	~
Desloca um bit para esquerda	<<
Desloca um bit para a direita	>>
XOR bit a bit	^

O operador “&” tem dois operandos e faz uma intercepção lógica bit a bit dos dois operandos

<?PHP

```
$a = 18;           // 18 é igual ao número binário    00010010
$b = 170;          // 170 é igual ao número binário   10101010
echo $a & $b;      // Retorna o valor                00000010 = 2 decimal

?>
```

O operador “|” tem dois operandos e faz uma reunião lógica bit a bit dos dois operandos,

<?PHP

```
$a = 18;           // 18 é igual ao número binário    00010010
$b = 170;          // 170 é igual ao número binário   10101010
echo $a | $b;      // Retorna o valor                10111010 = 186 decimal

?>
```

O operador “^” permite fazer o XOR de dois bytes, bit a bit. Só quando dois bits são diferentes o resultado é um bit a “1”

<?PHP

```
$a = 18;           // 18 é igual ao número binário    00010010
$b = 170;          // 170 é igual ao número binário   10101010
echo $a ^ $b;      // Retorna o valor                10111000 = 184 decimal

?>
```

O operador Complemento “~” altera todos os bits do byte, os bits que valiam 0 passam a valer 1 e vice-versa.

<?PHP

```
$a = 18;           // 18 é igual ao número binário    00010010
echo ~$a;          // Retorna o valor                11101101 = 237 = -19 decimal

?>
```

Os operadores de deslocamento “>>” e “<<” deslocam os bits para a esquerda ou direita. No exemplo seguinte, os bits são deslocados uma posição para a direita e depois são deslocados duas posições para a esquerda

<?PHP

```
$a = 18;           // 18 é igual ao número binário    00010010
echo $a >> 1;      // desloca para a direita um bit e retorna o valor 00001001 = 9 decimal
echo "<BR>";        // Muda de linha
$a = 18;           // 18 é igual ao número binário    00010010
echo $a << 2;      // desloca para a esquerda dois bits e retorna o valor 01001000 = 72 decimal

?>
```

7.3. Estruturas de controlo

A execução de um programa é sequencial, cada instrução é gravada na memória do uC em posições consecutivas e é executada pela ordem com que foi gravada, a menos que o programador tenha inserido no próprio código instruções que permitam alterar essa sequência de execução.

7.3.1. WHILE

A instrução WHILE(*condição*){ ... } permite repetir a execução de algumas linhas código, mais exactamente das linhas de código que tenham sido escritas entre as chavetas desta instrução. Estas linhas de código são repetidas enquanto a *condição* booleana do WHILE for verdadeira.

A *condição* do ciclo WHILE pode conter vários operandos e operadores mas o resultado final das operações tem de ser um valor booleano, verdadeiro ou falso. Enquanto a condição for verdadeira o ciclo repete-se.

| | | |
|---|--|-----------------------------------|
| <i>while</i> (<i>condição</i>)
{
instruções;
instructors;
} | <pre>\$i = 0;
while (\$i < 3){
 \$i = \$i+1;
}
echo \$i;</pre> | Depois de 3 ciclos, retorna \$i=3 |
|---|--|-----------------------------------|

7.3.2. FOR

O ciclo FOR{ ... } permite repetir a execução das linhas de código que estiverem escritas entre as chavetas do próprio ciclo. Este ciclo é semelhante ao ciclo WHILE, contudo a “condição” do ciclo WHILE pode assumir um valor falso em função da execução das instruções do próprio ciclo, fazendo com que o ciclo possa terminar mais cedo ou tarde, ao passo que o ciclo FOR tem um número de repetições predefinido.

| | | |
|--|---|-----------------------------------|
| <i>for</i> (<i>\$i=0; \$i < n; \$i++</i>)
{
instruções;
instruções;
} | <pre>for (\$i=0; \$i< 3; \$i++){
 }
echo \$i;</pre> | Depois de 3 ciclos, retorna \$i=3 |
|--|---|-----------------------------------|

7.3.3. IF

Quando se pretende executar algumas linhas de código se, e apenas se “*IF*”, uma determinada condição for verdadeira e pelo contrário “*Else*” executar outras linhas de código se a condição for falsa, o programador pode usar a instrução IF.

| | |
|---|---|
| <i>if</i> (<i>condição</i>)
{
//se condição verdadeira
instruções;
} <i>else</i> {
//se condição falsa
instruções;
} | <pre>\$i= 5;
<i>if</i> (\$i < 10)
{
 //Se verdadeira
 echo “verdadeiro”;
}<i>else</i>{
 //Se falsa
 echo “falso”;
}</pre> |
|---|---|

Em suma, esta instrução permite que apenas um de dois conjuntos de instruções seja executado, em função de uma *condição* ser verdadeira ou falsa.

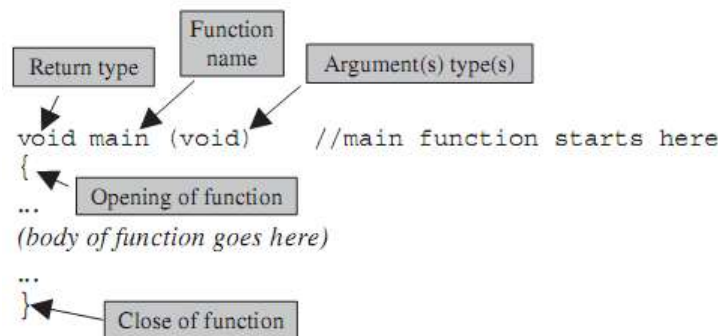
7.3.4. SWITCH

A instrução SWITCH permite que apenas um de muitos conjuntos de instruções seja executado, em função do valor numérico ou alfanumérico de uma variável.

| | |
|---|--|
| <pre><i>\$i</i> = 1; switch (<i>\$i</i>) { <i>case</i> 1: instruções; <i>break</i>; <i>case</i> 2: instruções; <i>break</i>; <i>default</i>: instruções; }</pre> | <pre><i>\$i</i> = 2; switch (<i>\$i</i>) { <i>case</i> 1: <i>echo</i> "1"; <i>break</i>; <i>case</i> 2: <i>echo</i> "2"; <i>break</i>; <i>default</i>: <i>echo</i> "default"; }</pre> |
| <pre><i>\$s</i> = 'a'; switch (<i>\$s</i>) { <i>case</i> 'a': instruções; <i>break</i>; <i>case</i> 'b': instruções; <i>break</i>; <i>default</i>: instruções; }</pre> | <pre><i>\$s</i> = 'a'; switch (<i>\$s</i>) { <i>case</i> 'b': <i>echo</i> "b"; <i>break</i>; <i>case</i> 'a': <i>echo</i> "a"; <i>break</i>; <i>default</i>: <i>echo</i> "default"; }</pre> |

7.4. Funções e procedimentos

As funções e os procedimentos permitem “agrupar” código, tornando mais fácil a sua organização e reutilização ao longo do programa, mas só as funções podem retornar valores.



7.4.1. Passagem de parâmetros por valor

As funções podem receber parâmetros por valor. Isto significa que quando a função é chamada o valor do(s) parâmetros é copiado para outra zona de memória do computador e essa zona de memória tem o(s) nomes definidos na função.

No exemplo seguinte, a função *duplica* recebe um parâmetro que é copiado para a zona de memória/variável *\$i*. Esta variável existe na memória do computador enquanto a função estiver a ser executada. Neste exemplo a função *echo* devolve o valor *10*.

```
<?php

    duplica(5);

    function duplica($i)
    {
        echo $i*2;
    }

?>
```

7.4.2. Passagem de parâmetros por referência

As funções e os procedimentos também podem receber parâmetros por referência. Neste caso o valor dos parâmetros não é copiado para uma nova zona de memória associada à execução da função, apenas é enviado como parâmetro(s) os endereços de memória onde esses valores se encontram armazenados. De facto, passam a existir dois nomes diferentes para a mesma zona de memória.

No exemplo seguinte a variável *\$str* e a variável *\$string* correspondem à mesma zona de memória do computador. Desta forma quando dentro da função a variável *\$string* é alterada estamos também a alterar a variável *\$str*. É por essa razão que a função *echo \$str* devolve o valor *Isto é uma string, e alguma coisa mais*.

```

<?php

$str = "Isto é uma string";
concatena($str);
echo $str;           // devolve 'Isto é uma string, e alguma coisa mais.'

function concatena(&$string)
{
    $string = $string . ' e alguma coisa mais';
}

?>

```

O exemplo seguinte ilustra a passagem de um parâmetro por referência, mais exactamente é passado o endereço de memória do array `$i[]`. Dessa forma, dentro da função, o array `$input` dentro da função refere-se à mesma zona de memória que a variável `$i[]`

```

<?php

$i[2];           // criação do array $i[2], este array tem dois elementos
$i[0]=1;         // é atribuído ao primeiro elemento de array o valor 1
$i[1]=2;         // é atribuído ao segundo elemento de array o valor 2

soma($i);        // é chamada a função "soma" e é passada a referência ao array $i

function soma(&$in)    // parâmetro passado por referência "&"
{ // dentro da função, o array $in corresponde à mesma zona de memória do array $i[]
    echo $in[0];      // devolve 1, pois o elemento $in[0] é o mesmo que $i[0]
    echo $in[1];      // devolve 2, pois o elemento $in[1] é o mesmo que $i[1]
}

?>

```

7.4.3. Retorno de valores

As funções podem retornar valores para o programa principal, a partir do qual foram chamadas. O exemplo seguinte ilustra uma função que recebe dois parâmetros por valor, ou por outras palavras recebe dois valores e devolve o resultado da soma desses dois valores.

```

<?php

$resultado= soma(5,7); // a função soma retorna o valor da soma dos dois argumentos 12
echo $resultado;       // a função echo devolve o conteúdo da variável $resultado 12

function soma($i,$j)
{
    return $i+$j;
}

?>

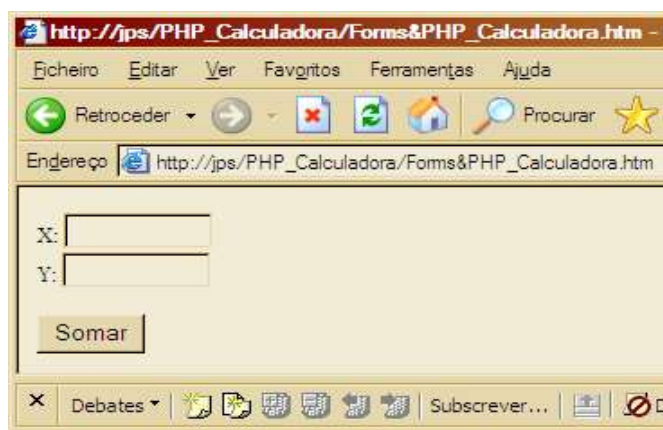
```

7.5. Exemplo PHP - Calculadora

Este exemplo apresenta uma calculadora desenvolvida em HTML e PHP. Esta calculadora apenas permite adicionar dois números.

O código em HTML necessário à criação da interface (“Form”) está gravado no servidor, mais exactamente no ficheiro “Calculadora.html” (canto inferior esquerdo), e o código PHP necessário ao processamento dos dados introduzidos nessa interface está gravado no ficheiro “Calculadora.php” (lado direito).

Para entendermos este exemplo necessitamos de saber utilizar a interface “form” disponível em HTML bem como as entradas “input” do tipo texto “Text”. Podemos ver no exemplo seguinte que uma das entradas de texto tem o nome “name” X e a outra o nome Y. Podemos também verificar que o utilizador ao seleccionar o botão “Somar” está de facto a executar (Action) o ficheiro escrito em PHP que irá processar os valores previamente introduzidos pelo utilizador no seu *BrowserWEB*. Quando o ficheiro PHP é executado no lado do servidor, ele pode aceder aos valores introduzidos pelo utilizador na interface “Form” usando o array \$_POST[nome do input da interface HTML]. Neste caso os elementos \$_POST[‘X’] e \$_POST[‘Y’] contêm os valores introduzidos pelo utilizador nas entradas X e Y da interface. A variável \$Z recebe o resultado da soma.



// ficheiro C:\Inetpub\wwwroot\ Calculadora.html

```
<html>
<body>
```

```
<form action="Calculadora.php" method="post" >
X: <input type="text" lenght="2" name="X" size="10"><br>
Y: <input type="text" length="2" name="Y" size="10">
<br><br>
<input type="submit" value="Somar">
</form>
</body>
</html>
```

// ficheiro C:\Inetpub\wwwroot\Calculadora.php

```
<html>
<body>

<?php
$Z=$_POST['X']+$_POST['Y'];
echo $_POST['X'],"+",$_POST['Y'] ,"=", $Z;
?>

<br><a href="Calculadora.html" > tentar outra vez
</a>
</body>
</html>
```

figura 122: Página PHP – Calculadora

7.6. Exemplo PHP – Acesso às bases de dados MySQL

A linguagem PHP permite estabelecer ligações TCP/IP com o gestor de bases de dados “MySQL server” e enviar-lhe comandos SQL. Quando o programa “MySQL server” recebe comandos de texto SQL na porta TCP 3306 executa esses comandos. Ao executar esses comandos o servidor MySQL pode por exemplo criar novas bases de dados, com tabelas, campos e registos.

Neste exemplo, vamos considerar a utilização de três computadores ligados à Internet. Num deles reside o Browser WEB, no outro reside o Servidor WEB e no terceiro reside o Servidor MySQL. As páginas WEB escritas em PHP são pedidas pelo Browser WEB mas são executadas/interpretadas no servidor WEB. Por sua vez o interpretador de páginas PHP, usado neste exemplo, estabelece uma ligação TCP/IP com o servidor MySQL e envia-lhe pedidos/comandos SQL, recebe as respostas do servidor MySQL e converte-as para a linguagem HTML. Nessa altura o servidor WEB envia essas páginas para o Browser WEB onde o utilizador poderá visualizar os dados presentes na base de dados MySQL.

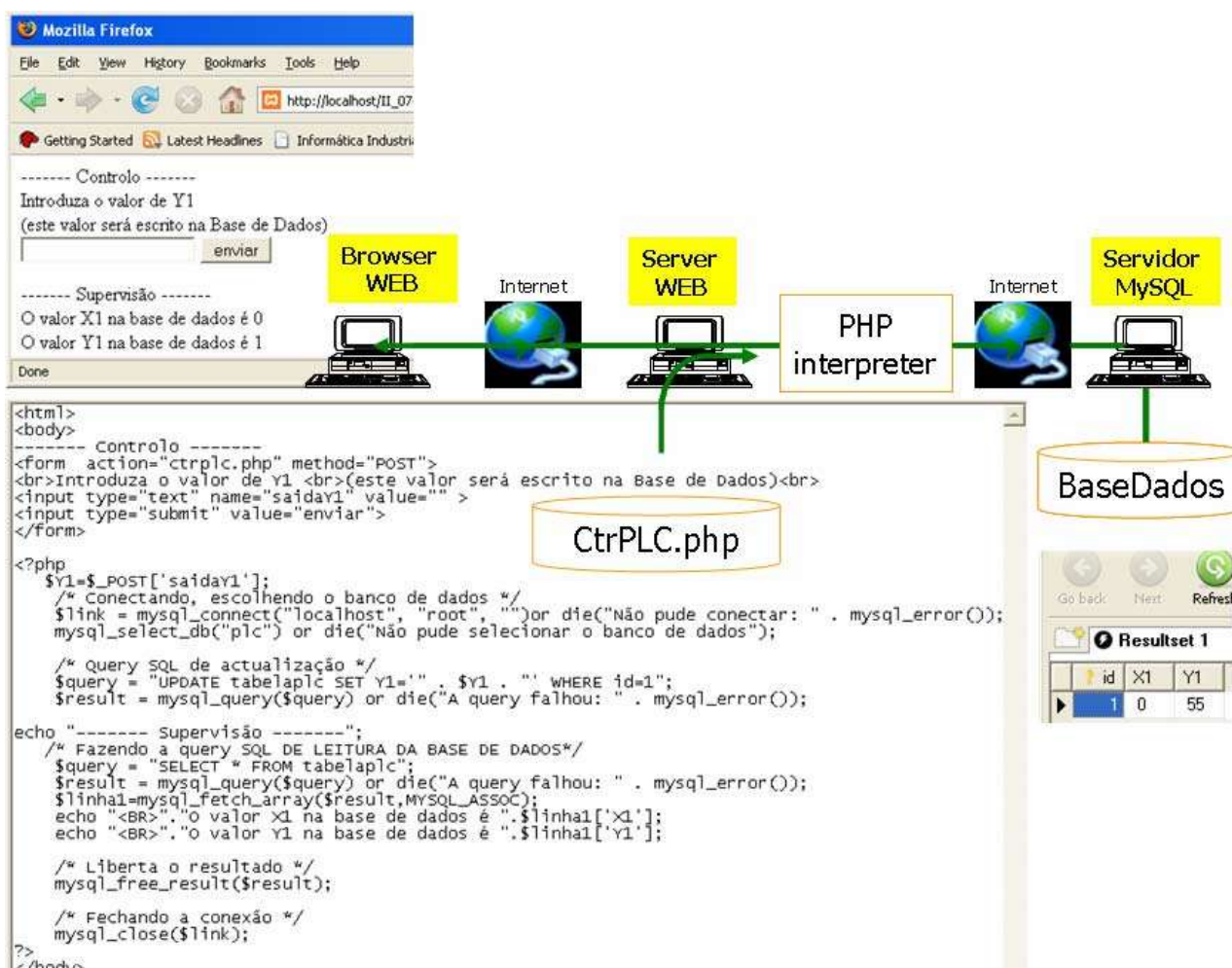


figura 123: PHP – Interações entre o browser WEB, o servidor WEB e o servidor MySQL

CtrPLC.php (usando o mysqli)

```
<html>
<body>
----- Controlo -----
<form action="ctrplc.php" method="POST">
Introduza os valores de Y1, <br>Este valor sera escritos na base de dados PLC"<br>
na tabela "tabelapl (id, X1, Y1)"<br><br>

<input type="text" name="saidaY1" value="1" >
<input type="submit" value="enviar">
</form>

<?php

$Y1=$_POST['saidaY1'];

/* Conectando, escolhendo o banco de dados */
$link = mysqli_connect("localhost", "root", "") or die("Nao pude conectar: " . mysqli_error());
mysqli_select_db($link,"plc") or die("Nao pude selecionar o banco de dados");

/* Query SQL de actualizacao */
$query = "UPDATE tabelapl SET Y1=" . $Y1 . " WHERE id=1";
$result = mysqli_query($link,$query) or die("A query falhou: " . mysqli_error());

echo "----- Supervisao -----<br>Os valores lidos na base de dados sao:";
/* Fazendo a query SQL DE LEITURA DA BASE DE DADOS*/
$query = "SELECT * FROM tabelapl";
$result = mysqli_query($link,$query) or die("A query falhou: " . mysqli_error());
$linha1=mysqli_fetch_array($result,MYSQLI_ASSOC);
echo " Y0:". $linha1['X1'];
echo " Y1:". $linha1['Y1'];

/* Liberta o resultado */
mysqli_free_result($result);

/* Fechando a conex.,o */
mysqli_close($link);

?>
</body>
</html>
```

7.6.1. Funções PHP para aceder ao Servidor MySQL

[mysql_affected_rows](#) -- Devolve o número de linhas afectadas na operação anterior com o [MySQL](#)

[mysql_change_user](#) -- Muda o utilizador

[mysql_client_encoding](#) -- Retorna o nome do conjunto de caracteres

[mysql_close](#) -- Fecha a ligação com o [MySQL](#)

[mysql_connect](#) -- Abre uma ligação com o servidor [MySQL](#)

[mysql_create_db](#) -- Cria um base de dados [MySQL](#)

[mysql_data_seek](#) -- Move o ponteiro interno do resultado

[mysql_db_name](#) -- Retorna os nomes das bases de dados

[mysql_db_query](#) -- Envia uma query ao [MySQL](#)

[mysql_drop_db](#) -- Apaga uma base de dados do [MySQL](#)

[mysql_errno](#) -- Retorna o valor numérico da mensagem de erro da operação anterior do [MySQL](#)

[mysql_error](#) -- Retorna o texto da mensagem de erro da operação anterior do [MySQL](#)

[mysql_escape_string](#) -- Escapa uma string para uso com o [mysql_query](#).

[mysql_fetch_array](#) -- Retorna uma linha do resultado da query sob forma de uma matriz associativa, matriz numérica ou ambas.

[mysql_fetch_assoc](#) -- Retorna uma linha do resultado e sob a forma de uma matriz associativa

[mysql_fetch_field](#) -- Retorna informação sobre uma coluna de um resultado como um objeto

[mysql_fetch_lengths](#) -- Retorna o tamanho de cada campo do resultado

[mysql_fetch_object](#) -- Retorna uma linha do resultado sob a forma de um objeto

[mysql_fetch_row](#) -- Retorna uma linha do resultado sob a forma de uma matriz numérica

[mysql_field_flags](#) -- Pega as flags do campo especificado no resultado

[mysql_field_len](#) -- Retorna o tamanho do campo

[mysql_field_name](#) -- Retorna o nome do campo especificado no resultado de uma query

[mysql_field_seek](#) -- Move o ponteiro do resultado para um campo especificado

[mysql_field_table](#) -- Retorna o nome da tabela onde esta o campo especificado

[mysql_field_type](#) -- Retorna o tipo do campo especificado em um resultado de query

[mysql_free_result](#) -- Liberta a memória do resultado de uma query

[mysql_get_client_info](#) -- Retorna informação da versão do cliente [MySQL](#)

[mysql_get_host_info](#) -- Retorna informação sobre o host do [MySQL](#)

[mysql_get_proto_info](#) -- Retorna informação do protocolo do [MySQL](#)

[mysql_get_server_info](#) -- Retorna informação do servidor [MySQL](#)

[mysql_info](#) -- Retorna informação sobre a última query

[mysql_insert_id](#) -- Retorna o ID gerado da operação INSERT anterior

[mysql_list_dbs](#) -- Lista as bases de dados disponíveis no servidor do [MySQL](#)

[mysql_list_fields](#) -- Lista os campos de uma tabela

[mysql_list_processes](#) -- Lista os processos [MySQL](#)

[mysql_list_tables](#) -- Lista as tabelas de uma base de dados [MySQL](#)

[mysql_num_fields](#) -- Retorna o número de campos do resultado

[mysql_num_rows](#) -- Retorna o número de linhas em um resultado

[mysql_pconnect](#) -- Abre uma conexão persistente com um servidor [MySQL](#)

[mysql_ping](#) -- Pinga uma ligação ou restabelece a ligação se não houver ligação

[mysql_query](#) -- Realiza uma query [MySQL](#)

[mysql_real_escape_string](#) -- Escapa os caracteres especiais numa string para usar em um comando SQL, levando em conta o conjunto actual de caracteres.

[mysql_result](#) -- Retorna dados do resultado

[mysql_select_db](#) -- Selecciona um banco de dados [MySQL](#)

[mysql_stat](#) -- Retorna o status actual do sistema

[mysql_tablename](#) -- Retorna o nome da tabela do campo

[mysql_thread_id](#) -- Retorna o ID da thread actual

Trab nº4 - O módulo IOT pede páginas PHP, para ler e escrever na BD

Para consolidar os conhecimentos adquiridos nas aulas teóricas, relativos ao desenvolvimento em PHP de páginas WEB dinâmicas, iremos utilizar um *ServidorWEB* (IIS ou Apache) e um módulo IOT (ESP).

Desenvolva o programa para o Módulo IOT atuar como um cliente HTTP/TCP/IP, pedir páginas PHP, através de mensagens HTTP (GET), a um computador remoto com uma base de dados, permitindo registar na base de dados as entradas e as saídas digitais e analógicas do Reservatório: Nível1(botão de flash -0) Nível2 (push button nº4), nível de água analógico (sensor de luz).

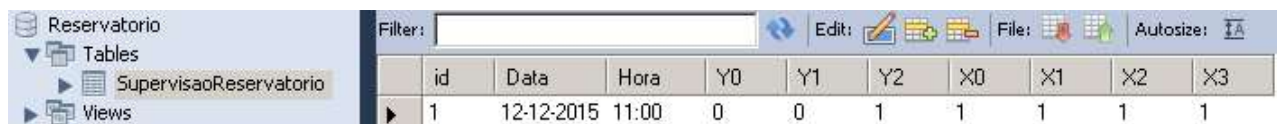
Pretende-se que o ESP:

- estabeleça uma ligação WiFi com o Router Wireless
- estabeleça uma ligação TCP/IP com o servidor APACHE, no computador
- peça ao servidor APACHE um documento “CtrPLC.php” e envie os parâmetros:
Y0 - saída digital do ESP
Y1- saída digital do ESP
Y2- saída digital do ESP
X2 - entrada digital do ESP (Botão Flash - nº0)
X3- entrada digital do ESP (Botão - nº4)
Nivel - entrada analógica do ESP (Sensor de luz - A0)

A base de dados “Reservatorio” contém uma tabela intitulada “supervisaoreservatorio” com os campos: Data, Hora, Y0, Y1, Y2, X0, X1, X2, X3.

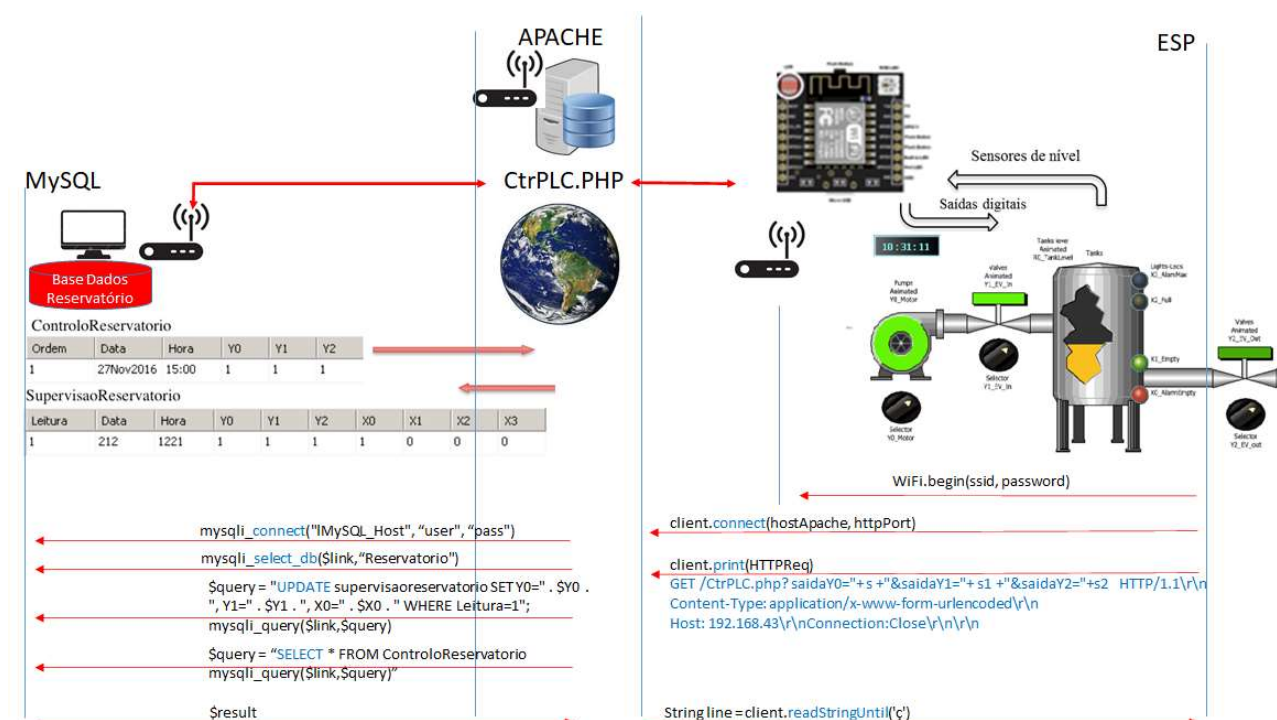
Uma página “CtrPLC.php” escrita em PHP acede à tabela “supervisaoreservatorio”.

Esta tabela tem apenas uma linha com o estado atual das entradas e saídas do ESP.



id	Data	Hora	Y0	Y1	Y2	X0	X1	X2	X3
1	12-12-2015	11:00	0	0	1	1	1	1	1

Para estes parâmetros ficarem aguardados na BD de segundo a segundo, linha a linha, noutra tabela da base de dados, chamada “histórico”, criando um histórico ao longo do tempo, o que deveria fazer?



Mensagens HTTP, do tipo GET e POST

Mensagem do tipo GET

Neste exemplo, o Apache responde com a página HTML “[Controlo.html](#)”, e o Browser apresenta a página seguinte, com um só botão do tipo **submit**:

```
<html>
  <body>
    <form action="Controlo.html" method="GET">
      <input type="submit" Name = "Y0" Value="Ligar_Y0">
    </form>
  </body>
</html>
```

Quando o utilizador prime o botão “**submit**”,
Browser envia para o Apache uma Msg HTTP do tipo GET:

```
GET /Controlo.html?Y0=Ligar_Y0 HTTP/1.1
Referer: http://192.168.1.6/Controlo.html
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: pt-PT,pt;q=0.5
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/70.0.3538.102 Safari/537.36 Edge/18.18363
Accept-Encoding: gzip, deflate
Host: 192.168.1.6
Connection: Keep-Alive
```

Mensagem do tipo POST

Neste exemplo, o Apache responde com a página HTML “[Controlo.html](#)”, o Browser apresenta a página seguinte, com um só botão do tipo **submit**:

```
<html>
  <body>
    <form action="Controlo.html" method="POST">
      <input type="submit" Name = "Y0" Value="Ligar_Y0">
    </form>
  </body>
</html>
```

Quando o utilizador prime o botão “**submit**”,
o Browser envia para o Apache uma Msg HTTP do tipo **POST**:

```
POST /Controlo.html HTTP/1.1
Referer: http://192.168.1.6/Controlo.html
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: pt-PT,pt;q=0.5
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
```


User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/70.0.3538.102 Safari/537.36 Edge/18.18363
Accept-Encoding: gzip, deflate
Host: 192.168.1.6
Content-Length: 11
Connection: Keep-Alive
Y0=Ligar_Y0

Importante, no cabeçalho desta mensagem POST, como o seu conteúdo tem os valores do <form> ...
</form> , neste caso Y0=Ligar_Y0
o “contente-type” deve ser igual a application/x-www-form-urlencoded

Content-Type: application/x-www-form-urlencoded

Nota: realizar um exemplo em que o ESP pede: um ficheiro “calculadora.html”, e um ficheiro PHP (ex calc.php) “passando” uma variável por POST, e outra por GET, ao Apache/InterpretadorPHP.

O ESP envia a mensagem HTTP do tipo POST para o servidor Apache,
a pedir o ficheiro Cal.php
e com os parâmetros X = 10 e de Y=10
no corpo da mensagem, que por isso é um conteúdo do tipo application/x-www-form-urlencoded

<pre>POST /calc.php HTTP/1.1 Host: 192.168.1.7 Content-Length: 9 Content-Type: application/x-www-form-urlencoded Connection: Keep-Alive X=10&Y=20</pre>	<pre>url= String("POST ") + "/Calc.php" + " HTTP/1.1\r\n"+"Host: " + host + "\r\n"+ "Content-Length: 9\r\n"+ "Content-Type: application/x-www-form-urlencoded\r\n" +"Connection: Keep-Alive\r\n\r\n"+"X=10&Y=20\r\n"; client.print(url);</pre>
---	--

O ficheiro “Calc.php”:

```
<html>
<body>
    <?php
        $Z=$_POST['X']+$_POST['Y'];
        echo $_POST['X'],"+",$_POST['Y'], "=", $Z;
    ?>
</body>
</html>
```

Exemplo: O ESP pede ao servidor WEB (Apache) páginas html, via Wifi 802.11

Neste exemplo, é utilizado um ESP, um router local e um computador com o servidorWEB (Apache). Tanto o ESP como o computador (c/ Apache) estão ligados ao Router.

O ESP pede ao servidorWeb, páginas html, enviando pedidos HTTP do tipo

```
GET /ola.html HTTP/1.1\r\n
```

(Para compreender este exemplo, o protocolo HTTP deve ser previamente estudado)

O router tem o SSID = **default** e não tem password.

O servidor WEB reside no computador remoto com o IP 192.168.1.10

Neste exemplo são usados dois objetos, um do tipo **WiFi**, e outro do tipo **WiFiClient**

Objeto do tipo **WiFi**

- .begin** (pede ligação wifi ao router)
- .status** (estado da ligação Wifi)
- .localIP** (retorna o end IP atribuído pelo Router ao ESP)
- .mode** (modo WIFI_STA)

Objeto do tipo **WiFiClient**

- .available** (devolve nº bytes recebidos)
- .readString** (lê vários bytes)
- .flush** (limpa a msg recebida)
- .print** (envia uma string para o dispositivo remoto)
- .connect**(host, tcpPort)

```

#include <ESP8266WiFi.h>
const char* ssid = "..."; // nome do router de casa
const char* password = "..."; // password do router de casa
const char* host = "192.168.1.2"; // computador de destino

void setup() {
    Serial.begin(9600);
    WiFi.mode(WIFI_STA);
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    } // while
    Serial.println(WiFi.localIP());
} // setup

int value = 0;

void loop() {
    delay(1000);
    ++value;
    Serial.print("connecting to ");
    Serial.println(host);

    // Use WiFiClient class to create TCP connections
    WiFiClient client;
    const int httpPort = 80;
    if (!client.connect(host, httpPort)) {
        Serial.println("connection failed");
        return;
    }

    // Pedido HTTP
    String HTTP_req = "GET /ola.html HTTP/1.1\r\nHost:192.168.1.2\r\nConection: Close\r\n\r\n";
    client.print(HTTP_req);
    Serial.println("Envio de msg para o Apache: ");
    Serial.println(HTTP_req);

    unsigned long timeout = millis();
    while (client.available() == 0) {
        if (millis() - timeout > 10000) {
            Serial.println(">>> Client Timeout !");
            client.stop(); return;
        } // if
    } // while

    // Read all the lines of the reply from server and print them to Serial
    while(client.available()){
        String line = client.readStringUntil('\r');
        Serial.println(line);
    } // while
} // Loop

```

Lista de Instruções ESP
 ESP.Restart()
 ESP.getFlashChipSize()

Exemplo: O ESP8266 como Cliente Wifi e Client TCP/IP

```
#include <ESP8266WiFi.h>
const char* ssid = "default"; // Nome da rede Wireless
const char* password = ""; // Password da rede Wireless
const char* host = "192.168.1.13"; // Endereço do PC do Apache
String url, url1; // Irá de conter a msg HTTP enviada para o Apache
String s; // Irá conter o estado do botão nº4 (0 ou 1)
WiFiClient client; // WiFiClient, permite enviar ou receber dados por TCP/IP
const int httpPort = 80;

void setup() {
  Serial.begin(115200);
  pinMode(4, INPUT); // Config o pino 4 como input, o sensor/botão está ligado no pino 4

  // Envia para o Serial Monitor (só para Debug)
  Serial.println(); Serial.println(); Serial.print("Connecting to "); Serial.println(ssid);

  WiFi.mode(WIFI_STA); // O ESP irá como Client Wireless
  WiFi.begin(ssid, password); // ESP tenta ligar ao Router
  while (WiFi.status() != WL_CONNECTED) { // status da ligação Wifi com o router= 0 - desligado 4- ligado ...
    delay(500); Serial.print("."); }

  // O ESP envia por porta série, para o PC local esta informacao:
  Serial.println("O ESP está ligado por WiFi ao Router, o Router deu-lhe o IP "); Serial.println(WiFi.localIP());
}

void loop() {
  // ----- TENTA LIGAR POR TCP/IP ao APACHE
  // Se o status da ligação TCP/IP entre o ESP e o computador remoto for igual a 0 (client.status= 0 - desligado)
  // Neste caso, o ESP tenta estabelecer ligacao TCP/IP com o Apache
  if(client.status()==0){
    Serial.print("connecting to "); Serial.println(host); // envia para o serial monitor
    if (!client.connect(host, httpPort)) { // ----- Tenta LIGAÇÃO TCP
      Serial.println("connection failed");
      client.flush();
      delay(1000);
    } //IF
  } // IF

  // ----- SE LIGACAO TCP ESTABELECIDA ...
  // Se o status da ligação for 4, significa que a ligação TCP/IP está estabelecida entre o ESP e o Apache
  if(client.status()==4){
    s=digitalRead(4); // Le botão n. 4
    url1="/update.php?saidaY0="+s+"&saidaY1=10000&saidaY2=15000";
    url= String("GET ") + url1 + " HTTP/1.1\r\n" + "Host: " + host + "\r\n" + "Connection: keep-alive\r\n\r\n";
    client.print(url); // Envia pedido HTTP, da linha anterior "url", para o Apache
    // GET /CtrPLC.PHP?saidaY0=0&saidaY1=1&saidaY2=0 HTTP/1.1\r\n" + "Host: " + host + "\r\n" + "Connection: keep-alive\r\n\r\n"
    Serial.print("Enviou msg HTTP para o Apache, a pedir o ficheiro update.php");
    Serial.println(url); // No Serial Monitor pode ser visto o pedido enviado para o Apache
    // Espera pela resposta do Apache
    delay(100);
  }

  // ----- SE CHEGARAM DADOS por TCP/IP ....
  if (client.available() > 0) { // Se o ESP tiver recebido bytes
    // Le todos os caracteres enviados pelo Apache até receber o caracter 'ç' ou exceder um tempo máximo.
    String line = client.readStringUntil('ç');
    Serial.print(line); // envia para o Serial Monitor (para Debug)
    client.flush();
  }

  Serial.println(); Serial.println(); Serial.println();
} // Fim do Loop
```

pagina HTML/PHP (CtrPLC.php)

Sempre que o cliente ESP, ou um Browser WEB, pedir a página CtrPLC.php fazendo “POST CtrPlc.php”, a página PHP deve atualizar os campo Y0, Y1, Y2 na Base de dados do computador e devolver uma pagina HTML ao Browser/ESP cliente, com o estado de todos os campos.

```
<html>
<body>
----- Controlo -----
<form action="ctrplc.php" method="POST">
Introduza os valores de Y0, Y1, e Y2 <br>Estes valores serao escritos na base de dados "Reservatorio,"<br>na tabela "SupervisaoReservatorio(Leitura,Data,Hora,Y0,Y1,Y2,X0,X1,X2,X3)"<br><br>
Y0:<input type="text" name="saidaY0" value="1" >
Y1:<input type="text" name="saidaY1" value="1" >
Y2:<input type="text" name="saidaY2" value="1" >
<input type="submit" value="enviar">
</form>

<?php
$Y0=$_POST['saidaY0'];
$Y1=$_POST['saidaY1'];
$Y2=$_POST['saidaY2'];

/* Conectando, escolhendo o banco de dados */
$link = mysqli_connect("localhost", "root", "")or die("N,,o pude conectar: " . mysqli_error());
mysqli_select_db($link,"Reservatorio") or die("N,,o pude selecionar o banco de dados");

/* Query SQL de atualizacao */
$query = "UPDATE SupervisaoReservatorio SET Y0='\" . $Y0 . \"', Y1='\" . $Y1 . \"', Y2='\" . $Y2 . \"' WHERE
Leitura=1";
$result = mysqli_query($link,$query) or die("A query falhou: " . mysqli_error());

echo "----- Supervisao -----<br>Os valores lidos na base de dados sao:";
/* Fazendo a query SQL DE LEITURA DA BASE DE DADOS*/
$query = "SELECT * FROM SupervisaoReservatorio";
$result = mysqli_query($link,$query) or die("A query falhou: " . mysqli_error());
$linhal=mysqli_fetch_array($result,MYSQLI_ASSOC);
echo " Y0: ".$linhal['Y0'];
echo " Y1: ".$linhal['Y1'];
echo " Y2: ".$linhal['Y2'];

/* Liberta o resultado */
mysqli_free_result($result);

/* Fechando a conex,,o */
mysqli_close($link);
?>
</body>
</html>
```

Trab nº5 - Browser WEB, Páginas WEB html, CSS, PHP, para aceder à BD

No Trabalho nº3 as páginas WEB (HTML, CSS) encontravam-se no ESP.

No trabalho nº 5, pretende-se que as páginas WEB (HTML,CSS) e páginas PHP residam no computador para permitirem o acesso a bases de dados e visualizar os dados a partir de um Browser WEB.

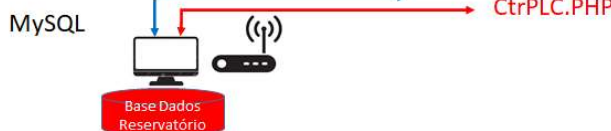
No browser WEB da figura seguinte (fluxos de informação em azul):

- pode observar-se na Frame central, os botões de controlo que permitem ligar o motor, a electroválvula de entrada e a electroválvula de saída. Pretende-se desenvolver uma página PHP “Novo.PHP” que permita “apenas” escrever na tabela “ControloReservatorio” o valor 1/0 consoante o utilizador remoto premiu um dos botões presentes na frame central.
- pode observar-se na frame da direita, o estado das saídas e entradas digitais.

Pretende-se desenvolver uma página PHP “Novo.PHP” que permita ler a tabela “SupervisãoReservatorio” e apresentar na frame da direita os seus valores.

TRAB5

Browser



ControloReservatorio					
Ordem	Data	Hora	Y0	Y1	Y2
1	27Nov2016	15:00	1	1	1

SupervisaoReservatorio								
Leitura	Data	Hora	Y0	Y1	Y2	X0	X1	X2
1	212	1221	1	1	1	1	0	0

TRAB4

ESP

ESP Local

