# sistemas de informação em rede

php

## introdução ao php

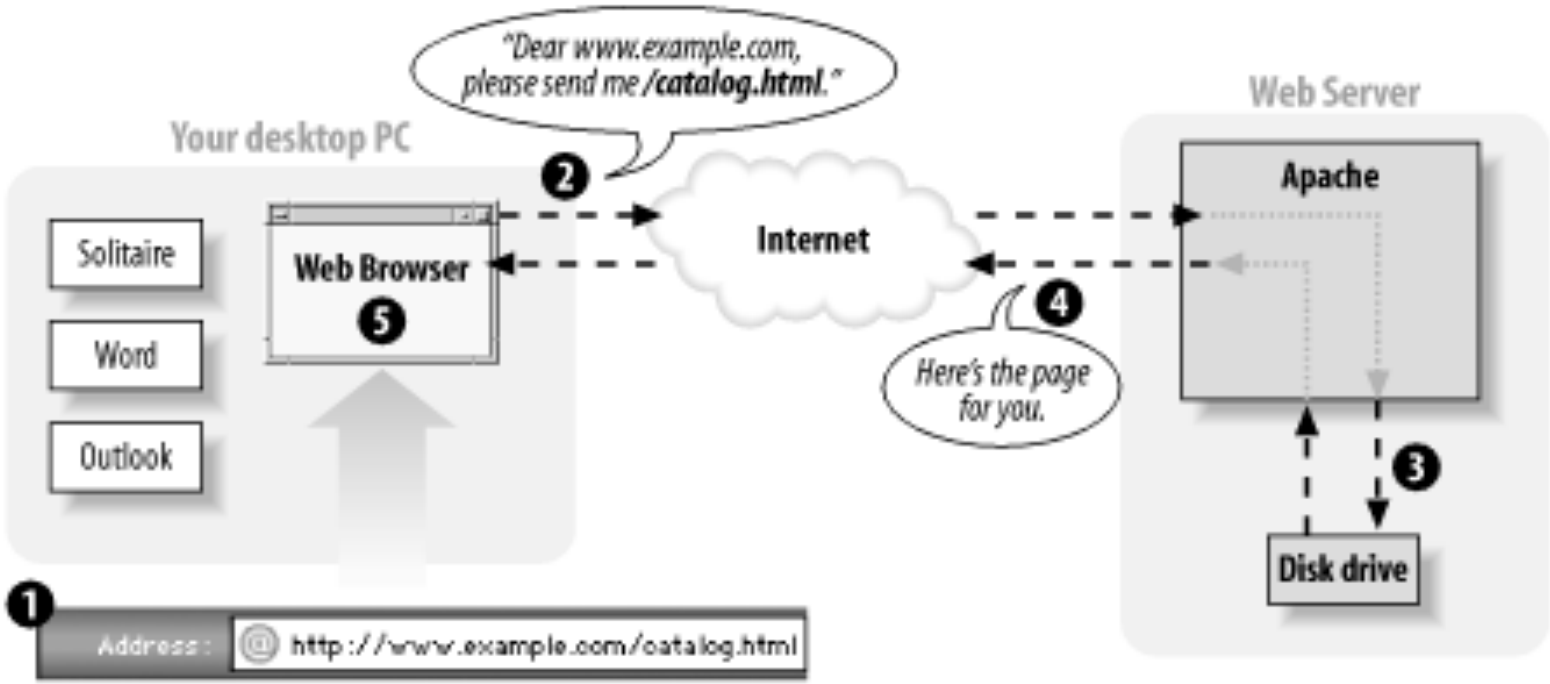
pedro moreira
2007-2016
Instituto Politécnico de Viana do Castelo
ESTG - IPVC
Escola Superior de Tecnologia e Gestão

revisão - João Nuno Azevedo (2024)

# php

- php
  - hypertext preprocessor
  - linguagem interpretada (script)
  - linguagem server-side (executada do lado do servidor)
  - utilização : desenvolvimento de conteúdos web dinâmicos
  - pode ser embutida em páginas HTML (desde que o servidor tenha o respectivo suporte para php).
  - gratuita, código aberto
  - grande comunidade de utilizadores / desenvolvedores

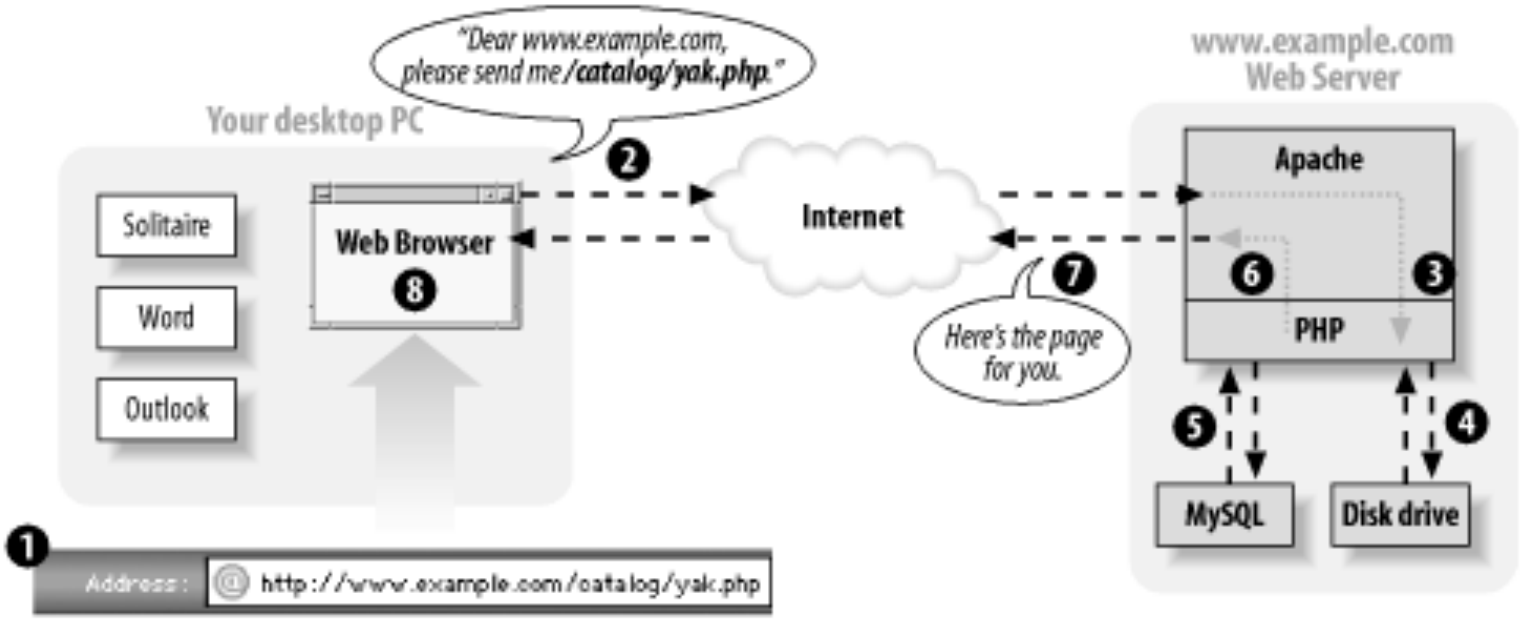
# comunicação cliente servidor sem php (apenas http & html)



# comunicação cliente servidor sem php (apenas http & html)

1. introdução de uma URI num agente cliente (browser) (p.ex. www.servidor.pt/pag1.htm)
2. o agente (cliente) envia uma mensagem pela Internet (via protocolo HTTP) para o servidor (www.servidor.com) pedindo o recurso (neste caso uma página – pag1.html).
3. um programa da máquina destino - servidor, um servidor HTTP (p.ex. apache) recebe o pedido e processa-o, acedendo ao ficheiro correspondente à página pedida no sistema de ficheiros do servidor.
4. o servidor http, envia (através do protocolo HTTP) o conteúdo do recurso pedido.
5. o agente do cliente (browser) recebe a informação e faz o "render" do recurso pedido.

# comunicação cliente servidor com php



# comunicação cliente servidor com php

1. introdução de uma URI num agente cliente (browser) (p.ex. www.servidor.pt/pag1.php)
2. o agente (cliente) envia uma mensagem pela Internet (para o servidor (www.servidor.com) pedindo o recurso
3. um servidor HTTP (p.ex. apache) recebe o pedido identifica-o como uma página contendo PHP, redireccionando o pedido para um interpretador de PHP.
4. O interpretador de PHP acede ao ficheiro correspondente.
5. O interpretador de PHP processa as instruções PHP (podendo no resultado destas trocar dados com outros ficheiros ou servidores , locais ou remotos).
6. O interpretador de PHP devolve a saída do programa PHP para o servidor HTTP
7. o servidor HTTP, envia (através do protocolo HTTP) o conteúdo processado do recurso pedido.
8. o agente do cliente (browser) recebe a informação e faz o "render" do recurso pedido.

# php : inserindo código php

- diversas formas (estilos) ambiente
  - estilo XML (preferido, no recomendado)

```
<?php
echo “<p>Order processed.</p>”;
?>
```

  - estilo curto

```
<?
php echo “<p>Order processed.</p>”;
?>
```

  - estilo script

```
<script language=“php”>
echo “<p>Order processed.</p>”;
</script>
```

# php : código (statements)

- terminados por ponto e vírgula (;)
- ignora sequências de espaços e linhas
- comentários (1) : **//**

```
// comentário até ao fim da linha
```

- comentários (2): **/* … */**

```
/* bloco de comentário que só
   termina qundo for encontrado o
   marcador de fim de comentário */
```

- terminados por ponto e vírgula (;)

# php: tipos de dados

- **Integer** :
  - números inteiro
- **Float**:
  - números não inteiros (vírgula flutuante)
- **String**:
  - cadeias de caracteres
- **Boolean**:
  - valores booleanos (**true** e **false**)
- **Array**:
  - usado para armazenar sequencias de dados numa estrutura linear.
- **Object**:
  - usado para armazenar instâncias de classes.

# php: operadores (aritméticos)

- **+** adição `$a + $b`
- **-** subtracção `$a - $b`
- ***** multiplicação `$a * $b`
- **/** divisão `$a / $b`
- **%** módulo `$a % $b`

# php: operadores (String)

- operadores de Strings
  - = **operador de atribuição**
  - . **operador de concatenação**

```
$a = "Sistemas";
$b = " de Informação em Rede";
$result = $a.$b;
```

- A variável **$result** contém agora a String:
  - "Sistemas de Informação em Rede".

# php: operadores (combinados)

- **+=** auto-adição `$a += $b // $a = $a + $b`
- **-=** auto-subtracção `$a -= $b // $a = $a - $b`
- ***=** auto-multiplicação `$a *= $b // $a = $a * $b`
- **/=** auto-divisão `$a /= $b // $a = $a / $b`
- **%=** auto-módulo `$a %= $b // $a = $a % $b`
- **.=** auto-concatenação `$a .= $b // $a = $a . $b`

# php: operadores
## pré e pós (decremento / incremento)

- **++$i** pré-incremento
- **--$i** pré-decremento
- **$i++** pós-incremento
- **$i--** pós-decremento

- exemplo:

```
$a = 4;          // $a com o valor 4
echo ++$a;       // $a ← 5, output 5
echo $a++;       // output 5, $a ← 6
```

# php: referências (&)

- uso normal por atribuição

```
$a = 5;          // a tem valor 5
$b = $a;         // b é uma nova var e possui o valor 5
$a = 6;          // a passa a 6, b continua com 5
```

- usando referências (&)

```
$a = 5;          // $a tem valor 5
$b = &$a;        // $b é uma referência a $a (alias)
$a = 6;          // $a = 6 , logo $b também devolve valor 6
```

  - quando usamos referências as variáveis ficam ligadas (como se fossem nomes diferentes para o mesmo objecto//variável).

# php: referências (&) : unset()

- unset()
  - podemos quebrar uma ligação de referência através da função unset (que "destrói" uma variável), neste caso destrói a ligação estabelecida por referência.

```
$a = 5;          // $a tem valor 5
$b = &$a;        // $b é uma referência a $a (alias)
$a = 6;          // $a = 6 , logo $b também devolve valor 6
unset($a);       // destruição de $a, mas não de $b
echo $b          // $b assume o valor de $a
```

# php: operadores de comparação

| | | |
|---|---|---|
| **==** | igualdade | $a == $b |
| **===** | identidade | $a === $b |
| **!=** | desigualdade | $a != $b |
| **!==** | não identidade | $a !== $b |
| **<>** | desigualdade | $a <> $b |
| **<** | menor que | $a < $b |
| **>** | maior que | $a > $b |
| **<=** | menor ou igual que | $a <= $b |
| **>=** | maior ou igual que | $a >= $b |

# php: operadores de comparação

- igualdade vs identidade
  - igualdade:
    - representam valores iguais (mesmo que as variáveis possam ser de tipo diferente), se necessário um dos operandos é convertido para o tipo do primeiro.
    - exemplo
      - 0 == '0', devolve true;
      - 0 == 'a', devolve true (ver nota sobre conversões)
      - 1 == '01', devolve true
  - identidade
    - os valores e tipos são idênticos
    - exemplo
      - 0 === '0', devolve false;

# php: operadores lógicos

- **!** negação lógica
- **&&** "E" lógico
- || "OU" lógico
- **and** "E" lógico
  - mesmo que && mas com menor precedência
- **or** "OU" lógico
  - mesmo que || mas com menor precedência

# php: operadores de bit

- **&** "e" bit-a-bit
- **|** "ou" bit a bit
- **~** negação bit-a-bit
- **^** "xor" bit-a-bit
- **<<** deslocamento de bits para a esquerda
  - $a << $b : desloca os bits de $a, $b posições para a esquerda
- **>>** deslocamento de bits para a direita
  - $a << $b : desloca os bits de $a, $b posições para a direita
- **nota importante (os operandos devem ser inteiros ou serem previamente convertidos)**

# php: ficheiros
# file modes

| | |
|---|---|
| r | read |
| r+ | read / write |
| w | write |
| w+ | write & read |
| x | cautious write |
| x+ | cautious write & read |
| a | append read only |
| a+ | append read/write |

# php: arrays generalidades

- o php suporta:
  - arrays enumerados
    - $products = array( ‘Tires’, ‘Oil’, ‘Spark Plugs’ );
  - arrays associativos
    - $prices = array( ‘Tires’=>100 );
    - $prices[‘Oil’] = 10;
    - $prices[‘Spark Plugs’] = 4;

# php: arrays operadores

**+** união $a + $b

- o array $b é adicionado ao fim de $a , com excepção dos índices conflituosos.

**==** igualdade $a == $b

- verdade se $a e $b contêm os mesmos pares cheve/valor.

**===** identidade $a === $b

- verdade se $a e $b são idênticos e elementos na mesma ordem

**!=** ou **<>** desigualdade $a != $b $a <> $b

**!==** não identidade $a !== $b

**exemplo**

$a = array(2,4,6,8);
$c = array(3=>6,2=>2,0=>4,1=>8)

$b = array(4,8,2,6);
$d = array(0=>2,1=>4,2=>6,3=>8)

$a == $b (falso)
$a == $c (verdade)

$a === $d (verdade)
$a === $b (falso)

# php: arrays
# iteração

**foreach** ($array **as** $valor) instruções

**foreach** ($array **as** $key => $valor) instruções

```
<?php
   $arr = array(1, 2, 3, 4);
   foreach ($arr as $value) {
       echo '-'.$value;
       }
?>
 // output:  -1-2-3-4
```

# php: arrays
# iteração:dica

**foreach** ($array **as** $valor) instrucoes

para modificar valor do array dentro de foreach iterar as referências (usando o operador &)

```
<?php
   $arr = array(1, 2, 3, 4);
   foreach ($arr as &$value) {
        $value = $value * 2;
        }
?>
 // conteúdo:  (2, 4, 6, 8)
```

# php: arrays
# tamanho

## count()

```
<?php
   $arr = array(1, 2, 3, 4);
   echo 'tamanho: '.count($arr);
?>

 // output:  tamanho: 4
```

nota importante: a função count só funciona correctamente com arrays numericamente indexados

# php: arrays ordenação

## sort()

- para arrays enumerados
- ordenação ascendente / alfabética

```
<?php
   $arr = array('ola', 'ole', 'aula', 'bom');
   echo 'tamanho: '.count($arr);
?>

// conteúdo: ('aula', 'bom', 'ola', 'ole')
```

## asort() e ksort()

- para arrays associativos
- asort() – ordena pelo valor
- ksort() – ordena pela chave
- ordenação ascendente / alfabética

# php: arrays
# implode() / explode()

**implode**($array, $delimitador)

converte array numa string com valores separados por delimitador (tipicamente um caracter ou string)

**explode**($string, $delimitador)

converte string de valores separados por delimitador num array com os valores

# php: arrays
# inserção no fim

```
<?php
   $prices = array(5, 10, 11);
   printf("<p>%s</p>\n", implode(',', $prices));
   $prices[] = 13;
   printf("<p>%s</p>\n", implode(',', $prices));
   array_push($prices, 4);
   printf("<p>%s</p>\n", implode(',', $prices));
?>
```

output

5,10,11

5,10,11,13

5,10,11,13,4

# php: arrays
# inserção no início

```php
<?php
   $prices = array(5, 10, 11);
   printf("<p>%s</p>\n", implode(', ', $prices));
   array_unshift($prices, 13);
   printf("<p>%s</p>\n", implode(', ', $prices));
   array_unshift($prices, 4, 5);
   printf("<p>%s</p>\n", implode(', ', $prices));
?>
```

output

5,10,11

13,5,10,11

4,5,13,5,10,11

# php: arrays
## remoção / inserção em pos. arbitrária

**array_splice($array, $offset, $count, $replacement)**

remove **$count** elementos do **$array** a partir da posição **$offset** e substitui por elementos definidos em **$replacement**

se $count = 0 então não remove

se $replacement não for definido ou null então apenas remove

# php: arrays
## remoção / inserção em pos. arbitrária

### array_splice($array, $offset, $count, $replacement)

```
$a=array(1,2,6,7);
$b=array(3,8,9,4,5);

$c = array_splice($b,1,2);
array_splice($a,2,0,$b);
array_splice($a,count($a),0,$c);
```

```
antes:
a: 1,2,6,7
b: 3,8,9,4,5
depois:
a: 1,2,3,4,5,6,7,8,9
b: 3,4,5
c: 8,9
```

# php: arrays
## pesquisa e informação e ...

**in_array($elem,$array,$strict)**

pesquisa $elem em $array, devolve valor da $key onde encontrado (null se falha). Se $strict for true então pesquisa com operador identidade senão faz pesquisa com operador igualdade.

**count($array)**

devolve número de elementos em $array

**array_key_exists($key,$array)**

determina se chave %key existe em $array

**is_array($foo)**

determina se $foo é do tipo array

**array_walk($array,$funct)**

aplica a funcao $funct a todos os elementos de $array

# php: arrays multidimensionais

```
$a = array(
array(1,2,3),
array(4,5,6),
array(7,8,9)
);

//antes $a[2][0] é igual a 7
$a[2][7] = 8 // depois foi substituído por 8
$a[3][0] = 1 // novo elemento na dimensão exterior
```

# mais utilidades: array_key_exists($key,$array)

- verifica se uma dada chave ($key) existe no ($array)
- pode ser de grande utilidade na validação de formulários inspeccionando $_POST ou $_GET

```
<?php
   if(array_key_exists('username',$_POST) {
   // ok!
   } else {
   // username não foi introduzido
   }
   ?>
```

# mais utilidades: empty($var)

- verifica se uma dada variável ($var) é vazia. devolve TRUE se $var for considerada "vazia". notar que o conceito de vazia é diferente do conceito de inicializada
- pode ser de grande utilidade na validação de formulários inspeccionando $_POST ou $_GET
- notar que vazia, pode ser:
  - "" : string vazia
  - 0: o número 0
  - "0" uma string com o número zero
  - NULL
  - FALSE
  - um array() vazio.
  - uma variável declarada mas não utilizada.

# algumas utilidades:
# @ (supressor de erros)

- impede o output de mensagens de erro na instrução
- exemplo:

```
@ $ligacao = new mysqli('localhost', 'teste', 'teste', 'test');
```

- se existir erro na ligação não vai ser feito o output por parte do interpretador PHP.

# algumas utilidades: get_magic_quotes_gpc()

- devolve true or false conforme a directiva **magic_quotes_gpc()** esteja ou não activa na configuração do PHP.
- esta directiva permite configurar o PHP para que todos os caracteres "especiais" recebidos através de formulários ou de bases de dados sejam convenientemente "escapados" por adição de uma barra invertida:
  - `plica '`
  - `aspas "`
  - `barra invertida \`
  - `NUL`

# algumas utilidades: addslashes()

- esta função permite reformatar uma string para que todos os caracteres "especiais" sejam convenientemente "escapados" por adição de uma barra invertida. Isto aplica-se particularmente no caso dos seguintes caracteres:
  - `plica '`
  - `aspas "`
  - `barra invertida \`
  - `NUL`
- esta função é particularmente útil em conjunto com o teste **get_magic_quotes_gpc()**.

# algumas utilidades: stripslashes()

- esta função permite reformatar uma string para que todos os caracteres "especiais" "escapados" sejam convenientemente reformatados para a sua forma original por remoção de uma barra invertida. Isto aplica-se particularmente no caso dos seguintes caracteres:
  - `plica '`
  - `aspas "`
  - `barra invertida \`
  - `NUL`
- esta função é particularmente útil quando se recebe dados de bases de dados e se pretende efectuar o seu output.

# algumas utilidades: htmlspecialchars()

- esta função permite reformatar uma string para que todos os caracteres "especiais" que sejam incompatíveis com o interpretador de HTML sejam convenientemente substituídos por sequências de escape equivalentes:
  - '&' (e comercial) → '&'
  - '"' (aspas) → '"' quando **ENT_NOQUOTES** não está activo.
  - ''' (plica) → ''' apenas quando **ENT_QUOTES** está activo.
  - '<' (menor que) → '<'
  - '>' (maior que) → '>'
- note que em algumas situações pode pretender fazer o output de strings contendo estes caracteres para HTML, o que poderia causar conflitos // problemas.
- **função inversa: htmlspecialchars_decode()**

# php/html forms
# method = “post”

- **$_POST**
- variável global
- array associativo que recebe a informação enviada por um formulário usando o método POST
- exemplo: $_POST[“sexo”] guarda o valor do campo de nome “sexo” do formulário enviado
- nota: a informação é tipicamente passada como string.

# php/html forms
# method = “get”

- **$_GET**
- variável global
- array associativo que recebe a informação enviada por um formulário usando o método GET
- exemplo: $_GET[“sexo”] guarda o valor do campo de nome “sexo” do formulário enviado
- nota: a informação é tipicamente passada como string.

# php/html forms
# method = “post”

- **$_FILES**
- variável global
- array associativo que recebe a informação (dos ficheiros) enviados por um formulário usando o método POST
  - **$_FILES[‘nome’][‘tmp_name’]** local onde o ficheiro foi temporariamente armazenado na máquina onde corre o servidor HTTP.
  - **$_FILES[‘userfile’][‘name’]** nome original do ficheiro (na máquina cliente).
  - **$_FILES[‘userfile’][‘size’]** tamanho em bytes.
  - **$_FILES[‘userfile’][‘type’]** é o tipo MIME type do ficheiro. Por exemplo: text/plain ou image/gif.
  - **$_FILES[‘userfile’][‘error’]** código de erro

# php/html forms
# dicas

- o nome dos campos de formulário que aceitem valores múltiplos:
  - <select multiple>
  - <input type="checkbox">
  - devem possuir um nome terminado em **[]**
  - **assim o PHP instanciará automaticamente a respectiva variável como um array.**
    - <input type="checkbox" name="interesses[]">
    - <select multiple name="hobbies[]">
  - **assim (e supondo envio por POST) as variáveis**
    - **$_POST['interesses'] e $_POST['hobbies'] serão elas próprias um array.**

# Acesso a BDs via Web
## (usando PHP & MySQL)

- **Referências**:
  - PHP and MySQL Web Development, 3rd ed, Luke Welling and Laura Thomson, SAMS publishing (capítulo 11)
  - manual do PHP (mysqli) : http://pt.php.net/manual/en/ref.mysqli.php

# ligação a bd : MySQL
## processo típico :1

1. Um utilizador (através de um browser web) envia um pedido HTTP para uma página web . Por exemplo, o utilizador envia um pedido de uma pesquisa sobre os livros escritos por determinado autor, usando um formulário HTML. Este pedido é enviado para uma página contendo código PHP.

2. O servidor HTTP recebe o pedido, acede ao ficheiro PHP, e passa-os ao interpretador PHP para processamento.

3. O interpretador PHP, inicia o processamento do script que contém uma instrução de ligação a uma base de dados, e uma interrogação que efectua a pesquisa em causa (em SQL). O interpretador executa estas instruções estabelecendo ligação com a bases de dados e efectuando respectiva interrogação (através de SQL).

# ligação a bd : MySQL
## processo típico :2

4. O servidor de Base de Dados (ex: MySQL) recebe a interrogação, processa-a e devolve os resultados ao interpretador de PHP.

5. O interpretador de PHP, recebe os resultados da interrogação à BD, processa-os, formata-os e devolve-os em HTML ao servidor HTTP.

6. O servidor HTTP envia o HTML resultante de volta ao browser, onde o utilizador poderá visualizar os resultados da sua pesquisa.

# interacção via web com BDs

- Passos básicos típicos de um script de acesso a uma BD:
  - 1. validar e filtrar dados introduzidos pelo utilizador.
  - 2. estabelecer a ligação com o SGBD / BD em causa.
  - 3. Interrogar a BD.
  - 4. Recolher os resultados.
  - 5. Processar os resultados e devolver informação ao utilizador

# php/mysql : ligação à BD

## abordagem OO

```
$ligacao = new mysqli('localhost', 'teste', 'pass', 'livros');
```

- devolve um objecto do tipo mysqli
- este objecto possui um conjunto de métodos que podem ser invocados. Nota: estes métodos possuem equivalentes procedimentais.
- exemplo:

```
$resultado = $ligacao->query("SELECT * FROM LIVROS");
```

# php/mysql : ligação à BD

## abordagem procedimental

```
$ligacao = mysqli_connect('localhost', 'teste', 'pass', 'livros');
```

- devolve um recurso que representa a ligação
- esta variável ($ligação) é depois usada como parâmetro em posteriores interacções com a BD.
- Nota: Tipicamente existem métodos/propriedades equivalentes na classe mysqli (mas não para todas as funcionalidades)
- exemplo:

```
$resultado = mysqli_query($ligacao,"SELECT * FROM LIVROS");
```

# php/mysql : ligação à BD

## verificação de sucesso (1)

- chamada a mysqli_connect_errno()
  - devolve código do erro (caso tenha existido) da última chamada a funções de ligação (ex: mysqli_connect)
- chamada a mysqli_connect_error()
  - devolve mensagem de erro do último erro de ligação.

```
/* tenta estabelecer ligação */
$ligacao = mysqli_connect('localhost', 'teste', 'pass', 'livros')
/* verifica sucesso na ligação */
if (mysqli_connect_errno()) {
  printf("ligação falhou com erro: %s\n", mysqli_connect_error());
  exit();
  }
```

# php/mysql : ligação à BD

## verificação de sucesso (2)

- teste directamente na chamada à função de ligação e utilização da construção **or die("msg")**
- na realidade a chamada a **die("msg")** é idêntica a uma chamada a **exit("msg")**.

```
/* tenta estabelecer ligação */
/* sai (morre) se não conseguir */
$ligacao = mysqli_connect('localhost', 'teste', 'pass', 'livros')
or die ("Não foi possível estabelecer a ligação");
```

# php/mysql : ligação à BD

## questões de segurança

- colocar o username e password em variáveis num ficheiro externo ao script externas à chamada à função.
- para segurança acrescida, o ficheiro com as variáveis de autenticação deverá ficar fora da árvore de publicação web do servidor.

# php/mysql : ligação à BD

## questões de segurança

```
// ficheiro pwds.inc
<?php
$user = "teste";
$password = "pass";
?>

// ficheiro acesso.php
//...
include("d:/secrets/pwds.inc");
$ligacao = mysqli_connect('localhost', $user, $password, 'livros')
   or die ("Não foi possível estabelecer a ligação");
//...
```

# php/mysql : ligação à BD

## ligação

- **procedimental:** mysqli **mysqli_connect** ( [string $host [, string $username [, string $passwd [, string $dbname [, int $port [, string $socket]]]]]] );
- **oo:** class **mysqli** {
  **__construct** ( [string $host [, string $username [, string $passwd [, string $dbname [, int $port [, string $socket]]]]]] )
  }

- nota:para indicar que o host é o mesmo em que o script corre:
  - host = “”
  - host = NULL
  - host = “localhost”

# php/mysql : ligação à BD

## selecção de BD

- **procedimental:** bool **mysqli_select_db** ( mysqli $link, string $dbname );
- **oo:** class **mysqli** {
  bool **select_db** ( string $dbname )
  }

# php/mysql : ligação à BD

## fechar ligação

- **procedimental:** bool **mysqli_close** ( mysqli $link )
- **oo:** class **mysqli** {
  bool **close** ( void )
  }

# php/mysql : ligação à BD

## tratamento de erros

### erros de conexão

- **procedimental:** string **mysqli_connect_error** ( void );
- **procedimental:** int **mysqli_connect_errno** ( void );

devolvem respectivamente o código e a mensagem de erro correspondentes ao (eventual) último erro numa operação de ligação à base de dados.

um código igual a zero (0) significa que não existiu erro.

# php/mysql : ligação à BD

## tratamento de erros

### outros erros (por exemplo de query, select_db, etc.)

- **procedimental:** string **mysqli_error** ( mysqli $link );
- **oo:** class **mysqli** {
  string **error**
  }
- **procedimental:** int **mysqli_errno** ( mysqli $link );
- **oo:** class **mysqli** {
  int **errno**
  }

devolvem o código e a mensagem de erro correspondentes ao (eventual) último erro numa função mysqli.

um código igual a zero (0) significa que não existiu erro.

# php/mysql : ligação à BD

**exemplo procedimental:**

```
<?php
   $ligacao = mysqli_connect("localhost", "teste", "teste", "test");

   /* verifica ligacao */
   if (mysqli_connect_errno()) {
       printf("ligação falhou: %s\n", mysqli_connect_error());
       exit();
   }

   /* mudar de BD */
   mysqli_select_db($ligacao,"biblioteca");
   if (mysqli_errno($ligacao)) {
       printf("mudanca de BD falhou: %s\n", mysqli_error($ligacao));
       exit();
   }

   /* fechar ligação */
   mysqli_close($ligacao);
?>
```

# php/mysql : ligação à BD

## exemplo oo:

```
<?php
   $ligacao = new mysqli("localhost", "teste", "teste", "test");

   /* verifica ligacao */
   if (mysqli_connect_errno()) {
       printf("ligação falhou: %s\n", mysqli_connect_error());
       exit();
   }


   /* mudar de BD */
   $ligacao->select_db("biblioteca");
   if ($ligacao->errno) {
       printf("mudanca de BD falhou: %s\n", $ligacao->error);
       exit();
   }


   /* fechar ligação */
   $ligacao->close($ligacao);
?>
```

# php/mysql : interrogando a BD

## interrogação à BD (query)

- **procedimental:** mixed **mysqli_query** ( mysqli $link, string $query [, int $resultmode] )
- **oo:** class **mysqli** {
  mixed **query** ( string $query [, int $resultmode] )
  }

- executa uma query (SQL) à BD
- devolve true ou false ou devolve um objecto de resultado no caso de interrogações SELECT, SHOW, DESCRIBE ou EXPLAIN
- $resultmode :
  - **MYSQLI_USE_RESULT**
    - **obrigatório mysqli_free_result() antes de voltar a interrogar a base de dados**
    - **potencialmente útil para resultados muito grandes (muitas linhas).**
  - **MYSQLI_STORE_RESULT** (defeito)

# php/mysql : interrogando a BD

## interrogação à BD (free_result)

- **procedimental:** void **mysqli_free_result** ( mysqli_result $result )
- **oo:** class **mysqli_result** {
  void **free** ( void )
  void **close** ( void )
  void **free_result** ( void )
  }

- liberta a memória associada a um resultado de uma query.
- deve ser sempre usado quando o resultado deixa de ser necessário

# php/mysql : acesso aos resultados

## acesso aos resultados (número de linhas):

- **procedimental:** int **mysqli_num_rows** ( mysqli_result $result )
- **oo:** class **mysqli_result** {
  int **num_rows**
  }

- devolve número de linhas do resultado
- nota: caso se tenha usado **MYSQLI_USE_RESULT** então o número de linhas poderá não corresponder aonúmero correcto de linhas (pelo menos até que todo o resultados tenha sido processado)

# php/mysql : acesso aos resultados

## acesso aos resultados (número de linhas):

- **procedimental:** int **mysqli_affected_rows** ( mysqli $link )
- **oo:** class **mysqli** {
  int **affected_rows**
  }
- devolve número de linhas afectadas pela query (no caso de INSERT, UPDATE, REPLACE or DELETE)
- no caso de um SELECT é idêntico a **mysqli_num_rows**.
  - -1 indica erro
  - 0 indica que query não afectou linhas ou que ainda não foi executada
  - >0 indica número de linhas afectadas

# php/mysql : acesso aos resultados

## acesso aos resultados (acesso às linhas):

- **procedimental:** mixed **mysqli_fetch_row** ( mysqli_result $result )
- **oo:** class **mysqli_result** {
  mixed **fetch_row** ( void )
  }
- devolve uma linha do resultado sob a forma de um array indexado numericamente (enumerado) (primeiro elemento : 0)
- chamadas consecutivas vão devolvendo linhas subsequentes
- devolve null quando não há mais resultados

# php/mysql : acesso aos resultados

## acesso aos resultados (acesso às linhas):

- **procedimental:** mixed **mysqli_fetch_assoc** ( mysqli_result $result )
- **oo:** class **mysqli_result** {
  mixed **fetch_assoc** ( void )
  }
- devolve uma linha do resultado sob a forma de um array indexado pelo nome do campo (associativo)
- chamadas consecutivas vão devolvendo linhas subsequentes
- devolve null quando não há mais resultados

# php/mysql : acesso aos resultados

## acesso aos resultados (acesso às linhas):

- **procedimental:** mixed **mysqli_fetch_array** ( mysqli_result $result [, int $resulttype] )
- **oo:** class **mysqli_result** {
  mixed **fetch_array** ( [int $resulttype] )
  }
- devolve uma linha do resultado sob a forma de um array enumerado, associativo, ou ambos.
- $resulttype pode ser uma das constantes:
  - **MYSQLI_ASSOC (associativo)**
  - **MYSQLI_NUM (enumerado)**
  - **MYSQLI_BOTH (defeito, ambos)**
- chamadas consecutivas vão devolvendo linhas subsequentes
- devolve null quando não há mais resultados

# php/mysql : acesso aos resultados

## acesso aos resultados (acesso às linhas):

- **procedimental:** object **mysqli_fetch_object** ( mysqli_result $result)
- class **mysqli_result** {
  object **fetch_object** ()
  }
- devolve uma linha do resultado sob a forma de um objecto, os campos podem ser acedidos como propriedades do objecto.
- chamadas consecutivas vão devolvendo linhas subsequentes
- devolve null quando não há mais resultados

# php/mysql : acesso aos resultados

## acesso aos resultados (acesso às linhas):

- **procedimental:** bool **mysqli_data_seek** ( mysqli_result $result, int $offset )
  **oo:** class **mysqli_result** {
  bool **data_seek** ( int $offset )
  }
- coloca o apontador de linhas do resultado numa linha arbitrária.
- offset deve estar contido entre {0, num_linhas – 1}
- devolve true ou false

# php/mysql : acesso aos resultados

## acesso aos resultados (acesso aos campos):

- **procedimental:** int **mysqli_num_fields** ( mysqli_result $result )
- **oo:** class **mysqli_result** {
  int **field_count**
  }
- devolve o número de campos de um resultado

# php/mysql : acesso aos resultados

## acesso aos resultados (acesso aos campos):

- **procedimental:** object **mysqli_fetch_field** ( mysqli_result $result )
- **oo:** class **mysqli_result** {
  object **fetch_field** ( void )
  }
- devolve a definição (objecto) de um campo do resultado
- chamadas consecutivas devolvem campos subsequentes
- devolve null quando não mais campos
- eventualmente útil para aceder a metadata sobre os campos
  - name (nome do campo); table (nome da tabela); type (tipo), etc.