▶ A scuola con PC Open

Web Developer PHP

di Federico Pozzato

1 La scelta di PHP

Citando testualmente la prefazione del manuale ufficiale di PHP.net: "L'obiettivo principale del linguaggio PHP è di permettere agli sviluppatori Web di scrivere velocemente pagine Web dinamiche, ma con PHP si possono fare molte altre cose".

È in questo spirito che *PC Open* introduce nelle sue pagine questo minicorso di PHP, potente linguaggio di scripting open source, pienamente integrabile con HTML (da cui la propedeuticità del corso Webmaster, fornito sul *CD Guida n. 2* unito a questo numero, utile anche se non necessaria) e indirizzato prevalentemente allo sviluppo dei siti Internet dinamici. PHP viene utilizzato anche per creare scripting di righe comando e applicazioni client-side **GUI** (Graphical User Interface) utilizzando le estensioni PHP-GTK. Tralasciando queste due applicazioni specialistiche, vedremo come PHP offra strumenti professionali molto evoluti per la gestione Web, pur mantenendo una struttura semplice, adatta anche a chi si avvicina per la prima volta a linguaggi di programmazione di questo tipo.

Alla fine di questo minicorso avremo gli strumenti di base per migliorare i nostri siti e per affrontare una delle più importanti funzionalità di PHP, ossia l'integrazione con i database e in particolare con MySQL.

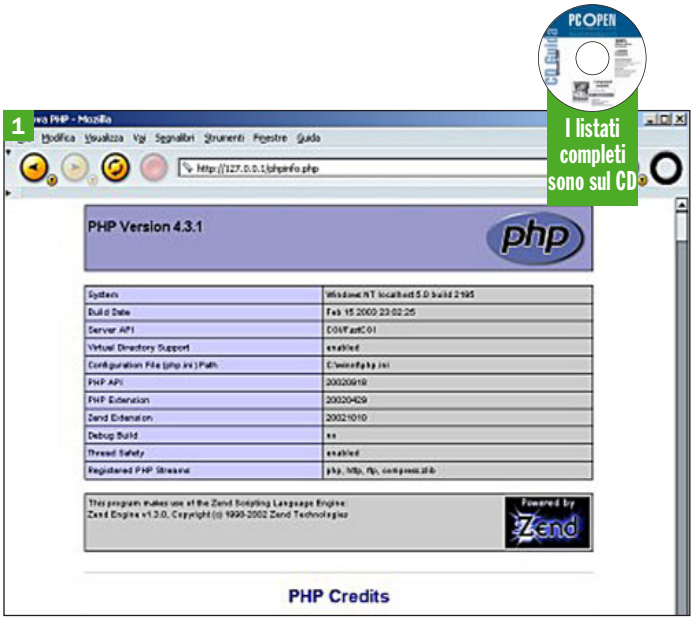
La storia del linguaggio PHP

PHP è un acronimo ricorsivo (tipico dell'ambiente open source) che significa *PHP: Hypertext Preprocessor*. Che cosa sia un ipertesto è chiaro, mentre può lasciare perplessi il significato di Preprocessor: PHP, come **ASP**, è un linguaggio **server-side**, ossia il codice PHP è prima elaborato dal server e solo dopo indirizzato al browser che ha chiamato la pagina. In tal senso PHP elabora la pagina "prima" della sua visualizzazione.

È una caratteristica totalmente diversa rispetto a linguaggi come Javascript che invece sono **client-side**, ossia interamente interpretati dai browser. Vedremo in seguito come questa caratteristica server-side consenta l'implementazione di strutture e funzionalità assolutamente inedite per chi è abituato a ragionare in termini di HTML puro.

PHP consente, inoltre, di creare immagini, file PDF, filmati Flash, generare file basati su XML, utilizzare i protocolli di posta POP3 e IMAP (ne vedremo un'applicazione nella seconda parte del corso), comprimere file in gzip e bz2, gestire sessioni, utilizzare funzioni e classi, fare uso della programmazione orientata agli oggetti e naturalmente interagire con tutti i database più diffusi. Per alcune di queste funzioni bisogna installare estensioni fornite con il file binario di PHP e quindi si dovrà verificare presso il proprio Web provider l'installazione delle specifiche librerie (ad esempio quelle grafiche).

PHP ha ormai quasi 10 anni di storia alle spalle (è stato creato nel 1994 da Rasmus Lerdorf e la prima distribuzione risale al 1995; sito di riferimento: www.php.net), sposa appieno la filosofia open source ed è distribuito con licenza GPL. È un **linguaggio multipiattaforma**, utilizzabile indifferentemente su macchine client con sistemi operativi Windows, Linux, BSD o Mac e interfacciabile con tutti i più popolari Web server.



La pagina phpinfo.php *è la prima pagina di questo corso realizzata in PHP*

Utilizzare PHP off line e scrivere i primi listati

I listati in PHP possono essere creati utilizzando un qualsiasi editor di testo e salvando le pagine con estensione *.php*. Proviamo a scrivere il nostro primo semplice listato (la parte PHP è in grassetto. Più avanti ne approfondiremo il significato), salviamolo col nome *phpinfo.php* (*listato1*) e poi visualizziamolo col nostro browser:

LISTATO 1 (phpinfo.php)

```
<html>
  <head>
    <title>Prova PHP</title>
  </head>
  <body>
    <?php
      phpinfo();
    ?>
  </body>
</html>
```

IL CALENDARIO DELLE LEZIONI

▶ Lezione 1:
- **La scelta di PHP**
- **PHP con un server off line**
- **La prima pagina PHP:** inseriamo PHP in HTML
- **Funzioni base e variabili**
- **I costrutti di controllo:** if else, while, do until, foreach
- **I form:** passaggio di dati coi metodi GET e POST
- **Esempi**

Le prossime puntate

Lezione 2: Approfondiamo PHP
Lezione 3: PHP e i database
Lezione 4: PHP e MySql
Lezione 5: Gestire un sito dinamico con PHP e MySql

Sul vostro browser vedrete o una pagina totalmente vuota oppure il listato esattamente come è scritto qui sopra. Cos'è successo? Abbiamo affermato che PHP è un linguaggio server-side, quindi per visualizzare le pagine create abbiamo bisogno di tre strumenti: un server Web, il supporto PHP (il **parser**, gli elementi di una frase o di un'istruzione) attivato da parte del server scelto e un browser.

Nel nostro esempio ci siamo serviti solamente del browser che, da solo, non è in grado di interpretare la pagina PHP e la fa vedere come fosse un normale documento HTML (quindi, in questo caso, apparentemente vuoto) oppure come un file testo. Nel primo caso la pagina sembra vuota, ma se andiamo a vedere il codice sorgente (ogni browser consente di farlo selezionando l'apposita voce presente nei menu) vedremo il *listato 1*. Per vedere invece la pagina PHP interpretata nella maniera esatta, avremmo dovuto per prima cosa caricare la pagina *phpinfo.php* su un server Web (abilitato per la traduzione delle pagine PHP) e poi richiamare la pagina col browser (vedi *disegno a fondo pagina*). Avremmo ottenuto quanto visibile *nell'immagine 1*: la nostra prima pagina ci restituisce tutte le caratteristiche della versione PHP installata sul server Web.

Per la maggior parte delle persone è certamente improponibile pensare di poter imparare a programmare in PHP restando connessi a Internet, vuoi a causa dei costi, vuoi per la scomodità di dover ogni volta caricare le proprie pagine via FTP (o con upload dedicati) per poi richiamarle tramite browser (vedi *disegno A*).

Risulta quindi necessario installare un server sul proprio PC, col supporto alle pagine PHP per le operazioni di parsing. L'installazione di un server sul proprio PC è già stata descritta nella 7ª lezione del corso Webmaster e la si dà per acquisita (l'installazione del server Web va fatta prima di PHP). Vediamo adesso come installare il parser, con indicazioni riguardanti tre server: Apache, Xitami e IIS di Microsoft.

La prima regola fondamentale è scaricare l'ultima release stabile di PHP per il proprio sistema direttamente dal sito ufficiale: www.php.net.

Al momento in cui scriviamo, la release stabile è una 4.3.x, in attesa del sospirato rilascio della release 5. Tralasciamo l'installazione di PHP su Linux, in quanto è un'operazione generalmente eseguita in automatico scegliendo gli opportuni pacchetti della propria distribuzione, e dedichiamoci all'ambiente Windows. Nell'area download di www.php.net si trovano due file utilizzabili con sistemi Windows:

- un file installer (come: *php-4.3.7-installer.exe*). Questo file ha dimensioni minime (circa 1 MB), è direttamente eseguibile nel sistema e configura automaticamente i server IIS e Xitami, consentendo l'installazione come versione CGI per il server Apache. L'installer non comprende le estensioni di PHP.
- un file zippato contenente il codice binario (come: *php-4.3.7-Win32.zip*). Questo file ha dimensioni ben più consistenti (circa 7 MB), ma consente l'installazione (manuale) di tutti i moduli.

Il server Apache

Come ASP richiamava il server IIS, così PHP richiama il server Apache. Tuttavia, non esiste al momento una procedura automatica per configurare il server Apache, sia esso 1.3.x o 2.0.x (supporto pienamente abilitato solo con una versione di PHP oltre la 4.3.1).

Si può procedere in due modi, configurando PHP in versione CGI (installazione più semplice) o come modulo (installazione consigliata per motivi di sicurezza).

Nel primo caso utilizziamo il file *Installer*. Eseguiamolo (ricordandoci la cartella dove PHP verrà installato: *C:\PHP* va benissimo, ma potete cambiarla a piacere) e arriviamo al termine dell'installazione (anche scegliendo l'opzione di configurazione automatica di Apache, il software vi risponderà che questa procedura non è ancora applicabile). Ora dovete editare il file *httpd.conf* del server Apache (*Start* > *Programmi* > *Apache* > *Configure* > *Edit the httpd.conf configuration file*) e, dopo esservi posizionati nella sezione *ScriptAlias*, aggiungete le seguenti righe:

```
ScriptAlias /php/ "c:/php/"
AddType application/x-httpd-php .php
Action application/x-httpd-php
"/php/php.exe"
```

sostituendo a *c:\php* la cartella di installazione. PHP viene installato nella versione CGI: fate ripartire il server Apache e poi verificate se la pagina *phpinfo.php* viene visualizzata nella maniera corretta.

Questo tipo di installazione è giudicato non sicuro e meno performante dell'installazione di PHP come modulo, però dal momento che ce ne serviamo solo per scopi didattici (esclusivamente off line) va comunque bene per i nostri scopi. Non è possibile, però, installare alcuna estensione.

Il secondo metodo richiede il file zip binario. Per prima cosa si deve scompattare il file in una cartella tipo *c:\php*, quindi copiare il file *php.ini-dist* nella *%SYSTEMROOT%* (per Windows 2000 è *C:\WINNT*) modificandone il nome in *php.ini*. Altra operazione da compiere è copiare il file *php4ts.dll* nella cartella *c:\php\sapi*. Ultimi sforzi: editare il file *httpd.conf* del server Apache (*Start* > *Programmi* > *Apache* > *Configure* > *Edit the httpd.conf configuration file*) e, dopo essersi posizionati nella sezione *LoadModule*, aggiungere le seguenti righe (per Apache 2.0.x):

```
LoadModule php4_module
c:/php/sapi/php4apache2.dll  [o
php4apache.dll per Apache 1.3.x]
AddType application/x-httpd-php .php
```

Facciamo le necessarie verifiche, dopodiché siamo pronti per iniziare.

IIS, PWS e Xitami

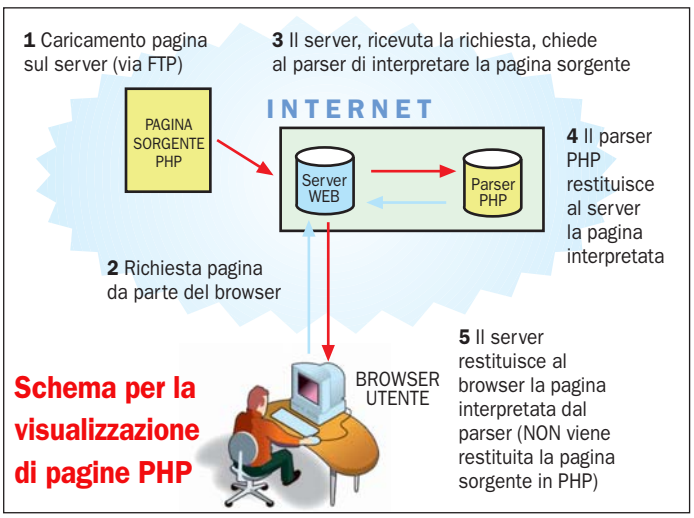
Con IIS, PWS e Xitami la vita è più semplice rispetto ad Apache, sempre che ci si accontenti dell'installazione minima (generalmente sufficiente per i nostri scopi).

In questo caso si deve solo eseguire l'installer PHP: durante la procedura di installazione ci verrà chiesto se è presente un server da configurare con PHP. Scegliete IIS o Xitami e attendete la conclusione dell'install shield: da adesso saremo in grado di verificare off line le nostre pagine PHP. Per verificare il funzionamento, fate partire il server, provate a inserire nella cartella scelta come root dei documenti il file *phpinfo.php* e richiamatelo col browser: dovete ottenere l'*immagine 1* della pagina a lato.

L'installazione delle estensioni implica l'utilizzo del file zip binario e qualche passo "manuale" aggiuntivo. Non la trattiamo in questa sede, ma è comunque presente il file *install.txt* che vi assiste passo passo.

Estensioni

Per utilizzare le estensioni si deve effettuare l'installazione del parser PHP tramite il file zip binario. Andrà poi modificato il file *php.ini* (dalla *%SYSTEMROOT%*) inserendo l'indicazione *C:\php\extensions* dopo l'uguale nella riga che inizia con *extension_dir =*. Fatto questo, sempre nel file *php.ini*, cercare la sezione *Windows extensions* e togliere il punto e virgola davanti alle estensioni da abilitare (ad esempio, per utilizzare le funzioni PDF bisogna abilitare l'estensione *php_pdf.dll*). ▷



Schema per la visualizzazione di pagine PHP

2 La prima pagina in PHP

La prima pagina PHP, in realtà, l'abbiamo già realizzata (la solita *phpinfo.php*) e la possiamo quindi analizzare. La parte iniziale e la parte finale comprendono solo tag e contenuti HTML, mentre la "sezione" PHP, quella interpretata dal parser tramite il server e restituita poi al client browser, viene chiaramente identificata dai tag *<?php* (inizio sezione PHP) e *?>* (fine sezione PHP).

Questi tag possono essere aperti e chiusi quante volte si vuole in un listato HTML: ai fini dell'efficienza nell'elaborazione della pagina, infatti, è meglio uscire dalla modalità PHP se si devono passare blocchi di contenuti HTML "puri" (vale specialmente per pagine complicate).

Tutto quello che si trova compreso tra i tag PHP viene quindi interpretato dal server e restituito al browser per la visualizzazione: la funzione *phpinfo()*, infatti, restituisce tutte le caratteristiche settate per il parser PHP installato nel server. Può essere curioso esaminare il sorgente di questa pagina: sono tutti contenuti HTML e della funzione *phpinfo()*, giustamente, non c'è alcuna traccia.

Notate il ";" al termine della riga compresa nei tag: tutte le istruzioni PHP terminano con questo segnale di fine riga. Il tag di chiusura *?>* fa anche da ";" e quindi per l'ultima istruzione prima della chiusura si potrebbe anche eliminare il ";", però è certamente meglio non prendere certe abitudini e usare tutta la sintassi normalmente richiesta.

Un secondo esempio, *hello.php*, ci consente di introdurre la funzione *echo* che ha lo scopo di restituire un output (*listato 2 - hello.php*):

LISTATO 2 (hello.php)
```
<?php
  echo "ciao PC OPEN";
?>
```

La funzione *print()* produce esattamente lo stesso output.

È possibile anche introdurre dei tag HTML all'interno della sezione PHP come si può vedere da *hello-html.php* (*listato 3*). Andranno inseriti all'interno di *echo* in quanto questi tag HTML devono diventare un output del parser:

LISTATO 3
```
<?php
    echo "<h1>Ciao PC OPEN</h1>";
?>
```

Per indicare inizio e fine della sezione PHP

Ci sono altri sistemi per indicare l'inizio e la fine della sezione PHP:

- **tag brevi** (short tag): basta indicare *<?* e *?>*. Per usarli deve essere appositamente configurato il parser PHP del server Web ospitante. Generalmente è così, ma l'uso di questi short tag è comunque sconsigliato per motivi di sicurezza: se, per qualunque motivo (manutenzione, aggiornamento, e così via), il parser non interpretasse più in maniera corretta gli short tag, il vostro codice diverrebbe visibile a tutti (come nell'esempio visto all'inizio).
- **indicazione di script:** *<script language="php">* e *</script>*. Non è sempre interpretato correttamente quindi evitiamolo.
- **ASP tag**: i tag sono gli stessi usati per ASP: *<%* e *%>*. Anche in questo caso deve essere attivata un'apposita opzione del parser (*asp_tags = On*), con gli stessi problemi di sicurezza già indicati per gli short tag.

In *tag.php* trovate riuniti insieme questi ultimi tre modi di indicare il codice PHP (vedi *listato 4*).

Per buona regola, è sempre bene utilizzare *<?php* e cercare di essere il più ordinati possibile, nell'eventualità di dover fare delle modifiche a distanza di tempo.

Per inserire dei commenti si premettono al commento le doppie slash *//*.

Curiosità

In un'ottica di filosofia open source si sente nominare l'acronimo **LAMP**. Esso indica un gruppo di prodotti divenuti, nel loro insieme, un riferimento per lo sviluppo di siti Web:

Linux
Apache
MySQL
PHP

3 Tipi di dato, variabili e operatori

Per iniziare realmente a usare PHP e scoprirne le caratteristiche, abbiamo bisogno di spendere ancora qualche minuto per introdurre la gestione delle variabili.

Diversamente da ASP, con PHP non serve dichiarare esplicitamente le variabili indicando nome e tipo, ma, più semplicemente, basta assegnare alla variabile il suo valore: sarà il parser ad attribuire alla variabile il giusto "tipo" (dichiarazione implicita). Tutte le variabili vanno indicate col simbolo *$* davanti al nome scelto (il primo digit dopo *$* non deve mai essere un numero).

Tipi di variabili scalari

Vediamo i quattro tipi di variabile scalare che abbiamo a disposizione:

```
$prova1 = 12;
// $prova1 è un tipo di dato intero (integer),
```

in questo esempio è un intero a base decimale, ma avremmo potuto fare una dichiarazione con le notazioni esadecimale o ottale. In caso di superamento del limite massimo previsto per un intero, PHP assegna automaticamente alla variabile il tipo float.

```
$prova2 = 1,23;
// $prova2 è un tipo di dato
a virgola mobile (float).

$prova3 = TRUE;
// $prova3 è un tipo di dato
booleano (boolean),
```

ossia assume solo valori vero (true) o falso (false). Il numero 0 o una stringa vuota corrispondono al valore booleno *false*; un numero diverso da 0 o una stringa non vuota corrispondono al valore booleano *true*.

```
$prova4 = "pippo";
// $prova4 è un tipo di dato
stringa (string).
```

La stringa può essere definita utilizzando le virgolette singole ' (*single quotes*) o le virgolette doppie " (*double quotes*). Nel proseguimento del corso si userà la seconda notazione che permette, tra le altre cose, di inserire dei nomi di variabili all'interno della dichiarazione

```
$1prova = 1;
// il nome della variabile non
è valido (numero dopo $)
```

Ridichiarando le variabili, o concatenandole, PHP sceglie sempre automaticamente il tipo di dato da gestire. Se volessimo sapere, ad esempio per motivi di debug, quale sia il tipo di dato di una certa variabile, si può usare la funzione **gettype**:

```
<?php
  $prova = 1,2;
  echo gettype ($prova); // si ottiene a
  video il tipo di dato: float
?>
```

Il tipo può essere imposto mediante l'istruzione **settype** all'atto già della dichiarazione

(esplicita) della variabile o attraverso l'imposizione (**casting**) del tipo (boolean, int, float, string, array, object, null):

```
<?php
 setype ($prova, "int");
 $prova = 1,2;
 echo $prova; // si ottiene a video
 il valore del tipo "int": 1
?>

<?php
 $prova = 1,2;
 $intero = (int) $prova;
 echo $intero; // si ottiene a video
 il valore del tipo "int": 1
?>
```

Variabili di tipo composto

Oltre alle variabili di tipo scalare, ci sono due variabili di tipo composto: array e object.

Array non è solo il classico vettore (ossia una lista di elementi individuati da un indice numerico avente numero iniziale 0), ma può essere anche una tabella di hash (una mappa), dove un elemento è individuato da una chiave non necessariamente numerica. Ciò lascia estrema libertà di utilizzo del tipo array, ma impone anche di prestare attenzione quando viene richiamato un valore. Le chiavi possono essere valori sia numerici, sia di tipo stringa; gli elementi dell'array possono essere di qualunque tipo (anche altri array).

Se vogliamo creare un vettore, lo possiamo fare in maniera esplicita (indico la parola chiave *array*) o implicita (assegno i valori alla variabile che sarà il vettore):

```
$vettore_a = array (2, 5, 7, 9, 11);
// dichiarazione esplicita con elementi
solo numerici
 $vettore_b = array (2, "pippo", true,
1.24, array (1,2,3)); // dichiarazione
esplicita con elementi misti
 $vettore_c = array (0 => 2, 1 =>
"pippo", 2 => true); // notazione per
indicare le chiavi del vettore (potevano
essere omesse senza problemi)
 $vettore_d[0] = 2; // dichiarazione
implicita del primo elemento del
vettore_d
 $vettore_d [1] = "pippo";
// dichiarazione implicita del secondo
elemento del vettore_d
```

Per visualizzare un array e le sue chiavi usiamo la funzione **print_r**, come indicato nell'esempio *vettore.php* (*listato 5*). Dagli esempi qui sopra otteniamo coppie che sono sempre identificate da un numero. Per fare riferimento a un elemento specifico (per stamparlo, modificarlo e così via) si usa il costrutto:

```
 $array[n]
 con n valore intero che parte da 0.
```

Creare un array come tabella di **hash** è altrettanto semplice e versatile, ma in questo caso ci dovremo preoccupare di indicare anche il valore (sia esso un numero o una stringa) da assegnare alla chiave dell'elemento dell'array:

```
$hash_a = array ("italia" => "roma",
     "francia" => "parigi", "spagna" =>
     "madrid", 12 => "pippo");
$hash_b["italia"] = "roma";
$hash_b ["francia"] = "parigi";
$hash_b[12] = "pippo";
```

La stampa dell'hash la possiamo vedere nell'esempio *hash.php* (*listato_06*). Per eliminare, modificare, stampare i dati dell'hash vi faremo riferimento con i due costrutti (diversi a seconda del tipo di chiave):

```
 $array[numero]
 $array["chiave"] // o: $array['chiave']
```

Si rivelerà utile questa possibilità che ci concede PHP: dopo aver dichiarato un array *hash_a*, possiamo inserire gli elementi anche senza fare riferimento a una chiave specifica:

```
 $hash_a[ ] = "topolino";
```

In questo caso la chiave dell'elemento sarà il numero intero successivo al numero intero più alto già presente nell'hash. Pertanto, come mostrato dall'array *hash_b* dell'esempio *hash-due.php* (*listato 7*), possiamo creare facilmente un array assegnando come chiave iniziale un numero diverso da 0. Vedremo come questo possa essere utile quando parleremo di date.

Il tipo di dato **object** lo analizzeremo nel prosieguo del corso parlando delle classi.

Tipi di variabili speciali

Per finire l'analisi, vi sono ancora due tipi speciali di variabili: **resource** e **null**. Il primo è creato e usato da una serie di funzioni ben definite (per approfondimenti vedere le appendici del manuale PHP), mentre il secondo serve ad assegnare a una variabile lo speciale valore *NULL*.

Nell'ambito delle variabili, oltre a quelle definite da noi troviamo le cosiddette **variabili superglobals**, arrays contenenti informazioni sul Web server, sull'ambiente di utilizzo e sugli input degli utenti. Alcuni esempi si possono vedere (coi relativi valori) aprendo la nostra prima pagina creata (*phpinfo.php*), e altri li vedremo introducendo i form e le sessioni. Quasi tutti questi array superglobali iniziano con un underscore: *$_POST, $_GET, $_SESSION, S_SERVER*.

Operatori

Gli operatori consentono di eseguire operazioni sulle variabili, operazioni non solo di natura numerica (esempio: calcoli aritmetici) e testuale (esempio: concatenazione), ma anche operazioni di controllo (esempio: logica binaria) e confronto (esempio: confronto di valori e tipi).

La concatenazione è senza dubbio una delle operazioni più usate: le variabili in PHP possono essere concatenate usando il segno ".":

```
<?php
 $a = "PC OPEN";
 $b = "una rivista di informatica";
 $c = $a.", ".$b;
 echo $c; // si ottiene come output:
PCOPEN, una rivista di informatica
?>
```

Notate l'inserimento, tra virgolette doppie, della virgola e dello spazio vuoto per meglio visualizzare a video le due variabili.

Lo stesso risultato lo si sarebbe ottenuto utilizzando opportunamente la funzione *echo* (senza però utilizzare la variabile *$c*):

```
 echo "$a, $b"; // equivalente a: echo
$a.", ".$b;
```

Si può anche "concatenare" una stringa con se stessa con questa sintassi abbreviata:

```
<?php
 $a = "PCOPEN ";
 $a. = "amico" // equivale a: $a =
$a."amico";
 echo $a; // si ottiene a video: PC
OPEN amico
?>
```

Vi sono poi operatori aritmetici, logici e di confronto. Per comodità sono stati riassunti nel riquadro della pagina successiva, indicando solo gli operatori che verranno utilizzati nel corso; per tutti gli altri operatori (e per la loro priorità) fare riferimento al manuale di PHP.net.

Costanti

Per alcune esigenze potremmo avere bisogno di definire delle costanti e non delle variabili. In questo caso il costrutto da usare è:

```
define ("nome", valore);
echo nome;
```

Valore può essere solo di tipo scalare; nome non è preceduto dal simbolo *$* ed è case sensitive.

Applicazione: immagine random

Vediamo finalmente un esempio pratico di quanto abbiamo imparato: immaginiamo di avere inserito in una certa cartella del nostro spazio Web una serie di immagini contenute nella directory *Immagini* e aventi nome *vacanza_x.jpg* dove *x* è un numero intero progressivo che parte dal valore *1*.

Sulla nostra home page è visualizzata una di queste immagini, ma noi vorremmo, per rendere il sito più attraente e farlo sembrare sempre diverso e aggiornato, fare apparire un'immagine diversa ogni volta che l'home page viene richiesta (o quando viene fatto il refresh). La situazione di partenza è visibile nella pagina *intro.html* (*listato 8*) che mostra l'immagine (un quadrato colorato) *vacanza_1.jpg* al centro della pagina.

Dovremo fare in modo che il valore x sia generato in maniera random a ogni chiamata della home page. Utilizzeremo a tale scopo la funzione **rand(y,z)** di PHP, supponendo di avere cinque immagini nel server Web. Il risultato è visibile aprendo la pagina *random.php* (*listato 9*): la parte che ci interessa, quella che genera il numero casuale compreso tra 1 e 5 e che visualizza l'immagine è la seguente:

```
<?php
 $num=rand(1,5);
 // rand(min,max) genera un numero
casuale intero compreso tra min e max
```

inclusi
```
echo "<img src='immagini/
vacanza_".$num.".jpg'>"; //
```
gli attributi HTML vanno indicati usando gli apici singoli e non le virgolette
```
?>
```

Riprenderemo l'esempio, per approfondirlo e migliorarlo, dopo aver introdotto il costrutto **while**.

Le date

Come avrete notato, PHP non propone nessun tipo di variabile *date* o *time*, ma molto probabilmente avremo spesso bisogno di utilizzare questi formati data e tempo.

A tale scopo useremo le funzioni di data e ora previste da PHP. La principale è senza dubbio

```
date (formato, timestamp);
```

in grado di restituire tutti i possibili valori in termini di data e tempo estraendoli da uno timestamp. Il **timestamp**, nato nel modo UNIX, rappresenta il numero di secondi trascorsi dal 1/1/1970 (la cosiddetta Unix Epoch) ed è valido fino al 2037 (va bene, quindi, per le nostre esigenze attuali). Se non indichiamo il timestamp, date considererà giorno e ora di quel preciso momento del server Web ospitante PHP. Se volete inserire un vostro timestamp, dovete usare la funzione *mktime()* con questa modalità:

```
mktime (ora, minuti, secondi, mese,
giorno, anno, ora_legale) // ora_legale
(valore -1 o 0) può essere omesso:
PHP cercherà di ricavare il valore dal
sistema
```

Date fornisce una lunga lista di formati tra cui scegliere, come:

```
$timestamp = mktime (12, 0, 0, 2,
11, 2004); // timestamp del
12/02/2004, ore 12.00

echo date ("d-m-y", $timestamp);
// a video: 11-02-04
echo date ("D d, F Y", $timestamp);
// a video: Wed 11, February 2004
echo date ("H:i:s", $timestamp);
// a video: 12:00:00
echo date ("w, z", $timestamp);
// a video: 3 (giorno della settimana),
41 (giorno dell'anno)
```

Può essere interessante anche studiare la funzione **getdate**, la quale inizializza un array con tutti i valori estratti da uno timestamp specificato.

Esempi di utilizzo di date in PHP

Trattare con le date non è semplice, ma può essere molto utile saperlo fare. Ecco quindi due esempi per fare un po' di pratica.

Oggi.php (*listato 10*): serve a esprimere la data odierna utilizzando i nomi estesi dei giorni e dei mesi, in italiano. La funzione *date* non è sufficiente perché estrae sì questi dati, ma in lingua inglese. Si potrebbe usare *setlocale* e *strftime*, ma preferiamo usare un altro sistema che ci consente anche si utilizzare gli array. *Date* ci consente di estrarre il valore numerico (senza zeri) sia del giorno della settimana (0=domenica) che del mese, quindi per prima cosa creiamo due array (quello dei mesi ha il primo numero della chiave corrispondente a 1) e poi il gioco è fatto:

```
<?php
 $giorno = array ("domenica",
     "lunedì", "martedì", "mercoledì",
     "giovedì", "venerdì", "sabato");
 $mese = array (1 => "gennaio",
     "febbraio", "marzo", "aprile",
     "maggio", "giugno", "luglio",
    "agosto", "settembre", "ottobre",
     "novembre", "dicembre");
 $giorno_mese = date ("d");
 $anno = date ("y");
 $mese_nome = $mese[date("n")];
// restituisce il nome del mese
 $giorno_nome = $giorno[date("w")];
// restituisce il nome del giorno della
settimana
 echo "Ciao, oggi è $giorno_nome
    $giorno_mese $mese_nome
    $anno";
 ?>
```

counter.php (*listato 11*): serve a organizzare un elemento che funga da contatore per indicare, ad esempio, quanti giorni mancano a una data particolare. Per prima cosa dovremo ricavare il *timestamp* della data odierna (in un orario x) e della data da raggiungere (sempre allo stesso orario x. Qui prendiamo come riferimento il 31 dicembre del 2004). Facendo la differenza tra questi timestamp otterremo il numero di secondi che separa le due date, quindi dividendo per il numero di secondi di un giorno (60*60*24 = 86.400 secondi) avremo il numero di giorni che ci separano dal nostro riferimento. I dettagli sono commentati nel listato:

```
<?php
 $inizio = mktime(12,0,0,date("n"),
      date("j"), date("Y"), 0);
 $fine = mktime
      (12,0,0,12,31,2004,0);
 $diff=($fine-$inizio)/86400;
 echo "Alla fine del 2004 mancano
      $diff giorni";
?>
```

Operatori aritmetici

Poniamo: **$a=5** e **$b=3**

```
$c=$a+$b;        // somma: $c=8
$c=$a-$b;        // sottrazione: $c=2
$c=$a*$b;        // moltiplicazione: $c=15
$c=$a/$b;        // divisione: $c=1,66
$c=$a%$b;        // modulo: $c=2
```

Pur non essendo un operatore va ricordato anche:

```
$c=int($a/$b);   // quoziente intero della divisione: $c=1
```

Utilizzeremo anche gli operatori di incremento e decremento:

```
$c=++$a;         // $c=6  (pre-incremento) e $a=6
$c=–$a;          // $c=4  (pre-decremento) e $a=4
$c=$a++;         // $c=5  (post-incremento) e $a=6
$c=$a–;          // $c=5  (post-decremento) e $a=4
```

Operatori logici

Poniamo: **$a=true** e **$b=false**

```
$c=$a and $b;    // AND: $c=false
$c=$a or $b;     // OR: $c=true
$c=! $a;         // NOT: $c=false
```

Operatori di confronto

Useremo questi operatori coi costrutti di controllo.

```
$a == $b;        // true se $a uguale a $b
```

NB: l'operatore **==** effettua un controllo di confronto senza modificare il valori di $a e $b, mentre l'operatore **=** effettua un'assegnazione di valore.

```
$a <> $b;        // true se $a diverso da $b
$a < $b;         // true se $a minore di $b
$a > $b;         // true se $a maggiore di $b
$a <= $b;        // true se $a minore o uguale a $b
$a >= $b;        // true se $a maggiore o uguale a $b
```

4 I form

Per interagire con chi accede alle nostre pagine abbiamo a disposizione, tramite HTML, una serie completa di campi da compilare. L'utente ci fornirà quindi i **dati** che verranno usati come base per elaborare le azioni successive.

Come webmaster dobbiamo compiere questi passaggi: creare il form di inserimento dati, acquisire i dati (registrandoli nelle forme opportune ed eventualmente validandoli), fornire una risposta all'utente. La creazione del form si effettua con HTML e la si dà per acquisita (vedi corso webmaster). Fondamentale per passaggio, registrazione e utilizzo dei dati è la prima riga dell'istruzione form, e in particolare la parola **action** e i due metodi **get** e **post**:

```
<form action="test.php"
      method="post">
```

Questa riga indica, una volta sottoscritti i dati con l'apposito controllo *submit*, che il browser verrà indirizzato alla pagina *test.php* sul CD (se action è omessa la pagina cui l'utente viene reindirizzato è la pagina stessa che comprende il form) e i dati passeranno col metodo post.

Il **metodo post** consente di trasmettere, in maniera totalmente trasparente all'utente, tutti i dati compilati nel form senza alcun limite di spazio, mentre il **metodo get** trasferisce i dati visualizzandoli sulla riga di indirizzo del browser (e col limite di soli 256 caratteri trasmissibili). Per vedere la differenza fate riferimento a *post.php* (*listato_12,* è un form con metodo post) e a *get.php* (*listato_13*, è un form con metodo get): entrambi hanno come attributo action la pagina *test.php*, ma la differenza di trasmissione dei dati la notate sulla riga del browser (vedi *immagini 2 e 3*).

Tre istruzioni

Una volta trasferiti i dati, dovremo essere in grado di accedervi: PHP mette a disposizione queste **tre istruzioni** "superglobali":

```
$_GET["dato"] // per form con
              metodo get

$_POST["dato"] // per form con
               metodo post

$_REQUEST["dato"]  // sia per form
                   con metodo get che post
```

$_REQUEST concede una possibilità che *$_GET* e *$_POST* non danno: una volta registrato un dato del tipo *$_REQUEST["nome_dato"]* possiamo poi richiamarlo nella pagina come fosse una variabile appena dichiarata, ossia con la forma:

$nome_dato

Metodo GET

Metodo POST

Conto alla rovescia modificato

Ora siamo in grado di interagire col nostro utente. Potremmo, ad esempio, aumentare l'utilità del "conto alla rovescia" presentato nella sezione precedente, consentendo all'utente di indicare lui stesso la data cui fare riferimento.

Dobbiamo solo stare attenti ai dati inseriti (sia in termini di formato che di valori consentiti) e quindi useremo dei controlli con caselle combinate (le liste a discesa), come nell'esempio *inserisci.html* (*listato 14*). I valori inseriti saranno passati attraverso il metodo post alla pagina *conto_rovescia.php* sul CD. È possibile anche inserire date senza senso (ad esempio 31 giugno): la funzione timestamp si occuperà di trasformare l'errato inserimento nella data più vicina e visualizzeremo questa data nella risposta. La data inserita dall'utente nel form viene così riportata, tramite *$_POST*, nella pagina indicata da action:

```
// $fine è il timestamp della data
richiesta
$fine = mktime (12,0,0,$_POST[
      'mese'],$_POST['giorno'],
      $_POST['anno'],0);
```

Sarebbe comunque da impedire l'inserimento di date non valide, ma per fare questo dobbiamo aspettare di aver introdotto i costrutti di controllo.

Modifica dello stile di una pagina Web

Un altro interessante esempio lo possiamo immaginare sfruttando le caratteristiche dei **fogli di stile**: vogliamo, infatti, dare la possibilità al nostro utente di visualizzare i nostri articoli riportati su pagina Web con carattere, dimensioni e colori preferiti.

Faremo transitare il nostro utente dalla pagina *introduzione.html* (*listato_15*) dove è presente un form attraverso il quale si dovrà scegliere il carattere poi visualizzato nella pagina *articolo.php* sul CD.

La scelta dei caratteri è ridotta alle quattro famiglie (arial, verdana, times, courier) generalmente visualizzate da tutti i browser, e i colori consentiti sono espressi in termini esadecimali, anche se l'utente vede delle descrizioni "normali" (bianco, nero, giallo, ...). Scelta la combinazione voluta, la pagina *articolo.php* verrà visualizzata di conseguenza tramite le indicazioni inserite nel foglio di stile incorporato:

```
<style>
span.font {
font-family: <?php echo
      $_POST['carattere']; ?>;
font-size: <?php echo
      $_POST['dimensione']; ?>px;
color: <?php echo $_POST['col_car']; ?>;
}
body {
background-color: <?php echo
      $_POST['col_sfondo']; ?>;
}
</style>
```

POST e GET in PHP

Fino alla versione 4.2.0, PHP concedeva sempre e comunque di richiamare i dati inseriti nei form con la forma *$nome_dato*. Ora questa possibilità, salvo il caso visto con *$_REQUEST*, non è più concessa per motivi di sicurezza. Per capirne le implicazioni, immaginate che qualcuno avesse chiamato la pagina *test.php* senza passare attraverso i form: semplicemente scrivendo http://www.xxxxx.it/test.php?nome=topolino questo utente smaliziato sarebbe riuscito ad attribuire alla variabile *$nome* il valore *topolino*. Vi sono casi in cui questa possibilità conduce a eventi assolutamente indesiderati, compreso l'accesso a pagine protette.
Per utilizzare il "vecchio" sistema bisogna fare una modifica su *php.ini*, sostituendo il valore *off* a *on* sulla riga *register_globals* (il valore valido per il vostro sistema lo potete vedere anche controllando la pagina visualizzabile usando la pagina *phpinfo.php*).
Sottolineiamo però, ancora una volta, come sia meglio comunque abituarsi a usare *$_POST* e *$_GET*, sia per evitare problemi di sicurezza, sia per evitare che il vostro provider, magari aggiornando il parser PHP, cambi il valore di *register_globals* da *on* a *off* lasciandovi un bel po' di problemi sulla visualizzazione del vostro sito!

L'esempio può essere ampliato coinvolgendo tutte le caratteristiche della pagina come indicato dalle proprietà dei CSS. Anche qui, però, mancano le strutture di controllo: cosa succede se si accede alla pagina senza passare attraverso l'introduzione? Sarà il browser a decidere la visualizzazione, ma chiaramente questa è una condizione inaccettabile per il webmaster.

Accessibilità dei siti Web

Nell'esempio indicato è possibile variare la dimensione (in pixel) del carattere.

Si potrebbe pensare che questa indicazione definisca la dimensione del carattere in maniera fissa, indipendente dal browser.

In realtà se aprite la pagina *articolo.php* con Mozilla (oppure Opera o Firefox) vedrete che usando le combinazioni *CTRL*+ + o *CTRL*+ - i caratteri aumenteranno o diminuiranno di dimensione.

Microsoft Explorer, invece, non modifica assolutamente la dimensione dei font cui è stata assegnata una dimensione fissa (in punti o pixel).

Le disposizioni del W3C

Chi sbaglia? Diversamente da quanto si potrebbe pensare, è Explorer a non rispettare i dettami stabiliti per i browser Internet: anche se questa possibilità di ridimensionamento manda in crisi i webmaster (desiderosi di mantenere il layout studiato per la pagina), essa è espressamente richiesta dal W3C (*World Wide Web Consortium*) per accrescere l'usabilità del Web a persone cui può essere assolutamente necessario scalare i caratteri.

Tenete poi presente che una certa dimensione in pixel può essere splendida per certe risoluzioni video, ma per altre può rivelarsi non adatta perché troppo piccola.

Non prendetevela, quindi, se dopo aver speso giorni e giorni a stabilire la giusta visualizzazione della pagina fissando tutti i font, un semplice *CTRL*+ + potrà scompaginare tutto.

5 Costrutti di controllo

Per costruire script avanzati bisogna padroneggiare assolutamente le strutture di controllo, ossia quelle istruzioni che consentono di eseguire delle azioni solo a fronte della verifica di particolari condizioni.

L'esempio tipico è dato dalla pagina protetta da password: volendo descrivere a parole l'azione connessa, essa suona come *Accedi alla pagina solo se la password che hai fornito corrisponde a quella registrata per il nome utente inserito*. "Solo se" è il costrutto di controllo.

PHP ci fornisce tutti i costrutti standard (*for, if, while, switch*) e un paio di istruzioni estremamente utili (*foreach* e *isset*) per accelerare i tempi di scrittura del codice.

If – else – elseif

La **condizione if** (con l'eventuale aggiunta di **else** o **elseif**) è la base dei costrutti di controllo. Viene definita una condizione: se (if) è rispettata (condizione vera) viene eseguito il codice tra le parentesi graffe dopo la if, altrimenti si esce dalla if senza eseguire il codice o viene eseguito del codice alternativo (else):

```
if (condizione) {
 codice da eseguire se condizione
è vera
}
else {
 codice da eseguire se condizione
è false
}
```

Le condizioni if possono essere nidificate senza problemi per creare strutture complesse. La condizione elseif consente di aggiungere ulteriori condizioni da verificare prima di arrivare al codice else o di uscire dal costrutto.

Un esempio di utilizzo lo possiamo vedere in *confronto.php* (*listato 16*): creiamo due numeri interi casuali *$a* e *$b* compresi tra 1 e 4 e verifichiamo quale dei due sia più grande:

In alcuni frangenti si dimostra utile usare la parola chiave *exit* all'interno di una condizione: quando viene incontrata questa istruzione, il server blocca la "traduzione" della pagina e la invia al browser client come definita in quel momento. Un esempio è disponibile in *controllo-exit.php* (*listato_17*): se la prima condizione è soddisfatta non posso più visualizzare quanto scritto dopo la chiusura del ciclo if:

```
    if ($a>$b) {
    echo "\$a=$a e \$b=$b:<br>";
    echo "\$a è maggiore di \$b";
    echo "<p><h3> Verificata questa
condizione blocco il codice successivo
usando exit.
Non potete sapere cosa c'è scritto
dopo il blocco if!</h3>";
    exit;
    }
```

While

Il costrutto del ciclo **while** è veramente semplice e al contempo potente: le istruzioni contenute nel ciclo sono iterate finché la condizione è vera, ogni volta andando a leggere (ordinatamente) il valore successivo della condizione. Quando la condizione diventa falsa il ciclo si arresta e prosegue oltre:

```
while (condizione) {
 codice
}
```

Questo costrutto è molto utilizzato per leggere le liste: ad esempio si usa while per mandare a video tutti i record di una tabella terminando l'esecuzione quando la tabella è vuota (vedremo questo utilizzo nelle lezioni dedicate a MySQL).

Con il costrutto while siamo finalmente in grado di migliorare l'esempio creato per la visualizzazione random di una foto (*random.php*, *listato_09*).

Cosa succede, infatti, se inseriamo altre immagini nella cartella Web che abbiamo scelto, sempre seguendo le stesse logiche di numerazione?

Dovremmo correggere off-line il listato di cui sopra, sostituendo al valore 5 il valore più alto delle nuove immagini inserite, per poi trasferire di nuovo la pagina sul server sovrascrivendo la precedente. Facile, ma è una possibile fonte di errori: PHP ci può risparmiare questa operazione rendendo la nostra pagina valida qualunque sia il numero di immagini presenti nella cartella.

Per questo sfruttiamo le funzioni per le directory (in particolare *opendir* e *readdir*) e i costrutti di controllo.

Innanzi tutto usiamo *opendir* per registrare (e validare) il percorso della cartella, e poi *readdir* per leggere il primo file della directory.

Qui interviene while: attraverso una condizione while su *readdir*, la directory viene letta file dopo file e ogni riferimento è registrato su un array. Esau-

LISTATO 16

```
<?php
 $a = rand (1,10);
 $b = rand (1,10);
    if ($a>$b) {
       echo "$a=$a e \$b=$b:<br>\$a è maggiore di \$b";  // il carattere
       escape (\) serve a visualizzare come output simboli particolari come $
       }
    elseif ($a==$b) {   // notare l'operatore di confronto ==
       echo "$a=$a e \$b=$b:<br>\$a è uguale a \$b";
       }
    else {
       echo "$a=$a e \$b=$b:<br>\$a è minore di \$b";
       }
 ?>
```

rita la lettura, il codice esce dal ciclo while e possiamo finalmente definire $max, ossia il numero di immagini che abbiamo trovato nella directory *Immagini*.

Ecco quindi la pagina *random-due.php* (*listato 18*).

La parte più utile da analizzare è la seguente:

```
while (false !== ($file = readdir($cartella))) {
    // il ciclo while verrà iterato finché
la cartella non sarà stata interamente
letta. A ogni iterazione $file assume
il nome del file successivo.
Con la prima iterazione creo l'array
e registro il primo nome di file,
poi l'array viene riempito con i nomi
dei file seguenti
    $lista[ ]=$file;
    }
    // count conta il numero
di elementi contenuti nell'array. Viene
diminuito di due perché l'array della
cartella contiene sempre le indicazioni
UNIX "." e ".."
    $max=count($lista)-2;
    if ($max==0) {
    echo "Attenzione: Non ci sono
          immagini nella cartella";
    }
 else {
    $num=rand(1,$max);
 }
```

Ricordiamo che PHP 5, al momento rilasciato solo come release candidate, prevede una nuova funzione, **scandir**, in grado di evitare questo passaggio con while: scandir sarà infatti in grado di leggere i contenuti di una cartella e registrarli direttamente in un array senza utilizzare altre funzioni.

For

Il ciclo for consente di iterare un'azione, definendo la condizione iniziale di un puntatore, il controllo da fare e l'incremento del puntatore dopo ogni ciclo. L'azione verrà eseguita (e rieseguita) finché il controllo darà risposta true alla verifica.

```
for (punto iniziale; condizione da
rispettare; tipo di iterazione) {
    codice da eseguire
 }
```

For si rivela utilissimo per costruire strutture di pagina di cui non possiamo conoscere con anticipo tutti gli elementi: stiamo parlando, quindi, di veri e propri siti dinamici.

Per capire meglio, possiamo costruire un esempio basandoci su quanto visto per il costrutto while con la directory *Immagini*.

In questo caso, però, il nostro scopo è ottenere la lista delle immagini contenute nella cartella, visualizzandole nella pagina Web *elenco-dir.php*. Qui trovate la parte del listato che si riferisce al costrutto **for** *(listato 19)*.

Possiamo migliorare la visibilità del nostro elenco visualizzando i file alternativamente su righe aventi colore di sfondo diverso (*elenco-dir-due.php, listato_20*). Qui l'attenzione va posta nel conteggio delle righe da mandare in output: se sono pari verrà eseguito un ciclo for e usciremo dalla pagina, mentre se sono dispari il ciclo for dovrà fermarsi prima dell'ultima riga, che andrà stampata a parte (*listato 20*).

Un buon esercizio per verificare di aver compreso bene i concetti di cui sopra può essere il seguente (è una variazione sul tema di quanto appena visto): creare una galleria delle immagini della directory, visualizzandole a due a due, una di fianco all'altra.

Chiaramente dovremo usare le tabelle e/o i CSS, ponendo attenzione all'eventuale ultima immagine dispari. La soluzione da me proposta la trovate nella pagina *elenco-galleria.php* (sul CD).

Vedremo nelle prossime puntate del corso come creare una galleria con un certo numero prefissato di immagini, o altri oggetti, per ogni pagina (ad esempio sei immagini per pagina) e una barra di navigazione per spostarsi tra le pagine.

LISTATO 19

```
echo "<h4>La cartella 'immagini' contiene questi file:</h4><p>";
        for ($i=2;$i<=count($lista)-1;$i++) {  // il contatore $i parte da 2
            perché i primi due valori registrati nell'array $lista sono
            "." e "..", e non voglio visualizzarli

        $ordine=$i-1;
        echo "————————————————————<br>";
        echo "<span class='output'>File $ordine: $lista[$i]</span><br>";
    }
```

LISTATO 20

```
if ($fine%2==0) { // controllo se nella cartella c'è un numero pari di immagini
        for ($i=2;$i<count($lista);$i=$i+2) {  // il ciclo di for visualizza due file
            ad ogni iterazione
            $ordine=$i-1;
            $j=$i+1;
            echo "<span class='dispari'>File $ordine: $lista[$i]</span>";
            echo "<span class='pari'>File $i: $lista[$j]</span>";
            }
        }
 else { // il numero di immagini è dispari, quindi si deve aggiungere l'ultima riga
         dispari
        for ($i=2;$i<$fine;$i=$i+2) { // il ciclo for visualizza due file per ogni
        iterazione e si ferma prima  dell'ultima immagine
            $ordine=$i-1;
            $j=$i+1;
            echo "<span class='dispari'>File $ordine: $lista[$i]</span>";
            echo "<span class='pari'>File $i: $lista[$j]</span>";
            }
        $j=$i-1;
        echo "<span class='dispari'>File $j: $lista[$i]</span>";
        }
```

Switch

Il costrutto **switch** somiglia a una serie di if di uguaglianza: una variabile (array e object esclusi) viene confrontata con dei valori predefiniti così da eseguire differenti blocchi di codice a seconda del valore assunto dalla variabile:

```
$mese=date('n');
switch ($mese) {
 case 1:
    echo “Siamo in Gennaio”;
    break;
 case 2:
    echo “Siamo in Febbraio”;
    break;
 case 3:
    echo “Siamo in Marzo”;
    break;
 .....
 }
```

Lo stesso risultato si sarebbe raggiunto con:

```
 $mese=date('n');
 if ($mese==1) {
    echo “Siamo in Gennaio”;
 }
 if ($mese==2) {
    echo “Siamo in Febbraio”;
 }
 if ($mese==3) {
        echo “Siamo in Marzo”;
 }
 .....
```

Dove sta la differenza? Nel primo caso la variabile $mese è valutata all'inizio del costrutto e poi il valore è solo confrontato con quelli delle istruzioni case, mentre nel caso di **if** la variabile $mese è valutata e verificata ogni volta, con uno spreco di risorse e tempo di elaborazione.

Notate la parola chiave *break*: va inserita al termine di ogni blocco di codice case, altrimenti il parser continuerebbe leggendo il blocco successivo invece di uscire dal costrutto.

È possibile definire un case con valore default: il codice sarà eseguito quando tutte le altre condizioni case si riveleranno false.

Isset

Abbiamo spesso la necessità di controllare, prima di compiere un'azione, se una variabile è definita (anche se magari con valore nullo) oppure se non lo è.

Questo controllo è assolto semplicemente da isset in combinazione con if: isset, infatti, restituisce un valore true se la variabile cui fa riferimento esiste:

```
 if (isset($variabile)) {
    codice
 }
```

Possiamo usare isset per migliorare l'esempio in cui modificavamo la visualizzazione di un articolo. ▷

LISTATO 21

```
<style>
 span.output {
    font-family:
    <?php
       if (isset($_POST['carattere'])) {
          echo $_POST['carattere'];
          // valido solo se è stato sottoscritto il form e quindi $_POST esiste
       }
       else {
          echo "Arial";
          // è la visualizzazione di default quando $_POST non esiste,
          ossia non è mai stato cliccato il bottone "invia" del form
       }
    ?>;
 }
 </style>
```

LISTATO 22

```
$lista=array ("pippo", "topolino", "paperino", "pluto");
 foreach ($lista as $copia) {
    echo "Un personaggio Disney: $copia<br>";
 }
```

Si può utilizzare foreach anche per definire un secondo array contenente le chiavi dell'array originale:

```
 $lista=array ("pippo", "topolino", "paperino", "pluto");
 foreach ($lista as $chiave=>$copia) {
    echo "Personaggio Disney numero $chiave -> $copia<br>";
 }
```

LISTATO 23

```
if (!isset($_POST['nome'])) {
 // serve solo alla prima chiamata della pagina
       die ("<p><h3>Tutti i campi vanno obbligatoriamente compilati</h3>");
    }
 foreach ($_POST as $ctr) {
 // foreach ci consente di controllare tutto l'array $_POST
    if (trim($ctr)==""){
    // trim toglie gli spazi vuoti eventualmente presenti in $ctr
       die ("<p><h3>Tutti i campi vanno obbligatoriamente compilati</h3>");
    }
 }
```

Invece di usare un form intermedio di passaggio (tra l'altro poco elegante, dal momento che il nostro utente potrebbe accedere direttamente alla pagina *articolo.php* una volta che ne conosca l'URL), potremmo definire dei valori di default validi finché non viene scelto un diverso carattere dal form cliccando su "invia".

Il controllo della definizione iniziale delle variabili è demandato proprio a isset come esemplificato in *articolo-due.php* (*listato 21*):

Foreach

Foreach, un po' come isset, ci aiuta a risparmiare tempo per compiere un'azione di cui spesso si ha bisogno: attraversare completamente un array, effettuando delle operazioni (assegnazione, confronto, visualizzazione, ...) su un secondo array "copia" appositamente creato.

La funzione più importante la si rileva quando bisogna operare su un array completo, anche solo per visualizzare tutti gli elementi.

Per fare questo potremmo usare un ciclo for (ma dobbiamo sapere quanti elementi compongono l'array) o un ciclo while (con una struttura non molto amichevole). Oppure un semplice ciclo foreach (pagina *lista-disney.php*, *listato 22*):

Un altro esempio concreto lo si ha in occasione della validazione dei dati passati attraverso un form: se vogliamo porre tutti i campi di un form come obbligatori, dovremo controllare tutti i campi per verificare che sia stato immesso un valore, inserendo quindi una serie di istruzioni if e/o isset. Foreach ci viene in aiuto per rendere la scrittura più agile e per evitare banali dimenticanze, come si vede dalla pagina *check.php* (*listato 23*):

Notate, nel ciclo if, la funzione **trim** il cui scopo è eliminare tutti gli spazi vuoti all'inizio di un campo: in questo modo se un utente compilasse il form solo con degli spazi vuoti, il nostro test comunque non ne validerebbe l'inserimento (una funzione if semplice, invece, l'avrebbe fatto passare).

Un controllo di questo tipo è sufficiente per la maggior parte delle nostre esigenze, come vedremo nella prossima puntata del corso.

Introduciamo qui anche la funzione **die** che interrompe l'esecuzione della pagina visualizzando come output il codice chiuso dalle parentesi.

Lo stesso compito era assolto anche da exit, vista in uno degli esempi precedenti.

Siti di riferimento

Se volete avere ulteriori approfondimenti su PHP potete visitare questi siti:

http://www.php.net
http://freephp.html.it
http://www.apache.org

Se state ricercando un suggerimento su una qualsiasi funzione di PHP, potete accedere velocemente alle pagine del manuale on-line scrivendo l'URL del sito seguito da uno slash con la parola da ricercare, ad esempio http://www.php.net/isset (sul manuale sarà cercata la parola isset).

È utile farlo perché, oltre alla descrizione del manuale, si trovano anche molti esempi pratici scritti dagli utilizzatori di PHP. ■

I corsi Webmaster disponibili nel CD

Nel *CD Guida 2* allegato a questo numero di *PC Open*, all'interno della cartella *PDF/Corsi*, trovate due corsi completi che possono essere un utile complemento al corso PHP. Uno è il corso *Web Developer ASP*, 97 pagine suddivise in quattro lezioni per capire come realizzare siti dinamici in tecnologia ASP.

Il corso *Webmaster* spiega, invece, in 88 pagine suddivise in otto lezioni tutto quello che bisogna sapere per costruire un sito e imparare il linguaggio HTML 4.01, i CSS (fogli di stile), Java Script e CGI. Il corso è completato da utili consigli per promuovere il proprio sito on line.

▶ A scuola con PC Open

Web Developer PHP

di Federico Pozzato

1 Approfondire PHP

Nella lezione 1 abbiamo esplorato le basi di PHP, dando tutti gli elementi per creare le nostre prime pagine e alcuni spunti per cogliere le potenzialità di questo linguaggio.

Nella seconda lezione metteremo in luce, invece, funzionalità avanzate che ci consentiranno di migliorare i nostri script, rendendoli più efficienti e utilizzabili anche per il futuro. Il discorso sarà poi concluso nella terza puntata, dove verranno trattate anche tematiche relative alla sicurezza e alla gestione degli errori.

Non parleremo ancora della connessione tra PHP e i database (con particolare riguardo a MySQL), in quanto questo argomento sarà oggetto di trattazione specifica e approfondita nelle prossime puntate del corso.

IL CALENDARIO DELLE LEZIONI

Lezione 1:
- PHP con un server off line
- Funzioni base e variabili
- I costrutti di controllo

▶ **Lezione 2:**
- **Approfondiamo PHP**
- **Include e require**
- **Funzioni e classi**
- **Proteggere una pagina**
- **Cookie e sessioni**
- **La funzione mail**

Le prossime puntate

Lezione 3: PHP e database
Lezione 4: PHP e MySQL
Lezione 5: Gestire un sito dinamico con PHP e MySQL

2 Include e require

I listati completi sono sul CD

Spesso capita di inserire in più pagine Web esattamente lo stesso codice: un esempio tipico sono le righe che formano la testata (*header*) o il piè di pagina (*footer*), ma possiamo prendere in considerazione anche le righe che compongono le barre di navigazione e dei menu, le chiamate ai database, la protezione di pagine con password, la chiamata dei CSS esterni e altro ancora.

Come agiamo in questi casi? Usando il "tradizionale" HTML, non abbiamo molta scelta: ci dobbiamo armare di pazienza ed effettuare una serie di copia-incolla di codice su pagine diverse! A parte gli errori sempre in agguato nell'operazione del copia-incolla, un grosso problema nascerà con la necessità di modificare una parte qualsiasi di questo codice ripetuto: prima o dopo, inutile negarlo, avremo bisogno di cambiare qualcosa, vuoi per migliorare le funzionalità e la navigazione, vuoi per aggiornamenti e refresh. Di fronte a questa esigenza dovremo nuovamente armarci di pazienza (molta più di quella che ci era servita quando abbiamo realizzato le pagine), aprire tutti i file interessati ed effettuare la modifica. L'operazione può essere più o meno agevole a seconda del tipo di cambiamento e del software utilizzato, e può divenire un ostacolo insormontabile se i listati delle pagine sono stati creati da terzi. Alcuni programmi, come Home Site, ci vengono in aiuto con una utilissima funzione che consente di operare il replace su più file contemporaneamente, però questo non è sufficiente.

Leggendo queste righe immaginiamo che qualcuno abbia pensato ai fogli di stile (CSS, *Cascading Style Sheet*) e ai problemi di modifica del layout di un sito Web. Prima dell'introduzione dei CSS, infatti, modificare gli attributi di struttura (anche solo la dimensione di una font) su tutte le pagine di un sito Web era un lavoro improbo, mentre adesso basta semplicemente cambiare un unico attributo in un CSS esterno e il gioco è fatto.

Include e **require** in PHP funzionano allo stesso modo di un CSS esterno: nel punto voluto del listato Web possiamo inserire una chiamata a un file esterno il cui contenuto sarà "trasferito" alla pagina chiamante e interpretato dal parser PHP come appartenente alla pagina di origine. Ogni eventuale modifica andrà effettuata, quindi, solo sul file esterno e automaticamente verrà estesa a tutte le pagine interessate.

PHP ci propone due funzioni simili: *include* e *require*. Si inseriscono in maniera estremamente semplice nella pagina chiamante, semplicemente indicando il nome (col percorso relativo) del file voluto:

```
<?php
   include ("testata.inc.php");
?>

<?php
   require ("testata.inc.php");
?>
```

La differenza tra le due funzioni è nella gestione degli errori: in caso di errore, *include* produce solo un avvertimento (*warning*) senza bloccare la pagina, mentre *require* blocca la compilazione del linguaggio (*fatal error*). Che cosa usare, quindi, è funzione degli obiettivi del Webmaster.

Un esempio semplice del vantaggio di usare gli include/require file lo possiamo ricavare direttamente dalla quotidianità. Immaginiamo di aver creato un sito dove sia necessario inserire in molte pagine un disclaimer a tutela del trattamento dei dati personali, come previsto dalle norme sulla privacy. Essendo Webmaster previdenti abbiamo incluso in ogni pagina interessata (come *privacy.php*, *vedi listato_01*) un file esterno col testo del disclaimer (*disclaimer.inc.php*, *vedi listato 2*):

```
<html>
   <head>
      <title>Privacy</title>
   </head>
```

```
 <body>
 <h3>Questa parte di testo è inserita
direttamente nella pagina
privacy.php</h3>
bla bla bla bla bla bla bla
<p>
<h3>Quella che segue, invece, è
inserita nella pagina disclaimer.inc.php
ed è visualizzata in questa pagina
utilizzando un'istruzione include:</h3>
<?php
  include("disclaimer.inc.php");
?>
  </body>
</html>
```

listato 1

```
<em>
Ai sensi della L. 675/96 ti informiamo
che tutti i dati in nostro possesso
verranno utilizzati solo per
pubblicizzare iniziative inerenti XYZWK.
Nessun dato verrà ceduto a terzi. In
ogni momento potrai chiedere di
essere cancellato dal nostro database
con una semplice comunicazione al
nostro indirizzo di posta elettronica.
</em>
```

listato 2

Quando il browser chiamerà *privacy.php*, l'effetto sarà di vedere riportato il testo del disclaimer come se appartenesse alla pagina originale *(immagine 1)*. Il vantaggio di questa struttura lo avremmo tangibilmente verificato all'inizio del 2004 quando è entrata in vigore la nuova norma di disciplina del settore (dalla Legge 675/96 al D.Lgs.196/03): invece di modificare tutte le pagine, l'unica operazione di modifica l'avremmo compiuta solo su *disclaimer.inc.php* semplicemente modificando il riferimento della Norma (vedi l'esempio corrispondente *privacy_new.php* con *disclaimer_new.inc.php*).

Bisogna porre molta attenzione al fatto che le pagine cui fanno riferimento *include* e *require* sono a tutti gli effetti pagine appartenenti al sito e richiamabili col browser. Se inseriamo note particolari in questi file esterni (magari anche password), dobbiamo essere sicuri che non siano visibili nel caso qualcuno ci finisca dentro direttamente. Il consiglio, quindi, è di nominare questi file inclusi come *nomepagina.inc.php*, dove il suffisso *inc* serve a noi per capire al volo che quella pagina è inclusa in un'altra, mentre l'estensione *php* indica al server di trattare la pagina utilizzando il parsing (è ovvio che nel listato dovranno essere presenti i tag PHP).

Se usassimo solo l'estensione *.inc* (o *.html* o *.txt* o nessuna) il browser ci farebbe vedere la pagina come fosse un testo, anche se contenesse i tag PHP. Buona norma è dunque provare ad aprire direttamente col browser tutti i file inclusi e vedere se è tutto OK per la sicurezza dei nostri dati.

Se volete una prova del rischio che si può correre, provate ad aprire la pagina *accesso.inc*: vedrete in maniera del tutto trasparente i dati di accesso a un database, password compresa *(immagine 2)*. Provate poi a chiamare *accesso.inc.php* e vedrete la differenza!

Un esempio pratico

Se il primo esempio serviva solo a entrare nell'argomento, la seconda proposta, invece, può trovare utili applicazioni pratiche nelle pagine del nostro sito: useremo include, gli array e i costrutti di controllo per costruire una barra di navigazione dinamica.

Lo scopo è sapere sempre in quale sezione del sito ci troviamo e dare la possibilità di spostarsi velocemente nelle altre sezioni. La barra verrà inserita come *include* in tutte le pagine: sarà quindi molto flessibile per permettere modifiche future nel caso di ampliamento del progetto.

Per il nostro esempio abbiamo bisogno di una pagina *main.php*, due sottopagine *arg1.php* e *arg2.php* che contengono le sezioni "viaggi e foto" e "curriculum", una pagina *menu.inc.php* (*listato_03*) per la barra di navigazione e un foglio di stile *menu.css* per definire l'aspetto della barra stessa.

La logica da applicare è la seguente:

- inizializziamo un array assegnando a ogni nome di pagina un testo da visualizzare nella barra che verrà creata;
- verifichiamo in quale pagina ci troviamo. Per fare questo utilizziamo una delle tante variabili superglobals di PHP (in questo caso *$_SERVER["PHP_SELF"]*) in unione con la funzione basename che restituisce solo il nome del file eliminando il resto del percorso
- creiamo una struttura di controllo di tipo *foreach-if*: la funzione *foreach* ci consente di navigare lungo tutto l'array creato, mentre *if* si occupa di "distinguere" la pagina attiva (quella dove ci troviamo) dai link cliccabili.

```
<?php
// inizializzo l'array
$naviga = array ("main.php"=>"Home
page", "arg1.php" => "Viaggi e foto",
"arg2.php" => "Curriculum");

// verifico in quale pagina mi trovo
$nome_file =
basename($_SERVER["PHP_SELF"]);
echo "<table><tr align='center'><td
class='lato'> </td>";

// struttura di controllo
foreach ($naviga as $chiave=>$menu)
{
echo "<td ";
if ($chiave==$nome_file) {
echo " class='nolink'>".$menu;
}
else {
echo " class='link'><a class='nav'
href='".$chiave."'>$menu</a>";
}
echo "</td>";
}
echo "</td><td
class='lato'> </td></tr></table
>";
?>
```

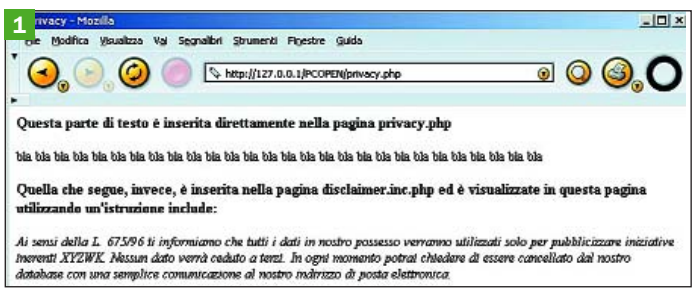
listato 3

È chiaro che, se aggiungo altre pagine, l'unica operazione da compiere sarà aggiungere la nuova pagina all'array *$naviga* nel file incluso *menu.inc.php*.

Il risultato ottenuto (uno dei tanti possibili a seconda di come definiamo il foglio di stile) è visibile nell'*immagine 3*: la pagina dove ci troviamo è visibile con sfondo verde e non è cliccabile, mentre lo sono le altre pagine. Fate clic per vedere cosa succede entrando nelle altre pagine.

La funzione *basename* consente anche di estrarre la radice del nome della pagina, eliminando l'estensione. Per fare questo, l'estensione da eliminare va indicata come secondo argomento di basename:

```
<?php
$nome_file =
basename($_SERVER["PHP_SELF"],
".php");
?>
```



Include e require *facilitano l'inserimento dei file di testo uguali in numerose pagine*

***Nella pagina* accesso.inc** *sono trasparenti anche i dati riservati*

La pagina è visibile con fondo verde *e non è cliccabile, ma lo sono le altre pagine*

3 Funzioni e classi

Spesso si scrivono righe di codice uguali per effettuare operazioni ripetitive e comuni a più pagine. Esempi possono essere gli script per la creazione di qualche menu, ma anche listati molto più semplici con i quali si vuole magari calcolare l'area di un triangolo o restituire un valore vero-falso dopo una verifica.

Una soluzione semplice e banale può essere usare la classica accoppiata copia-incolla, soluzione valida, però, solo se si sa esattamente dove recuperare il codice da copiare. Potremmo riuscire a risolvere velocemente il nostro problema, aprendo comunque la strada a possibili errori: copiatura errata del codice, difficoltà ad adattare il nome delle variabili con possibili dimenticanze sempre in agguato, propagazione di eventuali errori contenuti nelle righe originali, difficoltà nell'individuazione del codice che potrebbe essere diversamente incapsulato nelle pagine.

Per venire incontro alle esigenze dei Webmaster, PHP ci aiuta con funzioni e classi.

Le funzioni

Le funzioni sono script di codice da invocare e da cui ci si attende una risposta:

```
function esempio (eventuali
argomenti)
{
   codice
   return risultato restituito
}
```

La funzione deve avere un nome univoco (attenzione: non possiamo utilizzare i nomi riservati previsti da PHP, come ad esempio *print*) e va inserita (sembra una cosa ovvia, ma non sempre è rispettata) in un punto del listato precedente alla chiamata. Il risultato viene restituito utilizzando l'istruzione *return*: chiamata all'interno di una funzione, *return* termina immediatamente l'esecuzione della funzione corrente e restituisce il suo argomento come valore della funzione. Possiamo inserire più istruzioni *return* all'interno della stessa funzione, tenendo presente che, appena incontrato il *return*, il codice proseguirà al di fuori della funzione.

Finora abbiamo visto moltissime funzioni predefinite di PHP: *basename*, ad esempio, è una di queste. Basename, infatti, necessita di alcuni argomenti (il percorso della pagina e un'eventuale estensione) e restituisce di conseguenza il valore atteso, ossia il nome della pagina con o senza estensione.

Una funzione può restituire anche solo un valore vero-falso (*true-false*), da usare magari in combinazione con un *if* per creare un ciclo di controllo, o un qualsiasi altro tipo di variabile (valori, array, oggetti).

Usare le funzioni consente di concentrarsi sul codice puro ("astratto") della funzione, senza doversi preoccupare del significato delle variabili all'esterno della funzione stessa. Una volta scritta, la funzione sarà sempre utilizzabile e non dovremo neppure preoccuparci di reinterpretare il codice per adattarlo alle nuove variabili. Spesso passa molto tempo tra quando si scrive del codice e quando poi lo si riutilizza: anche in questo caso, con le funzioni evitiamo di doverci rinfrescare la memoria ricompiendo i passaggi logici del passato.

Basename è registrata nelle librerie di PHP ed è valida universalmente per tutti gli utilizzatori. Nel nostro piccolo, però, anche noi abbiamo bisogno di semplificarci la vita e quindi decidiamo di scrivere la nostra prima funzione *(listato 4)* per calcolare il prezzo netto di un bene, dato uno sconto espresso come valore da 0 a 100:

```
function prezzo_netto($prezzo,
$sconto)
{
if ($sconto<0 or $sconto>100) {
return "Questa funzione assume che lo
sconto debba essere un numero
compreso tra 0 e 100, mentre tu hai
inserito uno sconto uguale a $sconto";
}
$netto=$prezzo*(1-$sconto/100);
return $netto;
}
```

listato 4

Questa funzione può essere usata dovunque ce ne sia bisogno: l'applicazione pratica la si vede nella pagina *sconto.php* che contiene un form con due campi dove inserire i valori di prezzo e sconto sulla base dei quali calcolare il prezzo netto. Da notare è la chiamata della funzione *prezzo_netto* nella pagina *sconto.php*:

```
echo "Prezzo netto: ".prezzo_netto
($_POST['pre'],$_POST['sco']);
```

Come si vede, le variabili passate come argomento della funzione hanno un nome diverso da quello assegnato nello script della funzione, ma la cosa non ha la minima importanza perché questo è il senso delle funzioni: PHP, infatti, interpreta la prima variabile passata come fosse la variabile *$prezzo originale* e la seconda variabile come fosse la *$sconto originale*.

Va sottolineato che le variabili usate all'interno della funzione non possono essere richiamate all'esterno, quindi qualsiasi invocazione a *$prezzo* o *$sconto* nel corso del listato non produrrà alcun risultato (salvo, ovviamente, non esistano nel corpo della pagina due variabili con questo nome).

Se volessimo questa possibilità dovremmo definire le variabili della funzione come *$GLOBALS* (col loro nome) all'interno dello script della funzione stessa. L'esempio è riportato nella pagina *sconto_global.php* dove, all'interno della dichiarazione della funzione *prezzo_netto* è presente l'istruzione:

```
$GLOBALS["sconto"] = $sconto;
```

Adesso sarà possibile chiamare la variabile *$sconto* anche all'esterno della funzione, mentre non si potrà chiamare *$prezzo* perché non è stata dichiarata globale.

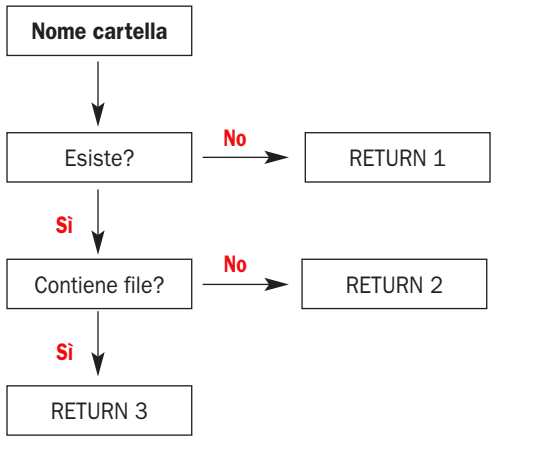
Possiamo definire anche fUnzioni senza argomenti: è il caso delle funzioni che effettuano dei controlli ripetitivi e restituiscono una stringa o dei valori true/false.

La funzione *controllo()*, proposta come esempio, verifica per prima cosa se esiste una cartella *varie* posta allo stesso livello della pagina Web su cui viene inserita. Se la cartella esiste, viene poi verificato se è vuota o se contiene dei file. La logica che sta alla base della funzione è esprimibile in forma di algoritmo, come si può vedere nell'*illustrazione 4*. Per funzioni semplici, disegnare un algoritmo può essere solo una perdita di tempo, ma è essenziale per funzioni più complesse ed è quindi buona abitudine imparare a usarli.

La funzione è riportata nel *listato 5*, mentre la pagina Web di esempio è la *checkvarie.php*. Provate a creare e cancellare la cartella *varie*, lasciandola vuota o inserendo qualche file.

```
function controllo()
// questa funzione controlla che in
una directory predefinita sia presente
```

4



Un algoritmo esprime la logica *che sta alla base della funzione* controllo ()

```
almeno un file
{
$cartella=@opendir('varie');
if (!$cartella) {
return "Attenzione: è inutile cercare
file... la cartella non esiste e quindi
devi prima crearla";
}
while (false !== ($file =
readdir($cartella))) {
$lista[]=$file;
}
$fine=count($lista)-2;
if ($fine==0){
return "Attenzione: Non ci sono file
nella cartella";
}
return “Prosegui pure: nella cartella
c'è almeno un file”;
}
```

listato 5

Per rendere più utilizzabile questa funzione, si può passare alla funzione *un argomento uguale al percorso della cartella da controllare*. La modifica è facile (la trovate in *checkvarie_plus.php*): basta definire la **funzione controllo** (*$directory*) e inserire *$directory* all'interno della funzione come argomento di *opendir*.

Ulteriore esercizio è prendere il percorso della cartella da controllare direttamente da un form: provate a costruire la pagina da soli e poi confrontatela con *checkvarie_form.php (vedi immagine 5)*. Adesso potete controllare tutte le cartelle che volete, l'unica accortezza è ricordarsi che il percorso della cartella è relativo e non assoluto.

Abbiamo detto all'inizio del paragrafo che la funzione originale deve essere presente nel codice prima di un'eventuale chiamata. Potremmo pensare di inserire (copia-incolla) la funzione in tutte le pagine che ci interessano, ed è una valida idea se le pagine che usano questa funzione sono poche, altrimenti ricadiamo nei problemi già descritti in precedenza. Una soluzione più efficiente (ed elegante) consiste, invece, nello salvare le funzioni in file separati includendoli poi come già visto. Ogni modifica fatta sulla funzione verrà quindi propagata automaticamente a tutte le pagine:

```
<?php
include (“funzione_controllo.inc.php”);
?>
```

Non è possibile, per chiarezza di comportamento del codice, ridefinire funzioni già dichiarate all'interno di uno stesso listato (overloading).

Le classi

Le funzioni sono costrutti semplici, adatti a lavori ripetitivi da cui ci si aspetta un unico risultato.

Non sempre ciò è sufficiente, basti pensare al caso di costruzioni di codice avanzato in grado di interagire con altre funzioni e variabili: un esempio può essere la creazione di un sistema di gestione avanzato di commercio elettronico, magari da esportare su più siti. In questo caso un'architettura demandata a funzioni stand-alone è possibile, ma diventa sempre più difficile da gestire all'aumentare della complessità richiesta.

Per esigenze di questo tipo ci viene in aiuto il concetto di **classe**, grazie al quale potremo addentrarci brevemente nella programmazione rivolta agli oggetti. La trattazione non può essere esaustiva perché l'argomento prevede conoscenze specialistiche, ma la cosa importante è riuscire a cogliere il concetto alla base della programmazione OO (**Object Oriented**) per poterlo poi approfondire con testi o corsi specifici.

Orientarsi agli oggetti significa porre l'accento sulla definizione di tutte le proprietà generiche di un certo oggetto e delle funzioni legate a queste proprietà. Lo faccio senza pensare all'oggetto specifico X o all'oggetto specifico Y, bensì guardando al concetto più astratto possibile grazie al quale posso poi definire gli oggetti specifici.

Una classe è, quindi, una collezione di variabili (proprietà) e funzioni (metodi) che utilizzano queste variabili:

```
class Nuova_classe
{
variabili (proprietà)
funzioni (metodi)
}
```

Definita una classe come vedremo in seguito, possiamo generare un oggetto utilizzando il costrutto new:

```
$oggetto = new Nuova_classe;
```

È il codice della classe che “definisce” l'oggetto, il quale deve essere creato, tramite l'istruzione *new*, con un nome diverso da quello di altri oggetti istanziati (cioè inizializzati) dalla stessa classe. Per il concetto stesso che sta alla base della programmazione OO, infatti, è possibile definire quanti oggetti si vuole basandoli su una stessa classe. Ricordate la scorsa puntata quando parlavamo dei tipi di variabili? Mancava solo il tipo *object*, ossia proprio quello definito usando la classe.

Definito un oggetto, a esso è possibile applicare tutti i metodi implementati nella classe di appartenenza, in maniera molto semplice e con una pulizia di codice inarrivabile per una programmazione standard.

Facciamo un esempio tratto dalla vita di tutti i giorni: se devo disegnare un rombo grande rosso e un rombo piccolo giallo, sostanzialmente devo fare due operazioni simili.

Qual è la differenza tra i due “oggetti” da disegnare? Solo la dimensione e il colore, dal momento che la definizione di rombo è unica! Bene, allora è sufficiente che sia dichiarata (una volta per tutte) una “classe” Rombo all'interno della quale siano introdotte le variabili dimensioni e colore che consentono di effettuare il disegno.

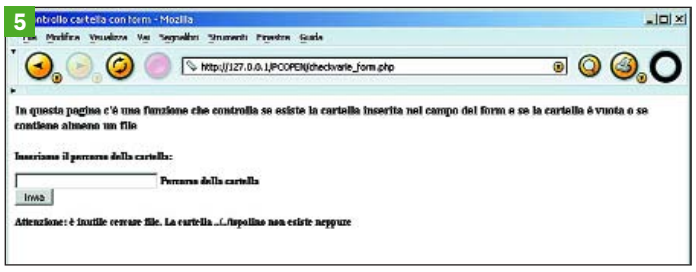
Alla classe non interessa il tipo di rombo da disegnare, ma solo definire che cos'è un rombo colorato. A questo punto per risolvere il nostro compito non ci resta che creare due “oggetti” facenti riferimento a questa classe, inizializzando semplicemente i due oggetti con le loro caratteristiche. Siamo in grado di creare infiniti rombi e, fondamentale, siamo in grado di farlo pur non sapendo nulla di cosa sia un rombo! Forniti i dati di colore e dimensioni, infatti, è la classe che si incarica di restituire gli oggetti in maniera del tutto trasparente per l'utente.

Vediamo di fare un esempio di programmazione di classe: supponiamo di creare una classe Quadrato grazie alla quale, una volta definito il lato, possano essere ricavati i valori di area, perimetro e diagonale. La definizione completa della classe la trovate in *classe_quadrato.inc.php*, mentre qui ci basta un estratto (*listato_06*) con solo il metodo “area”.

```
class quadrato
{
// dichiarazione delle variabili
utilizzate (da accompagnare con una
descrizione, per rendere più leggibile il
codice)
var $lat;
var $ar;
var $per;
var $diago;

function quadrato($lato)
// costruttore
{
   $this->lat=$lato;
}

function area()
// calcola l'area del quadrato
{
   $this->ar=$this->lat*$this->lat;
   return $this->ar;
}
}
```

listato 6

Ci sono due cose da notare: quando è definita una funzione con lo stesso nome della classe, questa funzione prende il nome di **costruttore**. Il costruttore è utile in certe classi perché inizializza il tipo di oggetto della classe stessa. In questo caso il costruttore serve a inizializzare l'oggetto col valore del lato del quadrato. Il costruttore può anche non essere indicato, ma se lo è sarà interno alla classe e avrà necessariamente il nome della classe stessa. ▷



Il risultato in pagina *della funzione controllo*

La seconda notazione importante è il modo di passare il nome delle variabili all'interno della classe: abbiamo detto che ogni classe può creare N oggetti basati su di essa, ma sempre con nome diverso. Non potendo a priori conoscere il nome dell'oggetto, all'interno della classe dovremo usare il costrutto *$this->* generico:

```
$this->nome_variabile
```

In questo modo tutte le variabili definite nella classe saranno disponibili per ogni oggetto creato in seguito.

Un esempio completo è riportato nella pagina *classe.php*, dove possiamo vedere come inizializzare gli oggetti e come chiamare variabili e funzioni. Un oggetto va istanziato (inizializzato) usando il costrutto *new*:

```
$a=5;
$nuovo_quadrato_1=new quadrato($a);
```

Creato l'oggetto *$nuovo_quadrato_1*, per chiamare le funzioni o le variabili useremo la notazione che già conosciamo (->), ricordandoci di indicare sempre il nome dell'oggetto appena creato *(vedi listato 7)*:

```
// creo un nuovo oggetto della classe quadrato con lato 5
$a=5;
echo "<h3>Quadrato_1:<p></h3>";
$nuovo_quadrato_1=new quadrato($a);

echo "<h5>Lato: ".$nuovo_quadrato_1->lat."<p>";
// per chiamare la funzione “area” e per ritornare il valore dell'area:
echo "Area: ".$nuovo_quadrato_1->area()."<p>";
// per chiamare la funzione “perimetro” e per ritornare il valore del perimetro:
echo "Perimetro: ".$nuovo_quadrato_1->perimetro()."<p>";
// per chiamare la funzione “diagonale” e per ritornare il valore della diagonale
echo "Diagonale: ".$nuovo_quadrato_1->diagonale()."<p></h5>";
```

listato 7

Questo esempio di classe è banale, ma permette di cogliere l'essenza della programmazione OO: definite proprietà e metodi di una classe sarà possibile, infatti, creare e gestire oggetti in maniera semplice e senza preoccuparsi di tutto quello che c'è nella classe.

Con una ricerca su Internet si trovano numerose classi pronte da usare. Una, ad esempio, serve a superare il limite del *timestamp*: non occorre capire cosa c'è sotto, ma solo sapere come istanziare l'oggetto e quali funzioni chiamare.

In *classe.php* abbiamo inserito due oggetti diversi (due quadrati di lato diverso) basati sulla stessa classe (*$nuovo_ quadrato_1* e *$nuovo_quadrato_2*): le chiamate a variabili e funzioni differiscono solo ed esclusivamente per il nome dell'oggetto, e ci consentono di ottenere facilmente tutti i dati utili di un “oggetto” quadrato.

Lasciamo per ultima una considerazione ormai ovvia: la classe va naturalmente “inclusa” nella pagina chiamante e posta prima della creazione di qualsiasi oggetto.

Una classe può essere anche la base di un'altra classe che ne estende le funzionalità. In questo caso nella seconda classe indicheremo solo il codice nuovo senza doverci preoccupare di funzioni e variabili già costruite nella prima classe. Il costrutto da cercare nel manuale di *php.net* è *extends*:

```
class quadrato_plus extends quadrato
{
  codice
}
```

4 Proteggere una pagina con password

Con la definizione di funzioni e classi abbiamo esaurito le nozioni fondamentali di PHP.

Arrivati a questo punto non dovrebbe quindi essere un problema affrontare un compito molto pratico (e utile), ossia elaborare qualche strategia per proteggere una o più pagine del nostro sito Web da occhi indiscreti. In pratica, vogliamo che a certe pagine possano avere accesso solo le persone dotate di una password, o meglio di una precisa accoppiata username-password.

I motivi di questa decisione possono essere i più vari: su una pagina possiamo avere inserito testi e immagini che solo i nostri amici più cari devono leggere, ma potrebbe anche essere una esigenza di lavoro che ci impone di tutelare certe informazioni, oppure una parte del Web site deve essere protetta perché da essa è possibile gestire gli aggiornamenti del sito stesso.

Qualunque sia il motivo, abbiamo la necessità di proteggere l'accesso a una pagina. Vediamo adesso un paio di esempi, rimandando poi l'approfondimento del discorso a dopo l'introduzione dei database (*lezione 4*).

PHP ci consente di ottenere la protezione di una pagina in molti modi. Quello scelto per esercizio implica la creazione di una funzione e l'uso di un ciclo di controllo (*listato_08*). La pagina di esempio è *protetta_1.php*, e al primo accesso ci fa vedere un form dove l'unico campo da compilare è quello relativo alla password da inserire.

La funzione controllo si occupa solo di restituire un valore *true* o *false*, a seconda che il confronto tra le due variabili dia esito positivo o negativo. Nel nostro caso, il valore restituito dalla funzione sarà assegnato a una variabile *$check,* la quale chiama la funzione usando come campi la password inserita nel form e la password definita da noi (*pluto*). Se *$check* avrà valore *true* allora la pagina sarà visualizzata, altrimenti ci troveremo di nuovo di fronte al form vuoto.

```
<?php

function controllo ($confronto, $pass) {
if ($confronto==$pass) {
return true;
}
return false;
}

$password=”pluto”;
// la password “pluto” l'abbiamo impostata noi

$check=controllo($_POST['pwd'], $password);
// $check assumerà valore true o false in dipendenza del valore inserito nel form

if ($check==false) {
// la password è errata o non è mai stata inserita
?>
<h3>Inserisci la password corretta per entrare nella pagina protetta</h3>
<form action="protetta_1.php" method="post">
<input type="password" name="pwd"> = password<p></p>
<input type="submit" name="invio" value="Connetti">
</form>
<?php
}
else {
// la password è esatta e quindi posso avere accesso alla pagina
  echo "<h3>Hai inserito la password giusta. La pagina è a tua disposizione</h3>";
}
?>
```

listato 8

Una variazione sullo stesso tema (meno elegante, ma ugualmente efficace) la potete vedere in *protetta_1_var.php*.

Per migliorare la costruzione presentata, bisogna quantomeno posizionare in un file esterno la funzione e il valore assegnato da noi alla password (vedere *password.inc.php* e *protetta_1_inc.php*).

In questo modo è possibile usare il costrutto *include* per inserire senza errori la stessa password in più pagine e se si modifica la password lo si fa in un unico punto.

Un altro limite, però, è proprio quello di avere una sola password per tutte le persone che hanno accesso alle pagine: se dovessimo cambiarla per motivi di sicurezza (poniamo che sbadatamente qualcuno abbia comunicato la password a persone non autorizzate) dovremo informare necessariamente tutti gli attori coinvolti.

Sarebbe invece più efficiente assegnare a ogni persona una diversa coppia username-password, in modo da modificare, eventualmente, soltanto la password di quella persona specifica.

Vediamo allora di modificare il codice precedente, introducendo un controllo username-password.

Usiamo la stessa logica vista nel primo esempio, con la novità di creare un array contenente le coppie username-password.

La parte di codice che ci interessa (con la nuova funzione di confronto e con l'array *$coppie)* è visibile in *listato_09*:

```
function controllo_coppia($confronto,
$user,$pass) {
foreach ($confronto as $chiave=>$val)
{
if ($chiave==$user and $val==$pass) {
return true;
}
}
}
return false;
}

// valori di username e password
decisi da noi
$coppie=array("pippo"=>"pluto",
"topolino"=>"minni",
"paperino"=>"paperina");

// $check darà un valore true o false
$check=controllo_coppia($coppie,$_
POST['user'],$_POST['pwd']);
```
listato_09

La parte che segue il listato 9 è uguale a quanto già visto e l'esempio completo è riportato sulla pagina *protetta_2.php*. Adesso siamo in grado di modificare username e password delle singole coppie.

Anche in questo caso, poi, sarà buona consuetudine portare in un file esterno la funzione e la definizione di *$coppie* (vedi pagine *password_2.inc.php* e *protetta_2_inc.php*).

Nel prosieguo del corso vedremo come usare i database per proteggere una pagina e come evitare di dover inserire username e password su ogni pagina protetta.

Ma per questi argomenti l'appuntamento è alle lezioni 4 e 5.

5 Cookie e sessioni

Il protocollo HTTP è un protocollo **stateless**, cioè non dispone di un metodo per mantenere una memoria relativa allo stato della comunicazione tra client e server. Per ovviare a questa mancanza vengono in nostro aiuto i **cookie** e le **sessioni**.

Sappiamo tutti bene cosa siano i cookie: piccoli file registrati sul nostro PC (lato client) e richiamati dalla pagina Web caricata (lato server). Lo scopo è recuperare informazioni precedentemente fornite dall'utente in maniera più o meno trasparente. Se usati coscienziosamente dai Webmaster, i cookie si dimostrano indubbiamente utili quando richiamano dati che altrimenti dovremmo ogni volta settare, però al contempo possono dimostrarsi deleteri se servono, come avviene col "cavallo di troia", per estrarre dal PC informazioni inerenti la nostra privacy e relative a siti da noi visitati, scelte effettuate o simili.

Proprio a causa della pessima pubblicità fatta ai cookie, molti utenti ne disabilitano completamente la registrazione nel proprio PC, o quantomeno la abilitano solo per siti assolutamente fidati.

Assumendo di utilizzare i cookie in senso positivo, dovremo tenere conto di queste possibili limitazioni e agire di conseguenza.

In questa sezione parleremo prima di cookie e poi introdurremo il concetto di sessione, assolutamente da conoscere sia per trasmettere dati all'interno del proprio Web domain che per superare, in alcuni casi, il problema della disabilitazione dei cookie.

Cookie

Poniamo di volere inizializzare un cookie in modo da riconoscere un utente che abbia già visitato il nostro sito. Magari potremmo avergli chiesto nelle precedenti visite quale sezione preferisce, in modo da indirizzarlo direttamente verso quella specifica pagina.

PHP imposta i cookie con la funzione setcookie:

```
setcookie (nome, valore, durata)
// valore e durata sono parametri
opzionali
```

e registra i valori nella variabile globale *$_COOKIE* (ricordate le variabili globali? Ne abbiamo parlato nella scorsa puntata) così da poterla richiamare in un secondo momento. Il nome del cookie è sempre obbligatorio, mentre non lo sono né il valore né la durata.

Lasciando vuoto il campo *valore*, il cookie specificato verrà inizializzato assegnandogli un valore *null*. Anche la durata è opzionale. Se non viene indicato nulla, il cookie sarà attivo solo fino alla chiusura del browser e poi verrà cancellato dal disco fisso del client. Se il nostro scopo è usare i cookie per recuperare i dati di accessi effettuati in momenti diversi, dovremo quindi assegnare una durata (un minuto, un giorno, un mese, un anno...) al "biscottino".

Le istruzioni che coinvolgono i cookie vengono inviate insieme con gli header HTTP (gli header sono le righe di intestazione HTTP che indicano al browser come comportarsi) e vanno pertanto chiamate (peculiarità degli header) prima di qualsiasi output di pagina ossia prima anche del tag *<html>*.

Un'altra importante peculiarità da ricordare è che, aperta una pagina, viene sempre recuperato il valore del cookie eventualmente registrato in precedenza: l'aggiornamento del cookie (o la sua registrazione) verrà letto solo quando la pagina sarà ricaricata successivamente. Questo ci facilita nella scrittura del codice dei nostri listati.

Un esempio è più chiaro di tante parole (vedi *cookie.php*): supponiamo di voler accogliere un navigatore che torna nel nostro sito indicandogli quando ci ha visitati l'ultima volta. Per fare questo dobbiamo innanzi tutto verificare se il navigatore possiede un nostro cookie precedentemente registrato, altrimenti dobbiamo creare il cookie per la prima volta. Il cookie dovrà essere valido sei mesi.

La prima parte da esaminare *(listato 10)* è il codice che precede l'inserimento del tag *<html>*. Con *setcookie* inizializziamo per la prima volta (o aggiorniamo se già esisteva) un cookie di nome *data*: esso contiene il valore *$data* che indica quando la pagina è stata vista l'ultima volta. La durata del cookie è espressa in secondi a partire da oggi, quindi usiamo la funzione *time()* aggiungendo il numero di secondi per arrivare a sei mesi circa.

```
<?php
$data = date("d-m-y")." alle ore
".date("H-i-s");
setcookie("data", $data,
time()+15800000);
?>
```
listato_10

Il codice del *listato 11,* invece, è contenuto tra i tag *<html>* dell'esempio proposto. Il ciclo *if* controlla se il cookie *data* esiste: in caso positivo ci dice quando abbiamo visto la pagina l'ultima volta, in caso negativo ci informa che è la prima volta che accediamo al sito.

Ricordiamo che la lettura del cookie si riferisce sempre allo stato precedente a quello eventualmente settato sulla pagina aperta.

```
<?php
if (isset($_COOKIE['data'])) {
echo "<h4>Ciao amico, hai già visitato
```

```
queste pagine. L'ultima volta che hai
aperto questa pagina è stato il
".$_COOKIE['data']."</h4>";
}
else {
echo "<h4>Ciao amico, è la prima
volta che visualizzi questa pagina
oppure hai cancellato il cookie dalla
memoria del tuo PC</h4>";
}
?>
```

listato 11

Un cookie può essere cancellato assegnandogli un valore *null* in questo modo:

```
setcookie ("data");
```

Sessioni

I cookie, lo abbiamo appena visto, salvano un piccolo file di informazioni all'interno del nostro PC, quindi lato client. Le sessioni, al contrario, sono un metodo lato server per archiviare informazioni relative a un singolo accesso.

Per avere un esempio, possiamo pensare ai siti di commercio elettronico in cui è presente un "carrello della spesa": il compito essenziale per il Webmaster è mantenere registrate le scelte effettuate da un cliente durante un collegamento (l'utente potrebbe acquistare N libri, spostandosi continuamente di pagina in pagina). La sessione serve proprio a questo, a mantenere memoria delle scelte fatte durante un collegamento da ogni singola persona, scelte che poi possono essere memorizzate o cancellate! Si potrebbero anche usare i cookie, ma in questo caso ci sarebbe il problema di cosa fare se sono stati disabilitati.

Con le sessioni, i dati sono registrati nel server che ospita il sito e vengono identificati (e differenziati gli uni dagli altri) tramite un identificativo univoco denominato **SID** (*Session Identifier*). Il SID (casuale e non prevedibile: è formato da 32 caratteri alfanumerici) viene assegnato a ogni utente che si collega al sito e la sessione viene mantenuta attiva (ossia viene mantenuto lo "stato") finché il SID viene passato tra server e client durante la navigazione del sito.

Fondamentale, dunque, è il passaggio del SID, operazione che può avvenire in due modi. Nel primo caso tornano utili i cookie: se sono attivi, il SID viene passato attraverso un cookie temporaneo che viene poi cancellato dal client quando viene chiuso il browser (o quando la sessione è esplicitamente chiusa con un'operazione di logout). Da parte del Webmaster non c'è da fare nulla di particolare.

Il secondo sistema, invece, consiste nel passare il SID attraverso l'URL di una pagina usando i metodi GET o POST, quindi ogni link di pagina sarà del tipo:

```
<a href="nuova pagina.php?<?php
echo SID; ?>">Clicca qui</a>
```

Quale scegliere? Il metodo "cookie" è più semplice dal momento che non bisogna fare nulla, però dovremo preoccuparci di prevedere la trasmissione del SID anche per quegli utenti che hanno disabilitato i cookie, quindi o si fa un controllo per vedere se i cookie di sessione sono attivi (basta verificare se esiste un cookie chiamato *PHPSESSID*) o si trasmette sempre e comunque il SID.

In realtà esiste anche un terzo metodo di passaggio del SID, completamente trasparente per programmatore e utente dal momento che il SID è trasmesso automaticamente. Questa possibilità, però, implica l'attivazione dell'opzione *trans_sid* nel *php.ini*, cosa non sempre possibile se il server non è nostro. Inoltre generalmente l'opzione è disattivata di default e quindi non tratteremo questo caso.

Cosa succede dei file sessione registrati sul server? Quanto tempo restano salvati? La durata di salvataggio dei dati sul server e anche la durata della sessione stessa sono definite nel *php.ini* e quindi dipendono dal gestore del server: questi valori sono visibili visualizzando la pagina *phpinfo* di cui abbiamo parlato nella prima lezione.

È chiaro, altresì, che un file di sessione potrà essere recuperato e riutilizzato in un secondo momento solo se si è tenuta nota del SID, altrimenti non vi si potrà accedere in nessun caso.

Va posto un minimo di attenzione al fatto che i file salvati sul server sono semplici file di testo non criptati: è bene, quindi, salvare nei file di sessione solo dati che possano, nel caso più malaugurato, essere letti da altri senza danni.

Giusto per fare un esempio, se si vuole usare la sessione per mantenere attivo, durante un collegamento, l'accesso a più pagine riservate, è bene non salvare come variabile di sessione un dato sensibile come la password. In ogni caso, niente allarmismi: ricordo che il SID identificante la sessione è composto da 32 caratteri alfanumerici casuali ed è quindi estremamente improbabile che un estraneo possa entrare accidentalmente in una nostra sessione!

Per usare le sessioni, le regole da conoscere sono relativamente poche.

Innanzi tutto, quando un visitatore accede al sito, PHP controllerà (su nostra esplicita richiesta formulata tramite la funzione *session_start*) se uno specifico ID di sessione sia presente.

In caso positivo il precedente ambiente salvato verrà ricreato, altrimenti verrà inizializzata una nuova sessione col suo specifico SID.

Come visto per i cookie, anche *session_start()* dovrà essere chiamata prima di qualsiasi output html. Le variabili di sessione saranno poi richiamate come variabili globali *$_SESSION*, e potranno essere "cancellate" in maniera molto semplice usando l'istruzione *unset*.

La sessione può essere del tutto cancellata con l'istruzione *session_destroy*.

Il carrello della spesa

E ora vediamo un esempio pratico che può dare molti spunti: il carrello della spesa. Supponiamo di avere una pagina all'interno della quale sia possibile scegliere tra libri e dischi da due elenchi (vedi la pagina *lista.php*), e una pagina che funga da riepilogo (vedi *carrello.php*) di quanto scelto durante la sessione in corso, potendo anche tornare in un secondo momento nella lista per ordinare altri prodotti.

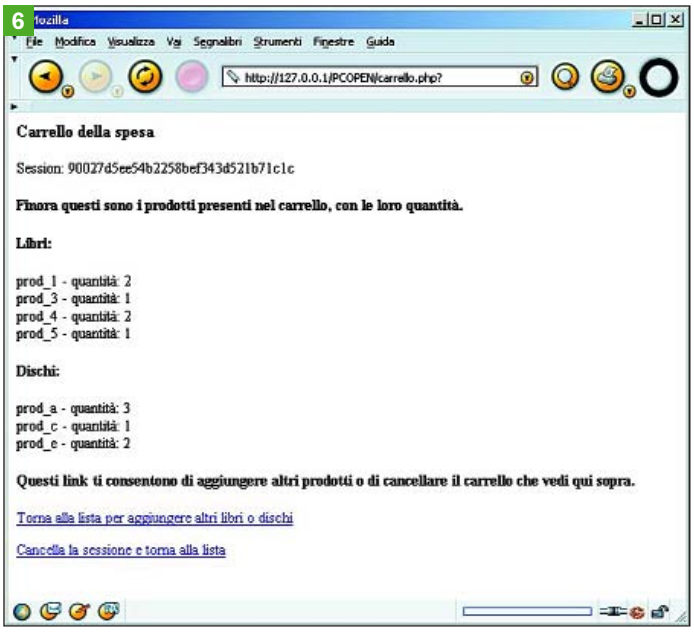
Sulla pagina *lista.php* va notata la chiamata della sessione e la trasmissione del SID all'interno della riga di intestazione del form. In questo modo non ci interessa sapere se un utente abbia o meno i cookie attivati:

```
<form action='carrello.php?<?php
echo SID; ?>' method="post">
```

Fatta la selezione e fatto clic sul pulsante *Vai* ci troviamo nella pagina *carrello.php* dove si vedono le selezioni effettuate fino a quel momento. Ciò è reso possibile dalle variabili di sessione, come si può vedere analizzando l'estratto visibile nel *listato 12*.

```
<?php
session_start();
// controllo se è stato selezionato un
libro
```

6



***La pagina* carrello.php** *presenta le selezioni effettuate dall'utente*

```
if ($_POST['libro']) {

//controllo se il libro selezionato
esiste già nel carrello
if (isset($_SESSION['libro'][$_POST['
libro']])) {
$_SESSION['libro'][$_POST['libro']]++;
}
 else {
 $_SESSION['libro'][$_POST['libro']]=1;
}
}
```

listato 12

Per prima cosa si verifica che un libro sia stato effettivamente selezionato, quindi viene creato (o modificato aumentando la quantità di 1) il vettore *$_SESSION['libro']* che ha una struttura del tipo:

```
$_SESSION['libro'] [libro selezionato] =
quantità
```

Stessa operazione viene fatta per il vettore *$_SESSION['disco']*, quindi i due vettori vengono stampati col classico ciclo *foreach* (dopo averli ordinati usando l'istruzione *ksort*). Il risultato è visibile nell'*immagine 6*.

Per aggiungere altri prodotti basta fare clic sul link di fondo pagina che ci riporta alla lista di selezione, mantenendo memoria dell'identificativo della sessione ed eventualmente trasmettendoli.

Il link che consente di cancellare la sezione ci fa passare attraverso la pagina *logout.php*, che serve unicamente a eliminare tutti i dati della sessione e a reindirizzarci nuovamente alla lista di partenza con l'istruzione:

```
header("Location: lista.php");.
```

Provate ad attivare e disattivare i cookie dal vostro PC per verificare il funzionamento delle sessioni. Un'altra annotazione che può essere utile: di default il nome della sessione è *PHPSESSID*, quindi per stampare le 32 cifre identificative basta chiamare:

```
echo $PHPSESSID;
```

Se volessimo recuperare una vecchia sessione (conoscendo il *PHPSESSID*), basta scrivere l'URL della pagina carrello in questo modo:

```
http://mio.sito/carrello.php?PHPSESS
ID=
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Adesso che questo semplice carrello della spesa è completo, magari potremmo voler spedire l'ordine direttamente via e-mail. Vediamo come fare questa operazione.

La funzione mail

Attenzione: per verificare il funzionamento off-line della funzione mail sul vostro PC deve essere installato anche un server mail. Poiché questo esula dalla presente trattazione, il consiglio è di verificare le pagine descritte caricandole su uno spazio Web on-line dove PHP sia attivato.

PHP comprende una funzione mail, fondamentale per poter creare form attraverso i quali gli utenti possano scriverci, per automatizzare risposte, per gestire mailing, per spedire cartoline elettroniche e chi più ne ha più ne metta.

La funzione preposta a tutti questi usi ha una forma estremamente semplice e compatta:

```
mail (destinatario, oggetto, messaggio,
header e messaggi addizionali)
// il quarto campo è opzionale e può
contenere tutti i vari header che
vediamo analizzando il codice sorgente
di una mail
```

Con questa funzione possiamo simulare in tutto e per tutto quello che facciamo col nostro programma di posta elettronica, quindi possiamo spedire mail di solo testo o con HTML, aggiungere allegati, inserire immagini in linea, spedire la mail a destinatari in *CC* (carbon copy) o *BCC* (blank carbon copy).

In realtà proprio la compattezza della funzione rende alcune delle operazioni sopra descritte alquanto complicate visto che bisogna conoscere la codifica MIME di spedizione delle mail. Fortunatamente in Internet si trovano classi già pronte all'uso per risolvere questi problemi.

Qui impareremo le nozioni fondamentali su come spedire un'e-mail in formato testo. Il caso tipico è quello della spedizione di un'e-mail a seguito della compilazione di un form predisposto su Internet (vedi *form_posta.php*). Il form contiene solo quattro campi da compilare: nome e indirizzo di posta del mittente, soggetto e corpo della mail stessa (*immagine 7*). Cliccando su *Spedisci* si apre una seconda finestra che ci conferma l'avvenuta spedizione della mail.

Nel codice di questa pagina (*spedisci.php*) inseriamo il nostro indirizzo (destinatario) e la funzione mail *(listato 13)*. Notiamo che il mittente va inserito usando un'istruzione *From* posta nel quarto campo della funzione mail (il campo addizionale). La funzione *stripslashes* serve a evitare problemi nel caso in cui chi compilasse il form usasse caratteri particolari come le virgolette, gli apici, le slash.

```
<?php
// destinatario della mail
$to="mioindirizzo@miosito.it";
// soggetto della mail
$subject=stripslashes($_POST['
soggetto']);

// oggetto della mail. In questo punto
il costrutto "\n" serve a far andare il
testo a caporiga nella mail che ci
arriverà al nostro indirizzo
$object=stripslashes($_POST['nome'])
."\n\n";
$object.="Oggetto:
".stripslashes($_POST['testo']);

// mittente della mail
$from="From:".$_POST['email'];

// funzione che spedisce la mail
mail($to,$subject,$object,$from);
?>
```

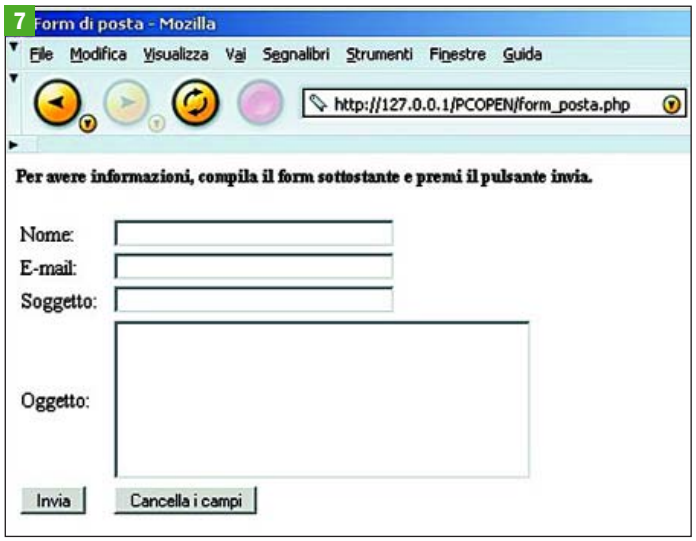
listato_13

Anche eventuali campi da inserire in copia o copia nascosta vanno inseriti nel quarto campo di *mail()*, separandoli con i caratteri speciali *\r\n* che determinano le azioni di "ritorno carrello" e "a capo". Un esempio è presente nel *listato 14*:

```
<?php
$to="mioindirizzo@miosito.it";
$subject=stripslashes($_POST['
soggetto']);
$object=stripslashes($_POST['nome'])
."\n\n";
$object.="Oggetto:
".stripslashes($_POST['testo']);

// inseriamo il mittente nel quarto
campo della funzione
$intestazioni="From:".$_POST['email']
."\r\n";

// aggiungiamo una coppia di mail
($cc_1 e $cc_2) in copia
$intestazioni.="Cc:".$cc_1.",
".$cc_2."\r\n";

// aggiungiamo una mail ($bcc_1) in
copia nascosta
$intestazioni.="Bcc:".$bcc_1."\r\n";

// funzione che spedisce la mail
mail($to,$subject,$object,$intestazioni
);
?>
```

listato 14

Per rendere più efficace la funzione *mail()* si può pensare di inserire nella pagina *spedisci.php* una serie di controlli per evitare che si possano lasciare campi vuoti nel form, e per verificare il corretto inserimento dell'indirizzo mail del mittente. ■



I quattro campi da compilare *del* form_posta.php

▶ A scuola con *PC Open*

Web Developer PHP

di Federico Pozzato

1 PHP e i database

La terza puntata del corso è divisa concettualmente in due sezioni ed è il trâit d'union tra i due oggetti di questa serie di lezioni per Web developer.

Nella prima parte concluderemo il corso di PHP sviluppato nelle due precedenti lezioni, approfondendo gli ultimi aspetti pratici di programmazione e introducendo alcuni suggerimenti generali in termini di gestione degli errori e di sicurezza.

La seconda parte, invece, è dedicata all'introduzione dei concetti fondamentali riguardanti i database, concetti senza i quali non saremo in grado di sfruttarne al meglio le caratteristiche.

Capiremo cosa siano un Database System e un DBMS (Database Management System) e vedremo i principi basilari per la corretta costruzione di una base di dati.

Al termine di questa lezione saremo quindi in grado di padroneggiare PHP e la teoria dei DB, pronti a unire queste due conoscenze nelle due puntate restanti del corso.

IL CALENDARIO DELLE LEZIONI

Lezione 1:
- PHP con un server off line
- Funzioni base e variabili
- I costrutti di controllo

Lezione 2:
- Approfondiamo PHP
- Include e require
- Funzioni e classi
- Proteggere una pagina
- Cookie e sessioni
- La funzione mail

▶**Lezione 3:** PHP e i database
- La funzione upload
- Lavorare con i file
- La gestione degli errori
- Accenni di sicurezza
- DataBase: il database system
- Siti Web dinamici
- Teoria dei database

Le prossime puntate

Lezione 4: PHP e MySQL
Lezione 5: Gestire un sito dinamico con PHP e MySQL

2 Upload di file

Nella prima lezione abbiamo visto come installare un server locale per poter testare off line in tutta calma le pagine PHP prima di caricarle sul nostro spazio Web.

L'accesso FTP del provider

Per compiere questa operazione il provider ci avrà assegnato un accesso FTP (File Transfer Protocol) protetto da username e password, accesso grazie al quale saremo in grado di spostare, aggiungere, cancellare, rinominare file direttamente sul server remoto come stessimo utilizzando il ben conosciuto Esplora Risorse di Windows.

Possiamo anche fare l'upload di immagini, filmati, file musicali; l'unico limite è generalmente dato dalla dimensione dello spazio a nostra disposizione.

Spesso solo webmaster, webdesigner e webcontent, ognuno per la sua area di competenza, si occupano di caricare contenuti nel sito, ma in alcuni casi potremmo voler dare la stessa possibilità anche ai nostri visitatori (o ai nostri clienti).

Pensiamo, ad esempio, al caso di un sito di viaggi amatoriali che voglia consentire agli utenti (meglio se registrati) di caricare (uploadare) le immagini della loro ultima vacanza, o ad un sito di un ente pubblico in cui una persona designata si occupi di caricare le delibere di interesse pubblico senza dover ogni volta passare attraverso il webmaster.

È chiaro che queste persone non devono assolutamente avere accesso tramite FTP alla struttura del sito Web, sia per motivi di sicurezza e privacy sia perché, accidentalmente o volutamente, potrebbero cancellarne tutti i contenuti.

Come effettuare l'upload dei file

La soluzione è quindi creare una pagina Web attraverso la quale compiere l'operazione di upload dei file. A seconda dei nostri scopi la pagina potrà essere o no protetta, consentendo l'accesso ai soli utenti autorizzati.

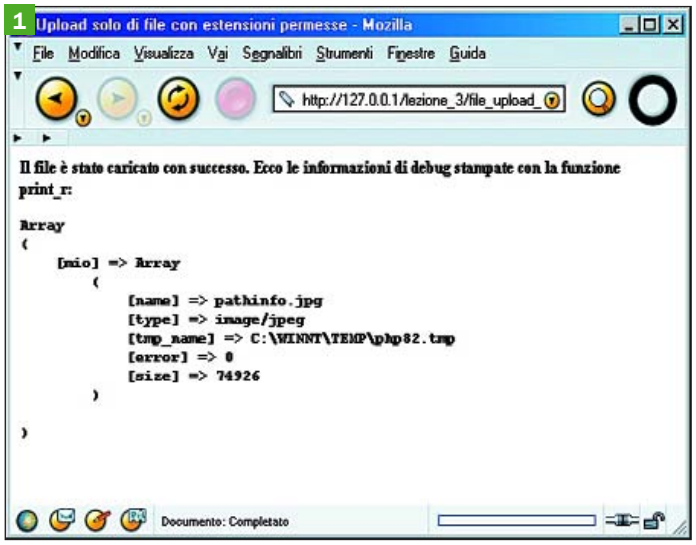
PHP è in grado di ricevere file (in forma testuale o binaria) direttamente da un form di una pagina Web e ci fornisce una serie di funzioni di autenticazione e manipolazione grazie alle quali abbiamo un completo controllo su quanto viene caricato e sulla sua gestione futura.

Costruiamo per prima cosa il form di inserimento, come visibile nella pagina *form_upload.php*.

Ci sono tre cose cui prestare attenzione (*listato 01*): la particolare istruzione **enctype** nell'intestazione del form (senza la quale l'operazione non viene svolta), l'imposizione della dimensione massima (in byte) del file caricabile e il type "file" dell'istruzione di input.

Listato_01
```
<form enctype="multipart/form-data"
    action="file_upload.php"
method="post">
<input type="hidden" name="MAX_
    FILE_SIZE" value="100000">
File da inviare: <input name="mio"
    type="file" size="40"><p>
<input type="reset" value="Cancella"
    > <input type="submit"
    value="Invia">
</form>
```

Il file individuato dopo aver premuto il bottone *sfoglia* (o *browse* se abbiamo impostato la lingua inglese di visualizzazione) viene caricato in memo-



Informazioni di debug utili *per l'upload di un file*

ria come file temporaneo e quindi indirizzato alla pagina *file_upload.php*. La dimensione massima del file è in realtà definita a livello server nel php.ini (valore della riga *upload_max_filesize*; di default è 2 MB) e fa testo indipendentemente dal valore scritto a livello di form. Definire qui un valore può però rivelarsi utile per due motivi: per restringere le dimensioni dei file che vogliamo far caricare o per evitare che un utente attenda il caricamento di un file per poi vedersi restituito un errore (le dimensioni del file sono confrontate con quanto indicato nel *php.ini* solo dopo il caricamento in memoria!).

Dopo il caricamento PHP ci mette a disposizione il vettore globale **$_FILES**, grazie al quale otteniamo una serie di informazioni sul file: il nome originale, il "mime type" (se il browser è abilitato a fornire questa informazione), la dimensione in byte, il nome del file temporaneo e il codice di errore.

Queste cinque chiamate sono riportate nel *listato 02*, e sono caratterizzate da cinque parole chiave (**name, type, size, tmp_name, error**) poste dopo il nome assegnato al file da caricare nell'operazione di input (nel nostro caso era <input name="mio" type="file" size="40">):

listato_02
```
<?php

 echo "Nome del file:
    ".$_FILES['mio']['name']."<br>";
    // esempio: "costruzione.gif"
 echo "Tipo di file:
    ".$_FILES['mio']['type']."<br>";
    // esempio: "image/gif"
 echo "Dimensione del file:
    ".$_FILES['mio']['size']."<br>";
    // esempio: 4783 bytes
 echo "Nome temporaneo file: ".$_
    FILES['mio']['tmp_name']."<br>";
    // esempio:
    "C:\winnt\temp\php22.tmp"
 echo "Tipo di errore:
    ".$_FILES['mio']['error']."<br>";
    // esempio: "0"
?>
```

Il valore "0" di $_FILES ['mio']['error'] indica che il caricamento in memoria è avvenuto in maniera corretta (vedi i codici risultanti dall'interrogazione più avanti nell'articolo). In questo momento il file è ancora nella memoria temporanea; per effettuare fisicamente l'upload utilizziamo una funzione che si occupa di spostare un file caricato in memoria nella sua destinazione finale:

```
move_uploaded_file (file_caricato,
destinazione);
```

Questa funzione restituisce *true* solo se l'operazione va a buon fine, quindi è necessario che il file sia correttamente caricato e (fondamentale) che la destinazione sia una cartella cui siamo abilitati in scrittura (in caso di dubbio chiedere al provider se e quali sono le cartelle cui si è abilitati in scrittura volendo caricare un file da form Web).

L'esempio completo lo potete vedere nella pagina *file_upload.php* (*listato 03*) e le informazioni restituite sono visibili nell'immagine 1; la cartella di destinazione scelta è "upload/" e il percorso è indicato in modo relativo. Terminata l'operazione il file temporaneo sarà cancellato dalla memoria.

Attenzione: *move_uploded_file* sovrascrive file esistenti senza chiedere alcun permesso.

listato_03
```
$dir="upload/".$_FILES['mio']['name'];
if (move_uploaded_file($_FILES
   ['mio']['tmp_name'], $dir)) {
      echo "Il file è stato caricato con
      successo.  Ecco le informazioni di
      debug:\n<pre>";
      print_r($_FILES);
}
```

Non è difficile impostare dei controlli basati sulla dimensione del file o sul tipo: come esempio potete studiare la pagina *form_upload_txt.php* (carica i file passando attraverso *file_upload_txt.php*) in cui è impostato un controllo per consentire solo il caricamento di file tipo testo.

Non sempre, però, il valore restituito da $_FILES['file']['type'] è sufficiente per definire il tipo di file che si sta caricando, ed inoltre questo valore in alcuni casi dipende, purtroppo, dal browser usato dall'utente. Per evitare problemi si può agire in modo da ricavare l'estensione del file (i tre ultimi caratteri dopo il "."), per poi confrontarla con un vettore di estensioni "ammesse".

L'esempio riportato in *form_upload_check.php* e *file_upload_check.php* tiene conto anche dei file con più estensioni (tipo "pippo.txt. php") estraendone solo l'ultima (quella effettivamente valida) e confrontandola con le estensioni consentite (listato 04).

L'esempio è utile anche per conoscere due nuove funzioni: **explode** (trasforma in vettore una lista una volta definito un separatore dei termini) e **in_array** (controlla se un valore è presente in un vettore).

listato_04
```
<?php

$nomi=explode('.',
$_FILES['mio']['name']);
    // il nome del file è stato diviso
    in più parti registrate nel vettore
$nomi. Il separatore è il punto.
$estensione=$nomi[count($nomi)-1];
    // estraggo l'ultimo valore del
    vettore: questa è l'estensione
    del file originale
$ammesse=array("jpg", "gif", "txt");
    // questo è il vettore con le
estensioni da me consentite
if (in_array($estensione,$ammesse)) {
    // con in_array controllo se
    l'estensione del file rientra tra
    quelle ammesse
[...]
```

Ricordiamo che va assolutamente impedito il caricamento di file con estensioni PHP dal momento che tali file potrebbero eseguire operazioni distruttive all'interno del nostro spazio Web: non va mai dimenticato, infatti, come PHP sia anche un potentissimo linguaggio di scripting.

È possibile anche caricare più file contemporaneamente utilizzando le caratteristiche dei vettori per inserire le istruzioni input da utilizzare:

```
<input name="mio[]" type="file">
```

Con qualche controllo nella pagina ricevente (se il codice errore è 0 allora il file in questione può essere caricato, altrimenti significa che c'è un errore o che la casella di caricamento è stata lasciata vuota) i nostri file verranno caricati nella directory scelta.

Gli esempi citati sono nelle pagine *form_upload_plus.php* e *file_upload_plus.php*, disponibili nel CD Guida.

I codici risultanti dall'interrogazione $_FILES['nome input']['error'] significano:

0 => il file è stato caricato correttamente in memoria

1 => la dimensione del file eccede quanto stabilito nel php.ini

2 => la dimensione del file eccede quanto indicato nel form

3 => il file è stato caricato parzialmente

selezionato nessun file nel form di upload

3 Lavorare con i file

Abbiamo già parzialmente visto come lavorare coi file e con le directory quando abbiamo introdotto le funzioni **basename**, **opendir** e **readdir** nelle lezioni precedenti, e lo stesso caricamento di file appena visto è un altro esempio di utilizzo delle funzioni filesystems di PHP. Abbiamo a disposizione numerosissime funzioni pronte all'uso (sempre che si abbia la pazienza di leggere il manuale), sta solo a noi trovare la maniera migliore di operare.

Nel paragrafo precedente, ad esempio, *move_uploaded_file* ha rivelato il difetto di sovrascrivere, senza avvertire, i file presenti nella cartella. Meglio predisporre, quindi, un controllo in grado di verificare la presenza o meno di un certo file nella directory esaminata. La funzione è:

```
file_exists ($nome);
   // $nome comprende il percorso
   di ricerca del file e il suo nome
```

Potremmo anche desiderare cancellare un file presente in una cartella: in questo caso la funzione da utilizzare è:

```
unlink ($nome);
   // $nome comprende il percorso
   di ricerca del file e il suo nome
```

Un esempio concreto è presente nella pagina *checkcanc.php*, attraverso la quale si può controllare se un file è presente nella cartella "upload" (ed eventualmente cancellarlo) semplicemente compilando il form. Potremmo migliorare l'esempio (pagina *checkcan_plus.php*) creando un form (vedi *listato 05*) con una casella a discesa nella quale siano presenti tutti i file della cartella "upload": selezionandone uno possiamo decidere di cancellarlo o di vederne alcune caratteristiche con la funzione pathinfo *(immagine 2)*.

Vedendo il listato originale, noterete che sono state usate @opendir e @unlink: nel prossimo paragrafo vedremo a cosa serve la @ davanti al nome di una funzione.

PHP consente di lavorare non solo con i file, ma anche "sui" file: possiamo, infatti, aprire e modificare i file scrivendo informazioni al loro interno. Per questi scopi si usano una serie di funzioni specifiche:

```
$handle=fopen ($nome, mode);
   // il descrittore $handle apre,
   tramite la funzione fopen, un file in
   lettura e/o scrittura, posizionando il
   puntatore all'inizio o alla fine del file.
   Usando un sistema windows è
   meglio usare sempre l'opzione
   aggiuntiva "t" come "mode" per i file
   testo e "b" per i file binari.
fread ($handle, length);
   // legge il contenuto di un file
   aperto dal descrittore $handle. Se
   length non è settato, il file viene
   letto fino alla fine.
fwrite ($handle, $content);
   // scrive $content sul file aperto dal
   descrittore $handle
fclose ($handle);
   // chiude il file aperto dal
   descrittore $handle
```

Con poche nozioni siamo in grado di creare un paio di esempi interessanti. Supponiamo di avere nel nostro sito una pagina di link (*immagine 3*) e di voler sapere quali sono effettivamente cliccati e quante volte. Per avere queste informazioni può essere sufficiente scrivere su un file di testo delle righe che contengano l'indicazione del link e le informazioni sul giorno e l'ora del clic: questo file potrà poi essere editato con un foglio elettronico per ricavarne delle statistiche.

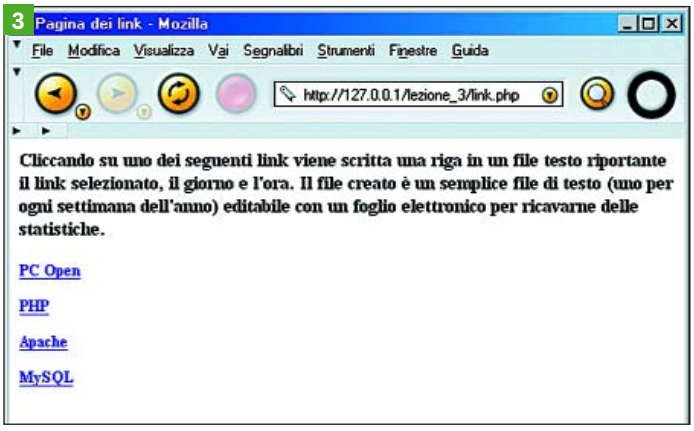
Il sorgente della pagina *link.php* ci mostra come ottenere questo risultato. Per prima cosa impostiamo la sezione HTML: non ci sono difficoltà, salvo l'accortezza di indicare ogni link in questo modo:

```
<a href="link.php?url=http://
www.pcopen.it">PC Open</a><p>
   // link.php è la pagina dove ci
   troviamo. L'url del link viene qui
   assegnato alla variabile
   $_GET['url']
```

La sezione PHP (in testa al codice sorgente della pagina *link.php*) contiene il codice che consente di scrivere nel file (*listato_06*). Per prima cosa viene controllato che esista $_GET['url'], il che vuol dire che qualcuno ha cliccato su uno dei link. Poi viene assegnato un nome al file di testo dove verranno registrati i dati per le statistiche: scegliamo di dargli un nome del tipo "annosettimana.txt", in modo da avere file separati per ogni settimana (usiamo le funzioni di data già viste nella lezione 1). Il successivo ciclo if si occupa di definire il descrittore $file e di aprire (eventualmente creare) il file annosettimana.txt, posizionando il puntatore di scrittura al termine del testo già inserito. Mi occupo poi di definire la riga da scrivere, inserendo il percorso del link, una tabulazione ("\t"), la data e l'ora del click e,

```
listato 05
<select name="nome" size="1"><option></option>
<?php
    $cartella=@opendir('upload');
    while (false !== ($file = readdir($cartella))) {
        $lista[]=$file;
    }
    $lista=array_slice($lista,2);
        // array_slice lo uso per eliminare le indicazioni . e .. presenti in readdir.
        È un sistema più veloce per compiere la stessa operazione vista
        nella prima lezione
    foreach ($lista as $nomi) {
        echo "<option>".$nomi."</option>";
    }
    ?>
    </select>
```

Form per ricavare informazioni *su un file e per cancellarlo*



Con PHP possiamo individuare *i link più cliccati di una pagina*

listato 06
```
<?php
    if (isset($_GET['url'])) {
        $nome_file = date("YW").".txt";
            // il file che verrà creato è del tipo annosettimana.txt,
        if ($file=fopen($nome_file, 'at')) {
            // apro il file o lo creo. $file è il "descrittore"; il mode "a" serve
            a posizionare il puntatore alla fine del file, il mode "t" serve per la
            modalità testo per Windows
             $ins = $_GET['url']."\t";
             $ins.= date('d/m/Y H:i')."\n";
              fwrite($file, $ins);
                // scrivo la stringa $ins passando alla funzione fwrite
               il descrittore
               fclose($file);
               // chiudo il file passando alla funzione fclose il descrittore
            }
            header("Location: ".$_GET['url']);
            // indirizzo l'utente al link su cui aveva cliccato
        }
?>
```

per ultimo, l'indicazione di fine riga ("\n").

Ora non ci resta che scrivere la riga con **fwrite**, chiudere il file con **fclose** e indirizzare il nostro utente (in maniera del tutto trasparente per lui) verso il link cliccato.

Il file di testo ottenuto sarà del tipo visibile in immagine 4. Adesso siamo in grado, con poche modifiche al file originale, di aggiungere a piacere tutta una serie di informazioni come il browser del visitatore e il suo indirizzo remoto. E adesso non ci resta che analizzare il nostro file settimanale delle statistiche dei link.

Certo questo è un buon sistema, ma non è ottimale e infatti si presta a piccole "manomissioni" esterne: ad esempio si può aggiungere una riga al file di testo anche senza cliccare su nessun link, semplicemente scrivendo un URL del tipo

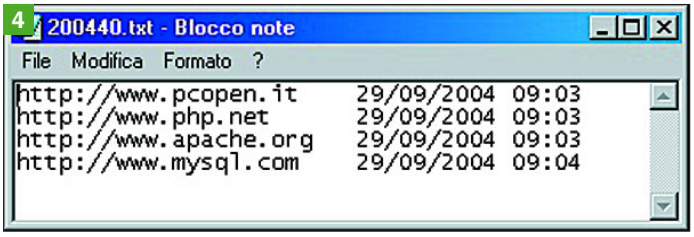
```
http://127.0.0.1/lezione_3/link.php?
url=http://www.linux.org
// scrive nel file di testo un link non
compreso nella pagina originale
```

Bisognerebbe dunque prevedere dei controlli di integrità, e senza dubbio un lavoro migliore lo si otterrebbe registrando i dati in un database. Ma questo lo impareremo nelle prossime lezioni; l'importante in questo frangente era solo dimostrare come non sia troppo difficile lavorare sui file con PHP.

Il secondo esempio (pagina "contatore.php") consente di creare un rudimentale contatore per le nostre pagine (*immagine 5*), mantenendo memorizzato il valore nel file di testo "conteggio.txt".

Rispetto all'esempio precedente, in questo caso dobbiamo prima leggere il contenuto del file conteggio.txt, assegnando il valore ad una variabile. Quindi, chiuso il file (è buona norma tenere i file aperti il meno possibile), incrementiamo il valore trovato e riscriviamo il file con questo nuovo valore. Queste nuove operazioni sono riportate nel listato 07. Ogni volta che la pagina contatore.php verrà caricata vedremo aumentare di una unità il valore visualizzato.

listato 07
```
    $conteggio= fread($file,filesize($visite));
    // assegno a $conteggio il valore che leggo nel file di testo
    (dove c'è solo un numero)
    fclose($file);
    $conteggio++;
    // incremento il contatore
    $file = fopen($visite,"w");
    fputs ($file,$conteggio);
    // scrivo il nuovo valore sul file eliminando il precedente
```

4


Il file testo ottenuto indica *quali link risultano più visitati*

Lavorare coi file può non essere sempre facile, ma abbiamo visto che possono rivelarsi estremamente utili anche se, per le loro caratteristiche, non possono sostituire i database. Come al solito, per approfondire e conoscere tutte le funzioni filesystems di PHP bisogna leggere l'ottimo manuale di PHP.net; questo anche per evitare, magari chiedendo aiuto in un newsgroup, di sentirsi rispondere con l'acronimo RTFM, ossia "Read the f... manual!".

5


Un semplice contatore *delle visite della pagina*

4 La gestione degli errori

Scrivendo del codice è sempre possibile fare qualche errore. Si può introdurre nella sintassi qualche costrutto errato o si possono eseguire operazioni proibite o bloccate dalle regole che le definiscono. Esempi possono essere: cercare di dividere un numero per zero, oppure chiamare una funzione con un nome inesistente o includere un file esterno non presente.

In tutti questi casi PHP ci viene in aiuto con una serie di messaggi di errore che spesso ci consentono di capire al volo cosa c'è di sbagliato nel codice. L'esempio *errori.php* genera i messaggi visibili in *immagine 6*:

Abbiamo ottenuto come risultato tre avvertimenti (**warning**) e un errore grave (**fatal error**). La differenza risiede nel fatto che un warning non blocca la lettura della pagina, mentre un fatal error ne causa l'immediata interruzione e quindi impedisce la lettura del codice successivo (la pagina *errori.php* dovrebbe mostrare una riga di testo alla fine).

Leggendo con attenzione il reporting di PHP, possiamo trarre i necessari suggerimenti per correggere il nostro codice, eliminando quindi il divisore zero, creando il file *pippo.txt* nella cartella *upload* e infine definendo la funzione *somma*. Questo debug è stato utile per noi programmatori, ma potrebbe essere fonte di problemi se venisse visto dai visitatori di questa pagina, oltre alla brutta figura che faremmo: l'utente, infatti, vedrebbe informazioni sui listati e sulle pagine, nonché nome e indirizzo di un file da includere (con qualche intuizione, potrebbe tentare di accedere direttamente ai file ▷

della cartella *upload*). Peggio ancora, potremmo dare indicazioni sull'indirizzo e sul nome utente del server su cui risiede il nostro database (lo vedremo nelle prossime lezioni).

Bisogna quindi porre attenzione alla gestione degli errori: la prima cosa da conoscere è il settaggio del *php.ini*, settaggio visibile, al solito, tramite la pagina *phpinfo.php* (vedi prima lezione). Probabilmente troveremo:

```
error_reporting = 2039
    (ossia tutti i messaggi di errore
    sono visualizzati tranne alcuni
    warning che non sono considerati
    comunque errati)
display_errors = On
log_errors = Off
```

Questo settaggio indica che tutti gli errori verranno visualizzati e non saranno registrati in un file di log. Se siamo utenti remoti c'è solo da tenere in considerazione questo fatto, mentre se siamo "padroni" del nostro server andrebbe presa in considerazione l'ipotesi di inibire la visualizzazione dei messaggi di errore consentendone, al contempo, la registrazione in un file di log. Dal momento, però, che la situazione standard è la prima (sul server remoto non abbiamo alcun potere), vediamo cosa possiamo fare per gestire al meglio gli errori.

Un intervento drastico è quello di impedire la visualizzazione di tutti i messaggi di errore che potessero manifestarsi in una pagina inserendo, all'inizio del codice:

```
<?php
error_reporting (E_NONE);
   // oppure error_reporting(0);
   // controllate sul manuale le opzioni
   di error_reporting per poter
   scegliere di visualizzare solo certi
   errori, tipo E_ERROR,
   E_WARNING,...
?>
```

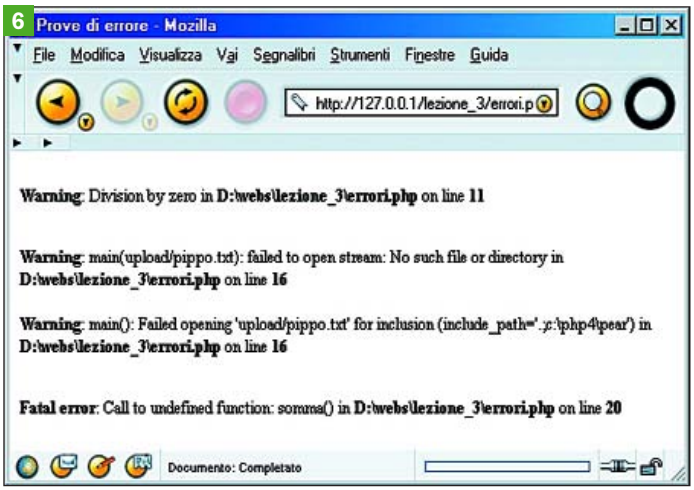
Provate, come verifica, a caricare la pagina *errori_no.php*, dopo aver controllato che il codice sorgente sia lo stesso della pagina *errori.php*: non verrà visualizzato nulla.

Questo, però, è un rimedio drastico, mentre PHP ci consente di usare anche delle soluzioni puntuali. Una sfrutta l'operatore di controllo errori "at" (@): la famosa chiocciolina @ davanti ad una espressione PHP impedisce l'eventuale visualizzazione di un messaggio di errore (*pagina errori_at.php*):

```
$b=0;
$c = @ ($a/$b);
   // non viene visualizzato il warning
   che ci aspetteremmo per avere
   diviso un numero per zero
```

@ funziona solo quando si trova davanti ad un'espressione che deve restituire un risultato, quindi va bene per chiamate di funzioni e include, ma non davanti a cicli condizionali come if o foreach. Su un esempio del paragrafo precedente si era usata @ davanti all'istruzione unlink per impedire che venisse visualizzato un warning se si tentava di cancellare un file non presente nella cartella: per lo scopo dell'esempio, infatti, il non trovare il file non era da considerarsi un errore e quindi non doveva comparire alcun messaggio.

Nel caso di uno script (ad esempio l'apertura di un file o connessione a un database), accanto ad @ si può usare il costrutto **die** (alias di exit) per bloccare il codice nel punto in cui lo script fallisce (*pagina errori_die.php*):



In questa schermata *sono raccolti alcuni errori in PHP*

```
$file = '/upload/pippo.txt';
$apri = @fopen($file, 'r') or
die("<h3>Non è stato possibile aprire
il file $file</h3>");
```

In questo esempio, la @ impedisce la visualizzazione del messaggio di warning, poi il codice, fallita la chiamata a *fopen*, è bloccato da *die*.

Alcuni errori, invece, li possiamo evitare sfruttando adeguatamente il codice. È perciò buona abitudine utilizzare dei cicli if insieme alle funzioni *file_exists, function_exists* e *method_exists*, per accertarsi che file, funzioni e metodi delle classi cui facciamo riferimento esistano realmente nel progetto che stiamo sviluppando.

5 I consigli per rendere il sito (più) sicuro

Sicurezza è un concetto ampio che coinvolge molteplici aspetti: i dati sul sistema e i dati in transito, i protocolli, il server hardware, il Web server software, le applicazioni Web e le persone. Tutti questi aspetti sono correlati, e basta una falla in un qualsiasi punto per rendere "insicuro" tutto il sistema.

Non è questa la sede per addentrarci in disquisizioni sulle politiche di sicurezza da adottare, né avrebbe senso parlare della sicurezza server-side di Apache e PHP, argomenti sui quali spesso non abbiamo controllo diretto e che lasciamo agli amministratori di sistemi.

Nel nostro piccolo, invece, possiamo dedicare qualche sforzo all'evitare errori banali di programmazione, cercando di fare quindi la nostra parte nella catena della sicurezza. Sono piccoli accorgimenti inerenti quanto abbiamo visto in queste puntate e di cui magari abbiamo già parlato, ma è bene richiamarli per concludere adeguatamente il discorso su PHP.

Come base di tutto, facciamo attenzione a cosa inseriamo nella **Web root**: teniamo i dati sensibili in cartelle diverse, magari adeguatamente protette impostando i permessi, ed evitiamo anche di lasciare nella root pagine come la *phpinfo.php* che può dare informazioni importanti ad un potenziale attaccante.

La prima accortezza, poi, consiste nel non usare mai, anche se il settaggio di *php.ini* lo consentisse, gli **short tag** <? e ?> o gli **asp tag** <% e %> per indicare al browser l'inizio del codice PHP. Un cambio di settaggio, infatti, renderebbe il codice visibile a tutti (magari insieme a password e parametri di connessione), oltre a impedire la corretta visualizzazione delle pagine Web. Usiamo quindi sempre i tag standard <?php e ?>.

Seconda accortezza: quando viene incluso un file (con *include* o *require*) in una pagina Web, teniamo presente che questo file esiste fisicamente nel nostro spazio ed è accessibile direttamente col browser se si conosce l'indirizzo remoto. Dal momento che gli *include* sono spesso usati per registrare dati importanti, funzioni e classi, bisogna impedirne la visualizzazione diretta. Ricordiamoci, quindi, di salvare il file con estensione finale .php, usando poi i tag php nel modo appropriato.

Anche i **messaggi di errore** possono causare falle nella sicurezza, quindi, oltre a controllare attentamente il codice scritto, si può disabilitare la visualizzazione dei messaggi nelle pagine caricate sullo spazio Web.

L'operazione di **upload di file** può essere pericolosissima se si lascia l'utente libero di caricare qualsiasi tipo di file, magari contenenti script PHP in grado potenzialmente di cancellare tutto il nostro lavoro. Essenziale, quindi, è porre un controllo sul tipo di file che è possibile uploadare e sulle cartelle dove permettiamo il caricamento.

Attenzione va posta ai **form**: in alcune circostanze abbiamo bisogno di passare alcuni valori "nascosti" e li inseriamo col *type="hidden"* in un'istruzione input. Ricordiamoci che questi valori sono perfettamente visibili a tutti: basta semplicemente visualizzare il codice sorgente HTML della pagina. Altro aspetto importante è controllare cosa viene inserito nei campi del form, validando sempre i dati. Immaginate, ad esempio, di avere creato un form per consentire la cancellazione di record da un database. Cosa succederebbe se un utente inserisse il carattere jolly del linguaggio di interrogazione "%"? Senza un controllo di validazione, il risultato finale sarebbe la cancellazione di tutti i record.

Le **sessioni** hanno un nome identificativo predefinito: PHPSESSID. A questo nome si fa spesso riferimento per propagare l'id di sessione o per registrarlo su un cookie. Sarebbe meglio, però, che il nome della sessione lo assegnassimo noi: cosa succederebbe, infatti, se un domani venisse cambiato nel php.ini questo valore predefinito? Il nostro codice non funzionerebbe più come avevamo previsto.

L'ultima raccomandazione fa riferimento al settaggio *register_globals* di *php.ini*. Fino alla versione 4.2 il valore di default era *On* e questo consentiva di registrare le variabili col loro nome, indipendentemente dall'input. In pratica, una variabile *$nome* presente in una pagina poteva derivare da un input di tipo *get* come da un input di tipo *post* o da un cookie o da una sessione.

Questo causava problemi di vulnerabilità, quindi il settaggio di *register_globals* è stato posto a *Off* e per riferirsi alle variabili si devono usare gli array superglobali $_POST[], $_GET[], $_SESSION[] e $_COOKIE[]. In questo modo si può distinguere, ad esempio, $_GET['nome'] da $_SESSION['nome'], dove il primo valore deriva dall'URL della pagina mentre il secondo deriva da una sessione inizializzata, e al contempo non è definita nessuna variabile globale *$nome*.

Anche se il vostro provider avesse mantenuto il valore di *register_globals* a *On*, è importante comunque usare sempre e solo gli array superglobali per non incappare nel problema visibile nella pagina *vulnerabile.php*: poniamo che in questa pagina venga fatto un controllo su una variabile *$autenticato* definita da un cookie. Se carichiamo la pagina veniamo bloccati (non abbiamo un cookie nella nostra memoria dove sia definita la variabile), ma possiamo bypassare il controllo semplicemente scrivendo sul browser l'URL in questo modo: *vulnerabile.php?autenticato=1* (*immagine 7*). Se invece il controllo fosse stato su $_COOKIE['autenticato'] non avremmo potuto fare nulla (vedere *vulnerabile_no.php*). Chiaramente se *register_globals* fosse stato *Off* anche il primo metodo non avrebbe più funzionato.

Con questi consigli il minicorso di PHP come strumento stand-alone si è concluso e adesso tocca ad ognuno approfondire la conoscenza di questo potentissimo linguaggio. Nelle prossime puntate vedremo, invece, come interfacciare PHP con un database in modo da aumentare considerevolmente le nostre azioni di webdeveloper. Ma per far questo dobbiamo capire bene cosa sono le basi di dati, come vediamo di seguito.

7 Vulnerabilita' - Mozilla
File Modifica Visualizza Vai Segnalibri Strumenti Finestre Guida
http://127.0.0.1/lezione_3/vulnerabile.php?autenticato=1
Queste informazioni dovrebbero essere leggibili solo da parte di chi sia stato autenticato
Documento: Completato

Un problema di vulnerabilità *facilmente evitabile*

6 I database

Le tecnologie basate sui database (letteralmente "raccolte di dati") giocano un ruolo critico in tutte le aree fondamentali della nostra vita: pensiamo, ad esempio, all'uso di database per consentire la gestione del nostro conto in banca, per prenotare delle visite in ospedale, per verificare gli esami effettuati all'università, per la gestione delle aziende (coi software ERP) e via dicendo.

Ne sentiamo tanto parlare, ma cos'è realmente un database? Volendo semplificare al massimo la definizione, possiamo vederlo come una collezione di dati tra loro correlati, dove per "dati" intendiamo dei valori conosciuti che possono essere registrati e hanno un significato implicito. Da questo punto di vista un'agenda personale è certamente un database, in quanto è una collezione di dati aventi un significato implicito (nome, indirizzo, telefono) e tra loro correlati. Anche la pagina di un libro, però, potrebbe essere vista come un database in quanto rappresenta una collezione di parole tra loro collegate. Così non è, dal momento che un database è considerato tale quando ha queste proprietà:

- rappresenta uno o più aspetti del mondo reale (il cosiddetto "minimondo");
- è una collezione di dati logicamente coerenti tra loro e coerenti con la descrizione della realtà scelta (non è, quindi, un insieme di dati messi a caso);
- è disegnato, costruito e popolato per un certo gruppo di utenti interessati ad interagire con esso compiendo determinate azioni (inserimento, recupero, aggiornamento, cancellazione dati). Un libro, quindi, non è un database, mentre lo è un dizionario.

Un database può contenere più o meno dati ed essere più o meno complesso: pensiamo, ad esempio, alla nostra agenda telefonica e alla banca dati di un gestore di telefonia. Una base di dati potrebbe essere gestita anche manualmente, ma all'aumentare della complessità avremo bisogno di un supporto tecnologico e informatico per costruire ed utilizzare il DB, e nel contesto di questo corso gli strumenti scelti sono il database MySQL e PHP. Dal punto di vista informatico un database è assimilabile ad una tabella tipo foglio elettronico, con i distinguo che vedremo in un paragrafo successivo.

Il database system

La rappresentazione grafica un ambiente di sviluppo "moderno" di un database è visibile in *figura 8*. A valle, memorizzato in un qualsivoglia mezzo elettronico, c'è il database fisicamente esistente, composto dai dati e dalle informazioni sulla sua stessa struttura (meta-dati).

Per creare il DB, accedervi e interrogarlo utilizzeremo un **DBMS** (DataBase Management Systems), ossia un'applicazione in grado di processare delle richieste e di accedere ai dati in scrittura e lettura. MySQL è un DBMS, come lo sono anche Microsoft Access e SQL Server, PostgreSQL, Oracle e moltissimi altri programmi.

Per utilizzare il DBMS, infine, servirà un'applicazione in grado di inviare una richiesta: po- ▷

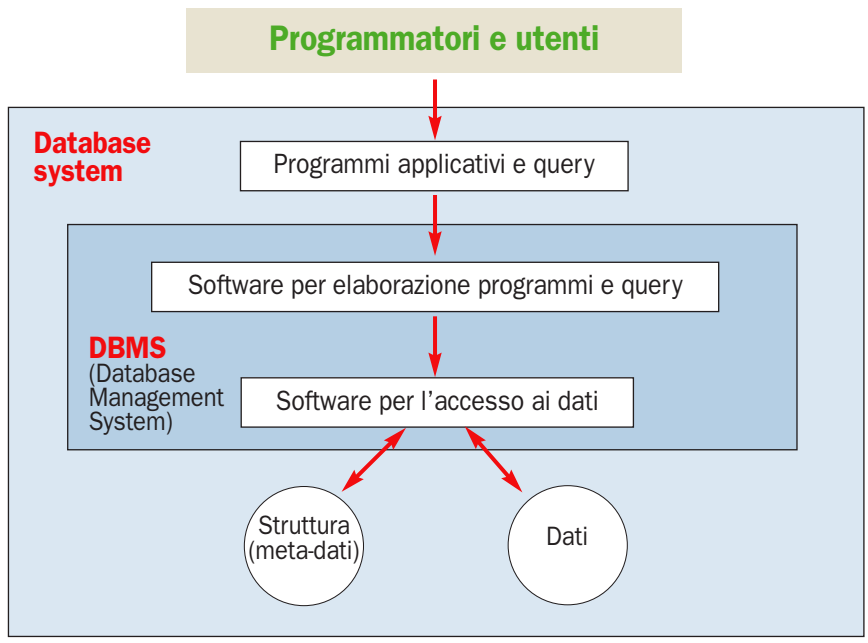
trebbe essere un'interfaccia dello stesso DBMS, ma può essere anche un software o un linguaggio esterno come PHP o ASP. Tutte le parti qui descritte vanno a formare il database system, ossia l'insieme di software, interfacce, applicazioni che servono all'utente per interagire totalmente con una base di dati.

Il DBMS è il componente fondamentale del sistema perché consente di gestire i processi di definizione, costruzione e manipolazione della base di dati.

Definire un database significa specificare il tipo, la struttura e i vincoli dei dati che saranno memorizzati, mentre **costruire** il DB implica il processo di registrazione dei dati sul "mezzo" controllato dal DBMS. **Manipolare** la base di dati significa, infine, creare le interrogazioni in grado di recuperare, aggiornare e cancellare i dati (riflettendo le variazioni del "minimondo" rappresentato dal database), nonché di generare dei report (a video o a stampa).

Fatte queste debite premesse, però, torniamo al nostro corso. Ci serve un database? Cosa ci consente di fare rispetto alle nostre esigenze di Web developer? Quali vantaggi ci porta? Capito questo dovremo affrontare la costruzione ex novo di un DB (un po' di teoria sarà necessaria per eseguire il lavoro nella maniera più efficiente) e imparare ad interrogarlo.

8



***L'"ambiente" globale** di sviluppo di un database*

7 Siti Web dinamici

Alla base della creazione di un sito Web sta la fondamentale attività della **progettazione**, il momento in cui le richieste e le idee del committente devono essere esplicitate, e spesso "tradotte" in un linguaggio concreto, in una serie di punti programmatici.

Chi ci affida il lavoro, infatti, ha ben chiaro (o dovrebbe averlo) il contenuto del messaggio da comunicare, il modo di comunicarlo e il target cui rivolgersi, ma probabilmente non ha mai pensato ai retroscena di questi aspetti per il presente e il futuro.

Chi crea i testi? Con quale cadenza potrei doverli cambiare? Se cambio un testo, quale sarà la procedura per il nuovo inserimento? È possibile/probabile avere esigenze di ampliamento dei menu del sito? Se ci sono delle news chi le inserisce? Ogni modifica cosa comporta? Chi fa la gestione?

Queste sono solo alcune domande che ci si deve porre. Alcune riguardano solo il committente, ma quasi certamente la risposta a queste domande riguarderà poi noi.

Ricordiamoci, e lo insegna ogni settore dell'economia, che è meglio "spendere" tempo in fase di progettazione (quando i costi della modifica sono bassi) piuttosto di accorgersi in fase avanzata che si potevano percorrere strade alternative: a questo punto fare delle modifiche diventerebbe molto costoso perché tante decisioni di struttura sarebbero ormai ultimate. Il costo della modifica è chiaramente massimo quando il progetto è realizzato. Il committente potremmo anche essere noi stessi e la validità di quanto scritto sopra resterebbe immutata.

Prendiamo come esempio un caso concreto: supponiamo di aver realizzato un sito avente lo scopo di essere la "vetrina" dei prodotti di un artigiano e di aver pattuito un compenso per un anno di aggiornamento delle pagine da effettuare quando il nostro cliente volesse esporre nuovi prodotti.

Se non abbiamo esplicitato bene il significato di questo aggiornamento, potremmo trovarci due problemi, opposti ma ugualmente potenzialmente gravi: dover fare troppi aggiornamenti, o doverne fare uno solo dopo molti mesi. Nel primo caso il problema per noi è dover continuamente prendere in mano il codice HTML, modificare le pagine e ricaricarle nello spazio Web; nel secondo caso dovremmo riguardare del codice scritto mesi prima, riesaminando il progetto (che magari ha condotto un nostro collaboratore che non lavora più per noi) e facendo infine l'agognato aggiornamento. In entrambi i casi il rischio che il costo del tempo utilizzato non sia stato coperto dal prezzo pattuito col cliente è alto.

Quale sarebbe potuta essere una soluzione migliore? Utilizzare un DBMS per creare un database contenente gli oggetti dell'artigiano da cui estrarre i dati da visualizzare sulla pagina Web.

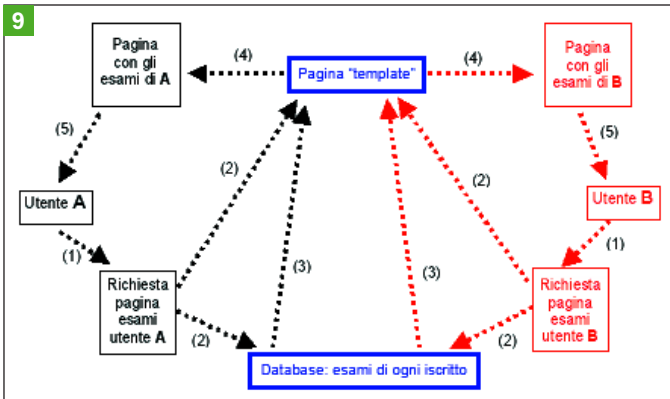
In questo modo ogni modifica/aggiunta/cancellazione ai prodotti andrebbe fatta direttamente sul database (un'operazione, come vedremo presentando il linguaggio SQL, molto facile e per la quale potremmo preparare un'interfaccia grafica, ad esempio con PHP) ed automaticamente la pagina Web sarebbe aggiornata. Certo questo sistema di lavoro ci "costerebbe" di più in fase realizzativa, ma ne beneficeremmo in seguito. Anzi, potremmo addirittura proporre al cliente, in cambio di un prezzo maggiore di realizzazione del sito, di fare lui stesso queste operazioni di modifica dandogli così anche una libertà generalmente molto apprezzata.

Potremmo stabilire delle aree nelle quali riservare al cliente l'aggiornamento (notizie, domande e risposte, comu-

nicati e via dicendo) e mantenere per noi, invece, l'aggiornamento delle altre. Insomma, un sistema di gestione a database può consentirci, pur pagando il prezzo di una progettazione più onerosa, di separare fisicamente il concepimento della pagina (**template**) dal suo riempimento, rendendo quest'ultima un'operazione più semplice e veloce in caso di aggiornamenti.

Ci sono poi i casi in cui utilizzare dei database è assolutamente necessario perché la mole di dati e l'utilizzo che se ne fa renderebbe impossibile la gestione di pagine "statiche". Un esempio, che poi useremo per costruire da zero un DB, possono essere le pagine di un ateneo che voglia mettere i propri studenti in grado di accedere alla lista degli esami fatti con la relativa votazione. È chiaro come non sia neppure concepibile pensare di creare tante pagine "statiche" (tutte protette da password) quanti sono gli studenti immatricolati, vuoi per esigenze di spazio (forse il problema minore) vuoi per tutte le implicazioni di aggiornamento che la scelta comporterebbe. In questo caso, quindi, l'unica soluzione è creare il template della pagina "esami"; la pagina verrà poi dinamicamente riempita con i dati dello studente che effettua la richiesta di visualizzazione del suo piano studi. I dati saranno ospitati in un database costantemente aggiornato in maniera veloce ed economica. La struttura dell'interrogazione e della risposta è visibile schematicamente in *figura 9*. Questa gestione porta anche un altro vantaggio: eventuali modifiche di layout della pagina sono eseguite, una sola volta, esclusivamente sul template e immediatamente sono attive per ogni richiesta di pagina.

9



La richiesta di una pagina gestita da un database *da parte degli utenti A e B (tra parentesi l'ordine delle operazioni)*

La stessa strutturazione la troviamo nei siti di home banking, in quelli di commercio elettronico, nei siti Web editoriale e l'elenco potrebbe continuare a lungo.

8 Teoria dei database

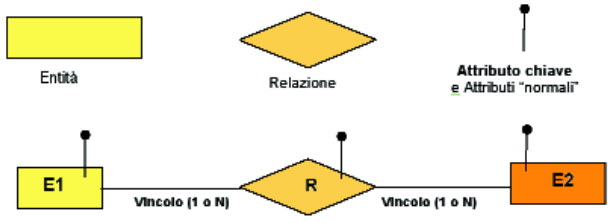
Avendo deciso di sfruttare i vantaggi offerti dalle basi di dati per il nostro sito, ci troviamo adesso nella necessità di creare un DB efficiente. Fondamentale, ai nostri fini, è che il database non abbia **informazioni ridondanti** (ossia non abbia duplicazioni di dati) perché questo comporterebbe spreco di memoria, minore efficienza del sistema e la probabile introduzione di inconsistenze e/o perdite di dati.

Per ottenere questo risultato si compiono delle **operazioni di modellizzazione** allo scopo di esplicitare la struttura finale del DB che andrà creata col DBMS scelto. Questo primo passo, da fare anche con carta e penna, è forse il più noioso, ma è assolutamente importante e vale quanto detto in precedenza: meglio dedicare molto tempo alle fasi progettuali piuttosto di accorgersi alla fine della necessità di modificare la struttura per renderla più efficiente.

Come esempio di lavoro vogliamo creare lo schema di un database il cui scopo è memorizzare i dati degli studenti universitari: in particolare ci interessa conoscere la situazione degli esami svolti, quindi dovremo registrare dati relativi agli studenti, dati relativi ai corsi ed agli esami e dati relativi ai professori.

Il primo passo è realizzare lo **schema concettuale**: è uno schema non utilizzabile direttamente dal DBMS, ma fornisce concetti vicini a quelli utilizzati da una persona nel percepire e concettualizzare la realtà oggetto di studio (il minimondo). Useremo, data la sua diffusione e la sua intuitività, il **modello Entità-Relazione** proposto da P. Chen nel 1976 (la *figura 10* riassume come rappresentare graficamente lo schema concettuale). Questo modello utilizza come base il costrutto **Entità**: rappresenta un oggetto della realtà che, ai fini dell'applicazione di interesse, ha una propria identità (è "distinguibile" dagli altri oggetti) ed ha una esistenza fisica (es: studente, professore) o concettuale (es: corso, piano di studi). Ogni entità è messa in **relazione** con altre entità, da cui il nome dello schema, allo scopo di definirne le reciproche corrispondenze. L'entità è accompagnata da una serie di attributi che ne descrivono le proprietà elementari (es: matricola, nome, data, voto), ed uno di questi attributi (o la combinazione di due o più) deve definire univocamente i

10



Lo schema concettuale*: modello Entità-Relazione*

dati registrati nell'entità (l'**istanza**). Ciò significa che non ci potranno essere due istanze con lo stesso valore: questo attributo è detto **chiave**. Anche una relazione può contenere degli attributi.

Ultimo sforzo è definire il vincolo di relazione tra due entità, vincolo classificabile con una di queste tipologie:

- 1:1 (**uno-a-uno**; one-to-one): ad ogni istanza di E1 corrisponde (può corrispondere) una ed una sola istanza di E2 e viceversa;
- 1:N (**uno-a-molti**: one-to-many): ad ogni istanza di E1 corrispondono (possono corrispondere) più istanze di E2, mentre ad ogni istanza di E2 corrisponde (può corrispondere) una sola istanza di E1;
- N:N (**molti-a-molti**; many-to-many): ad ogni istanza di E1 o di E2 corrispondono (possono corrispondere) più istanze dell'altra entità coinvolta.

Un esempio, con la rappresentazione dei tipi di relazione e degli attributi, è visibile invece in *figura 11*. Le entità coinvolte sono quattro (studente, piano studi, corso, professore) ed esemplificano i tre vincoli di relazioni previste:

- studente - inserisce - piano studi = relazione 1:1 -> ogni studente deve inserire un solo piano di studi, e ogni piano di studi deve essere riferito ad un solo studente. In questo esempio abbiamo assunto che non vi siano piani di studi standard tra i quali uno studente possa scegliere (la relazione sarebbe stata 1:N)
- studente - fa la tesi con - professore: relazione 1:N -> ogni studente può (se ha concluso

gli esami) fare la tesi con un solo professore (il relatore ufficiale), mentre ogni professore può avere N tesisti.
• studente - fa esami del - corso: relazione N:N -> ogni studente può dare gli esami di N corsi, ed ogni corso può essere oggetto di esami da parte di N studenti.

Terminato lo schema concettuale, per creare fisicamente il DB dobbiamo passare allo **schema logico** il quale sarà implementabile direttamente nel DBMS. Vi sono vari tipi di schema, ma il miglior compromesso tra qualità e semplicità è dato dal modello relazionale proposto da E. Codd (primi anni '70). Il modello si basa, semplicemente, su un unico costrutto detto relazione. Questa relazione è una tabella (useremo da adesso questo termine per non ingenerare confusione con lo schema concettuale) con le seguenti caratteristiche (*figura 12*):
• un numero prestabilito di colonne (detti anche campi o attributi) di cui va stabilito il tipo di valori (testo, numerico, ecc) e il loro eventuale dominio;
• un numero variabile di righe (dette anche **record** o **tuple** o istanze);
• uno o più campi formeranno la chiave primaria della tabella, il cui valore identificherà univocamente un record.

In linea di massima si potrebbe anche pensare di costruire un DB con una sola grandissima tabella, ma questo manifesterebbe immediatamente problemi di ridondanze e anomalie. Il modello di Codd, per evitare questo, si basa sulla teoria matematica dei sistemi e per garantire l'integrità e le coerenza dei dati vi sono delle regole aventi lo scopo di creare N tabelle, più piccole possibili, da mettere in collegamento tra loro attraverso i valori delle chiavi. Per passare dallo schema concettuale a quello logico si usa un algoritmo di "traduzione" (mapping):
• per ogni relazione 1:1 si creano due tabelle corrispondenti alle entità coinvolte. Tra i campi di una delle due tabelle si acclude, come chiave esterna, la chiave dell'altra tabella;
• Per ogni relazione 1:N si creano due tabelle corrispondenti alle entità coinvolte. Tra i campi della tabella del ramo N si acclude, come chiave esterna, la chiave della tabella del ramo 1;
• per ogni relazione N:N si creano tre tabelle corrispondenti alle due entità coinvolte ed alla relazione che intercorre tra loro. Quest'ultima tabella avrà come chiave primaria la combinazione delle chiavi delle altre due tabelle.

Lo schema che ne esce è visibile in *figura 13*. L'ultimo sforzo prima di avere uno schema efficiente si chiama **normalizzazione**: è un procedimento, in più fasi, che interviene sulla struttura delle tabelle di un database e sui collegamenti tra esse. La trattazione approfondita del tema esula dal presente corso, ma vale la pena accennare a tre consigli previsti dalle Forme Normali.

Il primo prevede di usare in tutte le tabelle attributi univoci (non ripetuti) e semplici (non composti). Questo vuol dire che un'ipotetica tabella "studenti" non deve mai comprendere una serie di colonne del tipo Esame1, Esame2... EsameN (l'attributo è sempre Esame, e non va mai ripetuto), né deve comprendere un campo Esame se all'interno di questo voglio registrare più di un dato (es: codice esame, votazione, data, professore). In entrambi i casi va prevista la creazione di una tabella da legare a quella studenti: in effetti è quello che abbiamo fatto nel nostro esempio.

Il secondo consiglio è di creare una nuova tabella cui fare riferimento se siamo costretti ad inserire continuamente valori ripetuti. Un esempio banale potrebbe essere il dipartimento di appartenenza di un professore: a forza di inserire il nome del dipartimento rischiamo di commettere errori di digitazione pregiudicando ricerche future, e inoltre se un dipartimento cambiasse nome dovremmo modificare uno a uno tutti i valori già inseriti.

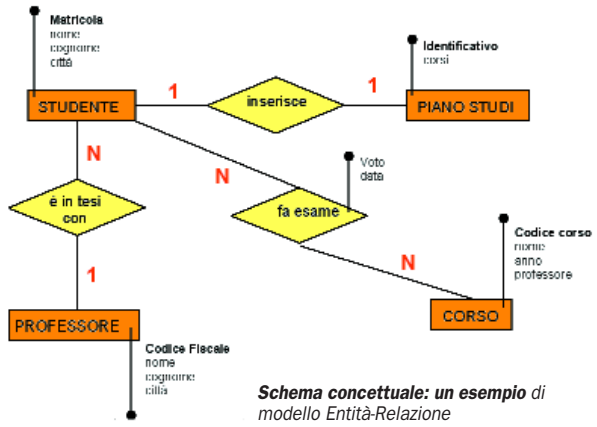
Meglio, quindi, creare una tabella "Dipartimenti" cui fare riferimento: in questo modo non ci sarebbero errori di inserimento, l'aggiornamento verrebbe fatto in un unico punto ed automaticamente propagato a tutti i record interessati della tabella Professori.

Ultimo consiglio è di non inserire in una tabella dei campi che non siamo direttamente dipendenti dai campi chiave della tabella stessa. Un esempio possono essere i campi CAP e Provincia che potremmo inserire, come anagrafica, nella tabella Studenti: questi campi dipendono solo dal Comune di residenza indicato, non dalla matricola dello studente che è la chiave della tabella. Anche in questo caso va prevista una tabella esterna che leghi un Comune al suo CAP ed alla Provincia di appartenenza.

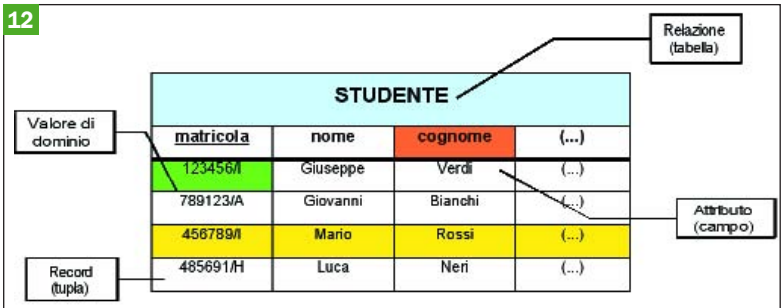
Un altro accorgimento, di buon senso, è di non memorizzare mai dati che possano essere calcolati, quindi mai registrare un campo "Età" se si ha a disposizione l'anno di nascita.

Se abbiamo rispettato quanto scritto, il nostro DB è già molto affidabile ed efficiente, ed è pronto per essere inserito nel DBMS.

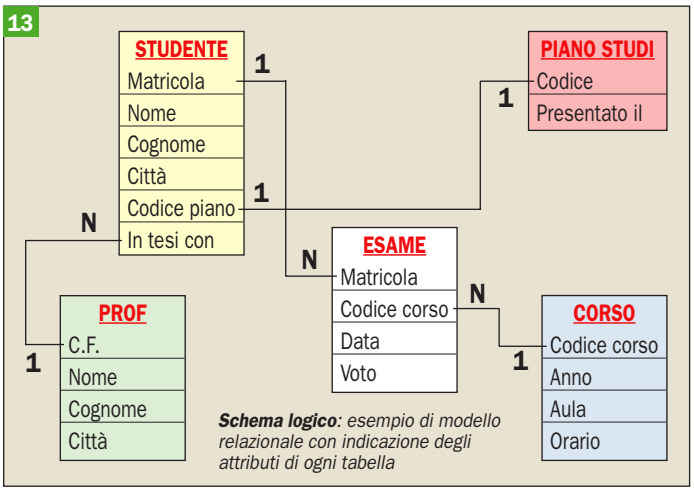
Nella prossima puntata vedremo come creare il DB con MySQL e come compiere le operazioni di inserimento, aggiornamento e cancellazione imparando ad usare il linguaggio universale di interrogazione delle base di dati: SQL, ovvero lo Structured Query Language. Infine vedremo come usare PHP per interfacciarsi con MySQL e portare su Web i risultati delle operazioni compiute. ■

11


***Schema concettuale: un esempio** di modello Entità-Relazione*

12


***Esempio di una relazione** (tabella) nel modello Relazionale*

13


***Schema logico**: esempio di modello relazionale con indicazione degli attributi di ogni tabella*

▶ A scuola con *PC Open*

Web Developer PHP

di Federico Pozzato

1 MySQL e PHP

La terza lezione del corso ci è servita per imparare i concetti fondamentali relativi alla costruzione di un database efficace ed efficiente. Prendendo spunto da un caso reale (creare una base di dati per memorizzare la carriera universitaria di uno studente), abbiamo visto come costruire uno schema concettuale ed il conseguente schema logico, tenendo conto delle tecniche di normalizzazione per garantire l'integrità del database e la rilevanza dei dati. Quanto descritto ha validità assolutamente generale: adesso, però, è il momento di implementare i nostri schemi su un DBMS e rendere interattiva la nostra base di dati.

Per queste operazioni useremo MySQL, le istruzioni SQL (Structured Query Language) e il linguaggio PHP che ci fornisce adeguati strumenti di interrogazione e gestione remota del DB.

Se qualcuno ha avuto difficoltà ad installare o configurare i programmi citati nelle scorse puntate e quelli introdotti in questa lezione (come Apache, PHP, MySQL e PhpMyAdmin), provi a fare riferimento al sito easyPHP (http://www.easyphp.org). Questo sito, di origini francese e tradotto anche in lingua italiana, è chiaro e aggiornato e fornisce in un unico pacchetto autoinstallante i programmi citati.

IL CALENDARIO DELLE LEZIONI

Lezione 1:
- PHP con un server off line
- Funzioni base e variabili
- I costrutti di controllo

Lezione 2:
- Approfondiamo PHP
- Include e require
- Funzioni e classi
- Proteggere una pagina
- Cookie e sessioni
- La funzione mail

Lezione 3: PHP e i database
- La funzione upload
- Lavorare con i file
- La gestione degli errori
- Accenni di sicurezza
- Il database system
- Siti Web dinamici
- Teoria dei database

▶ Lezione 4:
- PHP e MySQL
- MySQL, database opensource
- Costruzione e interrogazione di un database: il linguaggio SQL
- Interfaccia GUI: PhpMyAdmin
- Integrazione MySQL e PHP

La prossima puntata

Lezione 5: Gestire un sito dinamico con PHP e MySQL

2 MySQL, un database opensource

Lo schema logico di una base di dati va implementato utilizzando un DBMS (Database Management System), il componente software che consente di operare i processi di definizione, costruzione e manipolazione del database.

Per il corso, coerentemente con la scelta opensource effettuata nelle lezioni precedenti, è stato scelto MySQL, un **RDBMS** libero (*free*) distribuito con licenza GPL, basato sullo standard SQL e disponibile per ambienti Linux, Macintosh e Windows. L'acronimo non è sbagliato: la R indica la parola "Relational" a sottolineare come MySQL (così come PostgreSQL, MS Access, Oracle e altri) sia un DBMS basato sul concetto basilare di relazione tra tabelle.

MySQL nasce grazie alla svedese TcX (ora MySQL AB), una società cui serviva un DB veloce, flessibile e affidabile. Non trovandolo tra i prodotti esistenti, i progettisti svedesi decisero di crearselo distribuendolo poi con licenza GPL (vedi riquadro "Licenza GPL e licenza commerciale di MySQL"). Il risultato è un sistema in grado di gestire con efficienza e sicurezza enormi moli di dati, con una velocità di esecuzione di tutto rispetto. Ci sono ancora dei passi da fare per rendere MySQL veramente completo (vedi "il futuro di MySQL" più avanti nell'articolo), ma comunque le caratteristiche di questo database ne fanno un punto di riferimento assoluto del settore. Non solo sul Web, dove è uno dei principali attori, ma anche come DBMS per soluzioni interne ad un'attività, ad esempio per gestire la contabilità, le spedizioni, le buste paga o semplicemente per ordinare una volta per tutte la propria infinita collezione di libri. Il simbolo di MySQL è il delfino Sakila, per cui tuffiamoci nel processo di installazione del software e nella creazione del nostro primo database.

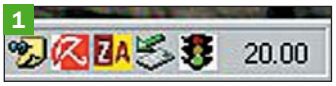
Installazione e avvio del server

MySQL è un cosiddetto database server, e quindi per poter funzionare necessita non solo del software installato ma anche di un server attivo (Apache, IIS, Xitami e via dicendo). Il nostro server è già impostato (vedi prima lezione), quindi concentriamoci sulla parte di mera installazione di MySQL. Scarichiamo da http://dev.mysql.com/downloads/ l'ultima versione stabile di *MySQL database server and standard clients* per Windows (disponibile anche nel CD Guida 1 nella cartella PDF\Corsi\PHP) e procediamo come al solito. Il processo è totalmente automatico se si accetta di installare il programma sulla cartella predefinita *c:/mysql*: approfittiamo di questa opportunità lasciando l'alternativa a chi voglia perdere qualche minuto in più per il settaggio successivo del file *my.ini*.

Terminata l'installazione facciamo doppio clic sul file *c:/mysql/bin/winmysqladmin.exe* per avviare per la prima volta il nostro server MySQL. Ci verrà chiesta una coppia user-pas-

sword: possiamo anche lasciarla in bianco dal momento che questi parametri servono solo in caso di amministrazione remota del server MySQL. A questo punto avremo accesso al pannello di gestione, rappresentato da una icona a forma di semaforo posta nella traybar (*immagine 1*). Se il semaforo è verde, siamo pronti a creare il nostro primo database.

1



Il semaforo verde *conferma l'attivazione del server MySQL*

Cliccando col tasto destro del mouse sul semaforo e scegliendo *show me* si apre la finestra di gestione: potremo vedere sulla linguetta *Environment* (*immagine 2*) l'indirizzo IP locale del nostro server (lo stesso di Apache o IIS) e sulla linguetta *my.ini Setup* i valori del file my.ini: la riga che più ci interessa è la *datadir* che ci fornisce il percorso di registrazione dei dati (utile saperlo per un backup). Sempre grazie al nostro semaforo potremo fermare e far ripartire il servizio che presiede al funzionamento del server MySQL. In alternativa per compiere la stessa operazione possiamo usare l'utility *Servizi* che abbiamo a disposizione in *Pannello di controllo, Strumenti di amministrazione*.

Client e sicurezza

Il server MySQL è stato avviato, ma da solo non basta: per la gestione e per tutte le operazioni inerenti i database abbiamo infatti bisogno di utilizzare un client. L'installazione appena effettuata ci mette a disposizione un client con una spartana interfaccia a caratteri, addirittura lanciata attraverso una finestra DOS.

La cosa non è così strana come potrebbe sembrare a prima vista: MySQL, infatti, è un'applicazione server e l'interfaccia client a caratteri consente di lavorare con qualsiasi computer indipendentemente dalle interfacce grafiche installate e permette di inviare facilmente comandi via telnet da un computer collegato in rete. Inoltre garantisce velocità e flessibilità, chiedendo solamente di conoscere pochi comandi e le basi del linguaggio SQL.

Naturalmente ci sono molte interfacce grafiche già pronte (una la vedremo tra poco) per semplificarci la vita, però almeno una volta è opportuno utilizzare lo strumento di default fornitoci, per capire le reali potenzialità di MySQL senza farci distrarre da nessun aspetto grafico.

Sulla barra di Windows clicchiamo su *Start, Esegui* e scriviamo *cmd* per aprire una console DOS. Al prompt scriviamo: *cd mysql\bin* per portarci sulla cartella di sistema di MySQL.

MySQL è studiato per applicazioni client-server, quindi, per motivi di sicurezza, utilizza la gestione dei profili di accesso degli utenti: a seconda delle autorizzazioni concesse (i cosiddetti "privilegi") gli utenti potranno, ad esempio, solo fare la gestione dei dati di database specifici oppure creare nuovi database ma senza poter accedere ad altri oppure solo inserire dati. L'amministratore del sistema, la persona cui è concesso il controllo assoluto di MySQL, è l'utente root. Windows crea automaticamente quattro profili identificati da user, password e server di provenienza (localhost e remoto). I profili, tutti generati senza password predefinite, sono:

- **root** connesso da localhost: tutti i privilegi;
- **root** connesso da remoto: tutti i privilegi;
- **utente anonimo** connesso da localhost: tutti i privilegi previsti per i database presenti nel server;
- **utente anonimo** connesso da remoto: privilegi da assegnare.

Questa configurazione è accettabile finché usiamo il DB in locale e lo curiamo solo noi, ma è chiaramente pericolosa nel momento in cui al DB avessero accesso altre persone o quando fosse messo in rete. La raccomandazione, quindi, è di cancellare subito l'utente anonimo connesso da localhost, assegnare le password a root e creare degli utenti reali cui assegnare i privilegi voluti.

Il manuale che si trova nella directory di installazione (*C:\mysql\Docs\manual.html*) è molto completo (salvato come file PDF occupa quasi 1.300 pagine di formato A4) e si rimanda alle pagine specifiche per queste operazioni. Per le esigenze didattiche di questo corso vediamo però come inserire la password "pluto" per l'utente root (connesso da localhost, ossia dal nostro server interno) scrivendo questa riga (dalla console DOS):

```
mysqladmin -u root -p password pluto
```

(verrà chiesto di inserire la password vecchia: era vuota quindi basta solo premere invio)

Fermiamo e facciamo ripartire il server MySQL per caricare i nuovi privilegi. Da adesso ci collegheremo al server e al client MySQL con l'identità di root e la password inserita.

Le prime istruzioni

Utilizzeremo l'istruzione **mysqladmin** per compiere operazioni "una tantum" che interessano il server nella sua generalità: interrogazioni sullo stato, cambiare password, ma anche creare e distruggere database (cosa possibile anche utilizzando il client) o fermare il servizio. Ad esempio per fermare il server si può scrivere:

```
mysqladmin -u root -p shutdown
```

(viene chiesta la password di root per rendere effettiva l'istruzione)

Useremo invece l'istruzione **mysql** per aprire il client e agire sul database prescelto secondo quanto previsto dai nostri privilegi di accesso. Per collegarci scriviamo:

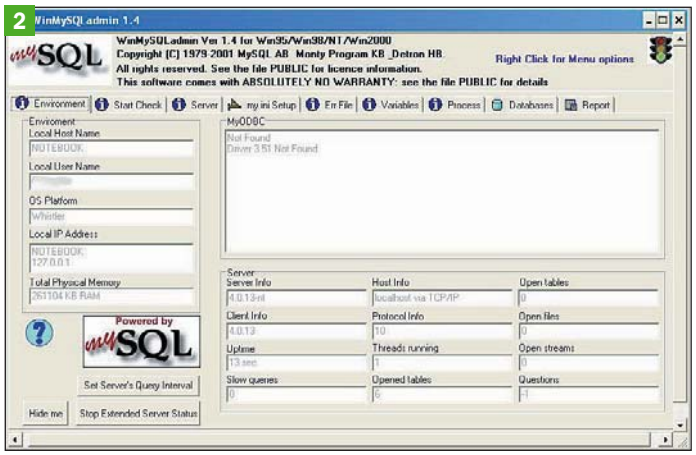
```
mysql -u root -p
```

e inseriamo la password. Se ci troviamo di fronte un prompt (*immagine 3*) del tipo:

```
mysql>
```

Siamo pronti a gestire un database. Il client accetta due tipi di inserimenti: comandi MySQL o istruzioni SQL (da concludere sempre con un segno di punto e virgola ";"). I comandi MySQL sono relativamente pochi (scrivere help per vederli) e ci interessano in particolar modo **USE** (per agire su un database specificato) e **QUIT** (per uscire dal client). Alle istruzioni SQL è invece dedicato il prossimo paragrafo. Come esempio scrivia- ▷

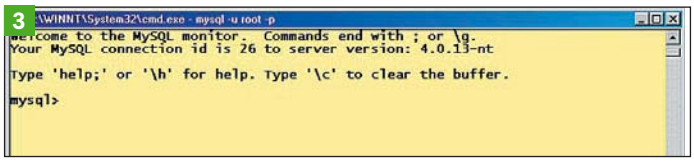
2



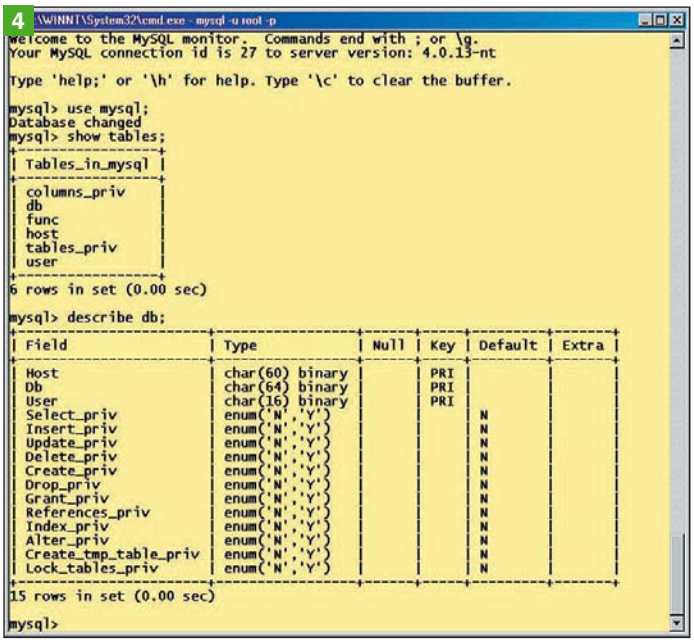
I pannelli di *configurazione del tool WinMySQLadmin*

Licenza GPL e licenza commerciale di MySQL

La licenza GPL (GNU General Public License) è la base del sistema GNU/Linux e del software libero. Compilata da Richard Stallman, questa licenza impone, tra le altre cose, di ridistribuire liberamente il codice sorgente di un programma che sia basato tutto, o in parte, su del codice soggetto a licenza GPL. È il cosiddetto "effetto virale" (ossia contagioso) della licenza GPL. Dal 2001 MySQL AB, società che detiene i diritti di MySQL ha deciso di fornire il DB con una duplice licenza: GPL e commerciale.
La versione commerciale (a pagamento) serve appunto a superare il vincolo di libera redistribuzione del codice: chiunque, pertanto, desideri usare MySQL per applicazioni di cui non vuole rendere pubblico il codice sorgente dovrà acquistare una licenza commerciale da MySQL AB.



il client mysql *è pronto ad accettare istruzioni e comandi*



Le risposte del server *alle nostre prime interrogazioni: show e describe*

▷ mo queste righe attraverso le quali accediamo al database "mysql" (è un DB installato in automatico e serve per la gestione del DBMS. Non va cancellato per nessun motivo) per poi vedere le tabelle che lo compongono e infine uscire dal client:

mysql > **USE** mysql
(accediamo al DB mysql. Attenzione è un DB di sistema)
mysql > **SHOW** tables;
(vediamo i nomi delle tabelle che compongono il DB)
mysql > **DESCRIBE** db;
(vediamo al struttura della tabella db: immagine 4)
mysql > **QUIT**
(usciamo dal client)

La prima e la quarta riga non necessitano del ";" perché sono comandi MySQL, mentre la seconda e la terza sono istruzioni SQL. Se dimenticassimo il ";" otterremmo solo l'effetto di andare a capo (col nuovo prompt "->") e potremmo continuare a scrivere l'istruzione come fossimo sulla riga precedente. La chiusura dell'istruzione necessita il ";" oppure il costrutto "\g".

Il futuro di MySQL

Pur essendo un eccellente prodotto per caratteristiche di velocità e di gestione dei dati, MySQL non è ancora completo sotto tutti i punti di vista.

L'integrità referenziale, ad esempio, è implementata solo per le tabelle di tipo **InnoDB** ma non per le **MyISAM**. Ricordiamo che l'integrità referenziale serve ad impedire che il valore di un campo di una tabella possa fare riferimento ad un valore di campo chiave non ancora esistente in una seconda tabella. Questa è una funzionalità importantissima per garantire la coerenza dei dati e non essendo ancora supportata dalle MyISAM dovremo, se vogliamo usare queste tabelle mantenendo l'integrità referenziale, creare dei controlli al momento di implementare l'interfaccia PHP.

Altro limite è il mancato supporto delle views: le "viste" sono delle tabelle virtuali generate da query e saranno utilissime per semplificare il codice e ottimizzare le ricerche. Per le prossime release MySQL AB ha già dato dei punti fermi: con la versione 4.1 troveranno supporto le sottoquery (serviranno a semplificare le chiamate nidificate di tabelle che hanno lo scopo di creare un filtro di ricerca), con la 5.0 le views e con la 5.1 l'integrità referenziale per le tabelle MyISAM.

3 Costruzione e interrogazione di un database: il linguaggio SQL

Per costruire un database e farlo funzionare adeguatamente abbiamo bisogno in primis di conoscere il tipo di dati che dovranno essere memorizzati, passo necessario per imparare a costruire una tabella. Dovremo poi assegnare il campo chiave, eventuali indici, valori predefiniti, il range di inserimento e così via. Successivamente dovremo apprendere come gestire la nostra base dati tramite le azioni di inserimento, modifica, cancellazione e interrogazione.

Un esempio è necessario, quindi partiremo dallo schema logico sviluppato nella scorsa lezione. Lo scopo è creare un database in grado di monitorare la situazione degli studenti di una ipotetica università (vedi lezione n.3 su *PC Open* di dicembre 2004, disponibile anche in formato PDF nel *CD Guida 3*).

Per compiere queste operazioni, la creazione e la gestione del database, è indispensabile conoscere il linguaggio SQL (Structured Query Language).

SQL, il cui nome originario era SEQUEL, fu inizialmente implementato nei laboratori IBM e venne poi standardizzato grazie allo sforzo congiunto di ANSI (American National Standard Institute) e ISO (International Standard Organization) che portò alla prima release standard denominata SQL1 (ANSI 1986). A questa sono seguite SQL2 (SQL-92), SQL3 (SQL:1999) e SQL:2003. Indipendentemente dalla release, la cosa fondamentale è avere a disposizione per tutti i database un linguaggio standard avente due specifiche caratteristiche:

- è un linguaggio di **definizione** dei dati (DDL, ossia DataDefinition Language)
- è un linguaggio di **manipolazione** dei dati (DML, ossia Data Manipulation Language)

Nei prossimi paragrafi vedremo nello specifico come usare SQL nel nostro RDBMS.

Tipi di dati

MySQL è in grado di gestire qualsiasi tipo di dato, e il termine "qualsiasi" comprende anche dati multimediali come filmati, immagini, file audio. È chiaro che l'efficienza del database sarà direttamente proporzionale alla corretta scelta del tipo di dato di ogni campo (colonna): minore sarà lo spreco di spazio, più veloci saranno tutte le operazioni compiute sul DB. Fondamentale, quindi, è conoscere i tipi di campo che abbiamo a disposizione e scegliere quello più adatto (attributi compresi) alle nostre esigenze. Rimandando al manuale di MySQL per i dettagli, mentre nel box "I formati dei dati" riassumiamo brevemente i tipi di dato.

La costruzione di una tabella

MySQL consente di creare diversi "tipi" di tabelle alle qua-

li corrispondono performance di risposta diverse. Le due più diffuse sono le **MyISAM** e le InnoDB. Le prime usano poca memoria e sono gestite in maniera velocissima, però non hanno ancora tutte le funzionalità che si potrebbero desiderare, mentre le **InnoDB**, pur essendo più complete, sono più lente e chiedono un maggior carico di memoria.

Inizialmente useremo le tabelle MyISAM, visto che sono quelle accettate di default. Apriamo un client MySQL collegandoci come root (in modo da avere tutti i permessi possibili): al prompt la prima operazione da fare è creare il database UNIVERSITAS:

```
NB: in MAIUSCOLO sono indicate le
istruzioni specifiche di SQL
mysql> CREATE DATABASE universitas;
```

Se volessimo cancellarlo l'istruzione sarebbe:

```
mysql >DROP DATABASE universitas;
```

Dobbiamo creare quattro tabelle: una con i dati degli studenti, una con i dati dei professori, una con i dati dei corsi e una con i risultati degli esami. Ogni tabella andrà inizializzata indicandone prima il nome e quindi tutti i campi che la compongono, tipi di dato e attributi compresi. La tabella STUDENTE sarà quindi implementata in questo modo (le istruzioni di creazione della tabella sono su più righe solo per rendere più comprensibile il listato):

```
mysql> USE universitas;
(entriamo nel database appena creato:
all'apertura del client è sempre
necessario indicare quale DB useremo
come riferimento per le istruzioni
successive)

mysql> CREATE TABLE studente (
-> matricola CHAR(8) NOT NULL,
-> nome VARCHAR(30) NOT NULL,
-> cognome VARCHAR(30) NOT NULL,
-> data_nascita DATE,
-> tesi_con CHAR(13),
-> PRIMARY KEY(matricola),
-> INDEX(cognome, nome);
-> );
mysql> EXPLAIN studenti;
(per vedere la struttura della tabella
appena costruita)
```

Abbiamo creato una tabella definendo tutti i campi (due hanno un numero di caratteri fissi, due hanno un numero di caratteri variabili, uno è una data di nascita), indicando il campo chiave (la matricola) e come indicizzare la tabella (prima per cognome e poi per nome). Gli indici sono fondamentali per abbreviare le fasi di ricerca, ma attenzione che indicare troppi indici porterebbe ad un peggioramento delle prestazioni.

La tabella ESAMI sarà invece così costruita:

```
mysql> CREATE TABLE esame (
-> matricola VARCHAR(8) NOT NULL,
-> cod_corso SMALLINT UNSIGNED
   NOT NULL,
-> voto TINYINT NOT NULL,
-> lode ENUM(no,si) DEFAULT no NOT
   NULL,
-> data DATE NOT NULL,
-> PRIMARY KEY (matricola,
   cod_corso),
-> );
```

Qui si è introdotto un valore booleano grazie al tipo di dato ENUM (i valori possono essere solo "no" o "si". Il primo è il valore di default). Volendo essere precisi, anche il campo voto avrebbe potuto essere un ENUM con valori compresi tra 18 e 30, ma visti i tempi di cambiamento lasciare libero il campo è forse meglio.

Capito il sistema, lasciamo ai lettori il compito di costruire le altre tabelle: l'unica accortezza è porre il campo codice nella tabella CORSO (*immagine 5*) come TINYINT con l'opzione AUTO_INCREMENT (è una chiave). Chi non volesse scrivere tutto il codice può approfittare del file *universitas_db.sql*: i file *.sql* sono dei file testo che consentono di effettuare, tra le altre cose, la procedura di backup. Grazie a questi file siamo in grado di esportare (e importare) la struttura del database (o della singola tabella) ed eventualmente anche i dati inseriti. Questa opportunità si rivelerà utilissima non solo per salvare i dati, ma anche per esportare su un server in Internet il nostro fiammante database creato off line sul PC di casa: tutti i provider fornitori di questo servizio consentono, infatti, sia l'importazione che l'esportazione di file .sql.

I comandi da eseguire, dal prompt del DOS una volta usciti dal client, sono:

```
mysqldump -u root -ppluto universitas
> universitas_db.sql
(backup di tutte le tabelle del database
"universitas", al momento senza dati.
Il file creato si chiamerà
"universitas_db.sql")
```

Attenzione: a causa di un ▷

5 C:\WINNT\System32\cmd.exe - mysql -u root -p

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 28 to server version: 4.0.13-nt

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> use universitas;
Database changed
mysql> explain corso;
+-----------+---------------------+------+-----+---------+----------------+
| Field     | Type                | Null | Key | Default | Extra          |
+-----------+---------------------+------+-----+---------+----------------+
| cod_corso | tinyint(3) unsigned |      | PRI | NULL    | auto_increment |
| nome      | varchar(100)        |      |     |         |                |
+-----------+---------------------+------+-----+---------+----------------+
2 rows in set (0.03 sec)

mysql>
```

***L'istruzione* explain** *ci mostra la composizione della tabella* CORSO

I formati dei dati

Il primo formato da esaminare è quello **numerico**, suddivisibile in numeri interi e decimali. Per i numeri interi (positivi e negativi) i campi disponibili (in ordine crescente di occupazione di byte) sono: TINYINT (1 byte), SMALLINT (2 byte), MEDIUMINT (3 byte), INT (4 byte) e BIGINT (5 byte). Questo significa che un dato SMALLINT avrà un range totale di 2(2x8) = 65.536 valori divisi tra negativi e positivi (da – 32.768 a +32767, 0 compreso). Se gli attribuissimo l'opzione UNSIGNED i valori sarebbero solo positivi, quindi da 0 a 65.535. Un altro importante attributo dei campi numerici interi è AUTO_INCREMENT: ogni nuovo record incrementerà di una unità il valore del record precedente, e quindi diventa un campo chiave. Per i numeri decimali abbiamo a disposizione FLOAT, DOUBLE e DECIMAL (M,D). L'ultimo è particolarmente utile perché consente di definire il numero di cifre decimali (D) e il numero massimo (M) di caratteri del numero, comprensivo di segno positivo o negativo e della virgola.

Secondo formato è quello **testuale** (**stringa**). Se la stringa contiene meno di 255 caratteri possiamo usare CHAR (M) o VARCHAR (M), con M numero massimo di caratteri. La differenza è che un campo CHAR occupa sempre M byte indipendentemente dalla stringa inserita nel record, mentre VARCHAR occupa un numero di byte uguale al numero di caratteri (più uno) della stringa. Se il testo è maggiore di 255 caratteri useremo TINYTEXT, TEXT, MEDIUMTEXT e LONGTEXT. Quest'ultimo consente di memorizzare una stringa di testo di 4 GB! L'attributo BYNARY trasforma il testo in stringa binaria: la conseguenza è che MySQL distinguerà, nell'ordinamento delle ricerche, i caratteri minuscoli dai maiuscoli.

Altro tipo di dato sono le **stringhe di testo predefinite**, rappresentate da ENUM e SET. In questo caso il valore del campo (unico nel caso di ENUM, anche multiplo nel caso di SET) può essere scelto solo tra valori predefiniti da noi in fase di creazione del campo.

Il formato **multimediale** è registrato come stringa di caratteri binaria nelle forme di TINYBLOB, BLOB (Binary Large Object), MEDIUMBLOB e LONGBLOB (4 GB di spazio!).

Date e ore possono essere salvate in forma completa o parziale. Nel primo caso abbiamo a disposizione TIMESTAMP (4 byte occupati, ma date limitate dal 1970 al 2037 su macchine a 32 bit) e DATETIME (8 byte occupati, dall'anno 1000 al 9999). Il formato di TIMESTAMP predefinito occupa 14 caratteri ed è memorizzato come AAAAMMGGHHMMSS; esistono comunque altri formati minori che possono essere maggiormente adatti alle esigenze di costruzione del DB. Il formato di DATETIME è variabile e dipende dai dati effettivamente inseriti. Date e ore "parziali" sono gestibili, pur nei limiti loro assegnati di default, con DATE (dal 01-01-1000 al 31-12-9999), TIME (consente valori negativi e supera abbondantemente le 24 ore: il formato è usabile, quindi, anche per calcoli sugli angoli) e YEAR (dall'anno 1901 al 2155).

Un piccolo spazio merita l'**attributo NULL** che può essere assegnato ad ogni tipo di dato. Cos'è un valore null? Null vuol dire proprio "nessun valore" e non è né una stringa vuota (per un testo) né il valore zero (per i numeri). Se sappiamo che un campo non potrà mai essere null è bene omettere questo attributo: in caso contrario verrebbe sempre salvato un bit extra per ogni campo che potrebbe essere null.

bug di alcune versioni (la 4.0.13 ad esempio) la password di root (la nostra "pluto") in questa istruzione va inserita immediatamente dopo l'opzione "-p" senza spazi intermedi. In generale è sempre comunque possibile inserire la password in questo modo in ogni istruzione di comando dalla shell del DOS. Se la versione che avete installato è esente dal bug, fate a meno di inserire la password sulla riga di comando, scrivendola invece solo quando richiesto come visto in precedenza. Altri comandi che potremmo scrivere sono:

mysqldump -u root -ppluto **-d** universitas studente corso > universitas_parz.sql
(backup dei soli dati delle tabelle 'studente' e 'corso' del database "universitas": il file creato è "universitas_parz.sql". L'opzione -t avrebbe fatto il backup della sola struttura senza i dati. Va inserita la password di root)

mysql -u root -p universitas < universitas_db.sql
(ripristino o inserimento, nel database già creato "universitas", delle tabelle oggetto di backup contenute nel file "universitas_db.sql". Le tabelle non vengono sovrascritte)

Eseguendo l'ultima istruzione avrete il vostro database universitas completo della struttura di tutte le tabelle e pronto ad essere gestito. Se il file *universitas_db.sql* non fosse nella cartella "bin" di mysql (quella dove ci troviamo per eseguire il comando), andrà indicato il percorso assoluto completo (ad esempio: *D:\backup\mio\universitas_db.sql*).

Da questo momento in poi daremo per scontato di essere connessi al client MySQL e di trovarci nel database universitas (istruzione *use universitas*).

La gestione dei dati

Il database è adesso solo uno scheletro vuoto, ma questa era la parte più difficile. Gestire i dati è, al confronto, un'operazione piuttosto agevole e vedremo come fare l'inserimento, la cancellazione la modifica dei medesimi usando sempre le istruzioni SQL.

L'inserimento può essere fatto in più modi:

INSERT INTO tabella **SET** campo_1= valore_1, campo_2=valore_2,...;

INSERT INTO tabella **VALUES** (campo_1, campo_2, ...);

La differenza tra le due istruzioni è che la prima consente di indicare (nell'ordine che preferiamo) solo i campi che vogliamo effettivamente riempire con dei valori (gli altri saranno completati in accordo con le caratteristiche definite per i campi della tabella), mentre la seconda istruzione ci impone di scrivere tutti i valori dei campi di un record (comprese quindi stringhe vuote, valori null o predefiniti) esattamente nell'ordine in cui è costruita la tabella. L'utilizzo dell'una o dell'altra istruzione dipende dal tipo di inserimento che dobbiamo fare.

Poniamo di voler inserire un valore nella tabella studenti:
mysql> INSERT INTO studente SET cognome="Rossi", nome="Antonio", matricola="912345IG", data_nascita="19780427";
(ho inserito nella tabella studente, nell'ordine che volevo, solo i dati cui volevo attribuire un valore. Svantaggio è che devo ricordare i nomi corretti dei campi)

Equivalente sarebbe stata l'istruzione:
mysql> INSERT INTO studente VALUES ("912345IG", "Antonio", "Rossi", "19780427", null);
(qui avrei dovuto inserire tutti i valori nel giusto ordine, compreso l'ultimo valore null perché questo studente ancora non è in tesi con alcun professore. Il vantaggio in questo caso è che non devo inserire i nomi dei campi)

Chiaramente se tentassimo di inserire una matricola (campo chiave) già esistente ci verrebbe segnalato un errore (*immagine 6*). Notiamo che l'inserimento della data va fatto nel formato corretto (AAAAMMGG) e che ad un eventuale professore con cui si fa la tesi ci si deve riferire tramite il codice fiscale, ossia il campo chiave della tabella "prof".

Non ci viene dato errore se inseriamo nel campo "tesi_con" un valore di riferimento alla tabella "prof" non esistente. La stessa cosa accade se, al momento di registrare il voto di un esame, inseriamo una matricola errata o un codice di un corso inesistente. Questo, è dovuto al mancato supporto dell'integrità referenziale delle tabelle MyISAM.

Se volessimo avere questo supporto avremmo dovuto definire tutte tabelle (ogni tabella ci sono riferimenti a valori di chiave esterni) utilizzando il "type" InnoDB e i comandi FOREIGN KEY - REFERENCES di SQL:
mysql> CREATE TABLE studente (matricola CHAR(8) NOT NULL, nome VARCHAR(30) NOT NULL, cognome VARCHAR(30) NOT NULL, data_nascita DATE, tesi_con CHAR(13), PRIMARY KEY(matricola), INDEX (tesi_con, cognome, nome), FOREIGN KEY (tesi_con) REFERENCES prof(cod_fisc)) TYPE=INNODB;
(il riferimento alla tabella prof è dato con l'istruzione FOREIGN KEY: bisogna però ricordare di indicare il campo "tesi_con" anche negli indici)

Così facendo siamo obbligati ad inserire nel campo "tesi_con" un valore già presente nella tabella "prof". A titolo di esempio, nel CD trovate il file *universitas_innodb.sql* che contiene la struttura del database costruita con tabelle InnoDB e tutte le integrità referenziali rispettate. Caricando queste tabelle si noterà l'impossibilità di caricare dati non coerenti con quanto definito in termini di integrità referenziale. Lo spazio occupato dalle tabelle, però, aumenta considerevolmente e calano le performance di velocità.

Torniamo al DB con tabella MyISAM: il file *universitas_dati.sql* contiene dei valori di esempio da caricare per popolare il database. Carichiamo questo file come abbiamo già imparato e proseguiamo la lezione.

La cancellazione dei dati di una tabella può essere fatta in maniera "completa" ossia cancellando tutti i record oppure, ed è il caso più frequente, indicando una condizione da rispettare per cancellare la riga o le righe in questione:
DELETE FROM tabella [**WHERE** condizione];
(la clausola where è opzionale: se c'è vengono cancellati solo i record che rispettano la condizione, altrimenti vengono cancellati tutti i record della tabella)

Ad esempio se volessi cancellare tutti gli studenti di cognome "Rossi", l'istruzione sarà:
mysql> DELETE FROM studente WHERE cognome="Rossi";

La modifica dei dati può essere fatta su tutti i record di una tabella o su delle tuple selezionate con un'apposita condizione:
UPDATE tabella **SET** campo_1= aggiornamento_1, campo_2= aggiornamento_2, ... [WHERE condizione];
(la clausola where è opzionale e serve a restringere il numero di record da aggiornare)

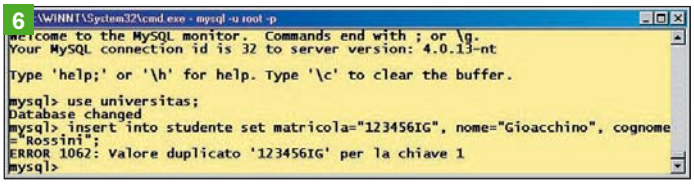
Un esempio potrebbe essere un cambio del professore con cui fare la tesi da parte di uno studente ben preciso, identificato con la sua matricola:
mysql> **UPDATE** studente **SET** tesi_con="BBBCCC52B41A123B" WHERE matricola="123456IG";

Altro esempio: supponiamo di dover trasformare tutti i voti (al momento espressi in trentesimi) in voti centesimali. Basta una modifica su tutta la tabella "esame" (*immagine 7*):
mysql> **UPDATE** esame **SET** voto=voto*10/3;

Esiste poi anche il comando REPLACE che combina le caratteristiche di INSERT e UPDATE e quindi può fare al contempo funzioni di inserimento e aggiornamento.

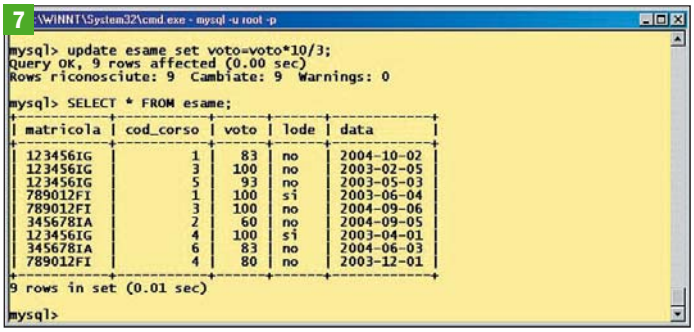
Interrogazione dei dati (query)

L'interrogazione (query, dal verbo "to query" = interrogare) del database è senza dubbio l'operazione più utilizzata nella gestione di un database, ed è al contempo facile e complessa. La facilità deriva dalla strutturazione dell'interrogazione standardizzata da SQL, mentre la sua complessità deriva dalla strutturazione relazionale della base di dati che ci impone spesso di dover estrar-

6


MySQL ci indica un errore *se tentiamo di inserire due studenti con la stessa matricola*

7



Risultato di un update *completo sulla tabella esame*

re informazioni da più tabelle tra loro collegate. Quest'ultima operazione prende il nome di **JOIN**.

Una query nella sua struttura più completa è così scritta (le clausole tra parentesi quadre sono opzionali):

SELECT campi_tabella/e
FROM tabella/e
[**WHERE** condizione];
[**GROUP BY** raggruppamento]
[**HAVING** condizione_raggruppamento]
[**ORDER BY** campi_tabella/e ASC/DESC]

Alcuni esempi sono:

```
mysql> SELECT cognome, nome FROM
  studente;
```
(selezioniamo solo le colonne cognome e nome, con quest'ordine, dalla tabella studente)

```
mysql> SELECT * FROM studente;
```
(con l'asterisco selezioniamo tutte le colonne dalla tabella studente)

```
mysql> SELECT * FROM studente
  WHERE cognome="Rossi";
```
(selezioniamo solo i record della tabella studente per i quali il valore di "cognome" è "Rossi")

Spesso, però, avremo bisogno di operare un join, ossia lavorare con più tabelle correlate. Supponiamo di voler sapere quali sono gli esami sostenuti dallo studente con matricola 123456IG. Una query come la seguente è formalmente esatta perché ci dà i codici dei corsi, però non ci restituisce i nomi degli esami corrispondenti (*immagine 8*):

```
mysql> SELECT cod_corso, voto, lode
FROM esame WHERE
matricola="123456IG" ORDER BY
data;
```

quindi per dare leggibilità al risultato dobbiamo fare un join ("to join" = unire) per sostituire al codice il corrispondente nome del corso ricavato dalla tabella "corso" (*immagine 9*):

```
mysql> SELECT corso.nome,
esame.voto, esame.lode
-> FROM corso, esame
-> WHERE corso.cod_corso=esame.
   cod_corso
->AND esame.matricola="123456IG"
-> ORDER BY data;
```

Un join, quindi, si fa estraendo i dati utili da più tabelle (i nomi dei campi vanno indicati anteponendo il nome della tabella cui fanno riferimento) e imponendo, nella clausola **WHERE**, una condizione di uguaglianza tra determinati valori delle tabelle: in questo caso il join è fatto sul campo che identifica il codice numerico del corso, campo chiave esterno per la tabella "esame" e campo chiave effettivo per la tabella "corso". Abbiamo introdotto anche la clausola **ORDER BY** che serve a ordinare i risultati (di default in maniera ascendente, altrimenti si deve aggiungere DESC) dell'interrogazione.

Introduciamo la parola chiave **DISTINCT**: serve ad eliminare da un risultato eventuali tuple duplicate che si otterrebbero. Supponiamo di volere sapere il numero di matricola degli studenti che hanno superato almeno un esame: queste matricole saranno presenti nella tabella esame, ma una query semplice presenterebbe duplicati se uno studente avesse passato più di un esame. Per evitarlo e avere il risultato voluto si scriverà:

```
mysql> SELECT DISTINCT matricola
FROM esame;
```

Molto utili, infine, sono le funzioni aggregate che agiscono sulla totalità dei record estratti, secondo una precisa condizione di raggruppamento: **COUNT** (conta il numero di record), **SUM** (somma i valori), **MAX** (trova il valore più alto), **MIN** (trova il valore più basso) e **AVG** (calcola la media). Se non sono specificati raggruppamenti queste funzioni agiranno tenendo conto di tutti i record estratti: ad esempio per sapere quanti sono gli studenti nati dal 1980 in poi:

```
mysql> SELECT COUNT(*) FROM
studente WHERE data_nascita >=
"19800101";
```

Il raggruppamento si rivelerebbe utilissimo se volessimo sapere la media voti di ogni studente (*immagine 10*). Uso anche la funzione alias **AS** che serve ad assegnare un nome nuovo momentaneo (in genere per motivi di maggiore chiarezza) ad un campo (esistente o creato con la query) o ad una tabella:

```
mysql> SELECT matricola, AVG(voto)
AS media
-> FROM esame
-> GROUP BY matricola;
```

Sulla clausola di raggruppamento potremmo porre delle ulteriori condizioni tramite HAVING per rendere ancora più interessanti le nostre interrogazioni.

Ad esempio poniamo di voler conoscere il nome degli studenti che hanno fatto almeno tre esami:

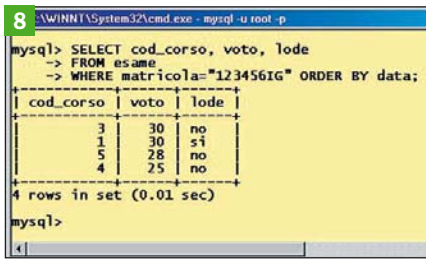
```
mysql> SELECT studente.nome,
studente.cognome FROM studente,
esame
-> WHERE studente.matricola=
   esame.matricola
-> GROUP BY esame.matricola
-> HAVING COUNT(*)>="3";
```
("having" agisce solo su una clausola "group by")

Queste sono le principali istruzioni di una query. SQL comprende poi una ulteriore serie di utili operatori su insiemi (UNION, EXCEPT, INTESECT, CONTAINS), operatori "quantificatori" (ALL, ANY, IN, NOT IN, SOME) e operatori di verifica (EXISTS, NOT EXISTS, UNIQUE) che si lasciano al libero studio degli interessati ad approfondire l'argomento.

Operazioni sulle query: comandi principali

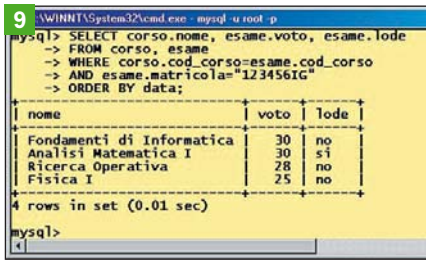
Sui record estratti da una query si può agire in un secondo momento, ad esempio intervenendo coi costrutti di PHP, oppure si può agire direttamente al momento dell'estrazione del risultato. SQL, infatti, fornisce una libreria completa

8



Una query esatta *ma che non consente una leggibilità immediata*

9

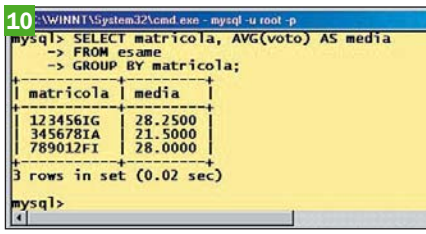


La stessa query *con un join diventa molto più significativa e leggibile*

di operatori aritmetici, logici, di confronto, di ricerca e binari. Abbiamo a disposizione funzioni numeriche (ABS, LOG, COS, ...), funzioni di controllo dei cicli (IF, IF NULL, CASE), funzioni per agire sulle stringhe (CONCAT, UPPER, ...), funzioni di data e ora e funzioni per la sicurezza (PASSWORD, ENCRYPT,...). Ad esempio per estrarre le date di nascita degli studenti rendendole "leggibili", potremmo usare (*immagine 11*):

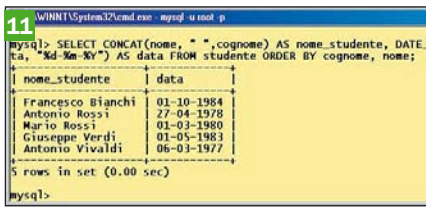
```
mysql> SELECT CONCAT(nome, "
  ",cognome) AS nome_studente,
  DATE_FORMAT(data_nascita,
  "%d-%m-%Y") AS data
-> FROM studente
-> ORDER BY cognome, nome;
```

10



Come calcolare *la media voti usando la clausola di raggruppamento e la funzione AVG di SQL*

11



Il risultato di una query *può essere sottoposto a varie operazioni prima di essere stampato*

4 Un'interfaccia GUI: PhpMyAdmin

Finora, per i motivi già scritti di universalità, abbiamo usato l'interfaccia a caratteri per agire su MySQL, però in certe occasioni un'interfaccia grafica può aiutarci a compiere certe operazioni di cui, magari, non ricordiamo perfettamente la costruzione.

Introduciamo quindi una utile GUI (*Graphic User Interface*) rilasciata sotto licenza GPL e totalmente scritta in PHP per interagire con MySQL: **PhpMyAdmin**.

L'installazione non comporta problemi: è sufficiente, infatti, scompattare il file zippato (lo trovate nel CD oppure su http://www.phpmyadmin.net/home_page/) in una cartella che chiameremo "phpMyAdmi" da porre nella "Document Root" del server Apache (ossia dove finora abbiamo posto le nostre pagine PHP per verificarle off-line, la cartella dove il nostro server va a leggere quando sulla barra dell'indirizzo scriviamo, ad esempio, http://localhost o http://127.0. 0.1).

Per far funzionare phpMyAdmin entriamo nella cartella di installazione tramite *Gestione Risorse* e apriamo, con un qualsiasi editor di testo, il file *config.inc.php*. Cerchiamo la riga contenente *$cfg['PmaAbsoluteUri']* e dopo il segno di uguaglianza indichiamo il percorso da raggiungere tramite server:

```
$cfg['PmaAbsoluteUri'] =
   'http://127.0.0.1/phpMyAdmin';
(o http://localhost/phpMyAdmin o ...)
```

Poi immediatamente dopo cerchiamo

```
$cfg['Servers'][$i]['host']
```

e scriviamo l'indirizzo IP del nostro server MySQL interno:

```
$cfg['Servers'][$i]['host'] = "localhost"
(o 127.0.0.1 o ...)
```

e poi di fianco a

```
$cfg['Servers'][$i]['auth_type']
```

scriviamo:

```
$cfg['Servers'][$i]['auth_type'] = "http"
```

A questo punto apriamo il nostro browser preferito e scriviamo l'indirizzo dove è installato phpMyAdmin: sarà qualcosa del tipo http://127.0.0.1/phpMyAdmin.

Si aprirà una finestra di richiesta di accesso. Indicando user e password (quelle di root o quelle di un utente precedentemente creato) avremo accesso al pannello di controllo (*immagine 12*) da cui possiamo partire per fare praticamente tutto quello che vogliamo (se ne abbiamo i permessi) su database, tabelle e singoli dati. Possiamo inserire i privilegi degli utenti, esportare e importare database salvati in formato .sql o .zip, scrivere query aiutati dalla grafica, inserire, modificare e cancellare dati, controllare lo stato del server MySQL, vedere l'ingombro delle tabelle e la loro velocità di risposta, e agire anche su dei comandi MySQL per analizzare e ottimizzare le tabelle della base di dati.

PhpMyAdmin è un ottimo strumento e vale sicuramente la pena usarlo per applicazioni interne, ma ricordate che il vostro provider potrebbe fornirvi un'interfaccia grafica diversa, o addirittura non fornirvela affatto.

Per tutte le opzioni di configurazione, veramente molte, si rimanda al manuale fornito col programma.

12

***Il pannello** di controllo di PHPMyAdmin*

5 L'integrazione fra MySQL e PHP

Abbiamo imparato a costruire un database partendo da schemi puramente concettuali per arrivare allo schema logico generale e infine all'implementazione fisica all'interno del Relational DBMS scelto per ospitare i dati. Siamo in grado di popolare il database e di gestire aggiornamenti, cancellazioni e interrogazioni anche di una certa complessità.

Il problema è che solo noi siamo in grado di farlo, e comunque non è sempre facile ricordare tutti i passaggi. Ma se ci fossero, ed è assolutamente normale sia così, altre persone coinvolte nella gestione della base di dati? E se dovessimo usare i dati "grezzi" estratti da MySQL per fornire informazioni a degli utenti esterni? Non possiamo certo pretendere che tutti imparino il linguaggio SQL per interrogare un database, anche perché comunque dovremmo prima spiegare loro gli schemi alla base della costruzione della base di dati.

Risolvere questi problemi equivale a fare l'ultimo passo verso l'ambiente "globale" descritto nella lezione 3 e quindi creare un completo Database System. Per fare questo dobbiamo unire le potenzialità di PHP a quelle di MySQL, allo scopo di creare delle soluzioni user-friendly di gestione del database. Nel caso particolare del nostro corso, il Web developer PHP dovrà essere in grado di:

- creare un database efficiente o comunque capirne il funzionamento nel caso l'architetto del DB sia un'altro professionista;
- interrogare il database utilizzando SQL;
- utilizzare i record risultato per comporre informazioni e/o pagine Web;
- fornire un facile sistema di gestione al web content.

PHP è un linguaggio fortemente votato al connubio con un database e può dare una marcia in più al Web. PHP, inoltre, non è "sposabile" solo con MySQL: potremo altrettanto facilmente usarlo per interfacciarci con database come DB2, Firebird, MSSQL, Oracle, PostgreSQL, Sybase e anche, tramite i driver ODBC, con Microsoft Access.

La prima cosa da fare è connettersi al database.

Connessione al server MySQL

È piuttosto semplice effettuare la connessione al server MySQL che ospita il nostro database Universitas. Sono richiesti tre parametri: indirizzo del server MySQL, user e password di un utente (root o altro):

```
mysql_connect (server, user,
   password);
```

In caso di successo la funzione restituisce un identificativo di connessione che rimarrà invariato fino alla chiusura della stessa, mentre in caso negativo viene restituito un valore *false*.

Effettuata la connessione al server, si deve indicare su quale database vogliamo agire. Con l'interfaccia a caratteri si usava il comando *use*, mentre con PHP utilizziamo:

```
mysql_select_db (nome_db,
identificativo di connessione);
```

Il risultato sarà un valore *true* o *false* (se il DB non esiste o se non si è riusciti ad effettuare la connessione al server), e nel primo caso il database scelto sarà attivo per tutte le prossime azioni.

Per connetterci al database universitas del nostro server locale (come utente root) possiamo scrivere un file PHP come quello visibile nel codice sorgente della pagina *connessione_prima.php*. La pagina ci dice se siamo connessi al database universitas, e in caso di errore (provate ad esempio a bloccare il server MySQL o a cancellare il database universitas) stampa una riga di scuse invece del messaggio di errore. Le righe di connessione sono visibili nel listato *connessione_prima.php*.

Le connessioni mysql_connect (non persistenti) restano aperte fino alla chiusura della pagina oppure fino alla chiamata della specifica istruzione che chiude la connessione indicata o comunque l'ultima aperta:

```
mysql_close();
// essendo senza argomenti, la
```

funzione chiuderà l'ultima connessione aperta

listato connessione_prima.php

```
<?php  //Creo una connessione con il database universitas
$nome_db = "universitas"; $id_connessione=@mysql_connect("127.0.0.1","root","pluto") or die("Non è possibile accedere al server");
// mysql_connect apre la connessione al server, e mysql_select_db la sfrutta per accedere ad uno specifico database
@mysql_select_db($nome_db,$id_connessione) or die("Non è stato possibile accedere al database");
?>
```

listato query_uno.php

```
$q="SELECT nome, cognome, matricola, date_format(data_nascita,'%d-%m-%Y') AS data FROM studente ORDER BY cognome, nome";$id_ris=@mysql_query($q) or die("Non è possibile eseguire la query");
// mysql_query restituisce un identificativo di risorsa. mysql_fecth_array estrae come array la prima riga del risultato della query. Con un ciclo while estraggo tutte le righe della query. Uso o chiavi numeriche o chiavi corrispondenti ai nomi usati nella select

while ($record=mysql_fetch_array($id_ris)) {
echo "Matricola: ".$record[matricola]."<br>$record[1] $record[0] - data di nascita: ".$record[data]."<p>";
}
```

listato query_due.php

```
$q="SELECT nome, cognome, matricola, date_format(data_nascita,'%d-%m-%Y') AS data FROM studente ORDER BY
    cognome, nome";
$id_ris=@mysql_query($q) or die("Non è possibile eseguire la query");
$numero_righe=mysql_num_rows($id_ris);
echo "Gli studenti presenti nel DB sono $numero_righe<p>";
mysql_data_seek($id_ris,$numero_righe-1);
// mysql_data_seek sposta il puntatore del risultato all'ultimo record, e mysql_fetch_array estrae un array corrispondente a questa posizione
$record=mysql_fetch_array($id_ris);
echo "Ecco l'ultimo studente presente nel database:<br>Matricola:
".$record[matricola]."<br>$record[1] $record[0] - data di nascita: ".$record[data]."<p>";
```

Interazione PHP-MySQL

La libreria MySQL di PHP consente di inviare query e comandi SQL e inoltre fornisce una serie di utili funzioni con le quali possiamo interrogare il server, interagire con esso e valutare le prestazioni del database.

L'istruzione fondamentale è **mysql_query**: grazie ad essa possiamo inviare una query al database attivo (o specificare una connessione diversa). Chiaramente mysql_query ci consentirà di effettuare sul database solo operazioni coerenti col nostro profilo utente.

La tipica costruzione di una query in PHP è:

```
$q = "SELECT * FROM studente";
$ris = mysql_query ($q);
```

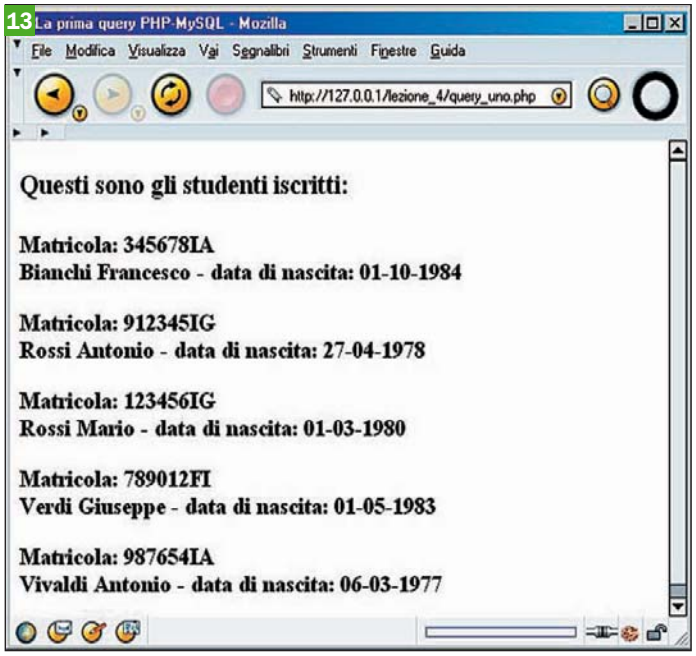
Il valore di $ris può essere:

- false in caso di errore: la query potrebbe essere errata;
- un identificativo di risorsa se la query è del tipo SELECT, SHOW, EXPLAIN o DESCRIBE. In questo caso nulla viene detto sul risultato ottenuto (ad esempio non si sa il numero di righe restituito);
- true in tutti gli altri casi: ad esempio con UPDATE, DELETE, DROP TABLE, ecc.

Un identificativo di risorsa può essere passato come argomento alla funzione **mysql_fetch_array** la quale carica tutti i campi del primo record del risultato in un array.

Per richiamare i valori dell'array possiamo fare riferimento alla solita chiave numerica (il primo elemento ha chiave zero) oppure usare come chiave il nome del campo dedotto dalla query. Una successiva chiamata alla stessa funzione mysql_fetch_array caricherebbe il secondo record, e così via fino a raggiungere l'ultimo record dopo il quale verrebbe restituito un valore false. Un ciclo while, quindi, è perfetto per consentirci di recuperare tutti i record di una qualsiasi query di selezione (listato *query_uno.php*), come si può vedere aprendo la pagina *query_uno.php* che elenca gli studenti presenti nel DB (*immagine 13*).

Se volessi recuperare il risultato di una riga ben specifica, ad esempio l'ultima, dovrei usare altre due funzioni: **mysql_num_rows** che estrae il ▷



***il risultato di una query SQL** recuperato con istruzioni PHP*

numero di righe risultanti dalla query, mentre **mysql_data_seek** sposta il puntatore del risultato ad una riga indicata nei suoi attributi (in questo caso la riga sarà l'ultima). L'utilizzo sarà chiarito dal seguente esempio (listato *query_due.php*):

Se vogliamo dare maggiore interattività alle nostre pagine, possiamo iniziare con un esempio utile e semplice: selezionare da una casella a discesa uno studente inserito nel database per ricavare tutte le informazioni su di lui (anni, esami sostenuti, media, se è in tesi con qualche professore). Costruiamo due pagine: la prima servirà alla selezione dello studente (*stud_select.php*), mentre nella seconda "stamperemo" la sua scheda (*stud_cv.php*). La query della prima pagina assomiglia a quanto visto nel listato *query_uno.php*, però abbiamo concatenato tre campi (matricola, cognome e nome). Nel tag "select" del form passeremo questo campo come valore da visualizzare, ma alla scelta fatta corrisponderà il valore della matricola che è il campo chiave (listato *stud_select.php*).

Selezionato lo studente, nella seconda pagina si tratta solamente di far eseguire al server MySQL, tramite PHP, alcune query con join: il listato *stud_cv.php* riprende solo una delle tre query previste (si rimanda al file per il codice completo). Il risultato, con una formattazione di massima, è visibile nell'*immagine 14*.

listato stud_select.php

```
<select name="stud_nome">
 <?php
   while ($record=mysql_fetch_array($id_ris)) {
       echo "<option value=".$record[matricola].">".$record[0]."</option>";
   }
  mysql_close();
 ?>
</select>
```

listato stud_cv.php

```
// Terza operazione: se lo studente è in tesi, stampare i dati del professore che lo segue
if (!is_null($record_1[tesi_con])) {
   $q_3="SELECT prof.cognome AS cognome, prof.nome AS nome, prof.dipartimento AS dipartimento FROM prof,studente
   WHERE studente.matricola='".$record_1[matricola]."' AND prof.cod_fisc=studente.tesi_con";
     $id_ris_3=@mysql_query($q_3) or die("Non è possibile eseguire la query");
     $record_3=mysql_fetch_array($id_ris_3);
     echo "<h3>Lo studente è in tesi col prof. ".$record_3[cognome]." ".$record_3[nome]." del Dipartimento di
       ".$record_3[dipartimento]."</h3>";
}
```

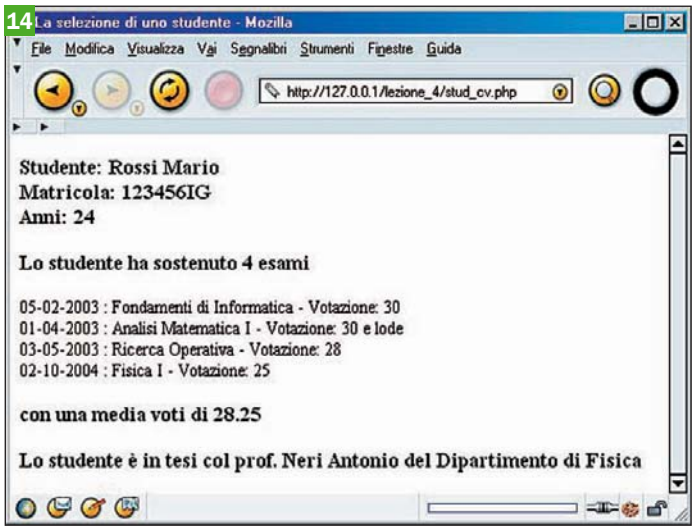
listato voto_gestione.php

```
if (isset($_POST['inserisci'])) {
  $q_3="INSERT INTO esame VALUES ('".$_POST['stud_nome']."','".$_POST['stud_es']."','".$_POST['voto']."',
    '".$_POST['lode']."','".$_POST['data']."')";
    mysql_query($q_3);
    if (mysql_affected_rows()==1) echo "Il voto è stato inserito";
     else echo "Attenzione: il voto non è stato inserito perché lo studente selezionato ha già sostenuto quell'esame";
 }
    elseif (isset($_POST['elimina'])) {
    $q_4="DELETE FROM esame WHERE matricola='".$_POST['stud_nome']."' AND cod_corso='".$_POST['stud_es']."'";
  mysql_query($q_4);
  if (mysql_affected_rows()==1) echo "Il voto è stato cancellato";
   else echo "Attenzione: il voto non è stato cancellato perché non era mai stato inserito prima";
}
```

L'ultimo esempio (file *voto_gestione.php*) riguarda l'inserimento e la cancellazione dei dati dal nostro database. A tale scopo predisponiamo una pagina dalla quale inserire (o eliminare) un esame svolto da uno studente. La pagina conterrà un form con due pulsanti per inserire o cancellare la votazione. Notate le due caselle a discesa di selezione: servono a preservare l'integrità referenziale del database, impedendo di inserire nella tabella "esame" una matricola non esistente o un corso non attivato.

Chiaramente se si tentasse di inserire un voto di un esame già sostenuto non si otterrebbe alcun risultato perché sarebbe una violazione della chiave della tabella. Per il controllo usiamo la funzione **mysql_affected_rows** che restituisce il numero di record interessati dalla query: se il valore ottenuto è zero significa che si è cercato di inserire un voto che già c'era o di cancellare un voto che non esisteva. Il listato *voto_gestione.php* fa vedere il particolare delle istruzioni di inserimento o cancellazione del record (attenzione all'uso "compatto" degli if interni). Si lascia al lettore la rifinitura della funzione, ad esempio per impedire di effettuare un inserimento se il voto non è stato scritto nel form, o impedire di assegnare la lode ad un voto minore di 30.

Con questa doverosa introduzione sul rapporto tra PHP e MySQL si chiude anche la quarta puntata del corso. La prossima sarà totalmente dedicata ad approfondire esempi pratici di applicazioni PHP-MySQL e vedremo come gestire in tutte le sue parti un sito Web dinamico.



***La scheda** dello studente selezionato*

Approfondimenti:

http://www.mysql.com
http:/www.mysql.com/documentation
http://www.mysql.com/downloads
http://www.php.net/mysql
http://www.phpmyadmin.net
http://www.easyphp.org

A scuola con *PC Open*

Web Developer PHP

di Federico Pozzato

1 La gestione di un sito dinamico

Le quattro lezioni pubblicate del corso "Web Developer PHP" ci hanno insegnato le basi del linguaggio PHP, basi necessarie per sviluppare un sito dinamico utilizzando come DBMS il database open-source MySQL. Abbiamo visto anche un po' di teoria dei database, dal momento che uno sviluppatore non può essere digiuno di un aspetto sempre più fondamentale nell'evoluzione del Web.

Quest'ultima lezione riunisce molti dei concetti visti dando loro un taglio ancora più pratico ed esemplificativo: lo scopo finale, infatti, è creare un completo **portale Web di commenti tematici**, comprensivo di una zona protetta di amministrazione e gestione. La sfida più interessante è rendere questo portale espandibile fin da subito.

L'esempio, capita la logica di fondo, potrà poi essere usato come spunto per Web site più complessi.

2 Progettare un sito Web di notizie e commenti

Innanzitutto per lavorare al meglio dobbiamo avere chiari i punti di partenza e di arrivo del progetto che ci apprestiamo a realizzare.

L'obiettivo (il **goal**) è la creazione di un sito nel quale inserire articoli, opinioni e approfondimenti (in generale: commenti) su una o più macro aree, a loro volta suddivise in argomenti: le aree (l'**object**), ad esempio, potrebbero essere la letteratura, il cinema, la politica, l'ambiente e via dicendo.

Inizialmente si partirà con un paio di aree (ambiente e informatica), da incrementare in seguito. Il sito si rivolgerà (il **target**) a visitatori desiderosi di approfondire argomenti specifici di loro interesse. Il portale deve essere assolutamente semplice e fruibile in maniera immediata da parte dell'utente, che in pochi attimi deve riuscire a individuare la sua area di interesse e a leggere i commenti inseriti.

Si deve anche prevedere un minimo di interattività, dando ai visitatori la possibilità di scrivere le proprie opinioni e di iscriversi ad una mailing list.

Gli articoli saranno affidata a più persone ("giornalisti"), esperte nel loro campo. Queste persone operano da luoghi diversi (anche da casa), quindi deve essere predisposto un sistema Web grazie al quale possano inserire i loro scritti nel portale dopo una autenticazione di accesso.

Bisogna poi prevedere un amministratore unico del sito, i cui compiti principalmente saranno:

- convalidare o cancellare commenti e foto inseriti dai giornalisti;
- inserire, modificare e cancellare le macro aree e gli argomenti (in accordo con l'editore del sito);
- aggiungere, modificare o eliminare i giornalisti;
- gestire la mailing list e spedire la newsletter;
- gestire i testi di presentazione delle pagine.

C'è un vincolo (un **mandatory**) importante: sia i giornalisti che l'amministratore sono utilizzatori Web, ma non sanno nulla di HTML, PHP, SQL o database. Va quindi fornito un portale "chiavi in mano", semplice e pronto all'utilizzo anche per quanto riguarda la gestione.

In termini progettuali, quanto scritto nelle righe precedenti rappresenta il **briefing** (vedi *figura 1*) del compito assegnatoci: "to brief", infatti, significa riassumere, dare istruzioni a qualcuno. Mancherebbero un paio di dettagli fondamentali come il tempo di realizzazione (il **timing**) e i soldi a disposizione (il **budget**), ma per il nostro esempio sono irrilevanti.

1

***Il ciclo** di realizzazione di un progetto Web*

IL CALENDARIO DELLE LEZIONI

Lezione 1:
- PHP con un server off line
- Funzioni base e variabili
- I costrutti di controllo

Lezione 2:
- Approfondiamo PHP
- Include e require
- Funzioni e classi
- Proteggere una pagina
- Cookie e sessioni
- La funzione mail

Lezione 3: PHP e i database
- La funzione upload
- Lavorare con i file
- La gestione degli errori
- Accenni di sicurezza
- Il database system
- Siti Web dinamici
- Teoria dei database

Lezione 4:
- PHP e MySQL
- MySQL, database opensource
- Costruzione e interrogazione di un database: il linguaggio SQL
- Interfaccia GUI: PhpMyAdmin
- Integrazione MySQL e PHP

Lezione 5:
- La gestione di un sito dinamico
- Il sito Web da sviluppare
- Il compito del WebDeveloper
- Il database
- La struttura del sito
- La costruzione del sito: la parte pubblica
- La costruzione del sito: le pagine di gestione

3 Il compito del Web Developer

Al primo briefing seguirà poi un lavoro ciclico di chiarimento (non sempre è possibile realizzare quello che il cliente vorrebbe alle sue condizioni), al termine del quale il Web Developer sarà pronto a realizzare il sito unendo, se necessario, le sue competenze con quelle di altri professionisti Web. Per semplicità assumeremo di essere in grado di creare il sito da soli.

Viste le premesse e i vincoli di progetto, è chiaro che dovremo sviluppare un portale dinamico utilizzando un DBMS come **MySQL** e integrandolo con il linguaggio **PHP**. Predisporremo due differenti profili di accesso alla gestione del database per amministratore e giornalisti, realizzando per loro un'apposita interfaccia grafica Web di gestione. Escludiamo di dare a queste persone accesso diretto al client MySQL (sia esso grafico o a caratteri), per motivi di praticità (le persone coinvolte sono solo utilizzatori del Web) e anche per motivi di sicurezza (mai da sottovalutare).

Partendo da questo punto, stabiliamo una via di sviluppo, tra le tante possibili, per la parte pubblica del portale.

L'**home page** dovrà presentare brevemente il sito, spiegarne l'obiettivo e dare delle indicazioni di navigazione. I testi saranno scritti dall'amministratore del sito e potranno essere modificati in qualunque momento. Servirà poi una sezione dalla quale accedere alla lista generale dei commenti, ordinati in due modi: per numero di click o per data di inserimento. Terzo punto da realizzare sarà l'elenco di tutte le aree e dei relativi argomenti, con link cliccabili. Per ogni area, inoltre, prevediamo di inserire due link diretti al commento più letto e all'ultimo commento inserito. È chiaro che questa terza parte sarà totalmente gestita dal database: una volta scritto il codice della pagina non dovremo più preoccuparci di fare nessun aggiornamento. In calce alla home page, infine, ci saranno i link a **due form**: uno per scrivere una mail e uno per iscriversi alla mailing list (o cancellarsi).

La **pagina principale**, quella che conterrà la cosiddetta "biblioteca dei commenti", avrà differenti tipi di visualizzazione dipendenti dalle scelte effettuate nella pagina stessa o nella home page. Si potrà, infatti, visualizzare (scegliendo uno dei due ordinamenti possibili) tutto l'elenco dei commenti, oppure l'elenco dei commenti di un'area o di un suo argomento, oppure vedere il commento stesso comprensivo di immagini. In ogni caso, questa pagina dovrà prevedere una barra di navigazione per potersi spostare tra le pagine del sito e un menu grazie al quale scegliere il tipo di visualizzazione e il tipo di ordinamento. Ulteriore attenzione va posta all'elenco dei commenti: per evitare che diventi troppo lungo (e quindi poco fruibile) useremo delle sottopagine, visualizzando i dati sintetici di 5 commenti per ogni sottopagina. Questa pagina è completamente gestita grazie al database sviluppato: la difficoltà, a questo punto, è solo di programmazione SQL.

Vi sono poi altre due pagine "accessorie":

- **scrivici**: consente di spedire commenti sul sito;
- **mailing list**: consente di inserire la propria mai per ricevere la newsletter o per cancellare l'iscrizione.

Le pagine saranno dinamiche, nel senso che la loro modifica sarà determinata direttamente dagli aggiornamenti del database e non da modifiche di codice sui listati HTML e PHP. A noi basterà quindi creare il template della pagina senza occuparci dei contenuti e del loro cambiamento.

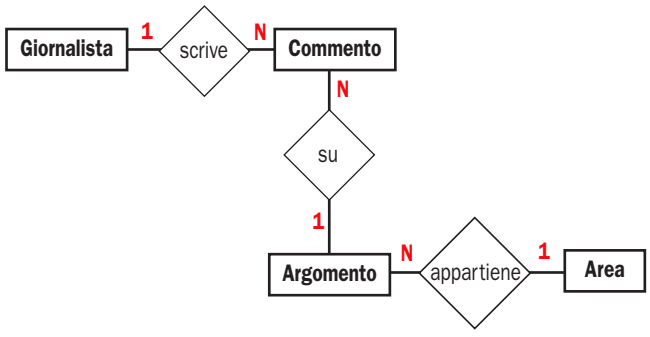
La parte amministrativa sarà creata tenendo presente la necessità di stabilire due differenti profili per amministratore e giornalisti. Il primo avrà accesso a tutte le parti configurabili del database, mentre chi scrive gli articoli avrà un accesso che gli consentirà solo di inserirli nel sistema e modificarli (immagini comprese).

4 Il database

La prima operazione da compiere, quella che poi necessariamente guiderà la realizzazione delle pagine, è la definizione della struttura della base di dati da realizzare.

Come visto nella lezione 3, si deve definire uno schema concettuale **Entità-Relazione** (vedi *figura A*), da trasformare successivamente in uno schema logico relazionale normalizzato. Le entità non sono molte: i commenti, le aree, gli argomenti e i giornalisti. Non ci sono le immagini: per loro useremo una gestione che non necessita di una tabella. Neppure l'amministratore è inserito nello schema perché non sono definite relazioni con altre Entità: chi amministra, infatti, sovrintende tutte le attività e pertanto non è legato direttamente a nessuna.

A



Lo schema Entità-Relazione *del database* Commenti

Dallo schema concettuale deriva lo **schema logico** (*figura B*) in cui è indicata la composizione di ogni tabella. I campi chiave sono in grassetto, mentre i campi con l'asterisco rappresentano una colonna che, pur non essendo chiave, contiene solo valori numerici unici ed è quindi utilizzabile come riferimento esterno per altre tabelle. Nello schema finale mancano alcune tabelle che, pur facendo parte del database, possiamo definire di servizio generale al sito e sono quindi dotate di vita propria indipendentemente dalla struttura definita per il DB.

Queste tabelle sono:

- **amministrazione:** nome (chiave), user, password, e-mail
- **mailing:** nome, e_mail (chiave)
- **testi_admin:** id_testo (chiave), descrizione, testo

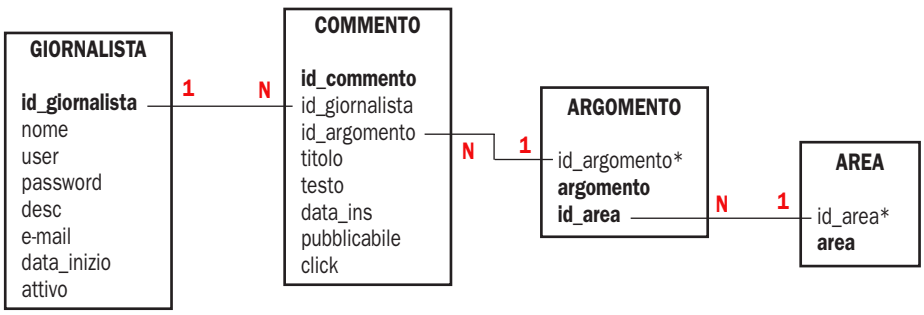
Costruire il DB è un ottimo esercizio (lezione 4), comunque nel CD allegato alla rivista è presente il file *commenti.sql* che contiene la struttura completa del database e alcuni dati di esempio. Aprendo il file con un editor di testo si può vedere nel dettaglio la struttura ed i tipi di dato di ogni tabella.

Se scegliamo di caricare il file, creiamo prima il database *Commenti*, quindi importiamo struttura e dati usando gli appositi comandi di PhpMyAdmin (vedi *immagine 1* e *imma-* ▷

▷ *gine 2*). Usando il client a caratteri di MySQL scriviamo invece dalla shell DOS (dalla cartella c:\mysql\bin):

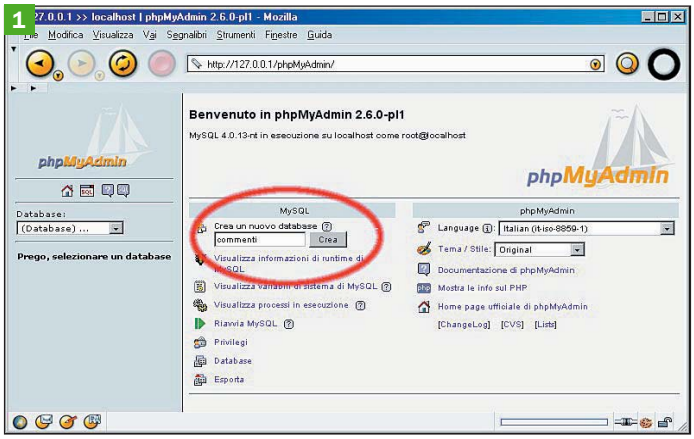
```
mysql -u root -p
// inseriamo la password  per entrare nel client
mysql> CREATE DATABASE commenti;
mysql> EXIT
// siamo usciti dal client mysql
mysql -u root -p commenti < commenti.sql
// eventualmente scrivere il percorso assoluto del file commenti.sql
```

B



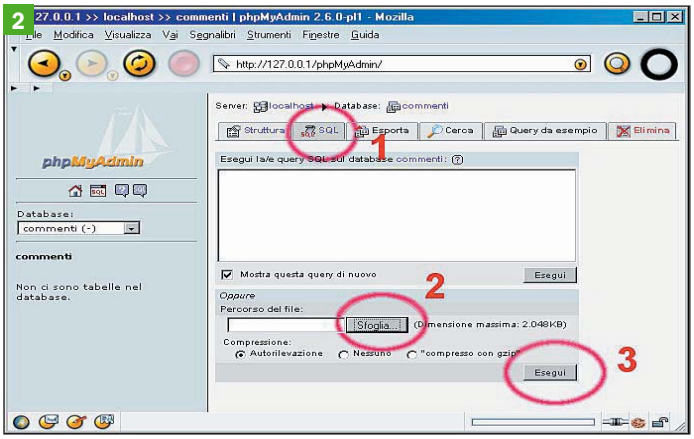
Schema logico relazionale *del database* Commenti

1



Creiamo *il database* Commenti

2



Azioni per importare struttura *e dati nel database* Commenti

5 La struttura del sito

Innanzitutto vediamo il significato delle cartelle presenti nella root del portale :

- nella root c'è solo l'home page "index.php" dalla quale la navigazione ha inizio. È sempre buona norma lasciare solo l'home page nella root;
- la cartella "pagine" ospita le pagine che ci consentono di vedere i commenti, spedire una mail, iscriverci alla mailing list. Sono le pagine "pubbliche" del sito;
- la cartella "servizi" ospita i dati della connessione, il foglio di stile responsabile del layout,

NB: le pagine del sito si trovano tutte all'interno della cartella *CD2/PDF/Corsi/PHP/lezione_5/commenti* nel CD Guida 2 che è la radice (root) del sito.
Ad essa si farà sempre riferimento indicando il percorso relativo della pagine

due icone e la pagina che ci restituisce i parametri del parser PHP installato (nel caso ci servisse controllare i settaggi di PHP;

- le cartelle "admin" e "giorn" servono per la gestione delle aree protette riservate ad amministratore e giornalisti;
- la cartella "img" contiene le immagini caricate per illustrare meglio i commenti. Questa cartella dovrà avere i permessi di scrittura abilitati in modo da poter effettuare un upload da una pagina Web.

Vista la loro importanza generale, guardiamo due pagine contenute nella cartella "servizi".

Il file *conn.php* (listato 1) contiene le impostazioni della connessione al server MySQL (lezione 4) che ospita il database "commenti". Chiaramente questa pagina deve sempre essere "inclusa" nelle altre prima di effettuare qualsiasi tipo di operazione sulla base di dati.

```
<?php
   $nome="commenti";
   $id_connessione=@mysql_connect('127.0.0.1','root','pluto') or
   die('Non è possibile accedere al database');
// dati connessione: user=root, password=pluto, indirizzo server=127.0.0.1, nome database=commenti
   @mysql_select_db($nome,$id_connessione) or die("Non è
   stato possibile accedere al database");
?>
```

Listato 1: *servizi/conn.php*

Il file *commenti.css* (vedi una piccola parte nel listato 2, con lo stile applicato per il tag body) ospita invece le definizioni delle formattazioni usate nel sito (tipici esempi possono essere i colori, i caratteri, le dimensioni e via dicendo). È un tipico foglio di stile e ci facilita nella scrittura della pagina e nella gestione delle opzioni di layout.

```
body {
   background-color: #FFFF99;
   font-family: Verdana, Geneva, Arial, Helvetica, sans-serif;
   font-size: 11px;
   width: 750px;
}
```

Listato 2: *servizi/commenti.css*

6 La costruzione del sito: la parte pubblica

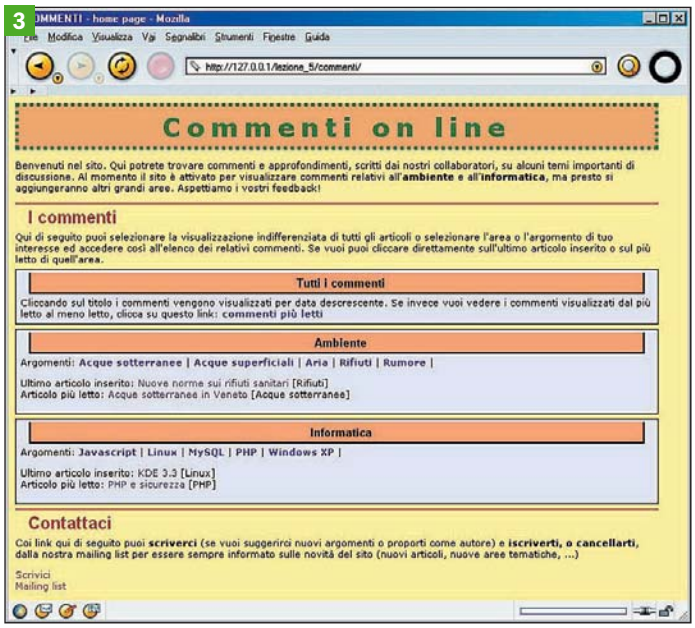
La home page "index.php" (immagine 3) sarà composta da:

- tre testi editabili dall'amministratore del sito. I testi sono quelli visibili su sfondo giallo;
- una sezione "testi commenti" che consente di vedere tutti gli articoli "pubblicabili" presenti nel database;
- N sezioni tante quante sono le aree trattate (nel nostro caso sono due, ambiente e informatica): per ogni area sarà presente l'elenco degli argomenti e due link veloci all'ultimo articolo inserito e all'articolo più letto;
- un'ultima sezione in basso coi link per scrivere i propri commenti al sito o per accedere alla mailing list.

Per visualizzare i tre testi scriveremo una query e memorizzeremo i risultati su un array $home da richiamare nei punti opportuni della pagina (listato 3). Avremmo anche potuto scrivere tre query diverse per estrarre ogni volta il valore del testo voluto, ma con la soluzione scelta si fa una sola richiesta al server. I testi sono memorizzati nella tabella di servizio "testi_admin" e hanno la peculiarità di contenere la stringa "home_" nell'id_testo che li definisce univocamente. Per sfruttare questa particolarità utilizziamo la clausola **LIKE** nella query SQL: LIKE consente di fare delle ricerche su stringhe di testo e il carattere di percentuale "%" indica il carattere jolly. L'array $home memorizzato in questo modo ha la caratteristica di avere come chiave l'identificativo del testo che si vuole stampare e come valore il testo stesso. È quindi semplice visualizzare il testo voluto con una istruzione echo (ne compariranno quindi tre in questa home page).

```
<?php
// estraggo le descrizioni della home
page e le salvo nell'array $home per
usare poi la descrizione dove serve
$sql="SELECT id_testo, testo FROM
  testi_admin WHERE id_testo LIKE
  'home_%'";
$ris=mysql_query($sql);
while ($tab=mysql_fetch_array($ris)) {
$home[$tab[id_testo]]=$tab[testo];
}
// visualizzo il testo che nella tabella
testi_admin corrisponde
all'identificativo 'home_p'
echo $home[home_p];
?>
```

listato 3: *index.php*



Home page *del portale dei commenti*

La sezione "testi commenti" contiene solo istruzioni HTML perché è sempre fissa: cliccando sul titolo si accede alla pagina *commenti.php* visualizzando i commenti in ordine decrescente di data di inserimento, mentre cliccando sul link nel prosieguo del testo si visualizzano gli articoli dal più letto al meno letto. In questo secondo caso avremo l'accortezza di inserire nell'indirizzo del link una variabile di tipo **GET** (lezione 1) di nome 'ord' e di valore 'plus':

```
// link ai commenti ordinati per data
decrescente
<a href="pagine/commenti.php"
class="tema">Tutti i commenti</a>
// link ai commenti ordinati dal più al
meno letto
<a href="pagine/commenti.php?
  ord=plus" class="arg">commenti
  più letti</a>
```

Più complesso è visualizzare ogni singola area con i relativi argomenti e link ai commenti. Dovremo usare delle query nidificate: con la prima estrarremo nome e codice di ogni record dalla tabella "area", poi utilizzando questo codice scriveremo una seconda query che estrarrà, dalla tabella "argomento", gli argomenti che si rifanno ad esso. Una terza e una quarta query, infine, sempre usando il codice dell'area, ci consentiranno di scrivere i link all'ultimo commento inserito e al più letto.

La prima query $sql_1 è ordinata in base al nome dell'area: introduciamo nel tag <a> del link una nuova variabile GET di nome "area" il cui valore sarà l'identificativo univoco dell'area (listato 4). Grazie a questa variabile cliccando sul link visualizzeremo solo i commenti di quell'area.

```
$sql_1="SELECT id_area,testo_area
   FROM area ORDER BY testo_area";
$ris_1=mysql_query($sql_1);
while ($tab_1=mysql_fetch_array
   ($ris_1)) {
// per ogni area estratta inserisco il
link dell'area stessa e scrivo una
query per ricavare gli argomenti relativi
all'area
echo "<div class='tem'><a href=
   'pagine/commenti.php?
   area=".$tab_1[id_area]."'
class='tema'>".$tab_1[testo_area]."
   </a>
```

listato 4: *index.php*

La seconda query, nidificata nel ciclo while della query precedente, serve a scrivere i link riferiti a tutti gli argomenti dell'area. Notiamo la presenza (listato 5) nella query $sql_2 di un join (legame tra una o più tabelle che fanno riferimento le une alle altre. Vedi lezione 4): la clausola **WHERE**, infatti, lega i valori dell'identificativo dell'area della tabella "area" e della tabella "argomento". I link contengono una nuova variabile GET di nome "arg" che ha come valore l'identificativo univoco dell'argomento.

```
$sql_2=
  "SELECT argomento.testo_argomento
   AS testo, argomento.id_argomento
  AS id FROM argomento,area WHERE
  argomento.id_area=area.id_area AND
  area.id_area='".$tab_1[id_area]."'
  ORDER BY
  argomento.testo_argomento";
$ris_2=mysql_query($sql_2);
echo "<span class='normale'>
  Argomenti: </span>";
while  ($tab_2=mysql_fetch_array
  ($ris_2)) {
// scrivo i link a tutti gli argomenti di
quell'area
echo "<a href='pagine/commenti.php
   ?arg=".$tab_2[id]."' class='arg'>".
   $tab_2[testo]."</a> | ";
}
```

listato 5: *index.php*

La terza query $sql_3 (listato 6) e la quarta $sql_4 estraggono un solo record dalla tabella "commento": rispettivamente il codice e il titolo dell'ultimo commento inserito e del più letto. Ovviamente il commento deve appartenere all'area selezionata ed essere stato definito "pubblicabile" dall'amministratore. Il link diretto al commento è reso possibile dalla presenza della variabile GET di nome "comm" che ha come valore l'identificativo univoco del commento:

```
$sql_3="SELECT commento.titolo,
  argomento.testo_argomento,
  commento.id_commento FROM
  commento,argomento WHERE
  commento.pubblicabile='si' AND
  commento.id_argomento=argomento.
  id_argomento AND
  argomento.id_area='".$tab_1
  [id_area]."' ORDER BY
  commento.data_ins DESC LIMIT 0,1";
$ris_3=mysql_query($sql_3);
$tab_3=mysql_fetch_array($ris_3);
echo "<p><span class='normale'>
  Ultimo articolo inserito: </span>
```

```
<a href='pagine/commenti.php?
comm=".$tab_3[2]."' class='art'>".
$tab_3[0]."</a> <span
class='normale'>[".$tab_3[1]."]
</span>";
```

***listato 6**: index.php*

Ultima annotazione: l'estrazione di un solo record è consentita dalla clausola LIMIT presente nel codice SQL:

```
LIMIT (X,Y)
```

Questa clausola limita il risultato ottenuto dall'esecuzione della query, estraendo Y record a partire dalla posizione X dei record estraibili.

La pagina "biblioteca dei commenti" (file *pagine/commenti.php, immagine 4*) sarà invece composta da:

- una barra di navigazione per spostarsi tra le pagine;
- un menu di navigazione diviso per aree tematiche, a loro volta suddivise per argomenti con l'indicazione del numero di articoli inseriti;
- un'area di visualizzazione: cliccando sul nome di un'area o su un argomento si potrà vedere la lista dei commenti, lista limitata a 5 commenti per sottopagina. Se un titolo attirasse l'attenzione, basterà cliccarci sopra per ottenere la visualizzazione dell'articolo stesso, completo di eventuali foto.

Per la costruzione della barra di navigazione si rimanda alla lezione 2. Il codice che contiene la barra è nel file *pagine/testata.php*: il file viene incluso in ogni pagina, adattandosi dinamicamente alla situazione (il nome della pagina in cui ci troviamo sarà scritto in grassetto e non sarà cliccabile).

Il menu di navigazione a sinistra segue la logica vista nella costruzione della home page. Le differenze riguardano l'aggiunta, a fianco di ogni scritta, di una icona col segno "+" e del numero di commenti contenuti in ogni argomento. Cliccando sull'icona + si ottiene la lista dei commenti ordinata dall'articolo più letto al meno letto: questo è reso possibile dall'aggiunta della variabile GET di nome "ord" e valore "plus" nel riferimento del link. Cliccando sulle scritte, invece, questa variabile non è presente e si otterrà una lista ordinata per data di inserimento decrescente. Ecco i due link a confronto:

```
echo "<a
href='commenti.php?area=".$tab_1[id
_area]."&ord=plus'><img
src='../servizi/bluplus.gif' alt='Più
letti' title='Ordina per i più
letti'></a>  <a
href='commenti.php?area=".$tab_1[id
_area]."'
class='tit'>".$tab_1[testo_area]."</a
></div>";
```

Interessante è capire come ricavare il numero di articoli di ogni argomento. Si usa la clausola SQL di raggruppamento **GROUP BY** (lezione 4), raggruppando tutti i commenti pubblicabili che hanno lo stesso argomento e operando su di essi una operazione di somma dei record (listato 7). Il suggerimento è salvare questi valori in un array $list che ha come chiave l'identificativo univoco dell'argomento e come valore il conteggio appena fatto. Sfruttando questo vettore posso ricavare anche quanti commenti ci sono in ogni area (creo il vettore $tot_area) facendo una semplice somma durante il ciclo while che visualizza gli argomenti di ogni area. I valori appena ricavati serviranno poi per creare le sottopagine di ogni scelta. Sommando tutti i valori dell'array (con la funzione PHP array_sum) posso anche ottenere il numero totale di documenti pubblicabili presenti nel database.

```
// conto quanti record ci sono
raggruppandoli per id_argomento, e
registro il numero nel vettore $list

$sql_3="SELECT id_argomento,
  COUNT(*) AS num FROM commento
  WHERE commento.pubblicabile='si'
  GROUP BY id_argomento";
$ris_3=mysql_query($sql_3);
while ($tab_3=mysql_fetch_array
  ($ris_3)) {
$list[$tab_3[id_argomento]]=$tab_3
  [num];
}
```

***listato 7:** pagine/commenti.php*

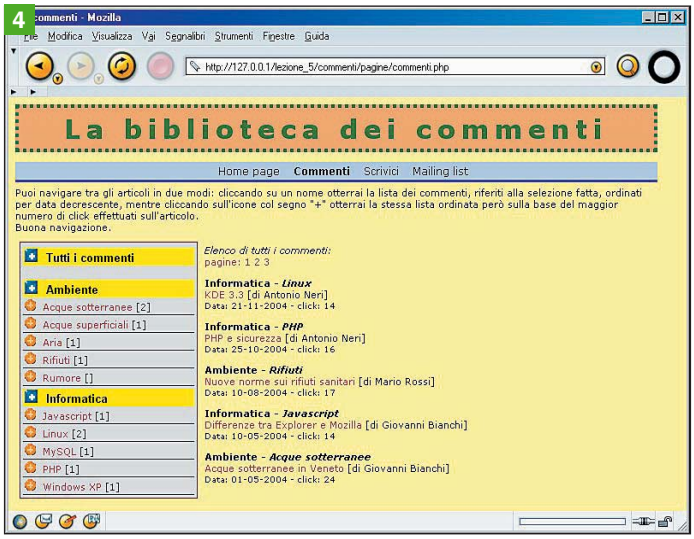
La creazione dell'area di visualizzazione di destra è la parte più complessa della pagina. Dobbiamo infatti controllare l'URL della pagina per verificare se ci sono delle variabili GET la cui presenza, col relativo valore, determina la costruzione della pagina. In particolare, alcune variabili escludono la presenza di altre a seconda della precedenza assegnata. Nel nostro caso la precedenza è dall'alto in basso:

- $_GET[comm]=x : verrà visualizzato (testo e immagini) il commento avente identificativo x;
- $_GET[arg]=y : visualizzazione della lista dei commenti che fanno parte dell'argomento di codice y;
- $_GET[area]=z : visualizzazione della lista dei commenti che fanno parte dell'area di codice z;
- nessuna della variabili precedenti: viene visualizzata la lista completa dei commenti.

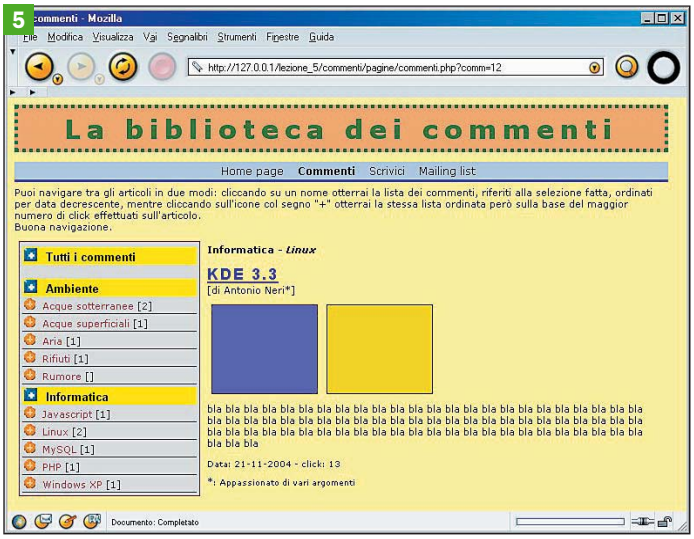
Altre due variabili possono trovarsi nell'URL e sono affiancate alle precedenti:

- $_GET[pg]=w : viene visualizzata la sottopagina w (se non c'è vuol dire che non ci sono sottopagine);
- $_GET[ord]=plus : ordinamento dall'articolo più letto al meno letto.

Cosa succede quando nell'URL c'è la variabile "comm"? Con una query estraiamo i dati che ci servono, quindi "stampiamo" questi dati per visualizzare l'articolo (*immagine 5*). Può essere interessante osservare il codice (listato 8) per notare un paio di particolari. Le immagini, ad esempio, possono essere al massimo due per commento ($imm1 e $imm2) e il loro nome è legato al codice del commento (es: il commento "12" può avere collegate due immagini, nella cartella "img", denominate "12_1.jpg" e "12_2.jpg"). Con la funzione "file_exists" controlliamo se l'immagine è effettivamente presente, e solo in caso positivo la



***La pagina di navigazione** di tutti i commenti pubblicati*



***La visualizzazione** di un articolo (con due immagini)*

```
$sql_r="SELECT area.testo_area, argomento.testo_argomento, commento.titolo, giornalista.nome, commento.testo,
date_format(commento.data_ins,'%d-%m-%Y'), commento.click, giornalista.email, giornalista.descri,
commento.id_commento FROM commento, argomento, area, giornalista WHERE commento.pubblicabile='si' AND
commento.id_commento='".$_GET[comm]."' AND commento.id_argomento=argomento.id_argomento AND
argomento.id_area=area.id_area AND commento.id_giornalista=giornalista.id_giornalista";
    $ris_r=mysql_query($sql_r);
    $tab_r=mysql_fetch_array($ris_r);
    echo "<strong>".$tab_r[0]." - <em>".$tab_r[1]."</em></strong><p><span
class='titoletto'>".$tab_r[2]."</span><br />[di ".$tab_r[3]."*]<br />";
// visualizzo le immagini solo dopo averne verificato la presenza
    $imm1="../img/".$_GET[comm]."_1.jpg";
    $imm2="../img/".$_GET[comm]."_2.jpg";
    if (file_exists($imm1)) echo "<img src='".$imm1."' class='vis'>";
    if (file_exists($imm2)) echo "<img src='".$imm2."' class='vis'>";
    echo "<br />".$tab_r[4]."<p><span class='normalemini'>Data: ".$tab_r[5]." - click: ".$tab_r[6]."<p>*:
".$tab_r[8]."</span><p>";
// incremento il valore dei click del commento scelto estraendo il valore attuale, aumentandolo di uno e facendo una
query di aggiornamento
    $cont=$tab_r[6]+1;
    $incr="UPDATE commento SET click='".$cont."' WHERE id_commento='".$tab_r[9]."'";
    mysql_query($incr);
```

listato 8: *pagine/commenti.php*

visualizziamo. Con questa gestione si può evitare di memorizzare sul database il legame tra il nome dell'immagine e il codice del commento. Avendo scelto di vedere l'articolo, inoltre, devo incrementare il valore dei "click" ad esso riferiti (questo valore serve poi a stabilire uno dei due ordinamenti dei commenti): ciò si ottiene con un'opportuna istruzione UPDATE.

Se invece nell'URL c'è una variabile "arg" o "area" o nessuna delle precedenti (il che vuol dire che vanno presi estratti tutti i commenti), dobbiamo scrivere tre query diverse per ricavare la lista degli articoli. Le query sono abbastanza simili, quindi esaminiamo quella che ci consente di vedere tutti i commenti lasciando ai lettori la costruzione delle altre due (c'è solo una condizione WHERE da aggiungere) visto che la logica è esattamente la stessa (si deve sempre fare un join che coinvolge le tabelle commento, giornalista, argomento e area):

```
$sql_r="SELECT area.testo_area,
argomento.testo_argomento,
commento.titolo, giornalista.nome,
commento.id_commento,
date_format(commento.data_ins,'%
d-%m-%Y'), commento.click FROM
commento, argomento, area,
giornalista WHERE
commento.pubblicabile='si' AND
commento.id_argomento=
argomento.id_argomento AND
argomento.id_area=area.id_area AND
commento.id_giornalista=giornalista.id
_giornalista ".$ordine." LIMIT
".$start.",".$vis;
```

La variabile $ordine indica il tipo di ordinamento dei record della query e si ricava da un ciclo if effettuato controllando il valore di $_GET[ord]: se è uguale a "plus" l'ordinamento si basa sui click, altrimenti l'ordinamento si basa sulla data di inserimento:

```
if (isset($_GET[ord]) AND
  $_GET[ord]=='plus') $ordine=
  "ORDER BY click DESC, data_ins
  DESC";
else $ordine="ORDER BY data_ins
  DESC, click DESC";
```

Per visualizzare N commenti per ogni sottopagina (in questo caso sono 5) usiamo una clausola LIMIT nella query: $vis è il numero di commenti da visualizzare su ogni sottopagina, mentre $start indica da quale record partire per ricavare i 5 commenti. $start è dinamica, nel senso che il suo valore dipenderà dal numero di sottopagina sulla quale ci troviamo. Il database ha 11 commenti pubblicabili (2 non sono ancora approvati), quindi avremo 3 sottopagine (vedi immagine_04): se ci troviamo sulla prima $start avrà valore 0 (il primo record ha sempre posizione 0), sulla seconda $start avrà valore 5, sulla terza $start avrà valore 10.

Il valore di $start sarà quindi dipendente dalla variabile $_GET[pg] che indica la sottopagina da visualizzare (di default la sottopagina è una sola ed ha quindi valore uno):

```
if(isset($_GET[pg]) AND $_GET[pg]>0)
$sottopagina=$_GET[pg];
else   $sottopagina=1;
// $vis=5 è il numero di commenti da
visualizzare per ogni pagina. $start è
usato dall'attributo LIMIT delle query
$vis=5;
$start=($sottopagina-1)*$vis;
```

Un ultimo sforzo (listato 9) ci consente finalmente di stampare l'elenco e creare i link per vedere le sottopagine.

Per prima cosa ci serve il totale dei commenti $tot, e questo valore lo ricaviamo sommando i valori dell'array $list del listato 6. Poi calcoliamo il resto della divisione (con l'operatore "modulo", lezione 1): se è zero il numero di sottopagine sarà dato dalla semplice divisione $tot/$vis, mentre se è diverso da zero dovremo aggiungere 1 al risultato (intero) che si ottiene dalla divisione.

Infine se le sottopagine sono più di una faremo eseguire un ciclo for per ottenere i link con l'opportuno valore assegnato alla variabile pg di tipo GET. Cliccando sul link della sottopagina, "commenti.php" viene ricaricata (le altre variabili GET restano invariate) e quindi il valore $start viene ricalcolato. Infine visualizzare la lista dei commenti con un ciclo while è ormai un gioco da ragazzi.

```
// $tot è il numero di commenti
presenti nel DB. Uso la funzione PHP
di somma dei valori di un array
$tot=array_sum($list);
$url="";
$ur="";
// $url e $ur sono usati per
completare il testo del link della
sottopagina. In questo caso sono
vuoti, ma se avessimo scelto di
visualizzare un'area o un argomento
avremmo dovuto dare loro i valori che
poi consentono di mantenere il
riferimento alla selezione
if ($_GET[ord]==plus)
  $url="ord=plus";
echo "<em>Elenco di tutti
  i commenti:</em><br />";
$mod=$tot%$vis;
if ($mod==0)   $pagine=$tot/$vis;
else   $pagine=(int) ($tot/$vis)+1;
if ($pagine>1) {
echo "<span class='colore'>pagine:
  </span>";
for ($i=1;$i<=$pagine;$i++) {
  echo "<a href='".basename
  ($_SERVER[PHP_SELF])."?".$url.$_
GET[$ur]."&pg=".$i."'>$i</a> ";
}
echo "<p>";
}
$ris_r=mysql_query($sql_r);
while
  ($tab_r=mysql_fetch_array($ris_r)) {
echo "<strong>".$tab_r[0]." - <em>
  ".$tab_r[1]."</em></strong><br />
   <a href='commenti.php?comm=
   ".$tab_r[4]."' class='cap'>".$tab_r
   [2]."</a> [di ".$tab_r[3]."]<br/>
  <span class='normalemini'>Data:
  ".$tab_r[5]." - click:
  ".$tab_r[6]."</span><p>";
}
```

listato 9: *pagine/commenti.php*

La pagina "scrivici" (file *pagine/formmail.php*) è un form per spedire una mail all'amministratore del sito (lezione 2). Compilati i campi, si apre una finestra (file *pagine/rispostamail.php*) che ci informa dell'avvenuta spedizione della mail o dell'impossibilità di spedirla se anche solo un campo è vuoto. C'è un controllo javascript a monte che verifica se nel campo mail c'è una @: va benissimo usare dei controlli javascript che verificano i campi prima di spedirli al server (PHP lo fa dopo!), però tenete presente che javascript può essere disabilitato da parte del visitatore e quindi i controlli PHP server-side vanno comunque inseriti! Per spedire la mail dobbiamo estrarre con una query l'indirizzo mail dell'amministratore: ormai dovrebbe essere un semplice esercizio.

Anche quando si compila la pagina della mailing list viene aperta una pagina di risposta ▷

▷ (file *pagine/rispostamailing.php*) che ci informa se tutto è andato bene (immagine 6) o meno. Questa pagina verifica la correttezza dei dati inseriti (per cancellarsi basta inserire solo la mail, per iscriversi serve anche il nome) e "capta" il tasto premuto sul form: se ci si vuole iscrivere viene eseguita una query di inserimento:

```
$query="INSERT INTO mailing
  (email,nome) VALUES
  ('".$_POST['email']."','".$_POST
  ['nome']."')";
mysql_query($query);
```

altrimenti viene eseguita una query di cancellazione:

```
$query="DELETE FROM mailing
  WHERE email='".$_POST['email']."'";
mysql_query($query)
```

In entrambi i casi viene spedita una mail di conferma (è una ottima abitudine farlo!) all'indirizzo mail inserito e all'amministratore.

L'iscrizione *alla mailing list è andata a buon fine*

7 La costruzione del sito: le pagine di gestione

Il sito pubblico è costruito, ma adesso bisogna gestirlo inserendo commenti, immagini, aree, argomenti, giornalisti, testi. In poche parole va creata l'area di gestione. Nel nostro caso la situazione è ancor più complessa perché di aree di gestione dovremo crearne due: una per l'amministratore e una per i giornalisti.

Area di gestione dell'amministratore

Tutte le pagine saranno chiaramente protette da accessi indesiderati grazie ad un sistema di autenticazione basato su username e password. Nella lezione 2 avevamo visto come implementare un sistema statico di accesso memorizzando user e password direttamente nella pagina Web, ma usare MySQL ci consente di registrare i dati su una tabella, rendendoli quindi facilmente modificabili in caso di bisogno.

La prima operazione da compiere, quindi, è validare l'inserimento dei dati, propagando poi l'accesso a tutte le pagine oggetto di gestione senza che si debbano reinserire i dati di autenticazione. Per fare questo useremo le sessioni: al login (to log in = connettersi) verificheremo i dati inseriti, e in caso di riscontro positivo salveremo questa risposta in una variabile di sessione che verrà cancellata al momento del logout (e comunque alla chiusura del browser)

Per accedere ad una pagina protetta dobbiamo per prima cosa creare un form di autenticazione (immagine 7) dei dati (file *admin/index.php*) composto da due caselle di testo "user" e "pwd" (listato 10):

```
<form action="admin.php"
   method="post">
<input type="text" name="user"
   size="30"> username<br />
<input type="password" name="pwd"
   size="30"> password<p>
<input type="submit" name="invio"
   value="Connetti">
</form>
```

listato 10: *admin/index.php*

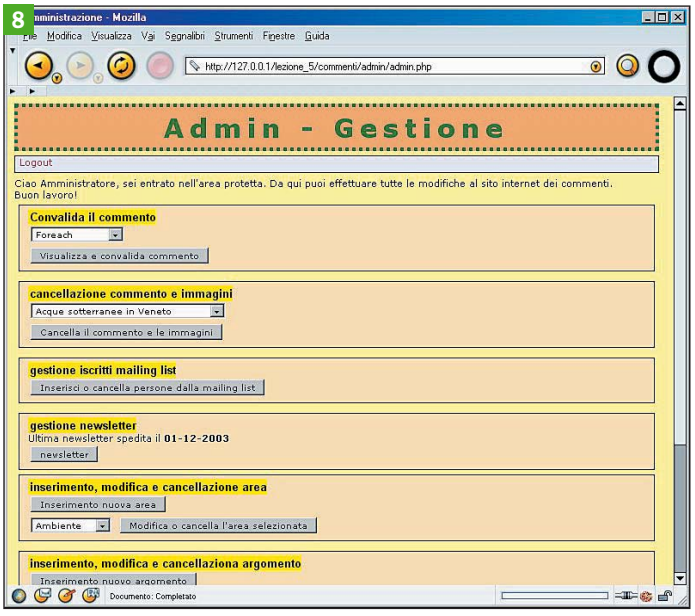
I dati inseriti (i nostri soliti "root" e "pluto") vengono spediti alla pagina *admin/admin.php* e lì controllati richiamando il file "admin/auth.inc.php". Questo file compie due controlli (vedi listato 11):

- per prima cosa avvia una sessione controllando se esista già la variabile di sessione $_SESSION['ok']. Se esiste, verifica che il suo valore sia "true" e in caso positivo concede l'accesso alla pagina. Questo controllo serve ad evitare di reinserire i dati di autenticazione nel caso tornassimo a questa pagina da una delle altre pagine di amministrazione;
- se il primo controllo fallisce (ossia non c'è alcuna sessione già iniziata che abbia una variabile di nome "ok" con valore "true") parte il processo di autenticazione vero e proprio. La prima verifica controlla che i campi del form esistano, quindi, dopo aver effettuato il collegamento al database, viene eseguita la query $sql che ritornerà, se i dati inseriti sono corretti, l'unico record della tabella "amministrazione". Verifico che il risultato dell'interrogazione contenga una sola riga: in questo caso è consentito l'accesso alla pagina e viene registrato il valore "true" per la variabile di sessione "ok", altrimenti il caricamento della pagina è bloccato e compare un messaggio di avvertimento.

Chiaramente il file *auth.inc.php* sarà incluso in testa ad ogni pagina di amministrazione.

È da notare l'uso nella query $sql della funzione PASSWORD la quale crittografa (a 16 bit) il valore inserito nel form e lo confronta col valore, anch'esso crittografato, contenuto nella tabella. Questo serve esclu-

Form di connessione *alle pagine di gestione dell'amministratore*



Una parte delle opzioni *di gestione dell'amministratore*

```
<?php
session_start();
// dopo aver inizializzato la sessione, faccio il primo controllo per verificare se era già stata fatta l'autenticazione
if (isset($_SESSION['ok'])) {
if ($_SESSION['ok']!==true) die ("interrompi tutto");
}
// se è la prima volta che accedo in questa sessione, verifico i dati inseriti nel form
else {
if (!isset($_POST['user']) or !isset($_POST['pwd'])) {
die("Per accedere a questa pagina devi inserire login e password");
}
// questo include mi rimanda alla pagina coi dati di connessione al database
include "../servizi/dati.php";
$sql="SELECT * FROM amministrazione WHERE user='".$_POST['user']."' AND pwd=PASSWORD('".$_POST['pwd'] ."')";
$ris = mysql_query($sql);
$righe=mysql_num_rows($ris);
if ($righe==0) {
// non c'era alcuna riga nella tabella che avesse quei valori di user e password
die ("Mi spiace, user e password non sono corrette. Controlla meglio l'inserimento");
}
elseif ($righe==1) {
// accesso consentito e contemporanea registrazione della variabile di sessione
$_SESSION['ok']==true;
}
}
?>
```

***listato 11**: admin/auth.inc.php*

sivamente per aumentare la sicurezza: anche se una persona, infatti, avesse casualmente accesso alla tabella "amministrazione", non avrebbe la possibilità di recuperare la password. Possiamo anche usare altre funzioni di crittografia come MD5() e SHA1().

Finalmente abbiamo accesso alla pagina generale di amministrazione (immagine 8, file "admin/admin.php"). La pagina è composta da tanti form che rimandano a pagine in cui si compiono le azioni specifiche deputate al ruolo di amministratore. Una della azioni, ad esempio, è quella di approvare e rendere pubblicabili i commenti inseriti dai giornalisti. Per far questo il form ci consente di scegliere uno tra gli articoli non ancora pubblicati grazie ad una casella a discesa alimentata da una query:

```
<select name="id" size="1"
  class="but">
<?php
$sql="SELECT id_commento,
  LEFT(titolo,40) FROM commento
  WHERE pubblicabile='no' ORDER BY
  data_ins";
$ris=mysql_query($sql);
while ($tab=mysql_fetch_array($ris)) {
echo "<option value='".$tab[0]."'>".
  $tab[1]."</option>";
}
?>
</select>
```

Nella casella a discesa sono visualizzati i primi 40 caratteri del titolo del commento, ma alla pagina successiva è passato con la variabile $_POST[id] il valore dell'identificativo del commento (ossia la chiave). La pagina che si apre (non c'è bisogno di reinserire i dati di amministratore perché la sessione è già stata validata in precedenza) riporta l'articolo come verrebbe pubblicato. L'amministratore può renderlo disponibile sul sito cliccando su "Approvato", oppure tornare alla pagina generale di amministrazione senza convalidarlo. Nel primo caso (listato 12) viene eseguita una query di aggiornamento su quel commento (il valore chiave del commento è memorizzato su $_POST[id] che dobbiamo continuare a propagare inserendolo come valore nascosto anche nel form di questa pagina. In alternativa avremmo dovuto salvare il valore in una variabile di sessione), modificando il valore del campo "pubblicabile" da 'no' a 'si', e viene anche mandata una e-mail all'autore del commento, dopo averne estratto l'indirizzo mail, per informarlo che il suo articolo è adesso disponibile per la lettura sul sito.

```
$approvato="UPDATE commento SET pubblicabile='si' WHERE id_commento='".$_POST[id]."'";
mysql_query($approvato);
echo "<span class='titoletto'>Il commento è stato approvato ed è quindi adesso disponibile sul sito Internet.
    E' stata spedita una mail di conferma all'autore.</span><p>";
// da qui in poi creo la mail automatica da mandare all'autore
$sql_g="SELECT commento.titolo, giornalista.email FROM commento, giornalista WHERE
   commento.id_commento='".$_POST[id]."' AND commento.id_giornalista=giornalista.id_giornalista";
$ris_g=mysql_query($sql_g);
$tab_g=mysql_fetch_array($ris_g);
$to=$tab_g[1];
$subject="Approvazione commento";
$object="Ciao, il tuo commento dal titolo \"".$tab_g[0]."\" è stato approvato ed è quindi disponibile da adesso nel
   sito internet.\n\nL'amministratore";
$sql="SELECT email FROM amministrazione";
$ris=mysql_query($sql);
$tab=mysql_fetch_array($ris);
$from="From:".$tab[email];
mail($to,$subject,$object,$from);
```

***listato 12:** admin/adminconv.php*

***Una semplice richiesta** di conferma (in javascript) ci può far evitare di commettere errori*

Cliccando su "Approvato" è comparsa (se avete javascript attivato) una finestrella di conferma (immagine 9) che vi chiede se siete sicuri della scelta fatta: questa finestra, utile per evitare di fare errori dovuti alla fretta, si ottiene semplicemente inserendo "onsubmit" nella riga di intestazione di un form:

```
<form method="post"
   action="adminnews.php"
   class="formadmin"
   onsubmit="return
   window.confirm('Sei sicuro?
   Confermi la scelta?');">
```

La pagina (file a*dmin/admincomm.php*) che consente la cancellazione di un commento e delle immagini ad esso collegato è molto simile alla precedente, salvo che la query sarà di cancellazione e le immagini, se ci sono, saranno cancellate con la funzione unlink (lezione 3):

```
$cancellato="DELETE FROM
  commento WHERE
id_commento='".$_POST[id]."'";
mysql_query($cancellato);
// cancello le immagini
$imm1="../img/".$_POST[id]."_1.jpg";
```

```
// percorso e nome dell'eventuale immagine 1
$imm2="../img/".$_POST[id]."_2.jpg";
// percorso e nome dell'eventuale immagine 2
if (file_exists($imm1))  unlink($imm1);
if (file_exists($imm2))  unlink($imm2);
```

Le pagine che consentono di inserire nuove aree (file *admin/adminarea.php*), nuovi argomenti (file *admin/adminarg.php*) o nuovi giornalisti (file *admin/admingiorn.php*) contengono un form compilato nel quale viene eseguita una query di tipo INSERT. Dalle stesse pagine possiamo anche modificare o cancellare aree, argomenti e giornalisti: in questo caso avremo di fronte un form vuoto e due pulsanti per modificare o cancellare i dati del record già selezionato nella pagina generale (l'identificativo del record è sempre trasmesso e propagato con una variabile POST). L'azione di modifica (il listato 13 si riferisce alla modifica di un argomento) si basa su una serie di if che controllano se i campi del form siano vuoti: se non lo sono viene aggiunta l'istruzione specifica che finirà poi nella query UPDATE effettuata sul record selezionato.

La cancellazione, invece, impone un controllo preliminare per conservare l'integrità del database: un giornalista, infatti, si può eliminare solo se non ha mai scritto commenti e, allo stesso modo, argomenti e aree possono essere cancellati solo se non sono già legati ad un articolo.

Per compiere la verifica si scrive una query come questa:

```
$sql_c="SELECT id_commento FROM commento WHERE
id_argomento='".$_POST[id]."'";
```

Se da questa query non è estratta alcuna riga (mysql_num_rows=0) allora posso effettuare la cancellazione, altrimenti l'operazione non può essere effettuata.

Nel caso dei giornalisti la cosa è aggirabile grazie al campo "attivo": modificando il valore di questo campo e ponendolo a 'no' è possibile bloccare l'attività del giornalista (non potrà più accedere all'area riservata e inserire articoli) senza cancellare i suoi dati dal database.

Ormai siamo esperti e modificare i testi o i nostri dati di amministratore (la nuova password va inserita due volte per evitare errori di battitura) e gli iscritti alla mailing list non ci spaventa.

Può essere interessante vedere come viene gestita la spedizione della newsletter. L'ultima data di spedizione è memorizzata nella tabella testi_admin, e questa data viene usata, al caricamento della pagina *admin/adminnews. php*, per estrarre la lista degli articoli (immagine 10) che sono stati pubblicati da quella data in avanti:

```
$sql="SELECT id_giornalista FROM giornalista WHERE attivo='si' AND user='".$_POST['user']."' AND
   password=PASSWORD('".$_POST['pwd']."')";
$ris = mysql_query($sql);
$tab=mysql_fetch_array($ris);
$righe=mysql_num_rows($ris);
if ($righe==0) {
die ("Mi spiace, user e password non sono corrette o non sei più inserito nel database dei giornalisti.
   Controlla meglio l'inserimento");
}
elseif ($righe==1) {
// accesso consentito e contemporanea registrazione della variabile di sessione di accesso e di una variabile di sessione che distingue un giornalista dall'altro
$_SESSION['ok']=true;
$_SESSION['id_giorn']=$tab[0];
}
```

listato 14: *giorn/auth.inc.php*

```
$mod="";
if (trim($_POST[arg])!="") $mod.= "testo_argomento='".trim($_POST[arg])."', ";
if (trim($_POST[area])!="") $mod.= "id_area='".trim($_POST[area])."', ";
if ($mod!=="") {
$mod=rtrim($mod,", ");
// questo rtrim elimina la virgola e il carattere vuoto che si trovavano nella parte destra di $mod
$modifica="UPDATE argomento SET".$mod." WHERE
   id_argomento='".$_POST [id]."'";
mysql_query($modifica);
echo "<span class='titoletto'> Aggiornamento dell'argomento effettuato
   </span><p>";
}
```

listato 13: *admin/adminarg.php*

```
$q="SELECT testo, date_format
   (testo,'%d-%m-%Y') FROM testi_admin
   WHERE id_testo='news'";
$r=mysql_query($q);
$t=mysql_fetch_array($r);
echo "L'ultima volta che hai spedito
   la newsletter era il giorno
   <strong>".$t[1]."</strong><br/>
   Dopo di allora sono stati pubblicati
   questi commenti:<p>";
$sql="SELECT titolo,
   date_format(data_ins,'%d-%m-%Y')
   FROM commento WHERE
   pubblicabile='si' AND
   data_ins>=".$t[0]." ORDER BY
   data_ins";
```

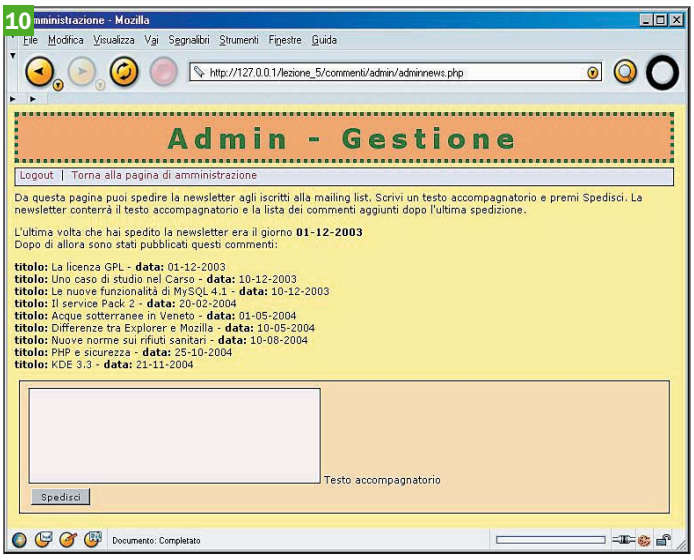
L'amministratore deve solo, se vuole, inserire un testo di accompagnamento alla newsletter e poi premere il tasto "Spedisci": automaticamente la newsletter verrà spedita agli iscritti (per rispetto della privacy tutti gli indirizzi saranno in campo nascosto Bcc) e conterrà l'indicazione dei nuovi commenti. Sempre automaticamente viene aggiornata la data di ultima spedizione della newsletter inserendo la data odierna (lezione 1):

```
// aggiorno la data dell'ultima spedizione con la data odierna
$agg="UPDATE testi_admin SET
   testo='".date('Y').date('m').date
   ('d')."' WHERE id_testo='news'";
mysql_query($agg);
```

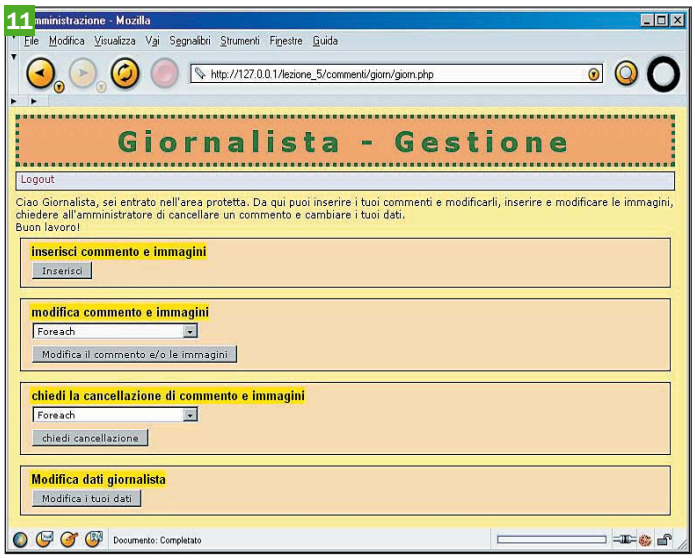
Adesso non ci resta che cliccare su logout (lezione 2) e uscire dalla gestione dell'amministratore.

Area di gestione del giornalista

Ogni giornalista presente nel database (in stato attivo) può accedere ad un'area di gestione specifica dalla quale potrà inserire e modificare commenti e foto, chiederne la cancellazione all'amministratore (non può farlo direttamente per scelta dell'editore) e infine modificare



Elenco dei commenti *pubblicati dall'ultima spedizione della newsletter*



***Le opzioni** di gestione del giornalista*

i propri dati di accesso.

La logica di accesso è uguale a quanto visto, ma diversamente dal caso dell'amministratore unico, l'autenticazione (il form di accesso si trova nel file *giorn/index.php*) dovrà memorizzare l'identificativo univoco del giornalista e verificare lo stato della sua registrazione bloccando l'accesso ai giornalisti disattivati.

Usiamo allo scopo il file *giorn/auth.inc.php* (listato 14) introducendo un nuovo controllo sulla query di accesso e inizializzando una variabile di sessione di nome "id_giorn" che memorizza il valore della chiave identificativa del giornalista.

Questa nuova variabile serve ad impedire che un giornalista possa vedere e modificare gli articoli e i dati degli altri giornalisti.

Entriamo nella pagina generale (immagine 11): inseriamo come user il nome del giornalista (es: "giuseppe") e come password il cognome (es: "verde"), entrambi minuscoli. Proviamo poi a vedere cosa succede usando il giornalista giovanni bianchi che è stato disabilitato.

Riguardo le pagine di gestione coinvolte, vale la pena spendere due parole sull'inserimento delle immagini (lezione 3), operazione contestuale all'inserimento del testo di un commento (file *giorn/giornins.php*).

Nel caso decidessi di inserire una o due immagini dovrò rispettare tre condizioni decise in fase di costruzione del sito: larghezza e altezza dovranno essere minori di 120 pixel e l'immagine dovrà essere di tipo JPG.

Per controllare il rispetto di queste regole uso la funzione PHP getimagesize, registrando i valori ottenuti (larghezza, altezza, tipo e attributi dell'immagine) in una serie di variabili grazie alla funzione list:

```
list($width1, $height1, $type1, $attr1)
  = getimagesize($_FILES
  ['immag']['tmp_name'][0]);
// l'immagine sarà caricata se
$width1<=120, $height1<=120 e
  $type1=2 (il valore 2 indica un file jpg)
```

Passati questi controlli devo caricare l'immagine sul server dandole come nome il codice del commento cui si riferisce. Per ricavare il codice identificativo (chiave) uso la funzione mysql_insert_id che mi restituisce proprio il codice che cerco (estrae il valore del campo autoincrementale presente nell'ultima query di inserimento dell'articolo), quindi carico l'immagine con move_uploaded_file e infine rinomino (funzione rename) l'immagine usando la codifica decisa in fase progettuale:

```
$percorso="../img/";
  $dir1=$percorso.$_FILES['immag']
  ['name'][0];
$id_c=mysql_insert_id();
if($_FILES['immag']['name'][0]) {
move_uploaded_file($_FILES['immag']
  ['tmp_name'][0], $dir1);
rename ($dir1,$percorso.$
  id_c."_1.jpg");
}
```

Contestualmente all'inserimento del commento viene mandata una mail all'amministratore (copia al giornalista) per avvertirlo che c'è un nuovo commento da approvare per la pubblicazione sul sito.

Un giornalista può poi modificare (file "giorn/giornmod.php") uno dei suoi articoli (comprese le immagini), però in questo caso il commento modificato viene reso nuovamente "non pubblicabile" e viene mandata una mail all'amministratore per informarlo che deve ricontrollare il commento modificato per approvarlo.

Il giornalista non può cancellare i propri articoli, ma può chiedere all'amministratore, attraverso il form della pagina "giorn/giorncanc.php", di farlo: anche in questo caso all'amministratore arriverà una mail con la richiesta.

Con il logout usciamo anche da questa area di amministrazione

Il sito dei commenti è adesso pronto per essere gestito e aggiornato, e tutte queste operazioni non saranno più compiute modificando il codice delle pagine. Il compito del WebDeveloper è finito!

Conclusioni

Il corso WebDeveloper PHP (e MySQL) finisce qui, dopo 5 lezioni, con questo esempio applicativo da studiare e riadattare secondo le proprie esigenze.

Lo scopo del corso era far intuire le potenzialità di un linguaggio open-source server side come PHP, utilizzabile con profitto sia in modalità stand alone sia in coppia con un qualsiasi DBMS allo scopo di sviluppare siti dinamici. E proprio in un'ottica di completezza di formazione del WebDeveloper, sono stati approfonditi anche i concetti di base, teorici e pratici, della creazione di un database.

Gli esempi trattati sono stati presi da esperienze reali "sul campo", ma, al solito, possono essere considerati solo un punto di partenza per raggiungere nuovi traguardi... il limite, generalmente, è dato dalla nostra fantasia.

Buon lavoro a tutti! ■

Gli altri corsi da Webmaster disponibili nel CD

Nel *CD Guida 2* allegato a questo numero di *PC Open*, all'interno della cartella *PDF/Corsi*, trovate due corsi completi che possono essere un utile complemento al corso PHP. Uno è il corso *Web Developer ASP*, 97 pagine suddivise in quattro lezioni per capire come realizzare siti dinamici in tecnologia ASP.

Il corso *Webmaster* spiega, invece, in 88 pagine suddivise in otto lezioni tutto quello che bisogna sapere per costruire un sito e imparare il linguaggio HTML 4.01, i CSS (fogli di stile), Java Script e CGI. Il corso è completato da utili consigli per promuovere il proprio sito on line.