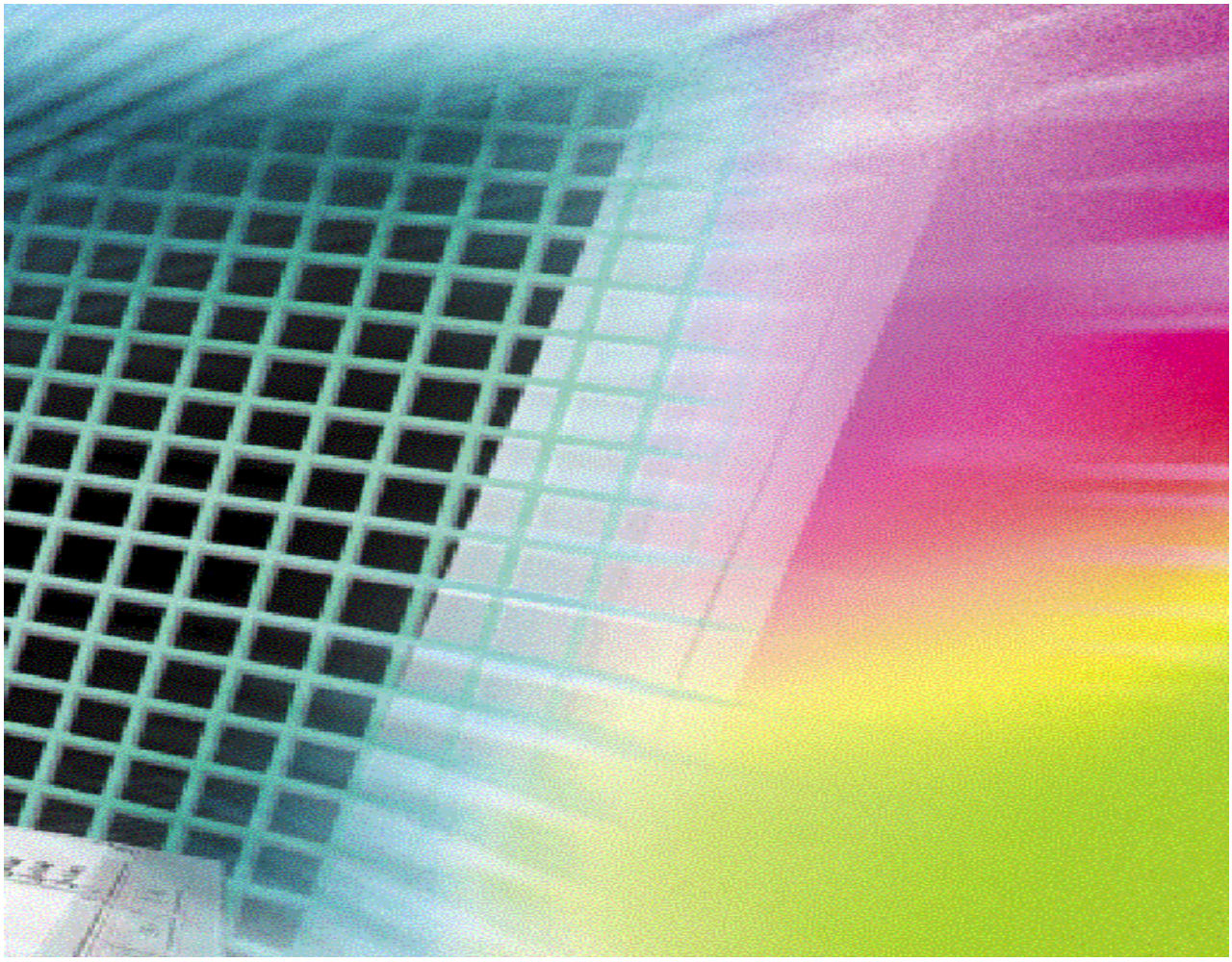
MASSIMO UBERTINI



PHP - JSP

WWW.UBERTINI.IT

PIATTAFORMA LAMP (LINUX, APACHE, MYSQL, PHP)

Apache (www.apache.org)

Scaricare i sorgenti dai rispettivi siti (www.apache.org , www.php.net, www.mysql.com). php-4.0.4pl1.tar.gz e apache_1.3.20.tar.gz.
Questi pacchetti contengono i file sorgente dei programmi, prima del loro utilizzo si devono compilare i file, assicurandosi di aver installato il compilatore C/C++.
Scompattare i file in una directory di lavoro: gunzip file php-4.0.4pl1.tar.gz; tar xvf php-4.0.4pl1.tar.gz; stessa cosa per Apache: gunzip apache_1.3.20.tar.gz; tar xvf apache_1.3.20.tar.gz.
Dopo queste operazioni si avranno due nuove cartelle, apache_1.3.20 e php-4.0.4pl1.
Per installare questi pacchetti si devono avere i permessi di amministratore (ossia accedere al sistema come root).
S'installerà il PHP direttamente all'interno di Apache e non come modulo.
Tutte queste operazioni devono essere eseguite dalla riga di comando.
Entrare nella cartella di Apache e scrivere.
./configure -prefix="path/in/cui/vogliamo/installare/apache" [invio]
Attendere che il file di configurazione crei il Makefile.
Andare alla cartella del PHP e scrivere.
./configure -with-apache="path-dei-sorgenti-di-apache" -enable-track-vars-prefix=path/in/cui/vogliamo/installare/php [invio]
Quando il makefile è stato creato scrivere.
make [invio];
e
make install [invio];
Se l'operazione di compilazione non da nessun errore l'installazione ha avuto successo.
Ora occorre tornare nella cartella dei sorgenti di Apache per ultimare la compilazione del server web, scrivere.
./configure -enable-track-vars [invio]
dopo aver riconfigurato Apache si può passare alla sua compilazione ed installazione.
make [invio]
make install [invio]
Alla fine della compilazione si ha un messaggio di successo.
Far partire Apache entrando nella sua sottocartella *bin* e digitare.
./apachectl start
Copiare per sicurezza questo file (*apachectl)* nella directory */bin* in modo da poter far partire Apache da qualsiasi posizione all'interno del sistema.

MySQL (www.mysql.com)

Una volta scompattato il pacchetto nella directory, bisogna far partire lo script che in automatico genera il make file:
./configure[invio]
se si vuole installare MySQL in una directory diversa da quella di default è possibile scrivere:
./configure --prefix=[percorso e nome della directory in cui volete installare MySQL]
Quando lo script finisce la configurazione automatica è possibile installare i file con:
make [invio]
e
make install [invio]
Una volta installato, inizializzare il database creando delle tabelle di sistema: spostarsi nella directory in cui si è installato MySQL, accedere alla sottodirectory /bin ed eseguire:

./mysqk_install_db
Infine non resta che far partire il demone del MySQL digitando:
./safe_MySQLd &
Il simbolo & serve per far eseguire il demone in background.
Se non si ricevono errori durante l'avvio del demone MySQL è stato installato ed avviato con successo nella macchina. Non rimane che abilitare le funzioni nel PHP per il MySQL: si deve ricompilare PHP.
Andare nella directory dove si trovano i sorgenti del PHP e riconfigurare l'engine in questo modo:
./configure --prefix-use-MySQL=[directory in cui avete installato MySQL] [invio]
reinstallare PHP scrivendo:
make [invio]
e
make install [invio]

PHP (www.php.net)

Configuriamo PHP attraverso il php.ini, attraverso questo file è possibile personalizzare alcune importanti impostazioni o abilitare/disabilitare molte funzioni del PHP.
La prima voce che si trova è *Language Options*. Altra parte molto interessante è quell'intitolata *Resource Limits*. Nella prima voce è possibile settare il numero di secondi per l'expired di uno script (30 secondi di default), nella seconda voce è possibile scegliere la quantità di memoria RAM da riservare al PHP. Nella parte successiva, *Error Handing and logging*, è possibile personalizzare i messaggi di errore in caso di sbagli nel codice. Nella sezione *Paths e Directories* ci sono varie voci molto interessanti da personalizzare. *doc_roor* è la cartella che contiene le pagine in PHP, *extension_dir* è la directory in cui si trovano le estensioni per altri servizi del PHP.
Nella sezione *File Upload* è possibile settare tutti i valori per l'upload dei file direttamente dalle pagine web: File Upload settato su *ON* permette l'upload dei file , diversamente per vietare questa possibilità scrivere *OFF*.
Upload_tmp_dir setta la cartella in cui riversare i file che arrivano dall'esterno, lasciare questo valore vuoto se si vuole settare all'interno dello script il nome di questa cartella. *Upload_max_filesize* setta la grandezza massima dei file permessa nell'upload in mega. Per coloro che hanno installato PHP sotto Win32 nella sezione *Windows Extension* si possono abilitare/disabilitare i file DLL che regolano alcuni servizi. Si trovano all'interno dell'elenco i file per abilitare l'IMAP, la manipolazione/ creazione dei PDF, la manipolazione delle GIF e vari altri servizi. Per abilitare il servizio occorre cancellare il punto e virgola iniziale, per disabilitarlo aggiungere il punto e virgola all'inizio. Nel *php.ini* sono presenti vari altri parametri: gestione della posta; direttive per MySQL; direttive per msql; gestione dei cookies; gestione dei log.

Apache (www.apache.org)

È buona regola effettuare tutte le modifiche ai file di configurazione, sia di Apache sia del modulo PHP, a server spento in modo da essere sicuri che al riavvio del server stesso siano caricate le nuove impostazioni.
Dopo aver attivato il server web, le pagine locali saranno raggiungibili, mediante il browser, all'indirizzo *http:///localhost* oppure *http://127.0.0.1*.
Apache, come server web, ha necessità di essere associato al numero della macchina su cui è installato in modo da poter rispondere alle richieste dei visitatori connessi. Teoricamente il problema sorge quando si decide di eseguire l'installazione in locale; infatti, presupponendo che di fare le prove off line, non si avrà un IP canonico da inserire. In che modo, allora, associare il server alla macchina?
La risposta, da quanto detto prima c'è già: i PC non connessi ad Internet, ma con un accesso remoto installato, hanno un indirizzo di default indicato dal numero *127.0.0.1* o, in alternativa, *localhost*. Volendo, è possibile raggiungere le pagine anche in un altro modo oltre a *localhost* o *127.0.0.1*. Bisogna installare una connessione di accesso remoto sul PC (di qualunque tipo: modem analogico, ISDN, XDSL). L'accesso remoto, infatti, installa dei pacchetti di rete che permettono un'ulteriore identificazione del PC stesso oltre ai nomi di default (*localhost*, *127.0.0.1*). In pratica, possiamo assegnare al PC qualsiasi nome; per fare questo si apre il *Pannello di controllo* e cliccare sull'icona *Prestazione e manutenzione*; selezionare *Sistema* nella finestra *Proprietà del sistema* nella scheda *Nome computer* si avranno davanti tre campi di testo:

1. *Descrizione computer.*
2. *Nome Computer.*
3. *Gruppo di lavoro.*

Se si decide di raggiungere le pagine in locale digitare, ad esempio, il nome *pippo* basterà inserire questa parola nel campo *Descrizione computer* e riavviare il PC.
Da questo momento, una volta installato, il server web sarà raggiungibile in tre modi diversi:

1. *http://localhost*
2. *http://127.0.0.1*
3. *http://pippo*

Installazione del server.

1. Cliccare sul file .EXE; dopo aver accettato i termini della licenza, si avrà di fronte una schermata con tre campi da riempire; anche se non è un'operazione fondamentale per i primi due (potrebbero essere lasciati vuoti visto che si è in *localhost*), inserire il nome del PC (*pippo*) mentre nel terzo campo, è fondamentale, scrivere l'indirizzo e-mail.
2. Spuntare il radiobox relativo a *Run as a service for All Users -- (Recommended)* e cliccare su *Next.*
3. Selezionare *Setup Complete* e andare avanti.
4. Scegliere la directory dove installare Apache. Per comodità lasciare quella di default: *C:\Programmi\Apache Group* .
5. Cliccare su *Install*, aspettare che finiscano le operazioni e riavviamo il PC.

Fare una prova: dal menu *Start* aprire la cartella *Apache httpd Server* e selezionare *Start Apache in Console*.
Se tutto è andato bene, si apriranno due finestre del DOS; una sarà richiusa immediatamente mentre nell'altra, rimasta attiva, ci sarà scritto *Apache/1.3.24 <Win32> running...*
Il web server è acceso e pronto a lavorare. Per un'ulteriore conferma aprire il browser e digitare *http:///localhost* o *http://pippo*: si avrà davanti una semplicissima pagina web.

Di default, Apache cerca le pagine da caricare in una cartella (all'interno della directory *Apache Group\Apache*) chiamata *htdocs* ed è qui che, in teoria, andranno i lavori in PHP per essere visualizzati. È possibile, comunque, variare il percorso di questa cartella e, ad esempio, sceglierne una a piacere dopo averla creata. Per esempio, se si volesse come directory base, la cartella *test*; dal menu *Start, Tutti i programmi, Apache httpd Server, Configure Apache Server*, selezionare il file *Edit the Apache httpd.conf Configuration File* (è il file che gestisce tutte le opzioni del server). Una volta aperto, con la funzione *Cerca*, si arriva alla riga *DocumentRoot* (di default dovrebbe essere *C:/Programmi/Apache Group/Apache/htdocs*) e cambiare *htdocs* in *test*. Poco più sotto si troverà *<Directory "C:/Programmi/Apache Group/Apache/htdocs">* anche in questo caso si dovrà sostituire *htdocs* in *test* (affinché le nuove impostazioni abbiano effetto sarà necessario riavviare il server). Per adesso sono finite le modifiche da apportare al file di configurazione del web server; d'ora in poi la cartella di lavoro sarà quella appena creata e sarà qui che si dovranno mettere tutte le pagine da provare. Per poter lavorare correttamente si devono conoscere i comandi base utilizzati da Apache. Nelle versioni precedenti, all'interno della cartella *Apache httpd Server*, c'erano delle icone che permettevano di far ripartire o fermare il server. A partire dalla 1.3.20, invece, è presente solo l'icona per lo *start* mentre le altre operazioni andranno eseguite tramite DOS (non usare CTRL + C che chiuderebbe la sessione invece di arrestarla). Aprire, quindi, il prompt del DOS e, digitare questa riga, spostarsi nella cartella dove è installato Apache: *C:\>cd programmi\apache~1\apache*. Una volta nella directory di Apache è possibile scegliere tra tre operazioni.

1. Fermare il server digitando *apache -k stop*
2. Far ripartire il server scrivendo *apache -k restart*
3. Attivare il server direttamente da DOS digitando *apache -k start*

Questi comandi saranno utili in seguito quando, apportate alcune modifiche per il PHP, so dovrà far ripartire la macchina. Per non avere sempre in primo piano la finestra del DOS quando sia avvia il server, creare un collegamento sul desktop dell'icona *Start Apache in Console*; cliccare con il tasto destro del mouse sul collegamento e scegliere *Proprietà*; dal menu a tendina del comando *Esegui* selezionate *Ridotto a icona* e dare l'OK in questo modo saranno più veloci le operazioni di avvio e non si avrà più il problema del prompt. Sempre relativamente al DOS impostare, come directory di lavoro, quella in cui è installato Apache; infatti, il DOS lo si usa spesso solo nei casi in cui si vorrà fermare o far ripartire il server in questa maniera non si dovrà spostarsi ogni volta ma ci si troverà direttamente nella cartella giusta. Per fare questo aprire un prompt e cliccare con il tasto destro del mouse sulla barra del titolo; selezionare *Proprietà* e, nel campo *Directory di lavoro*, scrivere il percorso per arrivare alla cartella di Apache: in questo caso sarà *c:\programmi\apache~1\apache*; infine cliccare su *Applica*.

MySQL (www.mysql.com)

MySQL è un **RDBMS** (*Relational DataBase Management System*) open source, per costruire pagine web dinamiche, i database permettono di organizzare i tanti dati che si raccolgono. una mailing list, un forum, un sito di e-commerce.
Il PHP contiene al suo interno numerose funzioni per la connessione dei database MySQL.
L'installazione e l'esecuzione di MySQL nei sistemi Win32 è semplificata perché l'eseguibile da scaricare da www.mysql.com si occupa d'installare automaticamente i file. Terminata l'installazione (la cartella d'installazione di default è *c:/mysql*) per avviare MySQL si deve accedere alla sottodirectory *c:/mysql/bin* attraverso DOS e digitare quanto segue: *start MySQLd-opt --skip-name-resolve --skip-grant-tables --language=italian [invio]*
Da questo momento MySQL è in esecuzione in background nel sistema, per terminare l'esecuzione digitare quanto segue sempre da DOS e sempre all'interno della cartella bin:
MySQLadmin -u root shutdown
Aprire il file *PHP.INI*. Spostarsi verso la fine del documento e cercare la sezione che riguarda MySQL basterà individuare il blocco di testo che inizia così:

[MySQL]
; Allow or prevent persistent links
Modificare le seguenti stringhe in modo da impostare questi valori:
mysql.allow_persistent = On
mysql.max_persistent = -1
mysql.max_links = -1
mysql.default_port = 3306
mysql.default_host = localhost
mysql.default_user = (potete lasciarlo vuoto)
mysql.default_password = (potete lasciarlo vuoto)
Fatto questo è possibile installare il database. La procedura è molto semplice: cliccare sull'icona *Setup.exe* e seguire le varie fasi del wizard; il consiglio è di cambiare la directory d'installazione da *C:\mysql* in *C:\Programmi\mysql* così facendo si avranno tutti i software nella stessa cartella.
Al contrario di ciò che accade normalmente, MySQL non comparirà direttamente nel menu *Start, Tutti i programmi*; per avviare il database si deve aprire la cartella *mysql* situata nella directory *Programmi*. Tutto ciò che interessa si trova nella cartella *bin*; più in dettaglio si lavorerà principalmente con un solo file .EXE: *winmysqladmin.exe*
Creare un collegamento sul desktop per sveltire le operazioni di attivazione del database. Fatto questo è possibile provare ad avviarlo: cliccare due volte su *wynmysqladmin.exe*; dato che è la prima volta che lo si lancia ci sarà chiesto di immettere *Username* e *Password*; riempire tutti e due i campi e dare l'OK.
Se tutto è andato per il verso giusto dovrebbe comparire un piccolo semaforo nella System Tray; naturalmente il semaforo dovrebbe avere la luce verde accesa viceversa un bel semaforo rosso starebbe ad indicare che c'è stato qualche intoppo.
Cliccare con il mouse sull'icona del semaforo e selezioniamo *Show Me* in modo da far aprire la finestra del programma con i vari menu a scelta. Il più importante è il menu *Databases*, ovvero la lista di tutti i *db* presenti e la relativa struttura.
Per chiudere MySQL premere sulla *X*; c'è un modo più dolce per dire al software che non si vuole più utilizzarlo: una volta che la schermata è aperta cliccare con il tasto destro vicino al semaforo e è possibile scegliere tra le due azioni principali (selezionare il rispettivo sottomenu in base al sistema operativo):
ShutDown this Tool
Ferma e chiude completamente l'applicazione
ShutDown this Server
Ferma momentaneamente l'applicazione e la mantiene in standby pronta per ripartire con l'opzione *Start the Server*

PHPMyAdmin (phpmyadmin.sourceforge.net/)

È un'interfaccia grafica, visualizzabile tramite browser, che permette d'interagire con MySQL. È possibile creare, editare, cancellare e modificare sia le tabelle sia i database. Scompattare lo ZIP e rinominare la cartella *PHPMyAdmin*. Copiare questa cartella nella directory principale del web server; in questo caso, quindi, si dovrà andare nella cartella *test* all'interno di Apache. Adesso è possibile iniziare a fare le modifiche; il file da modificare è il *config.inc.php*. Aprirlo, ci sono quadratini neri e il file è praticamente illeggibile. Il blocco note è un'applicazione per documenti di testo, ma c'è da aggiungere che il file che si è aperto è stato compilato su un sistema operativo diverso da Windows UNIX o derivati; ecco quindi il perché di quei caratteri strani, si sta facendo eseguire a blocco note un file compilato su un'altra piattaforma. Il problema si risolve facilmente e velocemente: aprire un qualsiasi altro editor di testo (Wordpad); copiare il testo illeggibile dal blocco note e incollarlo nel nuovo documento come per magia si avrà il testo in chiaro e pronto per essere modificato; adesso fare l'operazione inversa e riportare il tutto nel blocco note. Nel rincollare il testo pulito nel blocco note è probabile che rimangano uno o due spazi tra la fine del testo stesso e il fondo del documento. Questo non deve

assolutamente accadere in caso contrario si avrà un errore in PHPMyadmin; detto in altre parole tra la riga
set_magic_quotes_runtime(0);
?>
e la fine della pagina non ci deve essere nessuna riga vuota.
Dal file *config.inc.php* aperto; cercare le righe seguenti ed accertarsi che siano impostate così:
$cfgServers[1]['host'] = 'localhost';
$cfgServers[1]['port'] = '';
$cfgServers[1]['adv_auth'] = FALSE;
$cfgServers[1]['stduser'] = '';
$cfgServers[1]['stdpass'] = '';
$cfgServers[1]['user'] = 'pippo';
$cfgServers[1]['password'] = 'pippo';
$cfgServers[1]['only_db'] = '';
$cfgServers[1]['verbose'] = '';
$cfgServers[1]['bookmarkdb'] = '';
$cfgServers[1]['bookmarktable'] = '';
I campi *cfgServers[1]['user']* e *$cfgServers[1]['password']* devono essere settati con gli stessi valori utilizzati nel primo avvio di MySQL. Questa appena descritta è solamente una delle tante procedure possibili; ad esempio è possibile impostare più utenti per utilizzare il database, oppure si può impostare un maggiore livello di sicurezza prima di avere accesso a MySQL. Si è messa la cartella dello script all'interno della directory di Apache; a questo punto assicurarsi che il web server e il database siano attivi, aprire il browser; digitare *http://localhost/PHPMyAdmin* e si deve vedere una pagina divisa in due frames: a sinistra il nome di due database impostati di default (mysql e test) mentre a destra i vari comandi che permettono d'interagire per creare, modificare o cancellare tabelle. Il database MySQL non va assolutamente cancellato o modificato contiene, infatti, alcuni dati per l'installazione del database stesso.

PHP (www.php.net)

Per installare il modulo PHP, si deve scompattare il file in una qualsiasi directory sulla macchina; per esempio, *C:\Programmi* e, d'ora in poi, si farà riferimento sempre ad essa. Una volta scompattato si vede che la cartella ha nome *PHP-4.1.2-Win32* rinominarla semplicemente in *PHP*.
Aprire la cartella e cercare il file *PHP.ini-dist*; una volta individuato si deve copiarlo ed incollarlo nella cartella *C:\Windows* ricordarsi di rinominarlo in *PHP.ini*. Ora che è stato rinominato aprire il file con un qualsiasi editor di testo; con la funzione *Cerca* individuare la riga contenente quest'istruzione:
doc_root =
in verità questo parametro potrebbe anche essere lasciato in bianco ma, per completezza, riempirlo con il percorso che porta alla cartella base di Apache (quella che contiene i documenti da testare); facendo riferimento a quanto settato prima nel file di Apache, scrivere: *C:\Programmi\Apache group\Apache\test.*
Nella cartella PHP si deve cercare un file chiamato *PHP4ts.dll*, è fondamentale per il funzionamento del modulo PHP può essere considerato come il motore principale. Una volta trovato si deve copiarlo nella directory *C:\Windows\System*.
Ora si deve modificare nuovamente il file *httpd.conf* di Apache.

1. Cercare la riga *#LoadModule unique_id_module modules/mod_unique_id.so* e, subito sotto, aggiungere quanto segue: *LoadModule PHP4_module c:/programmi/PHP/sapi/PHP4apache.dll* (se necessario modificare il percorso che porta al file PHP4apache.dll)
2. Cercare la riga *AddModule mod_setenvif.c* e sotto aggiungere: *AddModule mod_PHP4.c*

3. Cercare *AddType application/x-tar.tgz* e aggiungere: *AddType application/x-httpd-PHP.PHP AddType application/x-httpd-PHP.PHP3*

Per completare l'opera non resta che impostare le pagine predefinite da caricare. Di default, digitando *http://localhost* (*127.0.0.1*), Apache cercherà la pagina chiamata *index.html*; se non è presente sarà visualizzato il contenuto della directory *test*.
Dato che l'interesse è quello di testare le pagine PHP, può essere comodo indicare al web server di cercare, oltre all'index con estensione .HTML, anche altri tipi di file.
Spostarsi, quindi, fino alla riga *<IfModule mod_dir.c>* e trasformiamo *DirectoryIndex index.html* in *DirectoryIndex index.PHP index.html index.htm index.PHP3 index.phtml.*
In altre parole si è aggiunto, separati da spazi, i nomi delle pagine che Apache deve cercare in automatico nella directory principale. È giunto il momento di vedere se tutto è a posto. Verificare l'esito della procedura con il seguente programma che permette di avere informazioni aggiuntive sull'installazione.

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

Salvare il documento nella cartella *test* con il nome di *info.php*; avviare il web server e digitare *http://localhost* (*http://pippo*, *http://127.0.0.1*).

EASYPHP (WWW.EASYPHP.ORG)

Installare

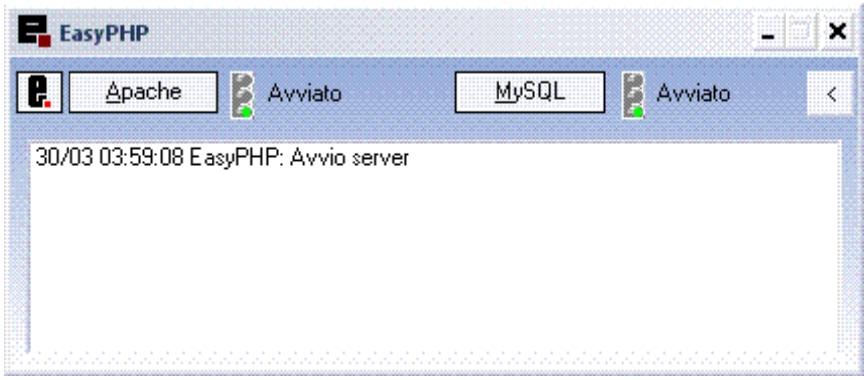
1. Scaricare EasyPHP.
2. Fare doppio click sul file scaricato.
3. Selezionare la cartella di destinazione e seguire le istruzioni.

Lanciare

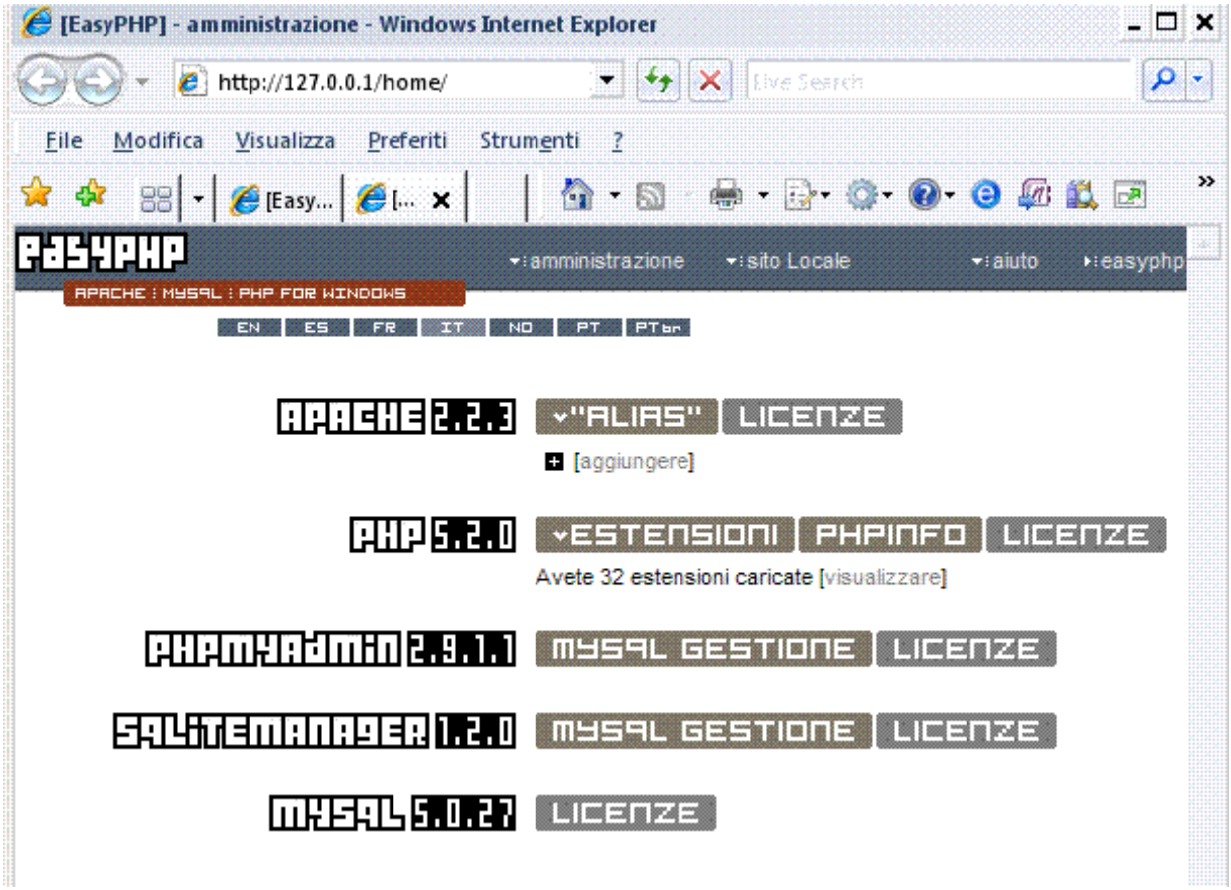
Non si può propriamente parlare di lanciare EasyPHP, il fatto è che sono avviati Apache e MySQL server. Dopo l'installazione, sarà stato creato un collegamento nella cartella *Start, Tutti i programmi, EasyPHP*. La prima volta che si avvia EasyPHP sarà aggiunta un'icona nella System Tray, vicino all'orologio. Cliccare su di essa per accedere ai menu.

1. *Aiuto*
2. *File di log*: riporta ogni errore generato da Apache e MySQL.
3. *Configurazione* : una semplice interfaccia per configurare EasyPHP.
4. *Esplora file F8:* apre la cartella *C:\EasyPHP\www*
5. *Amministrazione CTRL+A:* apre l'URL *http://127.0.0.1/home/* permette di gestire il DB e di amministrare gli alias.
6. *Sito locale F7*: apre l'URL *http://127.0.0.1/*
7. *Riavvia F*: avvia i server Apache e MySQL.
8. *Ferma F3:* ferma i server Apache e MySQL.
9. *Esci*.

Prima di aprire *Amministrazione* o il *Sito Locale*, verificare che *easyphp.exe* sia avviato *Start, Tutti i programmi, EasyPHP, EasyPHP* e che i server funzionino, doppio click sull'icona di EasyPHP a fianco dell'orologio: gli stati di Apache e di MySQL devono essere posizionati sul verde.



Amministrazione CTRL+A: apre l'URL *http://127.0.0.1/home/*



Uso della cartella www

Per fare in modo che lo script possa essere eseguito, ci si deve assicurare di aver posizionato il file della cartella *www*.
Il server Apache è configurato automaticamente per aprire un file indice *index.html* quando è digitato l'indirizzo *http://127.0.0.1.*
Questa è per definizione la pagina di partenza e verifica che EasyPHP è in esecuzione.
È consigliabile creare una cartella per ciascun progetto all'interno della cartella *www*; in questo modo diviene più semplice gestire diversi progetti.

La mia prima pagina in PHP

Questo esempio, *data.php*, mostrerà la data corrente.

```
<html>
<head>
<title>PHP</title>
</head>
<body>
Data Corrente : <?php print (Date("l F d, Y")); ?>
</body>
</html>
```

Salvare la pagina.
Creare una nuova directory all'interno di *www* oppure utilizzare la cartella creata durante l'installazione : *projet1*.
Salvare la pagina contenente lo script PHP in una delle seguenti estensioni: *.php, .php3, .php4*.
Queste estensioni sono state settate da EasyPHP.
Le estensioni possono cambiare con la società che offre l'hosting: ricordare di cambiare le estensioni se necessario.

Azioni da evitare.
Cercare di lanciare lo script facendo doppio clic sul file all'interno della cartella del progetto: questo genera una pagina di errore.
Azioni corrette.
Lanciare EasyPHP, connettersi tramite il browser a *http://127.0.0.1*, aprire la cartella del progetto, quindi cliccare su *data.php*.
A questo punto si deve vedere la pagina correttamente interpretata dal server Apache.

INSTALLAZIONE DI PHP IN IIS

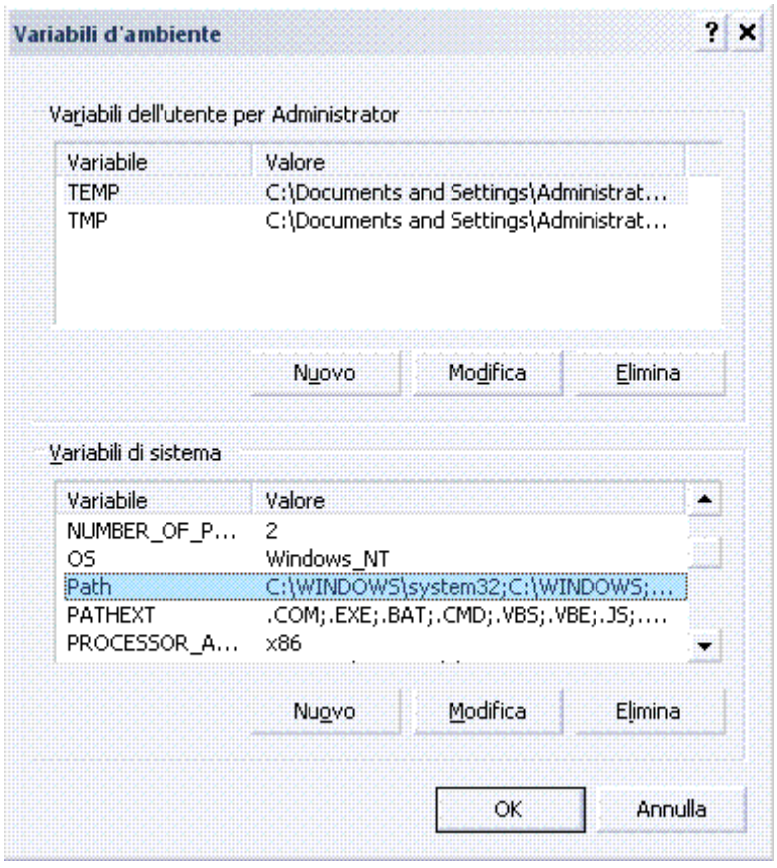
Introduzione

Può essere fatta in tre modi.

1. Filtro ISAPI: garantisce buone prestazioni nell'esecuzione degli script ma può riservare spiacevoli sorprese in quanto a stabilità.
2. CGI: attraverso le CGI IIS è in grado di chiamare il PHP ad ogni esecuzione di script, intercettarne l'output e inviarlo al browser. Se questa tecnica risulta particolarmente stabile, fornisce però prestazioni mediocri, adatte magari allo sviluppatore ma non sufficienti per la realizzazione di un server di produzione.
3. Fast CGI: è la migliore sia sul fronte delle prestazioni sia della stabilità.

Impostazione del PATH

Per evitare di dover spostare diverse librerie di PHP nelle cartelle di sistema, modificare la variabile di sistema *Path* in modo che contenga il percorso alla cartella di PHP appena creata. Aprire il *Pannello di controllo*, cliccare sull'icona *Sistema* e nella finestra che appare selezionare la scheda *Avanzate*. Selezionare *Variabili d'ambiente.*



Dall'elenco delle *Variabili di sistema* selezionare quella di nome *Path* e cliccare sul tasto *Modifica*. A questo punto si deve modificare il valore della variabile aggiungendo in coda la scritta *C:\EasyPHP\php5*; attenzione a non dimenticare il punto e virgola.
Fatto ciò si deve riavviare Windows per far sì che la modifica effettuata abbia effetto.

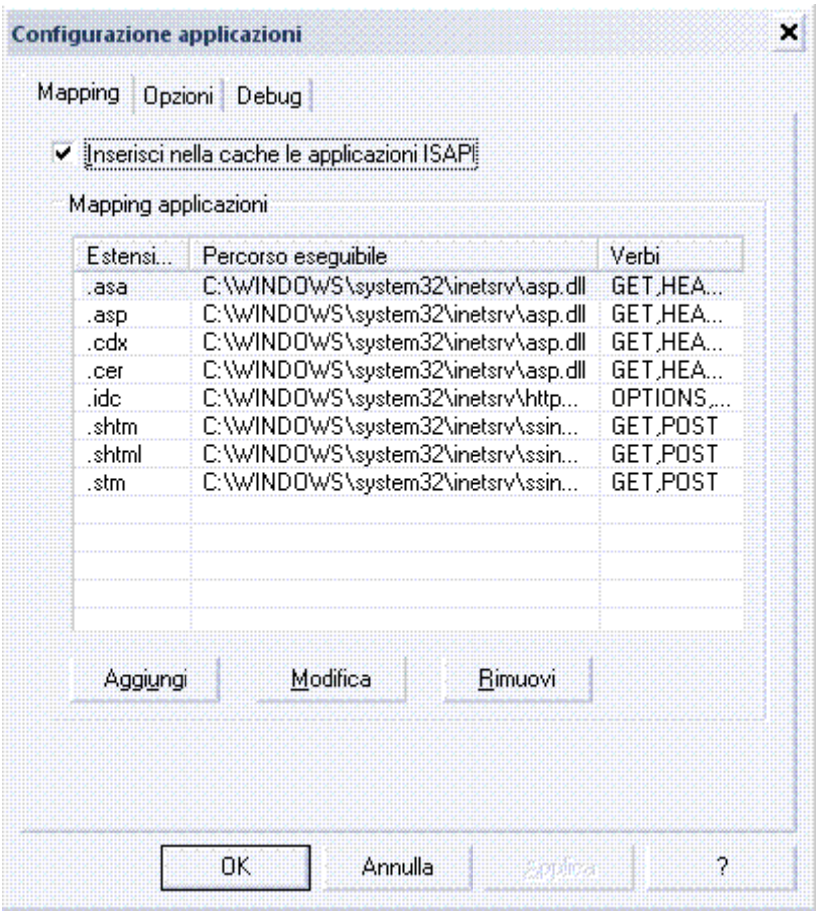
Modifica del php.ini

Nella cartella *C:\EasyPHP\php5* rinominare il file *php.ini-dist* in *php.ini*, è il file di configurazione di PHP, aprirlo con un editor di testi.
Modificare la direttiva *cgi.force_redirect = 0*

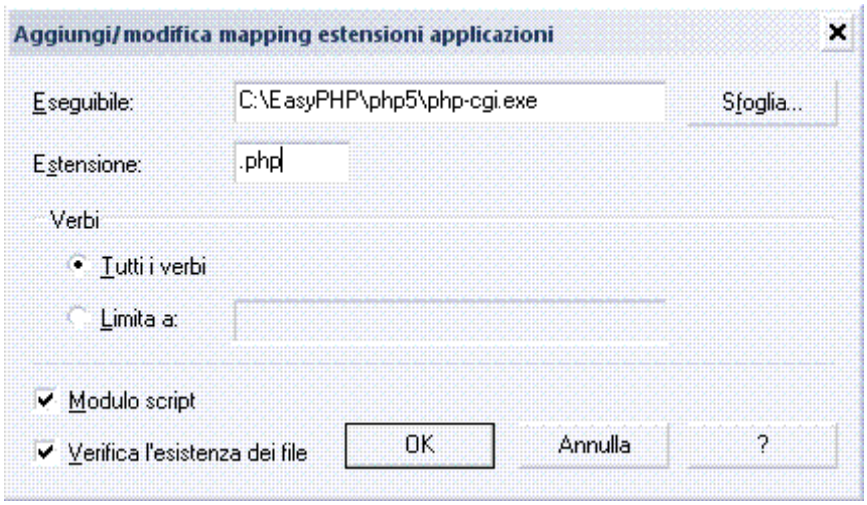
Modificare la direttiva *doc_root = "C:\Inetpub\wwwroot"*
Modificare la *direttiva extension_dir = "C:\EasyPHP\php5\ext"*
Togliere il punto e virgola all'inizio della direttiva *extension=php_mbstring.dll*. L'estensione mbstring è necessaria al corretto funzionamento di phpMyAdmin con i set di caratteri multibyte;
Togliere il punto e virgola all'inizio della direttiva *extension=php_gd2.dll* per caricare la libreria per la manipolazione delle immagini.
Togliere il punto e virgola all'inizio della direttiva *extension=php_mysql.dll*. L'estensione mysql è necessaria per l'interazione con l'omonimo database.
Modificare il valore della direttiva *session.save_path = "C:\Lavoro\Film\Temp"* in modo che punti ad una cartella di files temporanei del sistema.
Salvare il file.

Configurazione di IIS

Istruire IIS affinché usi l'interprete PHP (*php-cgi.exe*) per l'esecuzione dei files con estensione *.php*. Avviare la console di IIS con *Pannello di controllo, Strumenti di Amministrazione, Internet Information Services* espandere la struttura ad albero a sinistra fino a mostrare il *Sito Web predefinito*. Cliccare su questa icona con il tasto destro e selezionare la voce *Proprietà* del menù contestuale. Nella finestra che compare selezionare la scheda *Home directory*.
Cliccare sul bottone *Configurazione…* per accedere alle impostazioni di mapping.



Per aggiungere il supporto agli script PHP si deve cliccare sul pulsante *Aggiungi*.



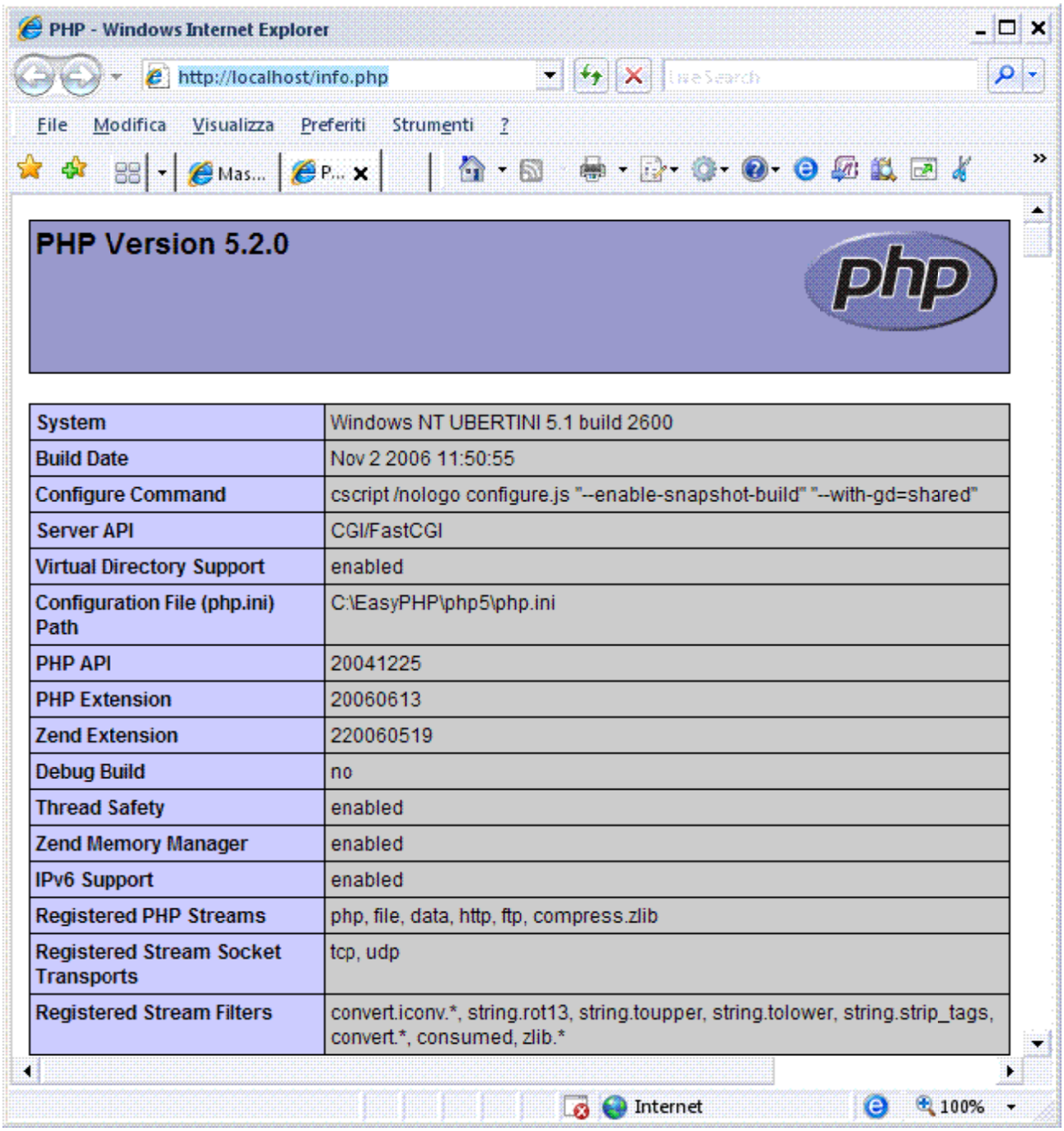
Test con phpinfo()

Verificare l'esito della procedura con il seguente programma che permette di avere informazioni aggiuntive sull'installazione.

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

Salvare il file creato nella cartella *C:\Inetpub\wwwroot* impostando il nome a *info.php*.
Aprire il browser e digitare l'indirizzo: *http://localhost/info.php.*
Attraverso questa finestra sono visualizzati i parametri di configurazione di PHP nel server in cui è eseguito il codice.
Si ha conferma che il file di configurazione *php.ini* è stato correttamente caricato.
Infatti il suo percorso sarà presente tra i primi parametri visualizzati.
Scorrendo la pagina ottenuta si può anche verificare il corretto caricamento delle estensioni (*mysql, mbstring, gd*) che si sono impostate nel *php.ini*.
A volte si verificano problemi di accesso negato durante l'esecuzione di script PHP.
Questi sono riconducibili principalmente a due fattori.

1. Possono dipendere o dalle normali impostazioni sui diritti di accesso dei files in partizioni NTFS, tali problemi si risolvono agendo sulle impostazioni di accesso di Windows XP relative ai file o alle cartelle coinvolte.
2. Possono essere legate alle politiche di accesso del server IIS, i permessi vanno modificati dalla console di amministrazione di IIS.



PHP Version 5.2.0

System	Windows NT UBERTINI 5.1 build 2600
Build Date	Nov 2 2006 11:50:55
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--with-gd=shared"
Server API	CGI/FastCGI
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\EasyPHP\php5\php.ini
PHP API	20041225
PHP Extension	20060613
Zend Extension	220060519
Debug Build	no
Thread Safety	enabled
Zend Memory Manager	enabled
IPv6 Support	enabled
Registered PHP Streams	php, file, data, http, ftp, compress.zlib
Registered Stream Socket Transports	tcp, udp
Registered Stream Filters	convert.iconv.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, zlib.*

INTRODUZIONE

Il PHP è un linguaggio di scripting, utilizzato per lo sviluppo di pagine web dinamiche, è stato rilasciato nella sua prima versione l'8 giugno 1995 da Rasmus Lerdorf.
L'acronimo significa.

1. *Personal Home Page*.
2. *Hypertext PreProcessor*.

Per funzionare, PHP necessita di un suo motore di scripting (script engine) che esegue le parti di codice prima che il server web invii la pagina all'utente. Quando un utente richiede una pagina PHP, il motore esegue il codice contenuto all'interno di quella pagina. Durante l'esecuzione, il codice produce delle informazioni in formato HTML. Infine il file in completo formato HTML è inviato all'utente. Se si prova a visualizzare il sorgente di una pagina in PHP si nota che non che non vi compare nessuna riga di codice PHP. Il vantaggio è proprio questo: nessun utente esterno, tranne il web master, può accedere al codice e modificarlo. Per l'utente esterno, la pagina PHP è esattamente uguale a una qualsiasi pagina in HTML.
Un altro punto a favore del PHP è la sua natura open source, quindi gratuita. Infine, l'alta portabilità del PHP; esso gira su tutti in principali server web.

SVILUPPO DI APPLICAZIONI DESKTOP

La **SAPI** (*Server Application Programming Interface*) chiamata **CLI** (*Command Line Interface*) che significa Interfaccia per la linea di comando, è mirata allo sviluppo di applicazioni shell o desktop con PHP. Notare che ci sono due differenti SAPI: CLI e CGI.
Per vedere quale SAPI.

```
C:\EasyPHP\php5>php -v
PHP 5.2.0 (cli) (built: Nov  2 2006 11:57:36)
Copyright (c) 1997-2006 The PHP Group
Zend Engine v2.2.0, Copyright (c) 1998-2006 Zend Technologies
```

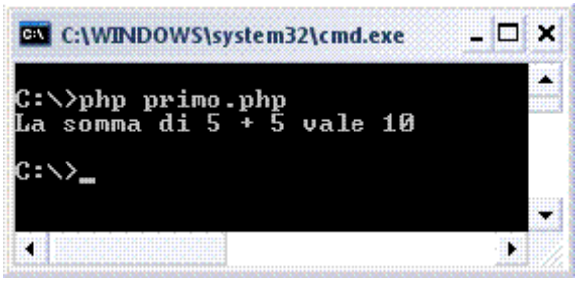
Elenco delle opzioni del PHP disponibili da linea di comando.

```
C:\EasyPHP\php5>php -h
Usage: php [options] [-f] <file> [--] [args...]
       php [options] -r <code> [--] [args...]
       php [options] [-B <begin_code>] -R <code> [-E <end_code>] [--] [args...]
       php [options] [-B <begin_code>] -F <file> [-E <end_code>] [--] [args...]
       php [options] -- [args...]
       php [options] -a
  -a               Run interactively
  -c <path>|<file> Look for php.ini file in this directory
  -n               No php.ini file will be used
  -d foo[=bar]     Define INI entry foo with value 'bar'
  -e               Generate extended information for debugger/profiler
  -f <file>        Parse and execute <file>.
  -h               This help
  -i               PHP information
  -l               Syntax check only (lint)
  -m               Show compiled in modules
  -r <code>        Run PHP <code> without using script tags <?..?>
  -B <begin_code>  Run PHP <begin_code> before processing input lines
  -R <code>        Run PHP <code> for every input line
  -F <file>        Parse and execute <file> for every input line
```

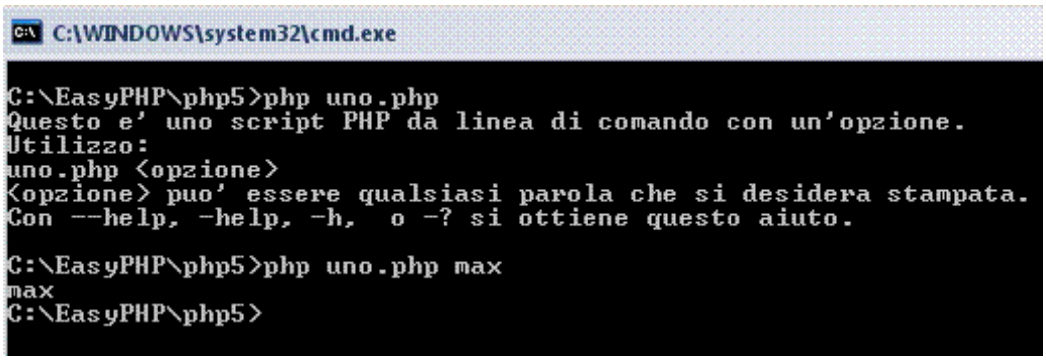
-E <end_code> Run PHP <end_code> after processing all input lines
-H Hide any passed arguments from external tools.
-s Display colour syntax highlighted source.
-v Version number
-w Display source with stripped comments and whitespace.
-z <file> Load Zend extension <file>.
args... Arguments passed to script. Use -- args when first argument starts with - or script is read from stdin
--rf <name> Show information about function <name>.
--rc <name> Show information about class <name>.
--re <name> Show information about extension <name>.

Programma che somma due numeri.

```
<?php
        $a = 5;
        $b = 5;
        $c = $a + $b;
        print "La somma di $a + $b vale $c";
?>
```




```
<?php if ($argc != 2 || in_array($argv[1], array('--help', '-help', '-h', '-?'))) { ?>
Questo e' uno script PHP da linea di comando con un'opzione.
Utilizzo:
<?php echo $argv[0]; ?> <opzione>
<opzione> puo' essere qualsiasi parola che si desidera stampata.
Con --help, -help, -h,  o -? si ottiene questo aiuto.
<?php
} else {
    echo $argv[1];
}
?>
```



Esistono due variabili che si possono utilizzare da linea di comando*: $argc* e *$argv*. La prima è il numero di argomenti più uno (il nome dello script). La seconda è una matrice contenente gli argomenti, iniziando dal nome dello script all'indice zero *($argv[0]).* Nel programma si è verificato se i parametri passati erano di più o di meno di uno. Inoltre se

l'argomento è *--help, -help, -h oppure -?,* si visualizza un messaggio di aiuto, visualizzando in modo dinamico il nome dello script. Se si riceve un argomento differente questo sarà visualizzato.

SINTASSI GENERALE DEL LINGUAGGIO

Affinché l'interprete PHP riesca a distinguere all'interno del codice il linguaggio da interpretare ed eseguire dall'HTML occorre utilizzare dei tag particolari.

```
<html>
<head>
<title>PHP</title>
</head>
<body>
<?php print " Ciao, mondo!"; ?>
</body>
</html>
```

Queste righe di codice stamperanno la parola *Ciao, mondo!.* Si nota immediatamente che non vi è nessuna traccia del codice originario. In altri termini, il client non ha alcun modo per risalire alle istruzioni PHP che hanno generato la pagina richiesta.
Fare clic su *Visualizz/HTML* e si ottiene

```
<html>
<head>
<title>PHP</title>
</head>
<body>
 Ciao, mondo!
</body>
</html>
```

È possibile distinguere il codice PHP delimitato dai tag *<?php e ?>.* L'interprete PHP sa che tutto ciò che si trova all'interno di questi delimitatori deve essere interpretato ed eseguito. Tutto ciò che si trova al di fuori dei tag PHP è normalmente eseguito dal browser.
Esiste anche la possibilità di utilizzare altri tag per distinguere il codice dall'HTML.
<? …… ?>
Questa sintassi è molto simile alla precedente, ma il suo uso deve essere abilitato all'interno del *php.ini*. Nella sezione *Language Option* si deve modificare il parametro *short_open_tag* ed inserire *On*.
<Script language="php" ……</script>
Questi tag sono attivi di default e possono risultare molto utili in quegli editor HTML visuali che non conoscono le estensioni PHP; per esempio Front Page.
<% ……. %>
Questa è la sintassi stile Microsoft ASP e deve essere attivata per poter essere utilizzata. Sempre nel *php.ini* nella sezione *Language Option* si deve modificare il parametro *asp_tags* su *On*. Altra regola fondamentale è il segno di fine comando composto dal ";". La dimenticanza del segno di fine comando genera un errore perché non fa concludere l'esecuzione della pagina. In presenza di errori di questo tipo il PHP fornisce un determinato messaggio *parse error*.

COMMENTI

I commenti rendono il codice più leggibile e modificabile, non sono eseguiti dall'interprete quindi una riga in più non cambierà la velocità di esecuzione dello script.
Il PHP supporta vari tipi di commenti.
Commenti in stile C: */*
Commento in stile C++: *//*
Commento in PHP: *i* oppure **/*

Commenti in stile PERL: #

COSTANTI DI DEFAULT

Ad un nome è associato un valore che si può utilizzare, ma che non si può modificare.
__FILE__
Il nome dello script che è sottoposto al parsing.
__LINE__
La linea dove è presente questo statement.
PHP_VERSION
La versione di PHP in uso.
PHP_OS
Il nome del sistema operativo su cui è fatto girare lo script.
TRUE
Un valore sempre vero.
FALSE
Un valore sempre falso.
E_ERROR
Un errore, differente dal solito errore di parsing (ossia, scrittura del codice), non recuperabile.
E_WARNING
Come sopra, ma in questo caso il PHP può continuare l'esecuzione del codice.
E_PARSE
Un errore del parser (ossia di sintassi del file).
E_NOTICE.
Qualcosa che potrebbe essere come non essere un errore; l'esecuzione non è interrotta.
Per esempio: *<? echo "Linea ", __LINE__ . " del file " ,__FILE__; ?>*
Restituirà: Linea XX del file NOMEFILE.
Queste sono le costanti di default del linguaggio, ma è possibile definire all'interno di uno script le proprie costanti.

VARIABILI

Le variabili possono sono dei contenitori di dati (numeri o lettere) che sono memorizzate all'interno di uno script per essere poi riprese durante l'esecuzione dello stesso. Ad ogni variabile, individuata con il suo nome, è associato un valore.
Le variabili in PHP sono prefissate con il segno "*$*" prima del nome, per esempio *$a = 5;*.

```
<?php
        // Programma che somma due numeri
        $a = 5;
        $b = 5;
        $c = $a + $b;
        print $c;
?>
```

Il PHP non obbliga a dichiarare preventivamente le variabili che s'intendono utilizzare all'interno del programma, ma quest'ultima avviene in contemporanea con l'assegnazione dei valori alla variabile stessa: **dichiarazione implicita**.
Se la variabile non esiste: l'esistenza è verificata con la funzione *isset()* restituisce TRUE se esiste.

```
<?php
        // stampa $n non esiste quindi ha valore nullo
        If (!isset($n)) { print ("$n non esiste"."</br>");}
        // stampa $n è vuota
        If (empty($n)) { print "$n è nulla"."</br>";}
        $m = $n + 5;
        print $m=$m;
```

```
?>
```
La variabile esiste, ma è impostata ad un valore nullo: la nullità è verificata con la funzione *empty ()* restituisce TRUE se è vero.
```
<?php
        // $n esiste ed è nulla
        $n=0;
        // stampa $n esiste
        If (isset($n)) { print '$n esiste</br>';}
        // stampa $n è vuota
        If (empty($n)) { print '$n è nulla</br>';}
        $m = $n + 5;
        print "\$m=$m <br>";
?>
```

Variabili dinamiche

Partiamo dall'assegnazione di una variabile.

$variabile = "Ciao";

Poniamo ora il caso di avere l'esigenza di creare una nuova variabile che abbia come nome il valore della variabile sopra assegnata *($variabile*).

È possibile fare questo utilizzando le variabili dinamiche.

$$variabile = "mondo";

Con quest'operazione il PHP esegue diverse operazioni: interpreta *$$variabile* partendo dalla parte interna dell'espressione *($variabile*) ed in questo modo crea una nuova variabile il cui nome è uguale a *Ciao* ed il valore è uguale a *Mondo*.

Nel PHP si possono annidare le variabili all'interno di altre variabili fino a livelli infiniti, ma è vivamente consigliato non spingersi oltre il secondo livello per non rendere il codice illeggibile.

A disposizione del programmatore c'è anche un certo numero di variabili predefinite, in pratica di variabili il cui valore è già impostato. Queste variabili possono essere divise in tre gruppi.

1 Variabili di Apache

È possibile leggere il valore di ogni variabile o con la funzione *phpinfo()* oppure con un semplice: *echo "NOME_DELLA_VARIABILE";*

GATEWAY_INTERFACE: la versione delle specifiche CGI utilizzate dal server, ad esempio "CGI/1.1".

SERVER_NAME: il nome del server, quello che in Apache si definisce in *ServerName* in *httpd.conf*, ad esempio myhost.com.

SERVER_SOFTWARE: il nome del software utilizzato per il web server.

SERVER_PROTOCOL: il nome e la versione del protocollo tramite il quale è stata richiesta la pagina, ad esempio HTTP/1.0.

REQUEST_METHOD: utilizzato nelle form, può essere GET, POST, HEAD, PUT.

QUERY_STRING: se presente, la stringa tramite la quale la pagina è stata richiesta.

DOCUMENT_ROOT: la DocumentRoot del server, come configurata in *httpd.conf*, ad esempio /var/www.

HTTP_ACCEPT: il contenuto dell'header Accept, ad esempio text/*, image/*, audio/*, application/*.

HTTP_ACCEPT_CHARSET: il charset accettato, ad esempio iso-8859-1.

HTTP_ENCODING: l'encoding della richiesta, se presente; ad esempio, gzip.

HTTP_ACCEPT_LANGUAGE: il linguaggio, ad esempio en.

HTTP_CONNECTION: il contenuto dell'header Connection, ad esempio Keep-alive.

HTTP_HOST: il nome dell'host, ad esempio localhost.

HTTP_REFERER: il nome della pagina dalla quale si proviene.

HTTP_USER_AGENT: il contenuto dell'header User_Agent, ad esempio Mozilla/4.72 [en]

(X11; I; Linux 2.2.14 i586).
REMOTE_ADDR: l'indirizzo IP dell'utente connesso alla pagina.
REMOTE_PORT: come sopra, ma riferito alla porta; ad esempio, le due variabili potrebbero avere un output del tipo: 127.0.0.1 1275.
SCRIPT_FILENAME: il nome dello script richiesto al server, ad esempio prova.php3.
SERVER_ADMIN: il nome dell'amministratore di sistema.
SERVER_PORT: la porta sulla quale il server è in ascolto, ad esempio la porta 80.
SERVER_SIGNATURE: l'eventuale signature del server.
PATH_TRANSLATED: il percorso per lo script richiamato, ad esempio /var/www/php/prova.php3.
SCRIPT_NAME: il path, a partire dalla DocumentRoot, dello script; ad esempio, /php/prova.php3.
REQUEST_URI: l'URI richiesto per accedere alla pagina.

2 Variabili d'ambiente
Dipendendo dalla shell utilizzata. Per leggerle, richiamare all'interno di uno script PHP.
echo "Il mio path è $PATH"; che visualizzerà il path sul sistema.

3 Variabili PHP
argv: un array contenente i parametri passati allo script.
argc: come sopra, ma se lo script è eseguito dalla linea di comando.
PHP_SELF: il nome dello script attualmente in esecuzione.
HTTP_COOKIE_VARS: un array associativo contenente le variabili passate allo script tramite i cookies HTTP.
HTTP_GET_VARS: un array associativo contenente le variabili passate allo script tramite una richiesta GET.
HTTP_POST_VARS: come sopra, ma riferito al metodo POST.
È da ricordare che le ultime tra variabili hanno senso solamente se, nel file di configurazione (*track_vars=On*) oppure all'interno dello script con la direttiva *php_track_vars*, è stato abilitato il tracking delle variabili.

TIPI DI DATO

Scalari

Integer

```
$a = 18;          // decimale
$a = -18;         // decimale negativo
$a = 022           // notazione ottale
$a = 0x12;        // notazione esadecimale
```

Per stabilire se una variabile è integer si usano le funzioni: *is_long(), is_int(), is_integer(), gettype()* restituisce una stringa che descrive il tipo di variabile.

```
<?php
    $a="Pippo";
    $b=5;
    If (is_long($a)) print "$a è un integer</br>";
    else print "$a non è integer</br>";
    print "$b è di tipo: ".gettype($b)."</br>";
    If (is_long($b)) print "$b è un integer</br>";
    else print "$b non è integer</br>";
?>
```

Per convertire un'espressione in un numero intero si usa la funzione *intval(),* oppure si può usare il casting.

Floating point

```
$a = 9.876;
```

```
$a = 9.87e6;
```
Per stabilire se una variabile è FP si usano le funzioni: *is_double(), is_float(), is_real(), gettype()* restituisce una stringa che descrive il tipo di variabile.
```
<?php
        $a="Pippo";
        $b=3.14;
        If (is_double($a)) print "$a è un FP</br>";
        else print "$a non è FP</br>";
        print "$b è di tipo: ".gettype($b)."</br>";
        If (is_double($b)) print "$b è un FP</br>";
        else print "$b non è FP</br>";
?>
```
Per convertire un'espressione in un numero FP si usa la funzione *doubleval(),* oppure si può usare il casting.

Boolean

Per stabilire se una variabile è booleana si usano le funzioni: *is_bool(), gettype()* restituisce una stringa che descrive il tipo di variabile.
```
<?php
        $a=TRUE;
        $b=5;
        If (is_bool($a)) print "$a è booleana</br>";
        else print "$a non è booleana</br>";
        print "$a è di tipo: ".gettype($a)."</br>";
        If (is_bool($b)) print “$b è booleana</br>”;
        else print "$b non è booleana</br>";
?>
```

Casting

Conversioni effettuate in modo implicito e trasparente.
```
<?php
        $a = 10;            // integer
        $b = $a + 1.5;      // real
        $c = "5 + $b";      // string
        $d = $c + 7;        // integer
        print "$a\n";
        print "$b\n";
        print "$c\n";
        print $d;
?>
```
Conversioni effettuate in modo esplicito: forzano una variabile ad un certo tipo, possono essere effettuate usando le funzioni predefinite *intval(), doubleval(), strval()* oppure il casting facendo precedere l'espressione di cui si vuole convertire il tipo dal nome del tipo di destinazione racchiuso tra parentesi tonde.
```
<?php
        $a = 10.8;          // real
        $b = (int)$a;        // integer
        print "$a\n";
        print "$b";
?>
```

Strutturati

Array

Un array è una variabile che contiene più valori.

$colori = array (“rosso”, “ verde”, “blu”);
L’ array *$colori*, non è altro che una variabile con più valori al suo interno.
In PHP esistono due tipi di array.

1. Array scalari ad indice numerico

In questo caso l’array *$colori* è un array ad indice numerico poiché il PHP assegna un numero univoco ai valori dell’array in base al loro inserimento.
$colori[0] sarà uguale a rosso; il primo valore nel PHP è sempre lo zero.
$colori[1] sarà uguale a verde.
$colori[2] sarà uguale a blu.
In PHP non occorre dichiarare preventivamente quanti elementi saranno contenuti all’interno dell’array, ma anzi è possibile aggiungere elementi anche durante l’esecuzione dello script. Per esempio, volendo aggiungere un quarto elemento all’interno dell’array *$colori* basta scrivere:
$colori[]= “viola”;
si utilizzano le parentesi quadre vuote, questo è molto utile quando si vogliano accodare degli elementi all'array senza ricorrere a specifiche funzioni e senza dover andare a leggere il numero di elementi contenuti nell'array: tutto sarà accodato automaticamente e correttamente.
Il PHP accoderà in automatico il valore viola agli altri presenti nell’array ed è possibile visualizzare il suo valore scrivendo:
print $colori[3];
Spazzolare gli elementi contenuti all’interno di un array ad indice numerico.

```
<?php
        $colori = array ("rosso", " verde", "blu");
        $colori[]= "viola";
        // visualizzare i valori dell'array precedentemente creato
        $n = count($colori);
        // la funzione count conta il numero di elementi presenti all'interno di un array
        for ($i = 0; $i< $n; $i++) {
                print $colori[$i]; print "\n";}
?>
```

In PHP, un array di valori può essere esplicitamente creato definendone gli elementi oppure la sua creazione può avvenire implicitamente inserendo valori all'interno dell'array.

```
<?php
        // Questo è un array di numeri interi che si crea esplicitamente
        $primi = array( 2, 3, 5, 7, 11 );
        // Questo array si crea implicitamente
        $pari[0] = 2;
        $pari[1] = 4;
        $pari[2] = 6;
?>
```

Il PHP gestisce le stringhe di caratteri come veri e propri array ad indice numerico.
$stringa =“ Il mondo è bello “;
è possibile raggiungere qualsiasi singolo carattere contenuto in *$stringa*. Per cui:
$stringa[0] sarà uguale al carattere I
$stringa[1] sarà uguale al carattere I
$stringa[2] sarà uguale al carattere “spazio”
e così via fino alla fine della stringa.
Contrariamente a quanto avviene in altri linguaggi di programmazione, in PHP gli elementi di un array possono essere di tipi diversi, in altre parole un array può contenere vari tipi di dato nello stesso array: lettere, numeri interi, numeri in virgola mobile

```
<?php
$colori = array ("Pippo", " TRUE", "3.14");
```

```
$colori[]= "7";
$n = count($colori);
for ($i = 0; $i< $n; $i++) {
        print $colori[$i]; print "\n";}
?>
```

2. Array associativi ad indice stringa

In questo caso è il programmatore a scegliere il nome dell'indice dei valori presenti all'interno dell'array: si basano su coppie *name-value*. Esempio di array associativo.

```
$colori[primo_colore] = "rosso";
$colori[secondo_colore] = "verde";
```

e così via….

In questo caso per visualizzare il secondo colore (verde) basterà l'istruzione seguente.

```
print $colori[secondo_colore];
```

Nella realizzazione di una rubrica personale, invece di definire una variabile per ogni voce della rubrica si può fare un array associativo che comprenda tutte le voci e dopo maneggiare i dati attraverso gli indici che si sono attribuiti alle varie voci.

```
<?php
$a = array(
        "nome" => "Mario",
        "cognome" => "Rossi",
        "email" => "mario@rossi.com",);
// è interessante la possibilità della funzione array di annidare le entries,
// come nell'esempio che segue.
$a = array(
        "primo" => array(
        "nome" => "Mario",
        "cognome" => "Rossi",
        "email" => "mario@rossi.com",),
        "secondo" => array(
        "nome" => "Marco",
        "cognome" => "Verdi",
        "email" => "mario@verdi.com",)
);
echo $a["secondo"]["email"];
?>
```

Per stabilire se una variabile è un array si usano le funzioni: *is_array(), gettype()* restituisce una stringa array.

Strings

La sintassi di base per le stringhe.

```
$string = "Io sono una stringa";
```

Se sono utilizzate le virgolette (""), il contenuto della stringa è espanso (o, tecnicamente, interpolato).

```
$num = 10;
$string = "Il numero è $num";
```

che visualizzerà: *Il numero è 10*.

Come in tutti i linguaggi, comunque, anche con il PHP ci sono i caratteri speciali che vanno fatti precedere da un simbolo di escape.

```
$num = 10;
$string = "Il numero è "$num"";
```

L'output di tale codice non è: *Il numero è 10*, lo script dà un errore di compilazione, le virgolette sono caratteri speciali, ma non per questo non è permesso utilizzarle; la sintassi corretta per il comando riportato sopra è questa.

$num = 10;
$string = "Il numero è \"$num\"";
Altri caratteri speciali sono i seguenti.
\n -> newline
\r -> carriage return
\t -> tabulazione
\\ -> backslash
\$ -> simbolo del dollaro
L'alternativa ai caratteri di escape, quando non ci siano contenuti da espandere, sono i apici (')
$string = '$ è il simbolo del dollaro';
visualizzerà proprio ciò che è contenuto fra gli apici. Attenzione a non cadere nel più consueto degli errori.
$num = 10;
$string = 'Il numero è $num';
non visualizzerà: *Il numero è 10,* ma *Il numero è $num*. Quindi si può affermare che, con gli apici, il contenuto della stringa è riportato letteralmente, ossia com'è effettivamente scritto al suo interno.
Per stabilire se una variabile è FP si usano le funzioni: *is_string(), gettype()* restituisce una stringa che descrive il tipo di variabile.
Per convertire un'espressione in una stringa si usa la funzione *strval(),* oppure si può usare il casting.

Object
PHP è un linguaggio procedurale, ma supporta anche il paradigma OOP.

OPERATORI

Aritmetici
$a + $b // (+) La somma di $a e $b
$a - $b // (-) La sottrazione fra $a e $b
*$a * $b // (*) Il prodotto di $a e $b*
$a / $b // (/) Il rapporto di $a e $b
$a % $b // (%) Il resto della divisione di $a e $b
Il risultato della divisione non è approssimato all'intero più vicino, ma riporta tutto il numero risultante; il numero dei caratteri dopo il punto da considerare è definito nel file *php.ini* alla riga *precision = 14.*
Quindi, saranno riportati solo 14 numeri dopo la virgola, a patto che ci siano.
$a = 12;
$b = 5;
$c = $a / $b;
echo $c;
il risultato è 2.4 e sarà visualizzato proprio come 2.4, non come 2.40000000000000 Quindi, i 14 decimali sono visualizzati solamente se esistono, e non indiscriminatamente come inutili zeri. Il resto della divisione è riportato da solo, senza il risultato della divisione stessa: se nell'esempio precedente avessimo utilizzato *%* al posto di /, il risultato sarebbe stato 2.

Relazionali
Gli operatori di confronto sono quelli che consentono di mettere in relazione tra loro due o più espressioni.
$a == $b //Confronta l'uguaglianza del valore tra $a e $b (stesso valore)
$a===$b //$a e $b sono identici: stesso valore e stesso tipo
$a != $b //$a e $b sono diversi
$a < $b //$a è minore di $b

```
$a <= $b      //$a è minore o uguale di $b
$a > $b       //$a è maggiore di $b
$a => $b      //$a è maggiore o uguale di $b
```

Stringhe

Operatore di concatenazione, denotato da un punto.

```
<?php
        $Nome = "Piero";
        $Cognome = "Bianchi";
        print ($Nome." ".$Cognome);
?>
```

Logici

```
$a & $b       //operatore And ($a e $b)
$a && $b      //come sopra, ma con una precedenza più alta
$a | $b       //operatore Or ($a oppure $b)
$a || $b      //come sopra, ma con una precedenza più alta
$a ^ $b       //operatore Xor ($a oppure $b ma non entrambi)
!$a           //operatore "Not" (vero se $a non è vera)
```

Espressioni

Linguaggio orientato alle espressioni.

```
$a = $b = 2;          //$a = 2 ma anche $b = 2
```

Espressione condizionale.

```
$a ? $b : $c
```

Il valore di questa espressione condizionale dipende dal valore di *$a*, infatti, la valutazione dell'espressione avviene in due passi.
Se *$a* è vera allora è valutata *$b* e quest'ultimo risultato sarà il valore dell'espressione.
Se *$a* è falsa allora è valutata *$c* e quest'ultimo risultato sarà il valore dell'espressione.

ISTRUZIONI

Semplici: non contengono altre istruzioni.

Assegnazione

Operazione con la quale ad una variabile è attribuito un valore, il simbolo = è l'operatore di assegnazione: *$a = 2;*

Assegnazione abbreviata.

```
$a += $b;     equivale     $a = $a + $b;
$a -= $b;     equivale     $a = $a - $b;
$a *= $b;     equivale     $a = $a * $b;
$a /= $b;     equivale     $a = $a / $b;
$a %= $b;     equivale     $a = $a % $b;
$a .= $b;     equivale     $a = $a . $b;
```

Assegnazione incremento e decremento.

```
++$a  //incrementa di uno $a e restituisce il valore incrementato
$a++  //restituisce $a e poi la incrementa di uno
--$a  //decrementa di uno $a e restituisce il valore incrementato
$a--  // restituisce $a e poi la decrementa di uno
<?php
        $i =1;
        print "$i\n";   // i=1
        $i++;
        print "$i\n";   // i=2
        $j = ++$i;
        print "$j\n";   // j=3 ma  i=3
```

```
        $k = $i++;
        print "$k\n";   // k=3
        print "$i\n";   // i= 4
?>
```

Strutturate: sono composte da più istruzioni.

Sequenza

Le istruzioni sono eseguite nello stesso ordine in cui sono scritte.

Selezione

Unaria

```
if (condizione) {
                blocco-istruzioni da eseguire nel caso l'istruzione risulti vera;
                }
```

Poniamo il caso di voler confrontare due variabili, solo se queste due variabili sono uguali si stamperà a video il loro valore.

```
<?php
        $a=5;
        $b=5;
        if( $a == $b) {
        print " Il valore di a è uguale a $a\n";
        print " Il valore di b è uguale a $b";
        }
?>
```

Binaria

```
if (condizione) {
                blocco-istruzioni da eseguire nel caso l'istruzione risulti vera;
                } else {
                blocco-istruzioni da eseguire nel caso l'istruzione risulti falsa;
                }
<?php
        $a=5;
        $b=5;
        if ( $a == $b) {
                        print " I valori $a e $b sono uguali";
                        } else {
                        print " I valori $a e $b sono diversi";
                        }
?>
```

Nidificata

Si usa quando si hanno molte condizioni da eseguire una dietro l'altra. Per esempio, vogliamo stabilire se, date due variabili *$a* e *$b*, *$a* è maggiore, minore o uguale a *$b*.

```
<?php
        $a=5;
        $b=5;
        If ($a<$b) {                                            // primo if
                        print "$a è minore di $b";
                } else {
                                if ($a>$b) {                    // secondo if annidato
                                        print "$a è maggiore di $b";
                                            } else {
                                                print "$a è uguale a $b";
```

```
                    }                   // fine secondo if
                }
?>
```
Questo metodo funziona, ma è troppo macchinoso e porterebbe a grosse difficoltà qualora le condizioni risultassero più di tre, in questo caso si usa il costrutto elseif.

```
if (condizione1) { blocco istruzioni 1 da eseguire per vera e si esce;}
elseif (condizione2) { blocco istruzioni 2 da eseguire per vera e si esce;}
elseif (condizione3) { blocco istruzioni 3 da per vera vera e si esce;}
else { le condizioni precedenti sono false;}
<?php
        $a=5;
        $b=5;
        if ($a<$b) { print "$a è minore di $b";}
        elseif ($a>$b) { print "$a è maggiore di $b";}
        else { print "$a è uguale a $b";}
?>
```

Multipla

Quando si hanno istruzioni mutuamente esclusive. La sintassi dello switch è la seguente.

```
switch (espressione) {
        case valore1:
                blocco istruzioni 1;
        break;
default:
        blocco istruzioni 1;
}
<?php
        $a=3;
        switch($a) {
                case 1:
                        echo "Uscito il segno 1";
                break;
                case 2:
                        echo "Uscito il segno 2";
                break;
                default:
                        echo "Uscito il segno $a";
        }
?>
```

Iterazione

Successione d’istruzioni eseguite ripetutamente.

1 Il programmatore specifica la condizione che determina la fine dell’iterazione
Ciclo a condizione iniziale

Finché una determinata condizione non diventa vera si eseguono le istruzioni comprese all’interno delle parentesi graffe. La sintassi del *while* è la seguente.

```
While( condizione ) {
                        istruzioni da eseguire
                        }
<?php
        // $a rappresenta uno dei due membri della moltiplicazione
        // (l'altro è il 2) poiché la tabellina parte dal numero 1
        $a = 1;
```

```
        // costrutto while
        while ($a != 10) {
                // tabellina del 2 fino a 10
                // finché ( while ) $a è diverso ( != ) da 10 (quindi finché la condizione
                // tra parentesi risulta vera) eseguire
                        $b = $a*2;
                        print "($a * 2)= $b\n";
                // prima di uscire dal costrutto while incrementare di un'unità la
                // variabile $a altrimenti loop infinito che finirà solo dopo il timeout
                        $a++;
}
?>
```

Ciclo a condizione finale

È prima eseguito il blocco istruzioni e poi è valutata la condizione. La sintassi del *do while* è la seguente.

```
do {
        blocco-istruzioni
        }
while (condizione)
<?php
        $a = 1;
        do {
                $b = $a*2;
                print "($a * 2)= $b\n";
                $a++;
        }
        while ($a != 10)
?>
```

2 Il numero d'iterazioni è noto a priori

Nel *for* le condizioni sono tre: per prima è valutata la condizione uno (una solo volta), generalmente questa prima condizione è un'espressione, da qui inizia l'iterazione vera e propria, è valutata la condizione due, se falsa si esce dal ciclo, se vera si esegue il blocco istruzioni. Al termine di ogni operazione è infine valutata la condizione tre.

```
for (condizione1, condizione2, condizione3 ) {
        blocco-istruzioni
}
<?php
        $a = 1;
        for ($a = 1; $a < 10; $a++ ) {
                $b = $a*2;
                print "($a * 2)= $b\n";
                                        }
?>
```

Il *foreach* è un metodo per attraversare un array. Esistono due notazioni sintattiche.

```
foreach(array_expression as $value) istruzione
foreach(array_expression as $key => $value) istruzione
```

La prima attraversa l'array dato da *array_expression*. Ad ogni ciclo, si assegna il valore dell'elemento corrente a *$value* e il puntatore interno avanza di una posizione (in modo tale che al ciclo successivo l'elemento corrente sarà il successivo elemento dell'array).

La seconda esegue lo stesso ciclo con la differenza che il valore dell'indice corrente è assegnato ad ogni ciclo, alla variabile *$key*.

```
<?php
```

```
        $array = array ("uno","due", "tre");
        while (list(, $value) = each ($array)) { echo "Valore: $value\n";}
?>
```
È equivalente al seguente codice.
```
<?php
        $array = array ("uno","due", "tre");
        foreach ($array as $value) { print "Valore: $value\n"; }
?>
<?php
        $array = array ("nome" => "valore", "nome2" => "valore2");
        while (list($key, $val) = each ($array)) { print "$key => $val\n";}
?>
```
È equivalente al seguente codice.
```
<?php
        $array = array ("nome" => "valore", "nome2" => "valore2");
        foreach ($array as $key => $value) { print "$key => $value\n";}
?>
```

FUNZIONI

La dichiarazione di funzioni consente al programmatore di estendere il linguaggio utilizzato. Una funzione può essere interpretata in due modi.

1. Come funzione matematica: relazione tra un input (i parametri passati) ed un output (il risultato della funzione).
2. Come un meccanismo per aggiungere nuovi comandi al linguaggio: è possibile raggruppare assieme un blocco d'istruzioni PHP che potranno essere richiamate ed eseguite.

Sintassi per la dichiarazione di funzione.
```
function nome (parametri formali) {
        blocco istruzioni;
        return ();
}
```
Esempio, sommare di due numeri e visualizzare il risultato.
```
<?php
        $a = 5;
        $b = 5;
        $c = $a + $b;
        print $c;
?>
```
Usando la funzione somma.
```
<?php
        $a = 5;
        $b = 5;
        function somma ($c,$d) { return  ($c + $d); }
        print somma($a,$b);
?>
```
Esempio, la funzione per la generazione di un numero random compreso tra dieci e venti.
```
<?php
        $a = 5;
        $b = 5;
        function random () { return (rand(10,20)); }
        print random();
?>
```
Ogni variabile è caratterizzata da una propria **visibilità**, in pratica dal contesto all'interno del quale essa è definita. In base alla visibilità si hanno due categorie di variabili.

1. Le **variabili globali**: dichiarate nella parte esterna della funzione, sono visibili in tutta la parte esterna dello script, ma non all'interno di funzioni o classi.
2. Le **variabili locali** dichiarate all'interno di funzioni o classi, che non sono accessibili al di fuori di essa.

```
<?php
        // Variabile globale non visibile all'interno della funzione
        $n=99;
        function conta(){ return $n;}
        print "Ci sono: ".conta()." persone";
?>
```

La funzione *conta()* restituisce un valore nullo perché *$n* non è visibile al suo interno.
È possibile rendere visibili le variabili globali all'interno di funzioni o classi dichiarandole *global*.

```
<?php
        // Variabile globale non visibile all'interno della funzione
        $n=99;
        function conta()
                { // $n è visibile all'interno della funzione
                  global $n;
                  return $n;
                }
        echo "Ci sono: ".conta()." persone";
?>
```

Le variabili globali sono accessibili tramite l'array associativo $*GLOBALS*, che è a sua volta una variabile globale.

```
<?php
        // Variabile globale
        $n=99;
        function conta()
        { return $GLOBALS["n"];}
        echo "Ci sono: ".conta()." persone";
?>
```

Oltre a contenere numeri una variabile può contenere caratteri e frasi.

```
<?php
        // Stampa una stringa
        $a = "Ciao Mondo";
        print $a;
?>
```

Funzioni predefinite del linguaggio

La funzione *echo()* serve per scrivere (o stampare) l'output che è inviato al browser del visitatore che accede allo script.

```
<html>
<head><title>echo())</title></head>
<body>
<?php echo "<h1>Benvenuto!</h1>"; ?>
</body>
</html>
```

Grazie ad *echo()* è possibile visualizzare il contenuto di variabili; per esempio, *$HTTP_HOST* è il nome del dominio del sito su cui lo script è eseguito.

```
<html>
<head><title>echo()</title></head>
<body>
<?php echo "<h1>Benvenuto su $HTTP_HOST!</h1>"; ?>
```

```
</body>
</html>
```
Le funzioni *exit()* e *die()* entrambe producono il risultato di arrestare l'esecuzione dello script, con la differenza che *die()* consente anche di stampare un messaggio passato come argomento prima di uscire.
```
<html>
<head><title>exit()</title></head>
<body>
<php exit(); ?>
<p>Questa frase non si vedrà</p>
</body>
</html>
```
Utili per gestire situazioni di errore che non consentono di proseguire l'esecuzione dello script PHP (*fatal error*). Per esempio, controllare la variabile *$n* e mostrare un messaggio di errore, bloccando l'esecuzione del programma, se questo è maggiore di uno.
```
<html>
<head><title>die()</title></head>
<body>
<?
        $n = 0;
        if ($n > 1) die("<h1>$n è maggiore di uno!!!</h1>");
        echo "<h1>$n è minore o uguale ad uno!</h1>";
?>
</body>
</html>
```
abs(): restituisce il valore assoluto di un numero.
```
<?php>
        $a = -3.4;
        print abs($a);              // ritorna3.4
?>
```
acos(): restituisce l'arcocoseno dell'argomento.
```
<?php>
        $arg = 1;
        print acos($arg);           // ritorna 0
?>
```
asin(): restituisce il seno dell'argomento.
atan(): restituisce l'arcotangente dell'argomento.
base64_decode(): decodifica una stringa codificata in MIME base64.
base64_encode(): codifica dati in MIME base64.
```
<?php>
        $str = "Ciao, io sono pippo";
        echo "$str\n";
        $enc_str = base64_encode($str);
        echo "$enc_str\n";
        $dec_str = base64_decode($enc_str);
        echo "$dec_str\n";
?>
```
si passa allo script la stringa *$str* che è prima codificata e visualizzata, poi decodificata e nuovamente visualizzata.
basename(): restituisce, dato un percorso, la componente di questo identificata da un nome di file.
```
<?php>
        $path = "/inettpub/wwwroot/uno.php";
        $base = basename($path);
```

```
        print $base;                // ritorna uno.php
?>
```
bcadd(): somma due numeri, prende come primi due argomenti due numeri e, come terzo argomento opzionale, il numero di cifre da visualizzare dopo la virgola.
```
<?php>
$num = bcadd(1.334, 4.44554, 2);
print $num;                         // ritorna 5.77
?>
```
bccomp(): compara due numeri, prende come argomento due numeri e, opzionalmente, un ulteriore numero che determina il numero di decimali da considerare dopo la virgola per considerare i due numeri uguali; restituisce 0 nel caso i due numeri siano uguali, +1 se il numero di sinistra è maggiore di quello di destra e -1 nel caso opposto.
```
<?php>
$comp = bccomp(0.334, 0.301, 2);
print $comp;                        // ritorna 1
?>
```
Ma se, al posto del 2 avessimo inserito uno oppure non avessimo inserito niente, il risultato sarebbe stato 0.
bcdiv(): divide due numeri, con le stesse modalità descritte per *bcadd()* e *bccomp().*
bcmult(): moltiplica due numeri, ed è possibile aggiungere un ulteriore parametro per limitare il numero di cifre dopo la virgola.
```
<?php>
        $molt = bcmul(2.31, 3.21, 2);
        print $molt;                // ritorna 7.41
?>
```
bcpow(): eleva a potenza due numeri, con la possibilità di specificare il numero di cifre dopo la virgola.
```
<?php>
        $pot = bcpow(2.3, 3, 2);
        print $pot;
?>
```
eleva 2.3 alla terza potenza, approssimando il risultato alla seconda cifra decimale: 12.16.
bcsqrt(): calcola la radice quadrata di un numero, con la possibilità di approssimare il numero di cifre dopo la virgola aggiungendo un secondo elemento alla funzione.
bcsub(): sottrae un numero da un altro, con la possibilità di approssimare le cifre dopo la virgola.
```
<?php>
        $num = bcsub(2, 5);
        print $num;                 // ritorna -3
?>
```
bin2hex(): converte una stringa di dati dal formato binario a formato esadecimale.
ceil(): restituisce il valore intero più alto riferito al numero passato come argomento alla funzione.
```
<?php>
$num = ceil(3.22112);
print $num;                         // ritorna 4
?>
```
chdir(): cambia la directory di lavoro, restituisce *TRUE* se l'operazione ha successo, *FALSE* in caso contrario, ad esempio nel caso la directory non sia leggibile.
```
<?php>
        $dir = "c:\compiler\borlandc";
        chdir($dir);
        print $dir;
?>
```

checkdate(): controlla che una data sia valida; per considerarsi valida, una data deve avere: l'anno compreso fra "0" e "32767"; il mese compreso fra "1" e "12"; il giorno è compreso fra "0" ed il numero relativo al numero di giorni del mese a cui si fa riferimento.
chgrp(): tenta di cambiare il gruppo di un file a gruppo; la funzione accetta come argomenti il nome del file cui si vogliono cambiare i permessi ed il nuovo gruppo di appartenenza, *chgrp(filename, gruppo); n*ei sistemi Windows non funziona ma restituisce sempre vero.
chmod(): è equivalente al comando Unix ed ha la stessa sintassi di *chgrp().*
chmop(): rimuove i whitespaces da una stringa; spesso è utilizzato per eliminare i caratteri \n quando si riceve un argomento dallo standard input; il carattere eliminato può essere letto con:
```
$carattere = chop($string);
echo "$carattere\n";
```
chown(): cambia l'owner di un file, come in Unix. Accetta come argomento il nome del file ed il nome del nuovo proprietario. Nei sistemi Windows, non fa niente e restituisce sempre vero.
```
$file = "prova.txt";
chown($file, $user);
```
chr(): restituisce il carattere ASCII specificato dal rispettivo numero.
```
<?php
        $ascii= "0126";
        $char = chr($ascii);
        print $char;                    // ritorna ~
?>
```
chunk_split(): divide una stringa in parti di n caratteri; il numero è passabile alla funzione dopo la stringa da dividere. Se non impostato, di default è assunto come 76.
```
<?php
        $string = "Questo è un corso per imparare il linguaggio PHP";
        $split = chunk_split($string, 5);
        print $split;
?>
```
restituirà.
```
Quest
o è u
n cor
so pe
r imp
arare
 il l
ingua
ggio
PHP
```
La funzione è utile per l'encoding MIME base64.
closedir(): chiude una directory precedentemente aperta con la funzione *opendir().*
copy(): crea la copia di un file.
```
$file = "prova.txt";
copy($file, "$file.bak");
```
cos(): restituisce il valore del coseno dell'argomento.
count(): conta gli elementi in una variabile.
```
<?php
        $arr[0] = "abc";
        $arr[1] = "def";
        $arr[2] = "ghi";
        $count = count($arr);
```

```
	print $count;
?>
```
restituirà 3, all'interno dell'array $arr sono presenti 3 elementi ($arr[0], $arr[1], $arr[2]).
crypt(): cripta una stringa.
In pratica, si deve passare alla funzione la stringa che dovrà essere criptata e, opzionalmente, il seme con sui criptarla; se questo non è passato alla funzione, sarà generato in maniera random dal PHP stesso. Un esempio di criptazione di una stringa.
```
<?php
	$var = "Questa è una variabile";
	$crypt = crypt($var, "aa");
	print $crypt;		// ritorna aax0a8ap9T9w
?>
```
current(): restituisce il primo elemento di un array:
```
<?php
	$arr[0] = "abc";
	$arr[1] = "def";
	$arr[2] = "ghi";
	$current = current($arr);
	print $current;			// ritorna abc
?>
```
date(): visualizza la data in formato che è possibile definire; la funzione riconosce come validi i seguenti formati.
a: am/pm;
A: AM/PM
d: giorno del mese in due cifre, da "0" a "31";
D: giorno del mese in formato testo, ad esempio "Mon";
F: mese, in formato testuale, ad esempio "March";
h: ora nel formato "01", "12";
H: ora nel formato "00", "23";
g: ora nel formato "1", "12";
G: ora nel formato "0", "23";
i: minuti, nel formato "00", "59";
j: giorno del mese nel formato "1", "31";
l: giorno della settimana, ad esempio "Monday";
L: specifica se l'anno è bisestile o meno ("1" oppure "0");
m: mese nel formato "01", "12";
n: mese nel formato "1", "12";
M: mese in formato testuale corto, ad esempio "Jan";
s: secondi da "00" a "59";
S: suffisso inglese per gli ordinali, "st", "nd", "rd", "th";
t: numero di giorni nel mese corrente, da "28" a "31";
w: giorno della settimana in formato numerico ("0"=domenica);
Y: anno in quattro cifre, ad esempio "2000";
y: anno in due cifre, ad esempio "00";
Ad esempio, *<?php print (date("l d F y H:i:s a")); ?>* stampa la data corrente
debugger_off(): disabilita il debugger PHP.
debugger_on(): abilita il debugger PHP.
decibin(): converte un numero da decimale a binario.
```
<?php
	$bin = decbin(10);
	print $bin;		// ritorna 1010
?>
```
dechex(): converte un numero da decimale a esadecimale.
decoct(): converte un numero da formato decimale a formato ottale.

define(): definisce una costante.

```
<?php
        define("COSTANTE", "Questa è una costante");
        print COSTANTE;
?>
```

defined(): controlla che una certa costante esista.

```
<?php
        define("COSTANTE", "Questa è una costante");
        if (defined("COSTANTE")) {
                print "La costante è definita\n";} else {
                print "La costante non è definita\n";}
?>
```

die(): visualizza un messaggio ed esce dal programma.

```
<?php
        define("COSTANTE", "Questa è una costante");
        if (defined("COSTANTE")) {
                print "è definita\n";}
                else { die (" non è definita; impossibile proseguire\n");}
?>
```

dirname(): quando si specifica un path, riporta il path senza il nome del file finale.

```
<?php
        $path = "c:\inetpub\wwwroot\uno.php";
        print (dirname($path));
?>
```

Fa l'esatto contrario della funzione *basename();* combinando le due funzioni, si può avere il path completo di un file, compreso il suo nome.

diskfreespace(): restituisce lo spazio di disco libero.

```
<?php print (diskfreespace("C:\inetpub")); ?>
```

each(): restituisce il primo valore di un array utilizzando le keys 0, 1, key e value; se l'array è associativo. La funzione *each()* è spesso utilizzata insieme *a list().*

```
<?php
        $array = array ("nome" => "valore", "nome2" => "valore2");
        while (list($key, $val) = each ($array)) {
                print "$key => $val\n";}
?>
```

che restituisce

nome => valore
nome2 => valore2

Se invece l'array non è associativo, il codice sarà.

```
<?php
        $array = array ("nome", "valore", "nome2", "valore2");
        while (list($key, $val) = each ($array)) {
                print "$key => $val\n";}
?>
```

ed il risultato

0 => nome
1 => valore
2 => nome2
3 => valore2

echo(): visualizza una o più stringhe.

ereg_replace(): sostituisce un'espressione regolare con determinati valori; alla funzione devono essere passati tre argomenti: il primo indica il testo da sostituire, il secondo è il testo utilizzato per la sostituzione ed il terzo è la stringa da modificare.

```
$stringa = "Questo è un corso su ASP";
```

echo ereg_replace("ASP", "PHP", $stringa);
ereg(): esegue il matching di un'espressione regolare.
```
<?php
        if (ereg("([0-9]{4})-([0-9]{1,2})-([0-9]{1,2})", $date, $regs)) {
        print "$regs[3].$regs[2].$regs[1]"; } else {
        print "IL formato della data non è valido: $date";}
?>
```
Tutto questo ciclo è fatto per controllare che una data sia in formato corretto. Il significato di "[0-9]{4})-([0-9]{1,2})-([0-9]{1,2}". È un'espressione regolare che significa "un numero da 0 a 9 ripetuto quattro volte seguito da un '-', da un numero da 0 a 9 ripetuto una o due volte, da un '-' e da un numero da 0 a 9 ripetuto una o due volte".
eregi_replace(): funziona esattamente come *ereg_replace(),* solamente che in questo caso l'espressione regolare è sostituita in maniera case insensitive, ossia ignorando i caratteri maiuscoli e minuscoli.
eregi(): funziona esattamente come ereg(), solamente che in questo caso l'espressione regolare è sostituita in maniera case insensitive.
error_log(): invia un messaggio di errore al file di log del web server, direttamente alla porta TCP dalla quale è arrivata la richiesta o in un file.
error_log(message, mesage_type, destination);
Message_type è un numero che specifica dove deve arrivare il messaggio. Può essere:
1. il messaggio è inviato al logger del PHP o nel file specificato da *error_log*;
2. il messaggio è inviato per e-mail al parametro specificato in *destination*;
3. il messaggio è inviato al debugger;
4. il messaggio è inserito in append al parametro specificato in *destination*.

escapeshellcmd(): se si richiama un comando esterno da una shell, con questo comando si fa in modo che i metacaratteri della shell siano fatti precedere da un carattere di escape per evitare che il comando produca degli errori.
exec(): esegue un programma esterno.
exit(): esce da uno script; è utile nei casi si voglia fermare uno script in caso qualcosa non soddisfi determinate condizioni. Non riporta un messaggio di errore come fa *die().*
exp(): eleva $e^{2.71828}$ alla potenza riportata come argomento della funzione.
<?php print exp(3); ?>
explode(): divide una stringa secondo un determinato pattern. Ad esempio, volendo dividere una stringa contenente tre nomi separati da virgole.
```
<?php
        $nomi = "Tizio,Caio,Sempronio";
        list ($nome1, $nome2, $nome3) = explode (",", $nomi);
        print "$nome1\n$nome2\n$nome3\n";
?>
```
che restituirà
Tizio
Caio
Sempronio
È una versione semplificata di *split().* Entrambe le funzioni, inoltre, sono molto utili nel caso ci sia la necessità di leggere determinati file contenenti delle liste.
fclose(): chiude un puntatore ad un file precedentemente aperto con *fopen().*
feof(): testa un puntatore ad un file per vedere se si è alla fine dello stesso.
fgetc(): restituisce il primo carattere del puntatore precedentemente aperto con *fopen();* se ad esempio il puntatore *$file* punta al file \tmp\prova.txt che contiene solamente la riga Ciao, un codice come il seguente:
```
$char = fgetc($file);
echo "$char\n";
```
restituirà C.
file_exists(): controlla se un file esiste, riportando *TRUE* in caso positivo o *FALSE* in caso

negativo.
if (file_exists($file)) { print "$file esiste;}
Può essere utile utilizzare questa funzione nel caso sia necessario agire su uno o più file, in modo da agire sullo stesso solo nel caso questo esista senza rischiare d'incorrere in inspiegabili anomalie dello script.
filegroup(): restituisce il gruppo al quale appartiene il file.
```
<?php
        $filename = "\boot.ini";
        $group = filegroup($filename);
        print "$filename appartiene al gruppo $group";
?>
```
filesize(): restituisce la grandezza di un file.
```
<?php
        $filename = "\boot.ini";
        $size = filesize($filename);
        print "$filename: $size";              // ritorna \boot.ini: 204
?>
```
filetype(): determina il tipo di file; i valori possibili sono: *fifo, char, dir, block, link, file e unknown*.
flock(): applica il locking ad un file; opera su un puntatore ad un file precedentemente aperto e le operazioni possibili sono:
1. per il lock in lettura;
2. per il lock in scrittura;
3. per rimuovere il lock, di qualsiasi tipo sia;
4. per impedire che *flock()* blocchi un file mentre applica il *lock().*

Ad esempio, per applicare *flock()* ad un puntatore *$file* precedentemente definito occorrerà scrivere.
```
flock($file, 2);                    // Per impedire che il file sia letto
.....
// Codice per lavorare sul file
flock($file, 3);                    // Per rimuovere il flock
```
fopen(): apre un file oppure un'URL.
fopen(filename, mode);
Ovviamente a *filename* corrisponde il nome del file o l'URL dello stesso, ed a *mode* la modalità con il quale questo deve essere aperto: si ha qui la possibilità di scegliere fra:
r - apre il file in sola lettura;
r+ apre il file in lettura ed in scrittura;
w apre il file in sola scrittura;
w+ apre il file in lettura e scrittura;
a - apre il file in sola scrittura e inserisce il puntatore alla fine del file (w lo inserisce alla fine);
a+ apre il file in lettura e scrittura inserendo il puntatore alla fine del file.
Ad esempio, per aprire un file locale in sola lettura.
$file = fopen("\tmp\prova.txt", "r");
Per un URL, invece.
$file = fopen("http://www.myhost.com/index.html", r");
Per tutte le successive operazioni sul file, poi, si deve agire direttamente sul puntatore *$file* e non direttamente sul file.
header(): invia un qualsiasi header HTTP.
Se volessimo inviare un header che rediriga il browser ad una determinata locazione.
header("Location: http://www.html.it");
Se invece volessimo inviare un messaggio di pagina inesistente oppure un'istruzione per fare in modo che il browser non utilizzi la cache, potremo scrivere.
header("http:/1.0 404 Not Found");

header("Pragma: no-cache");
hexdec(): restituisce il valore decimale di un numero esadecimale.
implode(): è l'opposto di *explode():* la sintassi è identica, ma in questo caso restituisce una stringa con i valori separati dal primo argomento della funzione.
in_array(): restituisce valore vero se in un array è presente un determinato valore.

```
<?php
        $numeri = array("1", "2", "3", "4", "5");
        $num = 2;
        if (in_array($num, $numeri)) {
                print "$num è presente nell'array \$numeri\n";}
?>
```

is_array(): controlla se una data variabile è un array.

```
<?php
        $numeri = array("1", "2", "3", "4", "5");
        if (is_array($numeri)) { print "\$numeri è un array\n";}
?>
```

La stessa cosa è fatta, con ovviamente la differenza dell'argomento, dalle funzioni.
is_dir(); is_double(); is_executable(); is_file(); is_float(); is_int(); is_integer(); is_link(); is_long(); is_object(); is_readable(); is_real(); is_string(); is_writeable().
isset(): restituisce 1 *TRUE* nel caso la variabile esista, 0 *FALSE* nel caso opposto; ad esempio, per vedere se esiste o meno una variabile.

```
$a = 10;
<?php
        $a = 10;
        print isset($a);
        print isset($b);
?>
```

join(): unisce gli elementi di un array con una stringa; l'uso è identico a quello *implde().*
key(): prende una chiave da un array associativo.

```
<?php
        $array = array("1" => "uno");
        $chiave = key($array);
        print $chiave;                  // ritorna 1
?>
```

Questa funzione è utile per estrarre tutte le chiavi di array associativi complessi.
link(): crea un hard link. Le informazioni sui link sono visualizzate con *linkinfo().*

```
link(target, link);
```

list(): assegna delle variabili come se fossero parti di un array.

```
<?php
        $array = array ("nome" => "valore", "nome2" => "valore2");
        while (list($key, $val) = each ($array)) { print "$key => $val\n"; }
?>
```

In questo caso, *list()* è utilizzato per stilare una lista di variabili che saranno estratte dall'array, senza ovviamente dare loro un valore ma lasciando alle parti successive del codice l'assegnazione dei loro valori. È inoltre utile notare che le variabili create da *list()* non assumono un solo valore, ma per ogni chiamata assumono un diverso valore, secondo gli elementi presenti nell'array.
mail(): funzione per l'invio di e-mail.

```
mail(To, Subject, Message, Altri_headers);
```

Supponendo di voler inviare un e-mail a *nome@host.com* con oggetto *Prova* e volendo specificare il nome del mittente.

```
mail("nome@host.com", "Oggetto", "Questo è il corpo dell'email",
"From: mittente <mittente@host.net>);
```

Ovviamente, nel file di configurazione, si deve specificare la locazione di *sendmail* (o

analogo programma per l'invio delle e-mail).
max(): restituisce il valore più alto di una serie di variabili, il reverso è *min(),* che adotta la stessa sintassi di *max().*

```
<?php
        $num = 1;
        $num2 = 23;
        $num3 = 0.3;
        $max = max($num, $num2, $num3);
        print $max;                    // ritorna 23
?>
```

mkdir(): crea una directory, di cui si deve specificare il percorso ed i permessi.
mkdir("\tmp\prova", 0777);
creerà la directory *\tmp\prova* con permessi impostati a 0777.
opendir(): apre una directory, della quale sarà possibile leggere gli elementi con *readdir()* e, poi, chiuderla con *closedir().*
parse_url(): prende in input la stringa contenente l'URL e restituisce un array associativo in cui ogni elemento corrisponde ad una parte dell'URL. Le chiavi inserite nell'array associativo sono le seguenti: *scheme, host, port, user, pass, path, query, fragment*.

```
<?php
        // URL da scomporre
        $url = "http://www.ubertini.it/sistemi.php";
        $p = parse_url ($url);
        // componenti URL
        print ("Protocollo usato:  ".$p["scheme"]."</br>");
        print ("Nome host        :  ".$p["host"]."</br>");
   print ("Percorso             :  ".$p["path"]."</br>");
?>
```

urlencode() e **urldecode()** permettono la codifica e la decodifica dell'URL, per esempio sostituiscono i caratteri non alfa numerici con un simbolo di % seguito dalle due cifre esadecimali del corrispondente codice ASCII.

```
<?php
        $l = "http://www.ubertini.it/sistemi.php";
        $c = "Reti e Programmazione";
        print ("<a href=\"$l?categoria=");
        print urlencode($c);
        print ("\">Clicca qui</a>");
?>
```

phpinfo(): visualizza informazioni sul PHP stesso.
phpversion(): visualizza la versione di PHP che si sta utilizzando.
popen(): apre un puntatore ad un processo che deve essere chiuso con *pclose().*
print(): visualizza una stringa a video come *echo().*
rand(): genera un valore numerico in maniera casuale.
range(): crea un array contenente un range di valori interi specificato; ad esempio, per creare un array con valori da 1 a 10.
$array = range(1, 10);
rename(): rinomina un file per rinominare oldname come newname.
rename("oldname", "newname");
rmdir(): rimuove una directory; questo può essere fatto solo se la directory è vuota e i permessi sulla directory lo consentono.
round(): arrotonda un numero.

```
$numero = round(2,3); // restituisce 2
$numero = round(2.5); // restituisce 3
$numero = round(2.6); // restituisce 3
```

I decimali da 0 a 4 sono approssimati all'intero precedente, da 5 a 9 all'intero successivo.

shuffle(): ordina in modo casuale gli elementi di un array; ad esempio, per poter visualizzare gli elementi di un array in maniera casuale.

```
<?php
        $num = range(0,10);
        shuffle($num);
        while (list(,$numero) = each($num)) { print "$numero "; }
?>
```

sin(): restituisce il seno dell'espressione.

sizeof(): calcola il numero di elementi presenti in un array.

```
<?php
        $array = array("1", "2", "3", "4", "5");
        $size = sizeof($array);
        if ($size <= 10) {
                        print "L'array contiene meno di 10 elementi\n";
                        } else {
                                print "L'array contiene più di 10 elementi\n";
                        }
?>
```

sleep(): mette lo script in pausa per un determinato numero di secondi, specificato come argomento della funzione.

split(): divide una stringa secondo un determinato pattern.

```
<?php
        $linea = "tizio||caio||sempronio";
        list ($uno, $due, $tre) = split("\|\|", $linea, 3);
        print "1 => $uno\n2 => $due\n3 => $tre\n";
?>
```

è stato necessario inserire un carattere di escape (\) prima di ogni "|" nell'espressione da utilizzare per splittare la riga.

sqrt(): restituisce la radice quadrata dell'argomento.

strcmp(): esegue una comparazione su due stringhe.

```
<?php
        $cmp = strcmp("Ciao", "Ciao a tutti");
        if ($cmp == "0") {
                                print "Le stringhe sono identiche\n";
                          } elseif ($cmp < 0) {
                                        print "La seconda riga è più lunga della prima\n";
                        } elseif ($cmp < 0) {
                                        print "La prima riga è più lunga della prima\n";
                        }
?>
```

restituisce La seconda riga è più lunga della prima. La funzione, infatti, restituisce 0 se le stringhe sono uguali, un valore minore di zero se la seconda è più lunga della prima e maggiore di zero se la prima è più lunga della seconda.

substr(): consente di estrarre parte di una stringa, si deve specificare che il carattere di partenza è quello in posizione *i* e il numero *n* di caratteri da estrarre.

```
<?php
        $s = "Ieri sono andato in bicicletta";
        print "$s\n";
        print "Estrazione dei primi 7 caratteri\n";
        print substr($s,0,7);
?>
```

Utilizzando un valore negativo come numero *n* di caratteri da estrarre, s'inizia a contare da destra verso sinistra.

```
<?php
```

```
	$s = "Ieri sono andato in bicicletta";
	print "$s\n";
	print "Estrazione degli ultimi 7 caratteri\n";
	print substr($s,-7);
?>
```

system(): esegue un programma di sistema, ne restituisce l'output e ritorna allo script.
tan(): restituisce la tangente dell'argomento.
unset(): elimina il valore di una variabile.
usleep(): come *sleep(),* ma questa funziona blocca lo script per n secondi.

Funzioni Apache

apache_lookup_uri(): questa funzione esegue una richiesta per un determinato **URI** (*Uniform Resource Identifier*: un indirizzo web) e riporta i risultati di tale richiesta: i risultati sono contenuti in un array.

```
$array = apache_lookup_uri($url);
```

Con questa funzione si hanno informazioni su:
status (che è poi un codice numerico); *the_request*: il tipo di richiesta, *method* il metodo utilizzato, *filename* il nome del file con il *path_info* path locale ed il protocollo utilizzato; *no_cache; no_local_copy; allowed; sent_bodyct; bytes_sent; byterange; clenght; unparsed_uri; request_time.*

getallheaders(): con questa funzione è possibile richiedere tutti gli headers relativi ad una richiesta: questi headers possono essere letti anche con la funzione *phpinfo().*

```
$array = getallheaders();
```

e, utilizzando un codice

```
$headers = getallheaders();
while (list($header, $value) = each($headers)) {
	print "$header: $value<br>\n";
	}
```

si può ottenere qualcosa di simile a

```
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
Accept-Charset: iso-8859-1,*,utf-8
Accept-Encoding: gzip
Accept-Language: en
Connection: Keep-Alive
Host: localhost
Pragma: no-cache
User-Agent: Mozilla/4.72 [en] (X11; I; Linux 2.2.14 i586)
```

virtual(): è l'equivalente di *<!--# include virtual="/file.txt"-->,* chiamata relativa ai server side includes. Con questa funzione, è possibile includere un qualsiasi file in una pagina generata dinamicamente con il PHP.

```
virtual(file);
```

Il file da specificare può essere qualunque file di testo e, di norma, è riferito alla directory radice dell'host; quindi, se si ha un file visibile via web con *http:/www.dominio.com/file.txt*, la sintassi da utilizzare è

```
virtual("\file.txt");
```

Funzioni di criptazione

PHP offre agli sviluppatori una serie di funzioni relative alla criptatura, legate alla libreria *mcrypt*; tale libreria supporta moltissimi algoritmi di criptazione, alcuni più utilizzati ed altri meno. Gli algoritmi sono: DES, TripleDES, Blowfish, 3-WAY, SAFER-SK64, SAFER-Sk128, TWOFISH, TEA, RC2, GOST, RC6, IDEA.
Per funzionare con tale libreria, il PHP deve essere stato compilato con l'opzione *--with-mcrypt*. I comandi fondamentali sono quattro, tutti con la medesima sintassi:
mcrypt_cfb(): cipher feedback; codifica byte per byte;

mcrypt_cbc(): cipher block chaining: utile per l'encoding dei file.
mcrypt_ecb(): electronic codebook: utilizzata per dati random.
mcrypt_ofb(): output feedback: simile a *cfb*, ma è data maggiore attenzione agli errori.
$encrypted = mcrypt_XXX(algoritmo, chiave, input, encode/decode)
dove:
XXX è il metodo che s'intende utilizzare (*cfb, cbc, cfb o ofb*);
algoritmo è l'algoritmo che s'intende utilizzare, con la sintassi *MCRYPT_ALGORITMO*
Ad esempio, si potrebbe utilizzare *MCRYPT_BLOWFISH* oppure *MCRYPT IDEA*
chiave altro non è che la chiave con cui si criptano i dati;
input sono i dati da criptare;
encode/decode indica alla funzione se si devono criptare o decriptare i dati; per questo, si usano rispettivamente *MCRYPT_ENCRYPT* e *MCRYPT_DECRYPT*
Volendo criptare una semplice stringa di testo con chiave di criptatura *La mia chiave* utilizzando CFB con l'algoritmo IDEA.

```
$stringa = "Una semplice stringa di testo";
$chiave = "La mia chiave";
$encrypted = mcrypt_cfb(MCRYPT_IDEA, $chiave, $stringa, MCRYPT_ENCRYPT);
```

Per leggere i dati criptati *($encrypted*) si deve ovviamente conoscere la chiave, il metodo e l'algoritmo utilizzati; quindi si deve scrivere:

```
$chiave = "La mia chiave";
$stringa = mcrypt_cfb(MCRYPT_IDEA, $chiave, $encrypted, MCRYPT_DECRYPT);
```

La variabile *$stringa*, quindi, conterrà *Una semplice stringa di testo*.

Funzioni FTP
ftp_connect():stabilisce una connessione FTP fra il PC ed il server FTP remoto.
$stream = ftp_connect(host, port);
dove *host* è il nome del server a cui ci si connettere e *port* (opzionale) è la porta alternativa alla quale ci si vuole connettere; se questa non è specificata, è utilizzata la porta di default per il protocollo FTP, ossia la 21. Nella variabile *$stream*, inoltre, è immagazzinato lo stream di dati che il client (in questo caso il PHP) riceve dal server, ossia i messaggi di connessione accettata (con i vari dettagli) o di connessione rifiutata. Ad esempio, per connetterci alla porta di default del server FTP *ftp://ftp.host.com* si usa.
$stream = ftp_connect("ftp://ftp.host.com");
ftp_login(): dopo la connessione, bisogna identificarsi in modo che il server permetta lo scambio dei dati.
$login = ftp_login(stream, username, password);
Se ad esempio in precedenza si era collegati all'host ftp.host.com, utilizzando la variabile *$stream*, adesso è possibile procedere al login vero e proprio con
$login = ftp_login($stream, "utente", "password");
La variabile *$login* servirà per capire se il login è andato o meno a buon fine e conterrà il valore 1 per il successo, 0 altrimenti. Per vedere se continuare lo scambio di dati in seguito all'autorizzazione si può utilizzare il valore assegnato a tale variabile e scrivere.

```
if ($login == "1") {
        ...// Fai il resto delle operazioni
    } else {
        echo "Autorizzazione non riuscita\n";
    }
```

Una volta connessi, si può sapere su che server si sta lavorando con la funzione
$system = ftp_systype($stream);
ftp_pwd():invoca il comando **PWD** (*Print Work Directory*) per vedere a che directory si è connessi dopo il login.
$directory = ftp_pwd($stream);
dove *$stream* è sempre la variabile che utilizzata per la connessione con *ftp_connect().*
ftp_cdup e **ftp_chdir()**:servono rispettivamente a muoversi alla directory superiore e a

muoversi in una determinata directory all'interno del server.
$var = ftp_cdup($stream);
$newdir = ftp_chdir($stream, "nuova_directory");
Se ad esempio al login si è nella directory / e si vuole anadre in */var/wwwdata* si può scrivere.
$newdir = ftp_chdir($stream, "/var/wwwdata");
ftp_mkdir e ftp_rmdir(): queste due funzioni invocano il comando *mkdir* e *rmdir*. La prima restituisce il nome della nuova directory, la seconda solamente i valori true o false.
$mydir = ftp_chdir($stream, "/var/wwwdata/");
// Posizionarsi in /var/wwwdata.
$newdir = ftp_mkdir($stream, "prova")
// Creiare la directory prova come sottodirectory di /var/wwwdata
$deleted_dir = ftp_rmdir($stream, $newdir);
// Cancellare la directory appena creata
// Controllare il tutto con:
if ($deleted_dir == "1") {
print "Operazione completata con successo.\n";
} else {
print "Qualcosa non è andato per il verso giusto.\n";
}
ftp_nlist():analoga al comando *dir*.
$list = ftp_nlist($stream, directory);
Ad esempio, posizionarsi nella directory */var/wwwdata* e leggerne i file con
$newdir = "/var/wwwdata";
$list = ftp_nlist($stream, $newdir);
I risultati sono contenuti in un array.
ftp_get():richiama il comando *GET*, per scaricare un file dal server remoto. Si deve specificare per la funzione, oltre allo stream, il nome del file locale, il nome del file remoto e la modalità di trasferimento *FTP_ASCII* o *FTP_BINARY*.
$file = ftp_get($stream, local_filename, remote_filename, mode);
Ad esempio, volendo scaricare dal server il file *data.txt* inserendolo nella directory */tmp* con nome *file.txt* in ASCII mode.
$file = ftp_get($stream, "/tmp/file.txt", "data.txt", FTP_ASCII);
Per vedere se l'operazione ha avuto o meno successo, si può operare in due modi: controllare se il file c'è nel disco oppure controllare il valore della variabile *$file*: se ha valore 1 allora l'operazione è stata completata, se ha valore 0 nessun file sarà stato scaricato sul disco.
ftp_put():fa esattamente il contrario di *ftp_put(),* ossia carica un file sul server.
$file = ftp_put($stream, remote_filename, local_filename, mode);
Caricare il file locale */tmp/file.txt* nella directory remota con il nome *data.txt*. Tutto in ASCII mode, ovviamente.
$file = ftp_put ($stream, "data.txt", "/tmp/file.txt", FTP_ASCII);
Anche qui, si può controllare in due modi: valutando il valore di *$file* oppure invocando la funzione *ftp_nlist()* per vedere se fra i file c'è anche *data.txt*.
ftp_size(): restituisce le dimensioni di un dato file.
$size = ftp_size($stream, remote_filename);
Per conoscere la dimensione del file *data.txt*.
$size = ftp_size($stream, "data.txt");
la variabile *$size* contiene le dimensioni del file *data.txt*.
ftp_mdtm(): restituisce la data di ultima modifica di un file, restituendola come Unix *timestamp*.
$date = ftp_mdtm($stream, remote_filename);
Volendo sapere la data di ultima modifica del file data.txt possiamo scrivere.
$date = ftp_mdtm($stream, "data.txt");

la variabile *$data* conterrà la data di ultima modifica del file oppure il valore -1 in caso di insuccesso.
ftp_rename() e **ftp_delete()**:servono per rinominare un file e per cancellarlo.
$name = ftp_rename($stream, oldname, newname);
dove *oldname* è il nome originario del file e *newname* è il nuovo nome da assegnare.
Ad esempio, per rinominare il file *data.txt* in *dati.dat* si può scrivere.
$name = ftp_rename($stream, "data.txt", "dati.dat");
La variabile *$name* conterrà 1 se l'operazione ha avuto successo, 0 altrimenti.
ftp_delete(), invece, si utilizza con sintassi.
$delete = ftp_delete($stream, file);
per eliminare il file dati.dat presente nella directory si può scrivere.
$delete = ftp_delete ($stream, "dati.dat");
la variabile può contenere valore 1 (il file è stato eliminato) o 0 (errore).
ftp_quit(): il lavoro sul server è terminato e ci si può disconnetters.
$quit = ftp_quit($stream);
È sempre consigliato invocare questa funzione invece di chiudere il programma in esecuzione.

Funzioni di networking
gethostbyaddr() e **gethostbyname()**: queste due funzioni permettono, rispettivamente, di ricavare il nome di un host partendo dal suo indirizzo IP e di ricavare l'indirizzo IP associato ad un nome di dominio.
$hostname = gethostbyaddr(IP_address);
$hostaddress = gethostbyname(hostname);
Ad esempio, *www.server.com* ha indirizzo IP 123.456.78.9: ma si vuole che sia uno script PHP a dirlo.
$hostname = gethostbyaddr("123.456.78.9");
$hostaddress = gethostbyname("http://www.host.com");
print "$hostname ha indirizzo IP $hostaddress.\n";
getservbyname():permette di ricavare il numero della porta alla quale è associato un determinato servizio.
$porta = getservbyname(service, protocol);
Ad esempio, volendo sapere a che porta è associato il protocollo FTP, si può scrivere.
$ftp_port = getservbyname("ftp", "tcp");
che con grande probabilità restituirà la porta 21 (quella di default per il FTP).

Standard POSIX
Il PHP mette a disposizione molte funzioni POSIX-compliant che permettono d'innestare una completa interazione con il sistema, per un'ottimale scrittura degli script.
posix_kill(): invia un segnale ad un determinato PID, restituendo *FALSE* in caso il PID non sia associato ad alcun processo, vero nel caso opposto.
posix_kill(PID, SIG);
dove PID è il pid del processo cui si vuole inviare il segnale e *SIG* è, appunto, il segnale.
Ad esempio, volendo inviare un segnale di *kill* a Netscape (PID 1234), si può scrivere.
$pid = "1234";
$signal = posix_kill($pid, KILL);
posix_getpid(): utilizzare la funzione sopra descritta lavorando da uno script è assai complicato per un semplice motivo: non avendo accesso alla shell, non si può sapere il PID associato ad una determinata applicazione. Ma il PHP aiuta anche in questo caso con questa funzione che permette d'inserire in una variabile il PID di un processo.
$pid = posix_getpid($application);
dove *application* è ovviamente l'applicazione delle quale interessa il PID. Utilizzando questa funzione con la precedente, si può scrivere qualcosa del tipo.
$pid = posix_getpid("netscape");

$signal = posix_kill($pid, KILL);
posix_getppid(): restituisce il padre del processo in uso.
$parent = posix_getppid();
print $parent;
si visualizza a video il padre del processo che si è appena lanciato: nel nostro caso, visto che si è lanciato il comando da una shell, il PID sarà proprio legato alla shell.
posix_getuid(), posix_geteuid(), posix_getgid(), posix_getegid(): sono simili e restituiscono rispettivamente:
Real user ID del processo;
Effective user ID del processo;
Real GID del processo;
Effective GID del processo.
$UID = posix_getuid();
print "UID = $UID\n";
$EUID = posix_geteuid();
print "EUID = $EUID\n";
$GID = posix_getgid();
print "GID = $GID\n";
$EGID = posix_getegid();
print "EGID = $EGID\n";
posix_getlogin():permette di recuperare il nome di login dell'utente che esegue lo script.
$login = posix_getlogin("/usr/sbin/apache");
print $login;
per fare in modo di aver visualizzato a video il nome dell'utente che esegue lo script.
posix_uname():è simile a *uname* dei sistemi Posix-compliant e restituisce informazioni sul sistema. Tali informazioni sono organizzate in un array che contiene le keys: *sysname, nodename, release, version, machine*. Ad esempio, lanciando uno script del tipo
$uname = posix_uname();
while (list($key, $value) = each ($uname)) { print "$key -> $value\n"; }
si otterrà come risultato
sysname -> Linux
nodename -> scatolinux
release -> 2.2.14
version -> #09 mar lug 7 00:06:07 CEST 2005
machine -> i586
posix_getpwname():permette di avere determinate informazioni su un utente del sistema partendo dal suo ID; allo stesso modo, la funzione *posix_getpwuid()* riporta le stesse informazioni partendo dal suo ID.
$user = posix_getpwnam("edo");
while (list($info, $value) = each($user)) { echo "$info -- $value\n"; }
che restituisce
name -- nome_utente
passwd – x
uid – NUM
gid – NUM
gecos -- ,,,
dir -- /home/nome_utente
dove *nome_utente* è il nome dell'utente e */home/nome_utente* è la sua home directory, NUM sono i due attributi numerici associati all'UID ed al GID e *gecos* sono informazioni addizionali sull'utente.
posix_getrlimit():restituisce un array contenente i limiti delle risorse del sistema.
$resources = posix_getrlimit();
while (list($key, $value) = each ($resources)) { echo "$key -> $value\n";}
che restituirà un output del tipo

soft core -> 0
hard core -> unlimited
soft data -> unlimited
hard data -> unlimited
soft stack - > 8388608
hard stack -> unlimited
soft totalmem -> unlimited
hard totalmem -> unlimited
soft rss -> unlimited
hard rss -> unlimited
soft maxproc -> 256
hard maxproc -> 256
soft memlock -> unlimited
hard memlock -> unlimited
soft cpu -> unlimited
hard cpu -> unlimited
soft filesize -> unlimited
hard filesize -> unlimited
soft openfiles -> 1024
hard openfiles -> 1024

Estensioni

La differenza fra le funzioni di libreria e le estensioni è sostanziale: le prime sono presenti direttamente all'interno del motore (built-in), le seconde sono presenti in librerie aggiuntive che devono essere installate sul sistema e richiamate in maniera particolare.
Meccanismo di caricamento dinamico di queste librerie: aprendo il file *php.ini* si vede un paragrafo dedicato alle *extension*, ossia estensioni nel linguaggio stesso: sono un insieme di librerie che sono richiamate al momento dell'esecuzione di uno script come avviene, in maniera analoga, per il caricamento dei moduli con un server web. La sintassi per il caricamento di queste estensioni di linguaggio è molto semplice: una volta che si sono installate le librerie sul sistema, non resta che aggiungere nel file *php.ini* la riga:
extension=libreria
È qui necessario un discorso mirato per i sistemi Unix ed i sistemi Windows: per entrambi la sintassi è identica, ma ovviamente il nome delle librerie e la loro estensione no.
Nei sistemi Windows, una libreria si riconosce dall'estensione *.DLL,* mentre per Unix questa è *.SO*: quindi, secondo il sistema, si deve utilizzare il corretto nome per la libreria e, soprattutto, la corretta estensione. Nello scrivere il nome della libreria che interessa caricare, non ci si deve soffermare sul percorso completo, ma è necessario solamente il nome della stessa, ad esempio *pgsql.so* per i database Postgres. Questo perché, nello stesso file, è presente un'altra linea di configurazione che definisce in quale directory sono presenti queste librerie: leggendo il file *php.ini* si può trovare la riga *extension_dir = directory* che istruirà il motore sulla locazione standard delle librerie. Quindi, quando si specifica con *extension* una libreria che si vuole sia caricata per l'esecuzione di uno script, sono di fatto uniti *extension_dir* e *extension*: se ad esempio *extension_dir* è */usr/lib/php* e si è impostato *extension=pgsql.so*, il PHP saprà che per caricare la libreria *pgsql.so* dovrà cercarla in */usr/lib/php3/pgsql.so*. Inoltre, con *extension= ....* è possibile specificare non solo una libreria ma tutta la serie di librerie che possono servire.
extension=pgsql.so
extension=mysql.so
extension=gd.so
extension=imap.so
extension=ldap.do
extension=xml.so
Come si nota dalla seconda riga, poi, è possibile specificare anche due o più librerie per

uno stesso campo in questo caso, ci sono due librerie per due database (Postgres e MySQL) che, oltre a non entrare in conflitto l'una con l'altra, potrebbero teoricamente anche essere utilizzate contemporaneamente. Nel caso si cerchi di richiamare una funzione non built-in all'interno di uno script, lo script visualizza una messaggio d'errore che avverte che si sta cercando di utilizzare una funzione non riconosciuta.

DATE

Le date sono rappresentate sotto forma di timestamp: è un numero intero che corrisponde al numero di secondi trascorsi dalla Unix epoach. Ad esempio, le ore 00:00:00 del primo gennaio 2001 corrispondono al timestamp 978303600.
Per conoscere il timestamp corrispondente ad una certa data basta invocare *mktime()* passando i parametri: ore, minuti, secondi, mese, giorno, anno. L'ultimo parametro, opzionale, impostato a zero ignora l'ora legale.
Timestamp delle ore 00:00:00 del primo gennaio 2009.

```
<?php echo mktime(0, 0, 0, 1, 1, 2009); ?>          // ritorna 1230764400
```

Per una forma più leggibile si usa *date(),*vuole due argomenti: il primo, obbligatorio, è una stringa che rappresenta il formato in cui codificare la data; il secondo, facoltativo, è il timestamp da formattare, se è omesso è considerato il timestamp corrente.
Questa istruzione stampa la data corrente nel formato gg/mm/aaaa

```
<?php print "Data di oggi " . date("d/m/Y"); ?>
```

Come prima ma senza lo zero prima di giorni e/o mesi di una sola cifra

```
<?php
        print "Data di oggi " . date("j/n/Y");
        print "\nSono trascorsi ".date("z")." giorni dall'inizio dell'anno.";
?>
```

Rappresentazione della data corrente in modo testuale, con i nomi dei giorni della settimana.

```
<?php
        $giorni = array( "Domenica", "Lunedi", "Martedi", "Mercoledi", "Giovedi", "Venerdi",
        "Sabato" );
        print "Oggi e': " . $giorni[date("w")];
?>
```

La *getdate()* restituisce informazioni sulla data e ora correnti in un array associativo.
Accertarsi che si tratti di una data valida; per esempio, sono date non valide il 31 aprile o il 29 febbraio di un anno non bisestile. La funzione usata è *checkdate().*
Verificare una data 31 aprile 2009.

```
<?php
        $giorno = 31;
        $mese = 4;
        $anno = 2009;
        print "La data $giorno/$mese/$anno ";
        if (checkdate($mese,$giorno,$anno)) print "è corretta.";
        else print "non è valida!";
?>
```

Confrontare due date per verificare se la prima è antecedente alla seconda: basta convertire il tutto in timestamp.
Prima data 7 luglio 2007, seconda data 7 luglio 2008

```
<?php
        $data1 = mktime(0, 0, 0, 7, 7, 2007, 0);
        $data2 = mktime(0, 0, 0, 7, 7, 2008, 0);
        print "La prima data è ";
        if ($data1 < $data2) print "precedente";
        else print "successiva";
        print " alla seconda.";
```

```
?>
```
Calcolare il numero di giorni che separa due date arbitrarie.
Prima data 7 luglio 2007, seconda data 7 luglio 2008
```
<?php
        $data1 = mktime(0, 0, 0, 7, 7, 2007, 0);
        $data2 = mktime(0, 0, 0, 7, 7, 2008, 0);
        print "Tra le due date ci sono ";
        print ($data2 - $data1)/(60*60*24);
        print " giorni.";
?>
<?php
        $id=mktime(0, 0, 0, 7, 7, 2007, 0);
        $fd=mktime(0, 0, 0, 7, 7, 2008, 0);
        $a=date('y',$fd)- date('y',$id);
        $g=date('z',$fd)- date('z',$id)+(((int)date('l',$id))?($a*366):($a*365));
        print "Anni fra due date: $a\n";
        print "Giorni fra due date: $g";
?>
```
La differenza tra i due timestamp, espressa in secondi, può essere convertita in giorni dividendo per 24 ore al giorno, 60 minuti all'ora, 60 secondi al minuto.

FILE

La lettura di un file è effettuata in due passi.

1. *fopen()* dato il path del file da aprire e la modalità di apertura restituisce un file pointer.
2. *fgets()* il file pointer è usato da altre funzioni, nell'esempio per la lettura del contenuto del file una riga alla volta.

mode **Descrizione**	
'r'	Apre in sola lettura; posiziona il puntatore all'inizio del file.
'r+'	Apre in lettura e scrittura; posiziona il puntatore all'inizio del file.
'w'	Apre il file in sola scrittura; posiziona il puntatore all'inizio del file e tronca il file alla lunghezza zero. Se il file non esiste, tenta di crearlo.
'w+'	Apre in lettura e scrittura; posiziona il puntatore all'inizio del file e tronca il file alla lunghezza zero. Se il file non esiste, tenta di crearlo.
'a'	Apre in sola scrittura; posiziona il puntatore alla fine del file. Se il file non esiste, tenta di crearlo.
'a+'	Apre in lettura e scrittura; posiziona il puntatore alla fine del file. Se il file non esiste, tenta di crearlo.
'x'	Crea ed apre il file in sola scrittura; posiziona il puntatore all'inizio del file. Se il file esiste già la chiamata a *fopen()* fallirà restituendo *FALSE* e sarà generato un errore di livello *E_WARNING*. Se il file non esiste si tenterà di crearlo. Questo equivale a specificare i flag *O_EXCL*|*O_CREAT* nella sottostante chiamata a *open()*.
'x+'	Crea ed apre il file in lettura e scrittura; posiziona il puntatore all'inizio del file. Se il file esiste già la chiamata a *fopen()* fallirà restituendo *FALSE* e sarà generato un errore di livello *E_WARNING*. Se il file non esiste si tenterà di crearlo. Questo equivale a specificare i flag *O_EXCL*|*O_CREAT* nella sottostante chiamata a *open().*

Esempio, aprire il file *dati.txt* in lettura.
```
<?php
        $n = "dati.txt";
        // apertura in modalità solo lettura
        $fp = fopen ($n, "r") or die ("Impossibile aprire il file");
        while ($r = fgets($fp,4096)){ print ("$r</br>");  }
```

```
        // chiusura del file
        fclose ($fp);
?>
```
Esempio, salvare il file *dati.txt*.
```
<?php
        // testo da salvare nel file
        $t = "Tutte le operazioni di scrittura,modifica o cancellazione.";
        $n = "dati.txt";
        // apertura in modalità scrittura
        $fp = fopen ($n, "w") or die ("Impossibile aprire il file");
        // scrittura
        fputs ($fp,$t) or die ("impossibile salvare il file");
        // chiusura del file
        fclose ($fp);
?>
```
Esempio, elencare il contenuto di una directory.
```
<?php
        // directory corrente
        $dir = ".";
        $d = opendir($dir);
        print ("File contenuti nella directory <b>$dir</b>:<br>");
        while ($f = readdir($d)){ print ("$f</br>");  }
?>
```

FORM HTML

Un form o modulo HTML è inserito in una pagina web mediante il tag *<form>* è costituito da un certo numero di oggetti quali caselle di testo, menu, caselle di spunta. Tali oggetti consentono al visitatore della pagina d'inserire delle informazioni esattamente come nel caso della compilazione di un questionario cartaceo. Una volta inseriti i dati richiesti questi sono inviati, tramite il tasto *Submit*, ad uno script che li elabora. La URL dell'applicazione è specificata con l'attributo *action* del tag form.
Digitare il seguente file: *inviarichiestaphp.htm*
```
<html>
<head>
<title>Modulo di richiesta</title>
</head>
<body>
<form method="get" action="richiesta.php">
<table>
<tr><td>Cognome</td>
<td><input type=text name="cognome"></td></tr>
<tr><td>Nome</td>
<td><input type=text name="nome"></td></tr>
</table>
</br>
<input type="submit" value ="Invia">
</body>
</html>
```
L'attributo *method* specifica il metodo da utilizzare per l'invio delle informazioni e può essere di due tipi.

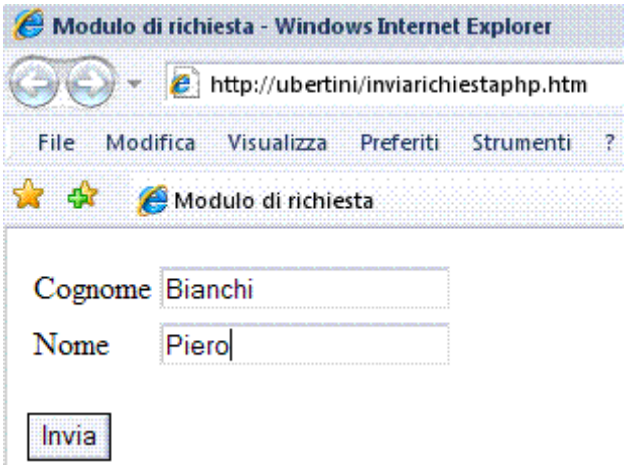
1. *GET*, le informazioni sono incluse nell'indirizzo URL e pertanto sono visibili nella barra degli indirizzi del browser, ma soprattutto sono vincolate dalla lunghezza massima di una URL, 256 caratteri.
2. *POST*, le informazioni sono scritte sullo standard input dall'applicazione destinataria,

non sono visibili nella barra degli indirizzi del browser e non ci sono limiti sulla quantità di dati inviata.

I due metodi sono equivalenti: l'interprete PHP analizza sia le informazioni passate tramite URL (metodo *GET*), sia quelle ricevute sullo standard input (metodo *POST*) e le rende immediatamente disponibili nello script di destinazione sotto forma di variabili globali. PHP è in grado di gestire anche parametri multipli quando si utilizza in un form il tag *<select>* con l'attributo *multiple* impostato.

In alternativa, si può accedere alle informazioni inviate con i metodi *GET* e *POST* utilizzando gli array associativi *$HTTP_GET_VARS* e *$HTTP_POST_VARS*.

Riprendiamo il form dell'esempio precedente, scrivere nelle caselle.



Premere il pulsante *Invia*. Il browser si sposterà alla pagina *richiesta.php.*

```
<html>
<head>
<title>Visualizza i parametri passati con la richiesta</title>
</head>
<body>
<h1>Parametri della richiesta</h1>
<?php
        print ("Nome:    ".$_GET['nome']."</br>");
        print ("Cognome: ".$_GET['cognome']);
?>
</body>
</html>
```

Poiché il form usa il metodo *GET* per trasferire i propri dati, questi sono codificati nell'indirizzo dello script cui devono essere inviati.

A questo punto è utile capire il modo in cui le informazioni provenienti da un form HTML

sono codificate per essere trasmesse con il metodo GET. Partire dall'indirizzo URL dello script cui si vuole inviare i dati; ad esso aggiungere a destra un punto interrogativo che separerà la URL vera e propria a sinistra dalla query string che si va a costruire. La struttura della query string consiste in una serie di coppie *nome/valore* separate da una *&*; i caratteri non ammissibili in un indirizzo URL, ad esempio gli spazi, sono sostituiti dal simbolo *%* seguito dal corrispondente codice ASCII in esadecimale.

Conversioni

Uno script PHP produce un risultato che consiste in una pagina web, in pratica l'output inviato al browser è codice HTML. Esiste una funzione che trasforma una stringa PHP nell'appropriata rappresentazione HTML.

Nell'esempio scrivere una frase nella casella di testo del form e premere il pulsante *Invia*.

```
<html>
<head>
<title>Convertitore interattivo in HTML</title>
</head>
<body>
<form method="get" action="converti.php">
<td><input type=text name="testo"></td></tr>
</br>
<input type="submit" value ="Invia">
</body>
</html>
```

Scrivere il file *converti.php*.

```
<?php
        print "hai inserito il testo?  ";
        print htmlentities($_GET['testo'])."</br>";
        print "che in HTML si scrive in questo modo:  ";
        print htmlentities(htmlentities($_GET['testo']));
?>
```

INCLUSIONE DI FILE ESTERNI

Possono essere sia dei semplici frammenti di codice HTML sia script PHP: nel primo caso si ha una situazione simile all'uso delle **SSI** (*Server Side Includes*); nel secondo, invece, si è in grado di condividere codice PHP tra più script.

È importante osservare che nel momento in cui PHP inizia ad analizzare il file incluso l'interprete si pone in modalità HTML, quindi eventuali frammenti di codice PHP, per essere correttamente riconosciuti ed eseguiti, dovranno essere racchiusi dai consueti demarcatori. Terminata l'analisi del file esterno si torna in modalità PHP e si continua l'elaborazione dello script principale.

Realizzare un sito in cui tutte le pagine devono contenere la barra di navigazione con i link alle varie sezioni del sito stesso. Per prima cosa si isola il frammento di HTML corrispondente alla barra di navigazione e lo si salva in un file *uno.htm*.

```
<!-- INIZIO -->
<body color="black" bgcolor="white">
<a href="uno.php">Home page</a> |
<hr size="1">
<!-- FINE -->
```

Si noti che c'è anche il tag <*body*>, in questo modo, tutte le pagine utilizzeranno lo stesso insieme di colori per il testo, lo sfondo.

Realizzare adesso le varie pagine del sito. Ogni pagina sarà uno script PHP. Una volta completata una pagina s'inserirà il codice PHP necessario per includere il file con la barra di navigazione. Scrivere il file *uno.php*.

```
<html>
```

```
<head>
<title>Pagina che include un file esterno</title>
</head>
<?php
        // Questa istruzione include ed esegue il file
        // libreria.php contenuto nella directory corrente
        require "libreria.php";
        // Usando include si scrive include "libreria.php";
?>
(...Contenuto della pagina...)
</body>
</html>
```

Nella pagina *uno.php* è stato rimosso il tag <*body*> in quanto già presente nella barra di navigazione inclusa.
In tutte le pagine PHP costruite in questo modo, l'istruzione
<?php require "libreria.php";?>
sarà rimpiazzata dinamicamente dal contenuto del file *libreria.php*. Qualunque modifica apportata a tale file sarà immediatamente riflessa in tutte le pagine del sito che la includono rendendone semplice la manutenzione.
Scrivere il file *libreria.php*.
<?php print "Questa è la libreria"; ?>
L'istruzione *include()* include e valuta il file specificato. La documentazione seguente si applica anche a *require().* I due costrutti sono identici in ogni aspetto eccetto per come essi trattano gli errori. *include()* produce un warning mentre *require()* restituisce un *Fatal Error*. In altre parole, usate *require()* se si vuole che un file mancante fermi l'esecuzione della pagina. *include()* non si comporta in questo modo, lo script continuerà nonostante tutto. Assicurarsi di avere un appropriato *include_path* impostato a dovere. Quando un file è incluso, il codice che esso contiene eredita lo scope delle variabili della riga in cui si verifica l'inclusione. Qualsiasi variabile disponibile in quella riga nella chiamata al file sarà disponibile all'interno del file chiamato, da quel punto in avanti.
Per esempio, questi quattro parametri sono forniti dall'amministratore del sito web per poter lavorare con il DB MySQL.
Creare una pagina di nome *config.inc.php* con queste righe.

```
<?php
        $db_host = "labs.fauser.edu";
        $db_user = "a2002bx";
        $db_pass = "a2002bx";
        $db_name = "db2002bx"
?>
```

config: indica che il file contiene dei dati relativi alla configurazione dello script.
.inc: indica che non è una pagina che sarà visualizzata direttamente, ma sarà inclusa all'interno di altre.
.php: per motivi di sicurezza, infatti se qualcuno cercherà di visualizzare questa pagina con il browser, vedrà solo una pagina vuota.
Per richiamare il file basta scrivere.
<?php include ("config.inc.php"); ?>

COOKIES

I cookies (biscotti) sono un meccanismo mediante il quale le applicazioni lato server possono memorizzare e recuperare informazioni sul lato client della connessione (browser). Utilizzando i cookie è possibile mantenere delle informazioni sul client in modo persistente e quindi condividerle tra gli script.
Tutte le operazioni di scrittura, modifica o cancellazione di cookies avvengono mediante una funzione *setcookie()* invocata prima del tag <*HTML*> perché la gestione è a livello di

intestazione HTTP per cui se si accede al cookie nella parte seguente del codice, esso risulta nullo perché il server inserisce il cookie nella sua risposta al client e di conseguenza soltanto alla successiva richiesta.
setcookie(Nome, Valore, Espirazione, Percorso, Dominio, Secure);
Gli argomenti che si possono passare alla funzione sono sei, ma solo i primi due argomenti obbligatori: il nome da assegnare al cookie ed il suo valore.

1. *Nome* è il nome del cookie, che può essere arbitrariamente scelto.
2. *Valore* è il valore del cookie, anch'esso arbitrario, da assegnare al cookie.
3. *Espirazione* è la data di scadenza del cookie, decorsa la quale scompare; se la scadenza non è indicata, scompare alla chiusura del browser.
4. *Percorso* è la directory, a partire dal dominio per la quale il cookie è valido.
5. *Dominio* è il dominio per il quale il cookie è valido.
6. *Secure* è un valore che imposta se il cookie debba essere inviato tramite una connessione HTTPS.

Inviare un cookie chiamato *test*, con valore *prova per il cookie test*, con espirazione di un minuto dal momento dell'invio, per la directory */nomeutente* del dominio *http://www.ubertini.it* senza utilizzare una connessione HTTPS: i parametri da passare a setcookie sono:
setcookie("test", "prova per il cookie test", time()+60, "/nomeutente", ".ubertini.it", 0);
Le cose da esaminare sono due: il tempo di espirazione e la connessione HTTPS. Per la seconda, s'imposta con 0 perché la connessione deve essere una normale connessione HTTP; se si volesse utilizzare il protocollo sicuro HTTPS, si deve inserire il valore 1. Il tempo di espirazione non può essere impostato come *tre minuti*, poiché la cosa non ha tempo: dobbiamo invece impostare il momento di espirazione a partire dal momento in cui il cookie è inviato all'utente; per questo, si utilizza la funzione *time()* alla quale si aggiunge il numero di secondi dopo i quali il cookie deve scadere. Ad esempio, se il cookie è inviato alle 13:20:00 e si vuole che esso espiri dopo 30 minuti (60x30=1800 secondi) si può scrivere come espressione di expire: *time()+1800*. In questo modo, *time()* riporta le 13:20:00 e *time()+1800* sarà 13:50:00.
La cancellazione di un cookie, infine, può avvenire in due modi: assegnandogli un valore nullo o impostando la scadenza ad una data passata.
Leggere da uno script PHP le informazioni memorizzate nei cookies: l'interprete PHP analizza automaticamente i cookies inviati dal browser e li rende disponibili ad altrettante variabili globali e all'array associativo *$HTTP_COOKIE_VARS* contenente tutte le coppie *nome/valore* relative ai cookies impostati.
Esempio di navigazione.

```
<?php
        $i = 0;
        if (isset ($HTTP_COOKIE_VARS) )
        { while (list ($nome, $valore) = each ( $HTTP_COOKIE_VARS ) )
                        { print ("Nome del cookie: $nome = Valore:  $valore\n"); i++;  }
        }
        Print ("Ho contato $i cookies impostati");
?>
```

salvacookie.php

```
<?php
        $nome="Colore";
        $valore="Giallo";
        setcookie($nome,$valore);
        $nome="Fiore";
        $valore="Primula";
        setcookie($nome,$valore);
?>
<html>
```

```
<head>
<title>Memorizzazione di cookie</title>
</head>
<body>
<h1>Cookie: memorizzazione</h1>
</body>
</html>
```

vedicookie.php

```
<html>
<head>
<title>Visualizzazione di cookie</title>
</head>
<body>
<h1>Cookie: visualizzazione</h1>
<?php
        // Una volta impostato un cookie è accessibile come una variabile globale
        print ("Colore:  ".$_COOKIE['Colore']."</br>");
        print ("Fiore:   ".$_COOKIE['Fiore']);
?>
</body>
</html>
```

SESSIONI

Il meccanismo delle sessioni è un espediente pensato per ovviare al problema della mancanza di stato del protocollo HTTP. Grazie al supporto per le sessioni è possibile associare uno stato ad ogni client, in pratica ad ogni visitatore del sito web e mantenerlo per più accessi.
Il mantenimento di una sessione richiede la collaborazione di client (il browser) e server. Il server si occupa della serializzazione della sessione, in altre parole della codifica delle informazioni di sessione in una forma adatta alla memorizzazione su file o database e del

suo ripristino alla successiva richiesta dello stesso client. Tutte le informazioni risiedono sul server.
Il client, da parte sua, deve fornire al server le informazioni necessarie ad associare alla richiesta la sessione corrispondente, questo è reso possibile da un identificativo di sessione **SID** (*Session IDentifier*) univoco che è generato al momento di avvio della sessione. Tecniche di propagazione del SID sono due.

Tramite cookie
È la più semplice, grazie ad un cookie l'identificativo di sessione è memorizzato direttamente nel browser del visitatore diventando accessibile in PHP sotto forma di variabile globale, per cui l'intera gestione della sessione è a carico dell'interprete PHP. Tale sistema, però, non è affidabile, per esempio il visitatore potrebbe aver configurato il browser in modo da rifiutare i cookie.

Tramite URL
Uguale al passaggio parametri con il metodo GET, per attuarla è necessario modificare i link alle pagine PHP in modo che contengano, nella parte di query string, il valore della costante SID, che è della forma
nome_sessione = ID_sessione
oppure è una stringa vuota, se non è avviata nessuna sessione

Una sessione consiste in una serie di accessi alle pagine PHP, effettuati in un determinato arco di tempo durante il quale è mantenuto uno stato. Ogni sessione è individuata da un identificatore, univoco, utilizzato per l'associazione tra client e relativa sessione.
Per utilizzare le sessioni in PHP si usano tre funzioni.
session_start()
È invocata per creare una nuova sessione o per ripristinarla se creata in precedenza; tenta anche d'impostare , nel browser, un cookie contenente l'identificativo di sessione, per cui è necessario che sia invocata all'inizio degli script, cioè prima del tag <*HTML*>.

session_register()
È usata per registrare delle variabili come variabili di sessione, ad esempio, se abbiamo una variabile *$nomeutente* e vogliamo renderla persistente per tutta la durata della sessione si lavora nel modo seguente.
```
// $nomeutente diventa variabile di sessione
session_register("nomeutente");
```

session_destroy()
È invocata per distruggere i dati relativi alla sessione, al log out.

Il programmatore deve prestare attenzione alla propagazione del SID (Identificatore di Sessione). In generale, è sufficiente memorizzare tale valore in un cookie nel browser del visitatore, PHP lo fa automaticamente, ma nel caso in cui i cookies non possono essere utilizzati perché il browser non li supporta o è stato configurato per rifiutarli, è il programmatore che deve farsi carico della propagazione del SID, modificando i link tra i vari script che condividono la sessione e passandolo come parametro.
```
<!--
Un esempio di link che propaga l'identificativo di sessione senza richiedere cookies
-->
<a href="altroscript.php?<?= SID ?>">Altro script</a>
```
Progettare uno script che propone al visitatore di cliccare su uno dei tre colori disponibili (bianco, rosso e verde), mostrando al contempo la sequenza delle selezioni effettuate fino a quel momento.

La sequenza dei clic sui vari colori è memorizzata in un array, registrato come variabile di sessione. La prima cosa da fare è l'avvio della sessione.

```
// Attivo (o ripristino) la sessione
session_start();
```

Il secondo passo è la registrazione dell'array $clicks come variabile di sessione.

```
// 'clicks' è una variabile di sessione: devo registrarla
session_register("clicks");
```

Passiamo ai parametri passati dall'utente. Per prima cosa verifichiamo se è stato cliccato il link "Ricomincia da capo", che comporta l'azzeramento della sequenza (dell'array).

```
// Devo azzerare?
if ($azzera) {
$clicks = array();
}
```

Se l'utente ha cliccato su uno dei colori lo aggiungiamo alla sequenza, inserendo un nuovo elemento in coda all'array.

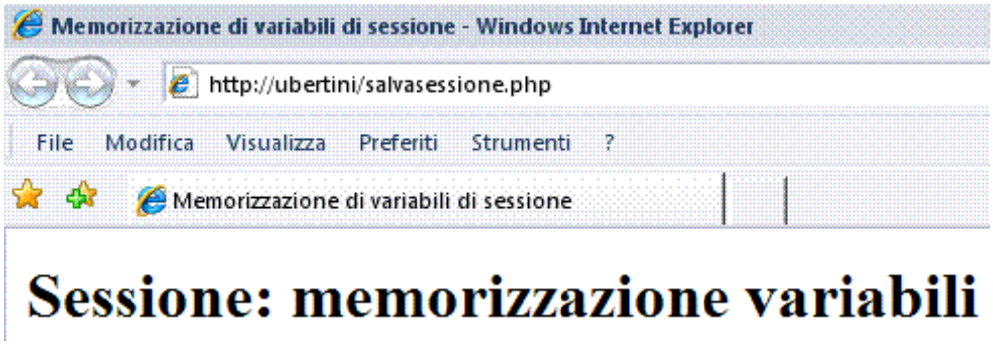
```
if ($click) {
$clicks[] = $click;
}
```

Infine, arriviamo alla visualizzazione della sequenza dei colori. Se l'array che rappresenta la sequenza contiene almeno un elemento, quindi se è stato scelto almeno un colore, sono visualizzati tutti i suoi elementi, diversamente è mostrata la scritta "sequenza vuota".

```
if (count($clicks)) {
foreach ($clicks as $colore) { echo "$colore "; }
} else {
echo "(sequenza vuota)";
}
```

salvasessione.php

```
<?php
        session_start();
?>
<html>
<head>
<title>Memorizzazione di variabili di sessione</title>
</head>
<body>
<h1>Sessione: memorizzazione variabili</h1>
<?php
        $_SESSION ['Cognome']="Rossi";
        $_SESSION ['Nome'] = "Piero";
?>
</body>
</html>
```



vedisessione.php

```
<?php
        session_start();
?>
<html>
<head>
<title>Visualizzazione delle variabili di sessione</title>
</head>
<body>
<h1>Sessione: visualizzazione variabili</h1>
<?php
        print ("SID di sessione:  ".Session_id()."</br>");
?>
<p>Valori di sessione</p>
<?php
        print("Cognome:  ".$_SESSION ['Cognome']."</br>");
        print("Nome:     ".$_SESSION ['Nome']."</br>");
?>
</body>
</html>
```

Contatore di pagine.

```
<?php
        // Avvia la sessione o la ripristina se già esistente
        session_start();
        // $contatore è una variabile di sessione
        session_register ("contatore");
        $contatore++;
?>
<html>
<head>
<title>Contatore di pagine visitate</title>
</head>
<body>
<h1>Numero di pagine visitate</h1><br>
<p>Numero di pagine visitate: <?php echo "$contatore"?><br>
</body>
```

```
</html>
```

AUTENTICAZIONE

Con la funzione *header* è possibile gestire senza la necessità di utilizzare i file *.htaccess* o comunque le autorizzazioni del server le pagine protette. Basterà inviare un header preciso al browser prima di passargli l'output per far si che, all'interno di una pagina, siano richiesti username e password per l'accesso.

```
<?php
        $username = "pippo";
        $pwd = "pluto";
        if(!isset($PHP_AUTH_USER)) {
                Header("WWW-Authenticate: Basic realm=\"Zona protetta\"");
                Header("HTTP/1.0 401 Unauthorized");
                print "Impossibile eseguire l'autorizzazione\n"; exit;
        } else {
                if (($PHP_AUTH_USER == $username) && ($PHP_AUTH_PW == $pwd)) {
                                        print "Autorizzazione riuscita per $username.";
                        } else { print "Autorizzazione fallita.";}
                        }
?>
```

Se i dati inseriti dall'utente nella maschera di autorizzazione coincidono con questi, l'autorizzazione riesce, altrimenti fallisce.

Prima di tutto, lo script controlla se sia impostata la variabile *PHP_AUTH_USER*, ossia l'username dell'utente che accede alla pagina protetta. Se questa esiste, probabilmente l'utente si è precedentemente autorizzato e, per questo motivo, essa è ricordata; in ogni caso, sia che l'utente si sia precedentemente autorizzato sia che sia la sua prima visita alla pagina, è controllato che l'username e la password precedentemente inserite o inserite al momento nella maschera di autorizzazione siano esattamente *$username* e *$pwd*: se combaciano, è possibile visualizzare l'output per l'avvenuta autorizzazione (quello che si è impostato come *Autorizzazione riuscita per $username*); nel caso contrario, è visualizzato un messaggio d'errore (*Autorizzazione fallita*). Se invece non esiste *$PHP_AUTH_USER*, è inviato un header per l'autorizzazione (il codice 401) che gli richiede username e password: una volta inserite, sarà eseguito il blocco descritto sopra e, se l'utente decide di non autorizzarsi, gli è presentata una pagina contenente il testo *Impossibile eseguire l'autorizzazione*. La funzione che controlla tutto questo meccanismo è *header(),* utilizzata per mandare un header al browser che ha inviato la richiesta; essendo quello che è inviato un header, è necessario che questo sia inviato prima di qualsiasi altro output.

OOP (*OBJECT ORIENTED PROGRAMMING*)

PHP è un linguaggio procedurale, ma supporta anche il paradigma OOP.

I concetti di base della OPP sono: **classe** e **oggetto** (**istanza**). La relazione tra le due nozioni è simmetrica: una classe è l'astrazione di un oggetto, mentre l'oggetto è istanza di una classe.

Una classe definisce un nuovo tipo di dato, caratterizzato da una propria struttura dati e da un insieme di funzioni che operano di di essa: le variabili dichiarate all'interno di una classe sono le **proprietà** (**attributi**), mentre le funzioni della classe sono i **metodi**.

La teoria della OOP stabilisce che le strutture dati interne ad una classe devono essere trasparenti all'utilizzatore della classe (il programmatore), che opererà su di esse soltanto tramite i metodi della classe e mai direttamente: gli stessi metodi devono essere utilizzati tenendo presente solo al loro semantica (cosa fanno) ed ignorandone l'implementazione (come lo fanno)

In PHP la dichiarazione di una classe avviene tramite l'istruzione *class*, seguita dal nome della classe e dal corpo racchiuso tra parentesi graffe. Il corpo della classe fornisce

l'implementazione ed è costituito da dichiarazioni di proprietà e metodi. Una proprietà è una variabile dichiarata all'interno di una classe in modo esplicito con la parola chiave *var*. L'accesso ad una proprietà avviene in modo diverso a seconda che ci si trovi
all'interno: *$this->nomevariabile*;
o all'esterno: *$nomeoggetto->nomevariabile*; della classe.
La dichiarazione di un metodo è uguale a quella di una funzione. Il richiamo di un metodo di una classe avviene in modo diverso a seconda che ci si trovi
all'interno: *$this->nomemetodo();*
o all'esterno: *$nomeoggetto-> nomemetodo();* della classe.
Se un metodo ha lo stesso nome della classe all'interno della quale è dichiarato diventa un costruttore: è un metodo che è invocato automaticamente ogni volta che è creata un'istanza della classe.
La parola chiave *new* consente d'istanziare una classe.

```
<?php
        // classe
        class visualizza {
                // proprietà
                var $i;
                // metodi
                function esegui_visualizza () {
                        $i = 7;
                        print "Visualizza un messaggio: $i";
                                                                }
                                        }
        // creazione di una istanza della classe
        $obj=new visualizza;
        $obj->esegui_visualizza();
?>
```

Per stabilire se una variabile è di tipo object si usano le funzioni: *is_object(), gettype()* restituisce una stringa object.

PAGERANK

Posizione di una pagina o di un sito web all'interno di un motore di ricerca. Il valore di PageRank di una pagina, non indica semplicemente il suo grado di popolarità sul web ma si spinge oltre fino a indicare un grado di autorevolezza. Il PageRank è uno dei tanti fattori che contribuisce a determinare la posizione della pagina stessa nei risultati delle ricerche: più il PageRank è alto e più sarà alta la posizione della pagina rispetto alle altre trovate con la stessa rilevanza. Il Pagerank è una caratteristica di Google che alcuni altri motori hanno cercato di imitare.

```
<?php
// fattore smorzante
$damping = 0.85;
// numero di iterazioni da compiere
$iterations = 10;
// valori iniziali del PR
$guess = 1;
// matrice che descrive quali pagine hanno link a quali altre pagine in questo esempio
// si vede che la pagina "1" ha un link verso la pagina 2 e un link verso la pagina 3
// la pagina 2 ha un solo link verso la pagina 3
$webMatrix = array
( array ("0", "1", "1", "0"),
 array ("0", "0", "1", "0"),
 array ("1", "0", "0", "0"),
 array ("0", "0", "1", "0") );
```

```
// fine configurazione utente
// popola l'array del PR iniziale
$pagerank = populate ($webMatrix, $guess);
// calcola il PR di tutte le pagine nella $webMatrix
for ($k = 0; $k < $iterations; $k++)
{ $pagerank = iteration ($webMatrix, $pagerank, $damping);}
// cosa esce
// Stampa informazioni relative al contenuto di una variabile in formato leggibile
print_r ($pagerank);
function iteration ($webMatrix, $pagerank, $damping){
        // aggiorna il PR di ogni pagina uno alla volta
        for ($i = 0; $i < count($webMatrix); $i++){
                $sum = 0;
                // calcola il PR di ogni pagina
                for ($j = 0; $j < count($webMatrix); $j++){
                        // trova quali pagine hanno un link alla pagina di cui stiamo
                        // calcolando il PR e calcola il PR che esce da queste pagine
                        ($webMatrix[$j][$i] != "0") ? $sum += $pagerank[$j] /
                        array_sum($webMatrix[$j]) : 0 ; }
                $temp = 1 - $damping + $damping * $sum;
                $pagerank[$i] = sprintf("%01.2f", $temp);  }
        return $pagerank;
}
function populate ($webMatrix, $guess){
        for ($i = 0; $i < count($webMatrix); $i++){$pagerank[$i] = $guess;}
        return $pagerank;
                                                                }
?>
```

ESEMPI

Visualizzare, ad ogni apertura di una pagina, un’immagine diversa

```
<?php
        // le immagini si devono chiamare con un numero esempio 1.gif
        // e si devono trovare in wwww\images
        $num = rand (1,10);
        print ("<img src=\"images/".$num.".gif\">");
?>
```

Spedire e-mail: *inviamail.htm*.

```
<html>
<head>
<title>Spedire mail</title>
</head>
<body>
<form method="get" action="invio.php">
<table>
<tr><td>Cognome e Nome</td>
<td><input type=text name="cognome"></td></tr>
<tr><td>E-mail</td>
<td><input type=text name="email"></td></tr>
<tr><td>Oggetto</td>
<td><input type=text name="oggetto"></td></tr>
<tr><td>Testo della email</td>
<td><input type=textarea name="testo" ROWS="5" COLS="10"
```

```
MAXLENGTH="200"></td></tr>
</table>
</br>
<input type="submit" value ="Invia">
</body>
</html>
invio.php
<?php
        $a = $_GET['cognome'];
        $b = $_GET['email'];
        $c = $_GET['oggetto'];
        $d = $_GET['testo'];
        if(isset($email)):
        // indirizzo di sposta cui spedire la mail
        $t="massimo@fauser.edu";
        mail($t,$b,"Nome: ".$a."\nOggetto: ".$c."\n\n".$d);
        endif;
?>
```

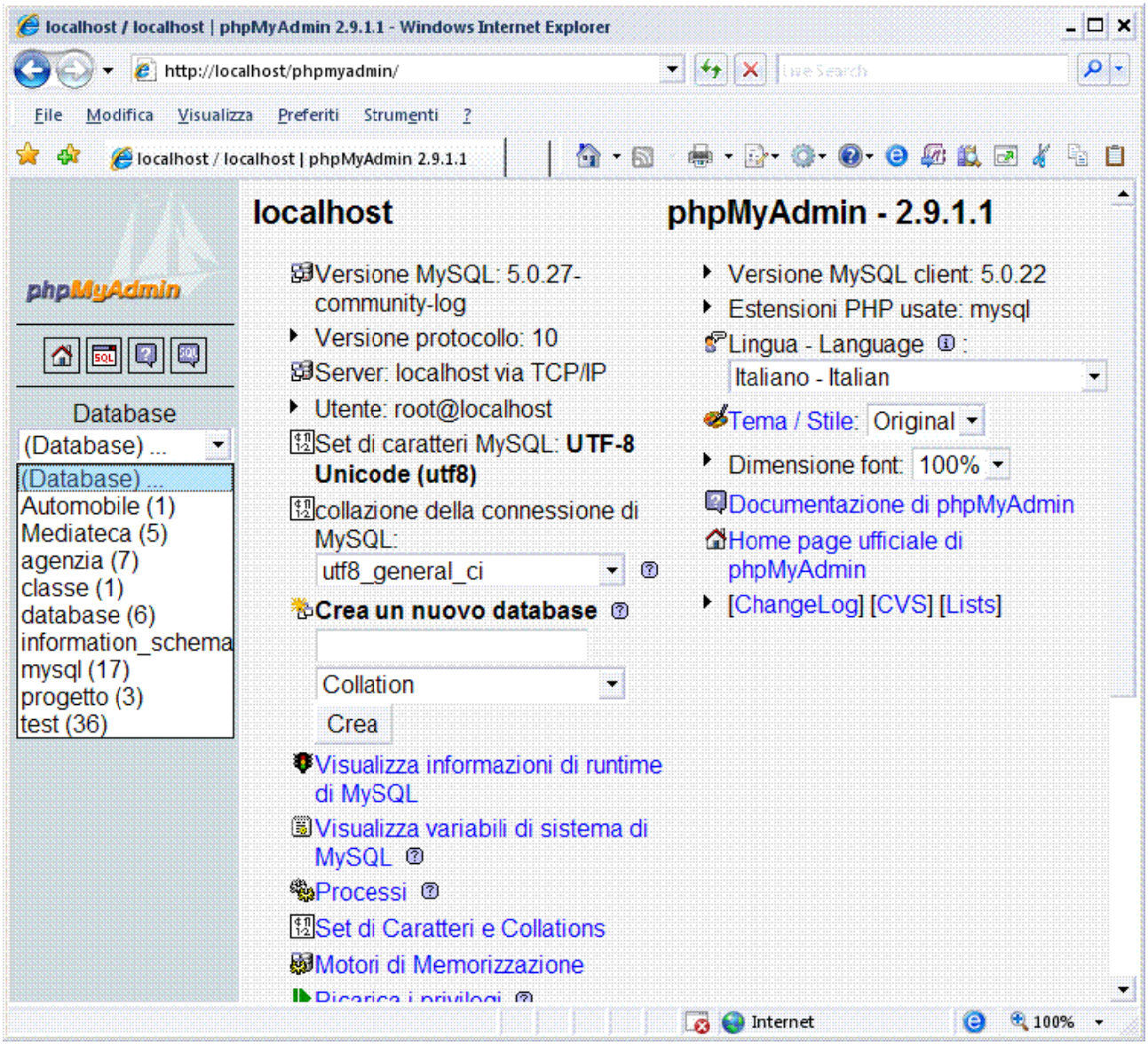
PHPMYADMIN

INTRODUZIONE

È un'interfaccia grafica che permette di amministrare MySQL, in pratica, si può visualizzare il contenuto del DB; creare, modificare, cancellare intere tabelle o singoli record; fare un backup dei dati contenuti; visualizzare informazioni sul DB.

INTERFACCIA GRAFICA

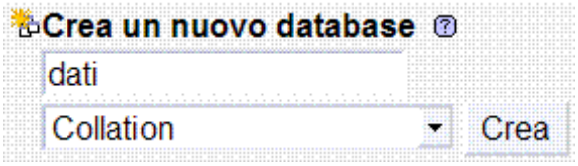
Digitare l'URL: *http://localhost/PHPadmin/.* È visualizzata una pagina composta da due frames; nella colonna di sinistra, ci sono i nomi di tutti i DB creati; se è la prima volta che si attiva MySQL ce ne sono solo due: *mysql* e *test*.
MySQL non va toccato perché contiene dati importanti per il funzionamento del DB.
Nella pagina centrale ci sono le risorse principali; per esempio, il form per creare un nuovo DB; ed una serie di link per visualizzare informazioni statistiche di runtime, di sistema; sono presenti, infine, collegamenti alla documentazione ufficiale.



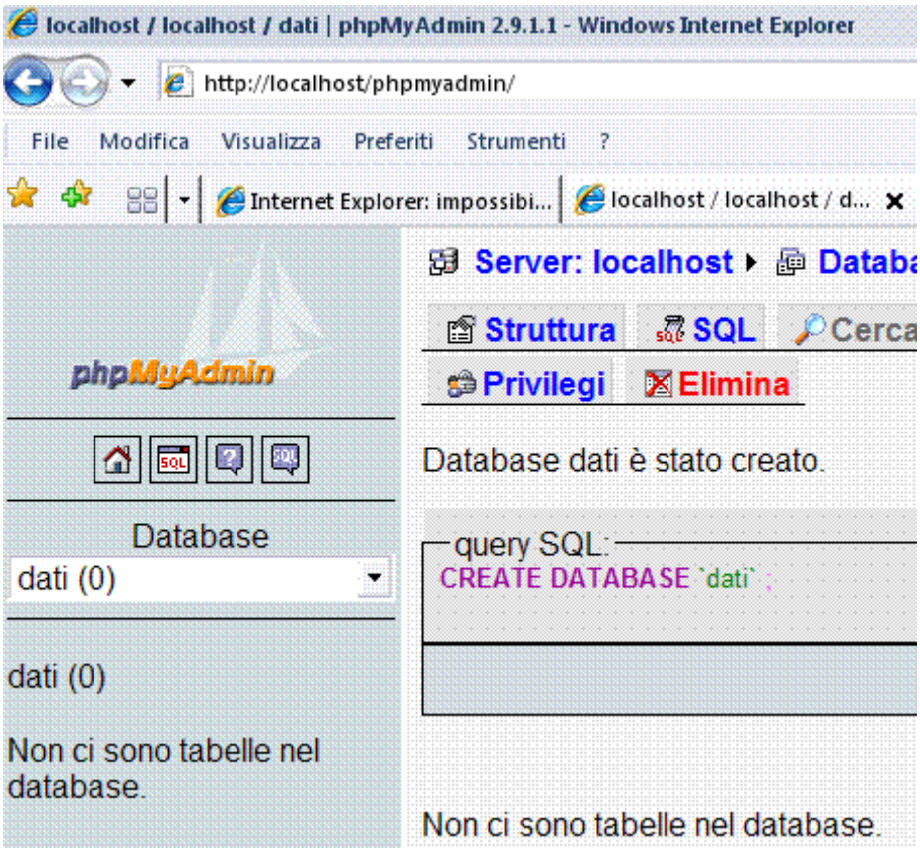
CREARE UN DB

Si deve creare un DB per ogni nuovo programma installato in modo da avere tutti i dati

suddivisi ed ordinati; è però possibile installare diversi script sullo stesso DB purché non ci siano tabelle con lo stesso nome.
Dal frame centrale: nel campo di testo sotto la scritta *Crea un nuovo database* inserire il nome del nuovo DB, per esempio *dati*.

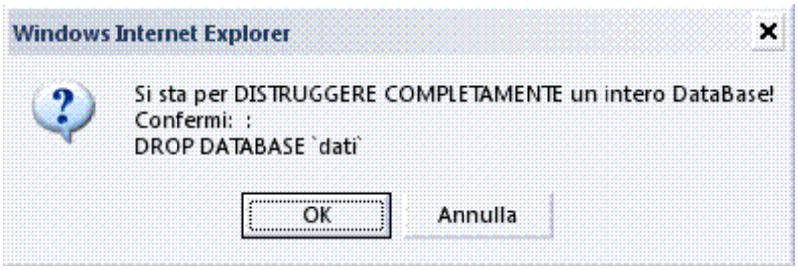


Una volta scritto il nome si preme sul pulsante *Crea* e, nel frame di sinistra, apparirà il DB che si è appena costruito; si noti anche la presenza di zero (0) e la scritta ad indicare l'assenza di tabelle all'interno del DB *dati*. Viceversa un numero indica il numero di tabelle all'interno di un DB, per esempio *mysql (17).*



CANCELLARE UN DB

Operazione inversa alla precedente. Cliccare una volta sul nome del DB da cancellare; se ci fossero delle tabelle al suo interno, queste sarebbero visualizzate sia nel frame di sinistra sia nella pagina centrale; in questo caso, invece, *dati* è vuoto. Fare clic sul pulsante *Elimina*, dare la conferma quando compare la seguente finestra.



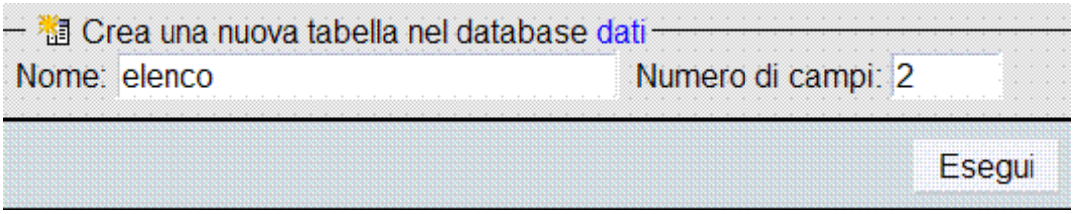
CREARE UNA TABELLA

Esistono due modi.

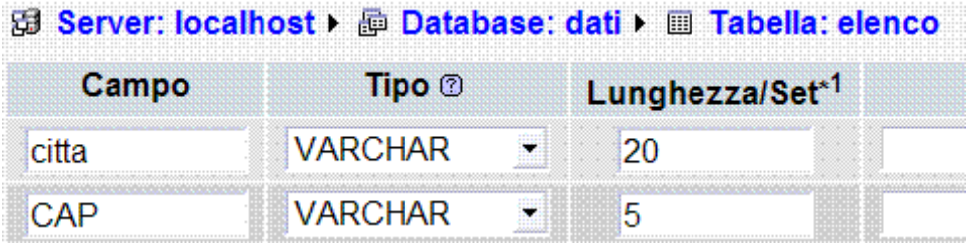
1. Manuale.
2. Automatico.

Manuale

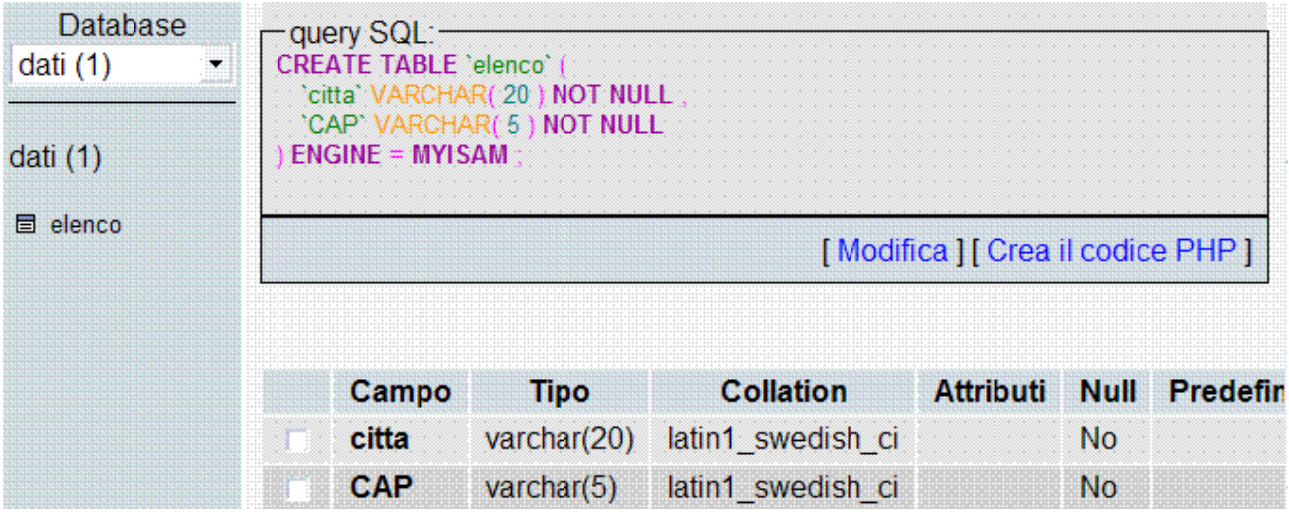
Usando il primo procedimento si deve impostare ogni singolo campo a mano, cliccare sul DB *dati*; nel frame centrale, a fondo pagina, si ha:
Crea una nuova tabella nel database dati



I due campi richiedono il nome da dare alla tabella e il numero di campi che questa tabella dovrà avere. Creare, la tabella *elenco*, con due campi; quindi cliccare su *Esegui*, PHPMyAdmin visualizzerà la struttura dei campi (2) della tabella (*elenco*).

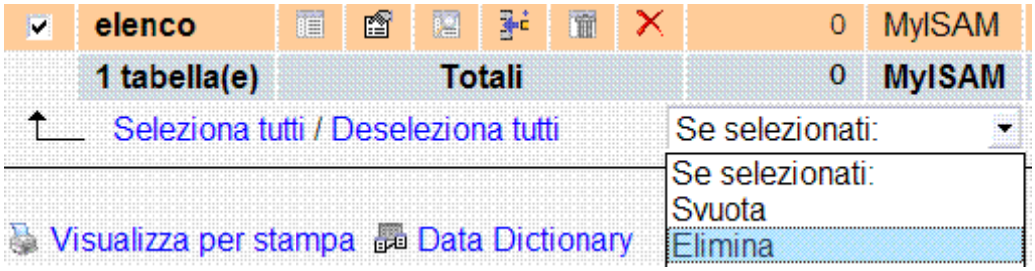


Riempire le voci *Campo*, *Tipo* e *Lunghezza.*
Fatto questo cliccare su *Salva* per completare l'operazione.



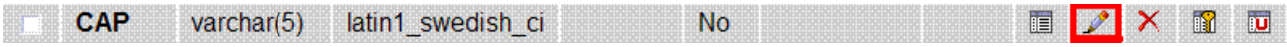
CANCELLARE O MODIFICARE UNA TABELLA

Segno di spunta sulla tabella, quindi fare clic sul menu a tendina *Se selezionati*:.



CANCELLARE O MODIFICARE UN CAMPO

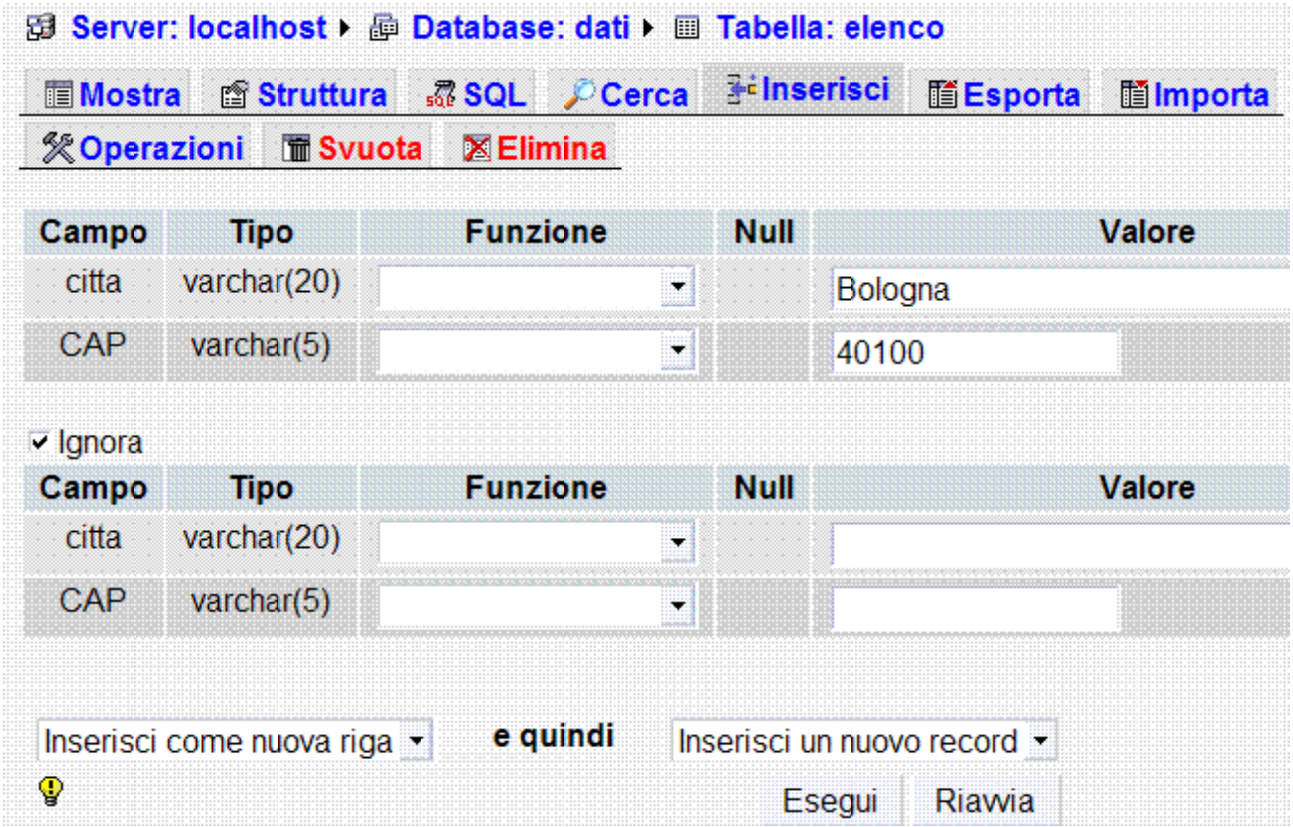
Nel caso si fosse commesso un errore nel compilare un campo si ha la possibilità di correggerli senza dover ripartire da zero. Cliccare sul nome del DB nel frame sinistro; dopo che il menu si sarà espanso, mostrando le tabelle presenti all'interno di *dati*, selezionare la tabella interessata (in questo caso *elenco*).



Selezionare l'icona *Modifica* e, nella schermata successiva, si fanno le modifiche; premere quindi *Salva* per aggiornare la tabella.

INSERIRE I DATI NEI CAMPI DELLA TABELLA

Cliccare sul nome del DB nel frame sinistro; dopo che il menu si sarà espanso, mostrando le tabelle presenti all'interno di *dati*, selezionare la tabella *elenco,* quindi fare clic sul pulsante *Inserisci.*



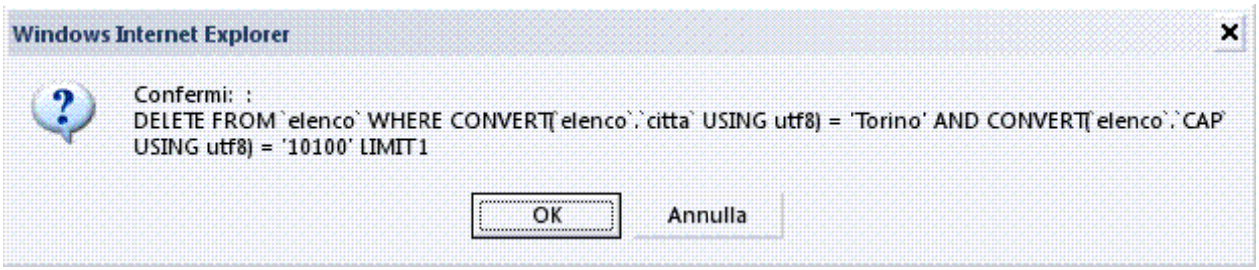
Selezionare *Inserisci come nuova riga e quindi Inserisci un nuovo record*, fare clic sul pulsante *Esegui*.

CANCELLARE O MODIFICARE I DATI NEI CAMPI DELLA TABELLA

Cliccare sul nome del DB nel frame sinistro; dopo che il menu si sarà espanso, mostrando le tabelle presenti all'interno di *dati*, selezionare la tabella *elenco,* quindi fare clic sul

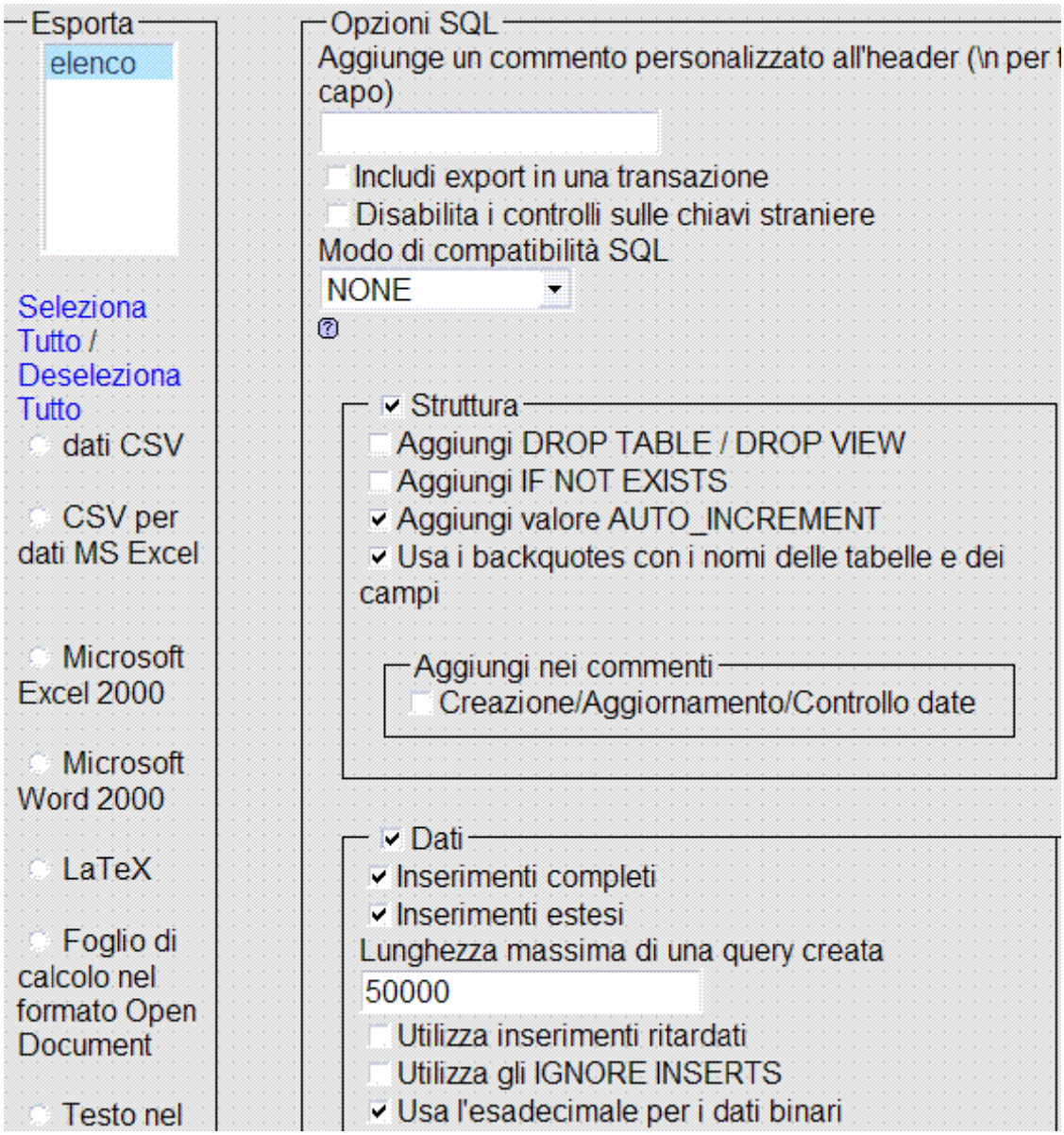
pulsante *Mostra.*
In fondo alla finestra compaiono i campi della tabella con i relativi dati.

Segno di spunta sul dato da modificare, quindi fare clic sull'icona *Modifica*, o sull'icona *Cancella*. dare la conferma quando compare la seguente finestra.



EFFETTUARE UNA COPIA DI BACKUP DEL DB

Cliccare sul nome del DB nel frame sinistro; dopo che il menu si sarà espanso, mostrando le tabelle presenti all'interno di *dati*, selezionare il DB *dati,* quindi cliccare sul pulsante *Esporta*. Compare una finestra con i checkbox già spuntati.



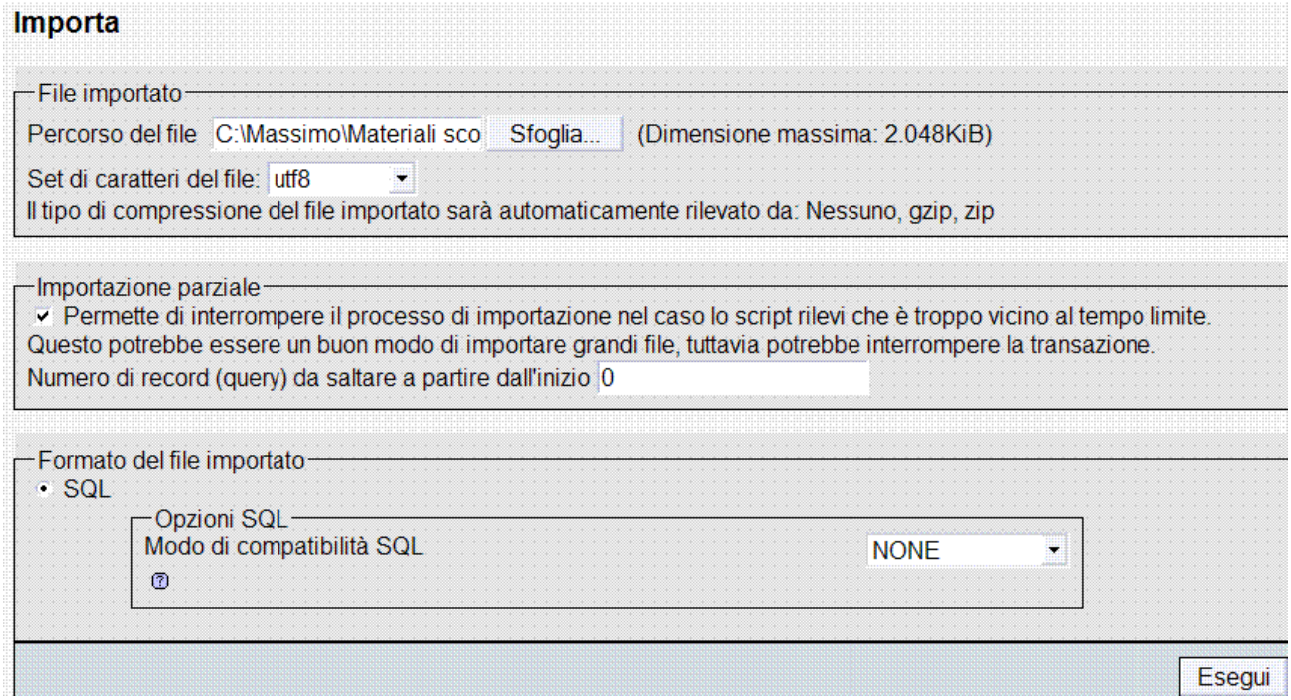
Una volta cliccato su *Esegui* e messo il segno di spunta *Salva con nome…* inizierà il salvataggio nel formato scelto del DB in modo da poter custodire una copia di sicurezza per ogni evenienza; in questo caso il file si chiama *dati.sql.*

Il contenuto del file *dati.sql* è il seguente.

```
-- phpMyAdmin SQL Dump
-- version 2.9.1.1
-- http://www.phpmyadmin.net
-- Host: localhost
-- Generato il: 7 Ago, 2008 at 04:29 AM
-- Versione MySQL: 5.0.27
-- Versione PHP: 5.2.0
-- Database: `dati`
-- Struttura della tabella `elenco`
CREATE TABLE `elenco` (
  `citta` varchar(20) NOT NULL,
  `CAP` varchar(5) NOT NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
-- Dump dei dati per la tabella `elenco`
INSERT INTO `elenco` (`citta`, `CAP`) VALUES
('Bologna', '40100'),
('Bologna', '40100'),
('Milano', '20100'),
('Roma', '00100'),
('Torino', '10100');
```

Reinstallare una copia di backup del DB

Cliccare sul nome del DB che si vuole ripristinare (se questo non esiste più basterà ricrearlo) nel frame sinistro; dopo che il menu si sarà espanso, mostrando le tabelle presenti all'interno di *dati*, selezionare il DB *dati,* quindi cliccare sul pulsante *Importa*.



Cliccare su *Esegui*. Il tempo d'importazione varierà a seconda della grandezza del file e alla fine del processo si devono ricevere questi messaggi:
Database xxxx
La query è stata eseguita con successo:
Il contenuto del file è stato inserito. (xxx Istruzioni)

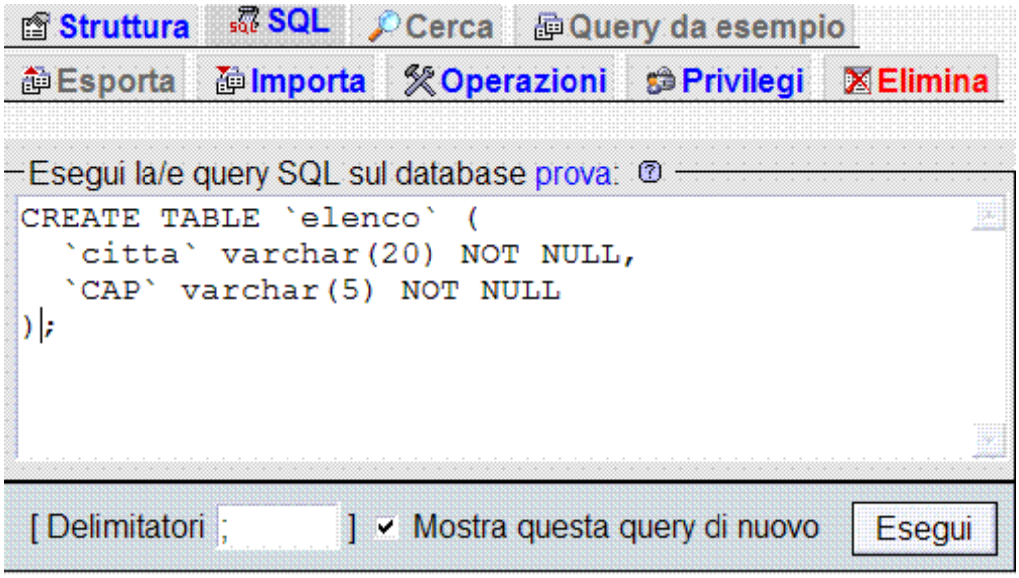
CREARE UNA TABELLA IN AUTOMATICO

Primo modo

La stessa procedura per reinstallare un backup è valida anche per importare un file *.sql*. Infatti, molti script, specialmente i più complessi, hanno un file *.sql* che permette di creare la struttura del DB senza dover impostare ogni singola tabella a mano. L'utente non dovrà fare altro che scegliere il DB appropriato e uploadare il file su MySQL. Ad esempio, è possibile rendere l'esercizio più veloce includendo lo script *dati.sql* già visto prima.

Secondo modo

Dal frame centrale: nel campo di testo sotto la scritta *Crea un nuovo database* inserire il nome del nuovo DB, per esempio *dati*.
Una volta scritto il nome si preme sul pulsante *Crea* e, nel frame di sinistra, apparirà il DB che si è appena costruito; si noti anche la presenza di zero (0) e la scritta ad indicare l'assenza di tabelle all'interno del DB *dati*.
Cliccare sul pulsante *SQL* ed inserire il comando, quindi fare clic su *Esegui* e la tabella sarà creata automaticamente.



Terzo modo: shell MySQL

C:\EasyPHP\mysql\bin>mysql -?
mysql Ver 14.12 Distrib 5.0.27, for Win32 (ia32)
Copyright (C) 2002 MySQL AB
This software comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to modify and redistribute it under the GPL license
Usage: mysql [OPTIONS] [database]
-?, --help Display this help and exit.
-I, --help Synonym for -?
--auto-rehash Enable automatic rehashing. One doesn't need to use 'rehash' to get table and field completion, but startup and reconnecting may take a longer time. Disable with --disable-auto-rehash.
-A, --no-auto-rehash No automatic rehashing. One has to use 'rehash' to get table and field completion. This gives a quicker start of mysql and disables rehashing on reconnect. WARNING: options deprecated; use --disable-auto-rehash instead.
-B, --batch Don't use history file. Disable interactive behavior. (Enables --silent)
--character-sets-dir=name

Directory where character sets are.
--default-character-set=name
Set the default character set.
-C, --compress Use compression in server/client protocol.
-#, --debug[=#] This is a non-debug version. Catch this and exit
-D, --database=name Database to use.
--delimiter=name Delimiter to be used.
-e, --execute=name Execute command and quit. (Disables --force and history file)
-E, --vertical Print the output of a query (rows) vertically.
-f, --force Continue even if we get an sql error.
-G, --named-commands
Enable named commands. Named commands mean this program's internal commands; see mysql> help . When enabled, the named commands can be used from any line of the query, otherwise only from the first line, before an enter. Disable with --disable-named-commands. This option is disabled by default.
-g, --no-named-commands
*Named commands are disabled. Use * form only, or use named commands only in the beginning of a line ending with a semicolon (;) Since version 10.9 the client now starts with this option ENABLED by default! Disable with '-G'. Long format commands still work from the first line. WARNING: option deprecated; use --disable-named-commands instead.*
-i, --ignore-spaces Ignore space after function names.
--local-infile Enable/disable LOAD DATA LOCAL INFILE.
-b, --no-beep Turn off beep on error.
-h, --host=name Connect to host.
-H, --html Produce HTML output.
-X, --xml Produce XML output
--line-numbers Write line numbers for errors.
-L, --skip-line-numbers
Don't write line number for errors. WARNING: -L is deprecated, use long version of this option instead.
-n, --unbuffered Flush buffer after each query.
--column-names Write column names in results.
-N, --skip-column-names
Don't write column names in results. WARNING: -N is deprecated, use long version of this options instead.
-O, --set-variable=name
Change the value of a variable. Please note that this option is deprecated; you can set variables directly with --variable-name=value.
--sigint-ignore Ignore SIGINT (CTRL-C)
-o, --one-database Only update the default database. This is useful for skipping updates to other database in the update log.
-p, --password[=name]
Password to use when connecting to server. If password is not given it's asked from the tty.
-W, --pipe Use named pipes to connect to server.
-P, --port=# Port number to use for connection.
--prompt=name Set the mysql prompt to this value.
--protocol=name The protocol of connection (tcp,socket,pipe,memory).

-q, --quick Don't cache result, print it row by row. This may slow down the server if the output is suspended. Doesn't use history file.
-r, --raw Write fields without conversion. Used with --batch.
--reconnect Reconnect if the connection is lost. Disable with --disable-reconnect. This option is enabled by default.
-s, --silent Be more silent. Print results with a tab as separator, each row on new line.
--shared-memory-base-name=name Base name of shared memory.
-S, --socket=name Socket file to use for connection.
--ssl Enable SSL for connection (automatically enabled with other flags). Disable with --skip-ssl.
--ssl-ca=name CA file in PEM format (check OpenSSL docs, implies --ssl).
--ssl-capath=name CA directory (check OpenSSL docs, implies --ssl).
--ssl-cert=name X509 cert in PEM format (implies --ssl).
--ssl-cipher=name SSL cipher to use (implies --ssl).
--ssl-key=name X509 key in PEM format (implies --ssl).
--ssl-verify-server-cert Verify server's "Common Name" in its cert against hostname used when connecting. This option is disabled by default.
-t, --table Output in table format.
-T, --debug-info Print some debug info at exit.
--tee=name Append everything into outfile. See interactive help (\h) also. Does not work in batch mode. Disable with --disable-tee. This option is disabled by default.
--no-tee Disable outfile. See interactive help (\h) also. WARNING: option deprecated; use --disable-tee instead
-u, --user=name User for login if not current user.
-U, --safe-updates Only allow UPDATE and DELETE that uses keys.
-U, --i-am-a-dummy Synonym for option --safe-updates, -U.
-v, --verbose Write more. (-v -v -v gives the table output format).
-V, --version Output version information and exit.
-w, --wait Wait and retry if connection is down.
--connect_timeout=# Number of seconds before connection timeout.
--max_allowed_packet=# Max packet length to send to, or receive from server
--net_buffer_length=# Buffer for TCP/IP and socket communication
--select_limit=# Automatic limit for SELECT when using --safe-updates
--max_join_size=# Automatic limit for rows in a join when using --safe-updates
--secure-auth Refuse client connecting to server if it uses old (pre-4.1.1) protocol
--show-warnings Show warnings after every statement.
Default options are read from the following files in the given order: C:\my.ini C:\my.cnf C:\WINDOWS\my.ini C:\WINDOWS\my.cnf C:\EasyPHP\mysql\my.ini C:\EasyPHP\mysql\my.cnf
The following groups are read: mysql client
The following options may be given as the first argument:
--print-defaults Print the program argument list and exit
--no-defaults Don't read default options from any options file

```
--defaults-file=#       Only read default options from the given file #
--defaults-extra-file=# Read this file after the global files are read
Variables (--variable-name=value)
and boolean options {FALSE|TRUE}  Value (after reading options)
auto-rehash                 FALSE
character-sets-dir          (No default value)
default-character-set       latin1
compress                    FALSE
database                    (No default value)
delimiter                   ;
vertical                    FALSE
force                       FALSE
named-commands              FALSE
local-infile                FALSE
no-beep                     FALSE
host                        (No default value)
html                        FALSE
xml                         FALSE
line-numbers                TRUE
unbuffered                  FALSE
column-names                TRUE
sigint-ignore               FALSE
port                        3306
prompt                      mysql>
quick                       FALSE
raw                         FALSE
reconnect                   TRUE
shared-memory-base-name     (No default value)
socket                      /tmp/mysql.sock
ssl                         FALSE
ssl-ca                      (No default value)
ssl-capath                  (No default value)
ssl-cert                    (No default value)
ssl-cipher                  (No default value)
ssl-key                     (No default value)
ssl-verify-server-cert      FALSE
table                       FALSE
debug-info                  FALSE
user                        (No default value)
safe-updates                FALSE
i-am-a-dummy                FALSE
connect_timeout             0
max_allowed_packet          16777216
net_buffer_length           16384
select_limit                1000
max_join_size               1000000
secure-auth                 FALSE
show-warnings               FALSE
C:\EasyPHP\mysql\bin>

C:\EasyPHP\mysql\bin>mysql -u root
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 53 to server version: 5.0.27-community-log
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
```

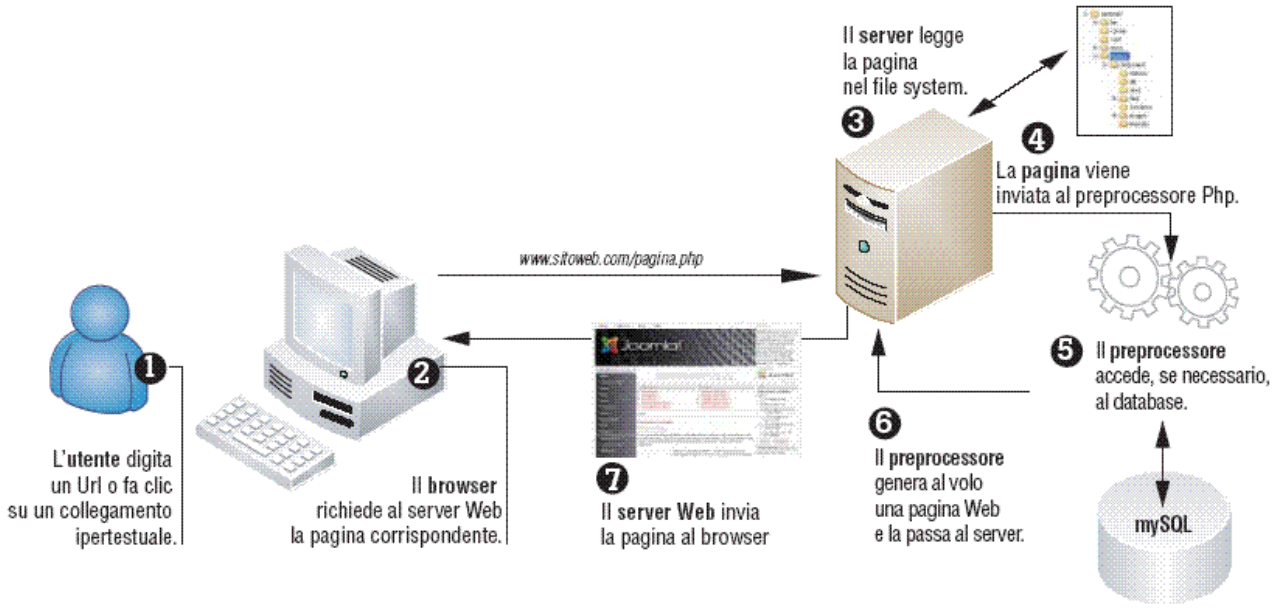
```
mysql> help
For information about MySQL products and services, visit:  http://www.mysql.com/
For developer information, including the MySQL Reference Manual, visit:
   http://dev.mysql.com/
To buy MySQL Network Support, training, or other products, visit:
   https://shop.mysql.com/
List of all MySQL commands:
Note that all text commands must be first on line and end with ';'
?         (\?) Synonym for `help'.
clear     (\c) Clear command.
connect   (\r) Reconnect to the server. Optional arguments are db and host.
delimiter (\d) Set statement delimiter. NOTE: Takes the rest of the line as new
delimiter.
ego       (\G) Send command to mysql server, display result vertically.
exit      (\q) Exit mysql. Same as quit.
go        (\g) Send command to mysql server.
help      (\h) Display this help.
notee     (\t) Don't write into outfile.
print     (\p) Print current command.
prompt    (\R) Change your mysql prompt.
quit      (\q) Quit mysql.
rehash    (\#) Rebuild completion hash.
source    (\.) Execute an SQL script file. Takes a file name as an argument.
status    (\s) Get status information from the server.
tee       (\T) Set outfile [to_outfile]. Append everything into given outfile.
use       (\u) Use another database. Takes database name as argument.
charset   (\C) Switch to another charset. Might be needed for processing binlog
with multi-byte charsets.
warnings  (\W) Show warnings after every statement.
nowarning (\w) Don't show warnings after every statement.
For server side help, type 'help contents'
mysql>quit
Bye
C:\EasyPHP\mysql\bin>

C:\EasyPHP\mysql\bin>mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 105 to server version: 5.0.27-community-log
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> create database dati;
Query OK, 1 row affected (0.00 sec)
mysql> use dati
Database changed
mysql> CREATE TABLE `elenco` ( `citta` varchar(20) NOT NULL, `CAP` varchar(5) NO
T NULL);
Query OK, 0 rows affected (0.00 sec)
mysql>quit
Bye
C:\EasyPHP\mysql\bin>
```

PHP E IL RDBMS MYSQL

INTRODUZIONE

L'impiego più importante di PHP si ottiene nel campo delle interrogazioni di DB presenti sul server. Consente al programmatore d'interfacciarsi con numerosi DB: PostgreSQL, MySQL, Oracle, Adabas, filePro, ODBC.



PRIMA FASE: CONNESSIONE

L'utilizzo dei comandi per la connessione ai DB prescinde dalla conoscenza dei DB stessi. La fase di connessione comprende anche l'autenticazione: nome utente e password sono indispensabili perché MySQL è un DB molto sicuro e non si può accedere ad esso senza aver prima creato un account. Occorrono le seguenti informazioni.

1. Il nome dell'host (*hostname*) del server MySQL.
2. Il nome utente (*username*).
3. La password (*password*).
4. Il nome del DB su cui si vuole lavorare.

Se si sta lavorando in locale.

1. Il nome dell'host (*hostname*) del server MySQL: *localhost*.
2. Il nome utente (*username*): *root*, oppure "".
3. La password (*password*): "".
4. Il nome del DB su cui si vuole lavorare.

Se si sta lavorando in remoto.

1. Il nome dell'host (*hostname*) del server MySQL: *$db_host = "labs.fauser.edu";*
2. Il nome utente (*username*): *$db_user = "a2002bx";*
3. La password (*password*): *$db_pass = "a2002bx";*
4. Il nome del DB su cui si vuole lavorare: *$db_name = "db2002bx"*

Se invece si devono provare gli script nel sito si deve richiedere il nome utente e la password all'amministratore di sistema.

```
<?php
        // apre la connessione
        $conn=mysql_connect("localhost","root","");
        if (!$conn) die ('Impossibile connettersi al DB:'.mysql_error());
        print "Connessione eseguita";
```

?>
La funzione, in caso di successo, restituisce un intero positivo che ha il significato di identificativo della connessione e che sarà utilizato successivamente in altre chiamate di funzione, come *mysql_query()* e *mysql_close().* La restituzione di un valore nullo, invece, denota il fallimento del tentativo di connessioen.
La funzione *die* permette di stampare a video la scritta compresa tra virgolette nel caso la connessione non vada a buon fine. Nel caso PHP non riesca a connettersi a MySQL si vede la scritta *Impossibile connettersi al DB*, nel caso contrario si vede la scritta *Connessione eseguita*.
Il passo successivo è la selezione del DB su cui si opererà. MySQL permette, per ogni account, illimitati DB.

```
<?php
        // seleziona il DB
        mysql_select_db("dati") or die ('Impossibile aprire il DB:'.mysql_error());
?>
```

Il risultato restituito dalla funzione è vero in caso di successo, falso in caso altrimenti.

Secondo fase: interazione

Si opera sul DB sia a livello di schema di DB (creare, modificare, cancellare tabelle), sia sui dati (inserimento, modifica, cancellazione di record). Per comunicare con il DBMS si utilizza il linguaggio **SQL** (*Structured Query Language*).
Per esempio, per inviare interrogazioni si può usare il seguente comando.
mysql_query (string query [,resource identificativo_connessione [,int modo_risultato]])
mysql_query() invia una query al DB attualmente attivo sul server associato all'identificativo di conmnessione specificato. Se *identificativo_connessione* non è specificato, è considerata l'ultima connessione aperta. Se nessuna connessione è aperta, la funzione prova a stabilire una connessione come se *mysql_connect()* fosse richiamata senza argomenti ed usa questa. Il paramentro opzionale *modo_risultato* può essere *MYSQL_USE_RESULT* e *MYSQL_STORE_RESULT* (valore predefinito), così il risultato è bufferato.

Terza fase: chiusura della connessione

Terminata la sessione di lavoro si chiude la connessione con il server.

```
<?php
        // chiude la connessione
        mysql_close($conn);
?>
```

Visualizzare i dati della tabella elenco del DB dati

```
<html>
<head>
<title>Visualizzazione dei dati della tabella elenco del DB dati</title>
</head>
<body>
<h1>Visualizzazione dati</h1>
<?php
        // apre la connessione
        $conn=mysql_connect("localhost","root","");
        if (!$conn) die ('Impossibile connettersi al DB:'.mysql_error());
        print "Connessione eseguita";
        // seleziona il DB
        mysql_select_db("dati") or die ('Impossibile aprire il DB:'.mysql_error());
?>
```

```
<br></br>
<table border="1">
<tr><th>Città</th><th>CAP</th></tr>
<?php
        // visualizzo i dati
        $risultato = mysql_query ("SELECT *from elenco");
        while ($riga=mysql_fetch_array($risultato))
                {
                print("<tr>");
                print ("<td>".$riga["citta"]."</td>");
                print ("<td>".$riga["CAP"]."</td>");
                print("</tr>");
           }
        // chiude la connessione
        mysql_close($conn);
?>
</table>
</body>
</html>
```

INSERIRE DATI NELLA TABELLA ELENCO DEL DB DATI

Questa pagina HTML è utilizzata per l'inserimento di una nuova città ed un nuovo CAP nella tabella *elenco* del DB *dati.*

```
<html>
<head>
<title>Inserimento dati nella tabella elenco del DB dati</title>
</head>
<body>
<h1>Inserimento dati</h1>
<form method="get" action="inserimentoDB.php">
<table>
<tr><td>Città</td>
<td><input type=text name="citta"></td></tr>
<tr><td>CAP</td>
```

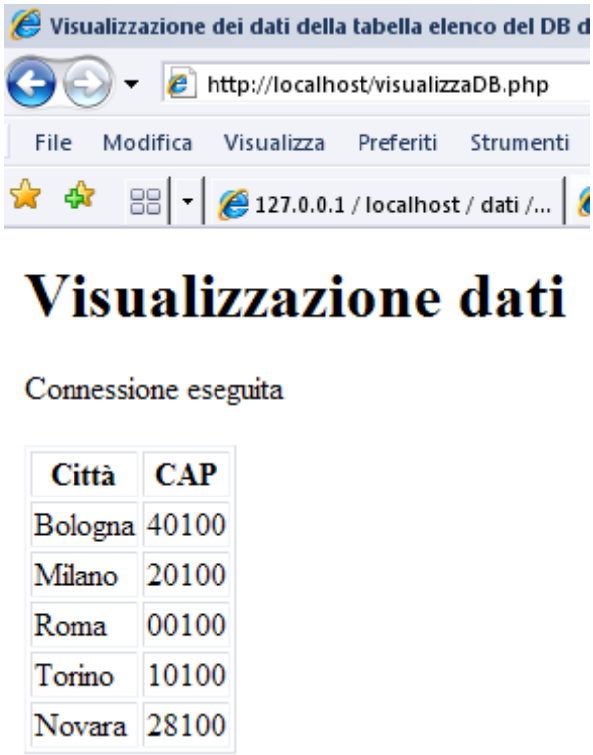
```
<td><input type=text name="CAP"></td></tr>
</table>
</br>
<input type="submit" value ="Inserisci nel DB">
</body>
</html>
```

Si riceve in input la citta e il CAP dalla pagina HTML, quindi s'inserisce il tutto nella tabella *elenco* del DB *dati.*

```
<html>
<head>
<title>Inserimento dati nella tabella elenco del DB dati</title>
</head>
<body>
<h1>Inserimento dati</h1>
<?php
	$ci = $_GET['citta'];
	$ca = $_GET['CAP'];
	// apre la connessione
	$conn=mysql_connect("localhost","root","");
	if (!$conn) die ('Impossibile connettersi al DB:'.mysql_error());
	// seleziona il DB
	mysql_select_db("dati") or die ('Impossibile aprire il DB:'.mysql_error());
	if (trim($ci)=="") {
?>
<p>La città deve essere specificata</p>
<a href="inviarichiestaDBphp.htm">Ripeti</a>
<?php
	}
	else {
		$insert="insert into elenco(citta,CAP)values ('$ci','$ca')";
		if (mysql_query ($insert)) print ("Inserimento riuscito");
		else print ("Inserimento non riuscito");
	}
	// chiude la connessione
	mysql_close($conn);
?>
</body>
```

</html>

Nuova visualizzazione dei dati della tabella *elenco* del DB *dati.*



RICERCARE DATI NELLA TABELLA ELENCO DEL DB DATI

Questa pagina HTML è utilizzata la ricerca di una città nella tabella *elenco* del DB *dati.*

```
<html>
<head>
<title>Ricerca dati nella tabella elenco del DB dati</title>
</head>
<body>
<h1>Ricerca dati</h1>
<form method="get" action="ricercaDBphp.php">
<table>
<tr><td>Città</td>
<td><input type=text name="citta"></td></tr>
<tr><td>CAP</td>
<td><input type=text name="CAP"></td></tr>
</table>
</br>
```

```
<input type="submit" value ="Ricerca nel DB">
</body>
</html>
```

Si riceve in input la citta dalla pagina HTML, quindi si ricerca il tutto nella tabella *elenco* del DB *dati.*

```
<html>
<head>
<title>Ricerca dati nella tabella elenco del DB dati</title>
</head>
<body>
<h1>Ricerca dati</h1>
<?php
        $ci = $_GET['citta'];
        $ca = $_GET['CAP'];
        $conn=mysql_connect("localhost","root","");
        if (!$conn) die ('Impossibile connettersi al DB:'.mysql_error());
        mysql_select_db("dati") or die ('Impossibile aprire il DB:'.mysql_error());
        if (trim($ci)=="") {
?>
<p>La città deve essere specificata</p>
<a href="ricercaDBphp.htm">Ripeti</a>
<?php
    }
    else {
?>
<br></br>
<table border="1">
<tr><th>Città</th><th>CAP</th></tr>
<?php
                $cur=mysql_query ("select * from elenco where citta='$ci'");
                while ($riga=mysql_fetch_array($cur)){
                        print("<tr>");
                        print ("<td>".$riga["citta"]."</td>");
                        print ("<td>".$riga["CAP"]."</td>");
                        print("</tr>");
                                        }
        }
```

```
        mysql_close($conn);
?>
</body>
</html>
```

FUNZIONI PHP - MYSQL

MySQL supporta una moltitudine di tipologie dei campi secondo i dati che si dovranno inserire. Per quanto riguarda i numeri MySQL supporta svariati tipi di archiviazione, i tipi più importanti sono i seguenti.

TINYINT *1 byte*
SMALLINT *2 byte*
MEDIUMINT *3 byte*
INT *4 byte*
BIGINT *8 byte*
FLOAT *4 byte*
DOUBLE *8 byte*
CHAR (numero_caratteri)
VARCHAR

Mentre in un campo *CHAR* la lunghezza è fissa ed è stabilita dal programmatore durante la creazione della tabella, nel campo *VARCHAR* il numero di caratteri è variabile e non bisogna specificare preventivamente il numero di caratteri massimo che dovrà contenere il campo. Naturalmente i campi di tipo *VARCHAR* occuperanno più memoria e la lettura al loro interno impiegherà qualche decimo di secondo in più rispetto ai campi *CHAR*.

Supponiamo di voler costruire una tabella che raccoglie una serie d'indirizzi e-mail per una eventuale mailing-list di un sito web. Si deve costruire una tabella con tre campi: un campo per un numero identificativo per indicizzare i dati, un campo per il nome e il cognome degli utenti ed infine la colonna che conterrà le e-mail.

mysql_create_db(): tenta di creare un nuovo DB nel server associato all'identificativo di conmnessione specificato, restituisce TRUE in caso di successo, FALSE in caso di fallimento.

bool mysql_create_db (string nome_database [, resource identificativo_connessione])

mysql_insert_id(): restituisce l'ultimo ID (se presente) della riga interessata dall'ultima operazione di *insert*.

```
<?php
        $dati = mysql _query("insert into mail (id_utente, nome_cognome, mail values ('2',
        'Mario Rossi', mario@tiscalinet.it') ");
        $ultimo_id = mysql _insert_id();
?>
```

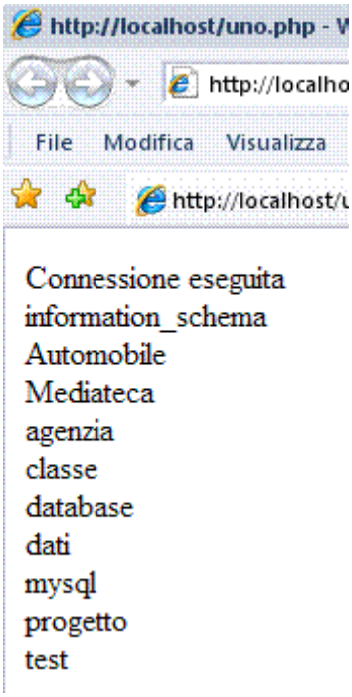
mysql_drop_db(): elimina un database MySQL.

```
<?php
        mysql _drop_db("nome_database_da_eliminare");
?>
```

mysql_list_dbs(): restituisce un risultato puntatore contenete i DB resi disponibili dal demone *mysql*. Usare la funzione *mysql_tablename()* per esplorare questo risultato puntatore o qualsiasi funzione per i risultati delle tabelle, come *mysql_fetch_array().*

```
<?php
        $conn=mysql_connect("localhost","root","");
        if (!$conn) die ('Impossibile connettersi al DB:'.mysql_error());
        print ("Connessione eseguita</br>");
        $lista = mysql_list_dbs($conn);
        while ($riga = mysql_fetch_object($lista)) {
        print $riga->Database . "</br>";
}
?>
```

L'esempio riportato produce il seguente output.



mysql_num_rows(): restituisce il numero di righe interessate dall'istruzione SQL.

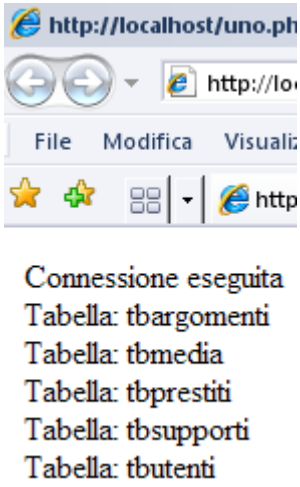
```
<?php
        $dati = mysql _query("select * from mail");
        $numero_righe = mysql _num_rows($dati);
?>
```

mysql_list_tables(): restituisce la lista delle tabelle presenti nel DB selezionato.

```
<?php
        $nome = "Mediateca";
        $conn=mysql_connect("localhost","root","");
        if (!$conn) die ('Impossibile connettersi al DB:'.mysql_error());
        print ("Connessione eseguita</br>");
        $risultato = mysql_list_tables($nome);
        if (!$risultato) {
                print "Errorore DB, Impossibile elencare le tabelle\n";
                print 'Errore MySQL: ' . mysql_error();
                exit;
        }
```

```
        while ($riga = mysql_fetch_row($risultato)) { print "Tabella: $riga[0]</br>";}
        mysql_free_result($risultato);
?>
```

L'esempio riportato produce il seguente output.



mysql_affected_rows(): ottiene il numero di righe coinvolte nelle precedenti operazioni MySQL, restituisce il numero di righe coinvolte nell'ultima query INSERT, UPDATE o DELETE associata a *identificativo_connessione*.
int mysql_affected_rows ([resource identificativo_connessione])
mysql_escape_string(): aggiunge le sequenze di escape a *stringa_senza_escape*, in modo che sia sicuro usarla in *mysql_query().*
string mysql_escape_string (string stringa_senza_escape)
mysql_errno(): restituisce il numero di errore dall'ultima funzione MySQL, oppure 0 (zero) se nessun errore è intercorso.
int mysql_errno ([resource identificativo_connessione])
string errore_mysql (): restituisce il testo dell'errore dall'ultima funzione MySQL, oppure la stringa vuota se nessun errore intercorre.
string errore_mysql ([resource identificativo_connessione])

PROGETTAZIONE DB CON COMANDI SQL

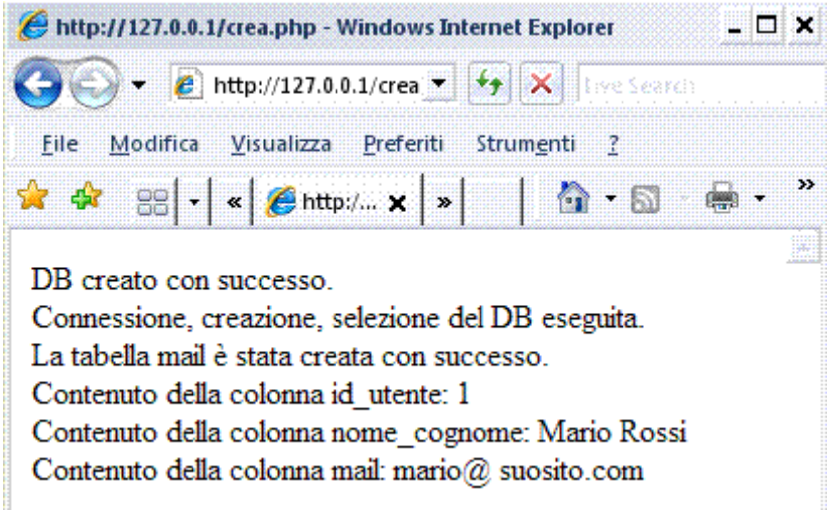
Progettare il DB di nome *sito* con una sola tabella di nome *mail* usando i comandi SQL.
La tabella deve essere composta dai tre campi descritti in figura.

Campo	Tipo
id_utente	int(10)
nome_cognome	char(100)
mail	char(50)

Dopo aver creato il DB *sito* e la tabella *mail* inserire un nuovo indirizzo all'interno della stessa tabella con il comando SQL *insert into*
'1','Mario Rossi','mario@suosito.com''
Non bisogna specificare a MySQL di creare un nuovo campo in quanto questo avviene in automatico con l'uso dell'istruzione la cui sintassi è la seguente.
insert into nome_tabella (nome_campi) values (dati_da_inserire_nei_campi);
L'elenco dei dati presente nella seconda coppia di parentesi tonde deve corrispondere all'ordine dei campi inseriti nella coppia delle prime parentesi. Ripetere lo script tutte le volte che si deve inserire un nuovo campo.
Ora che la tabella contiene un dato è possibile leggere al suo interno per sapere quali dati ci sono, per far questo si usa l'istruzione SQL *select*, che chiede i dati di una riga della tabella selezionata nell'istruzione *from*. Per poter utilizzare i dati che MySQL invia si deve utilizzare la funzione *mysql_fetch_array()* che crea un array associativo che ha come indice il nome delle colonne.
Visualizzare i dati estrapolati dal DB.

```
<?php
        // apro la connessione
        $conn=mysql_connect("localhost","root","");
        if (!$conn) die ('Impossibile connettersi al DB: '.mysql_error());
        // creo il DB
        if (mysql_query("CREATE DATABASE sito") ) {
                print ("DB creato con successo."."</br>");
        } else { printf ("Errore nella creazione del DB: ", mysql_error());}
        // seleziono il DB appena creato
        mysql_select_db ("sito") or die ('Non riesco a selezionare il DB: '.mysql_error());
        print ("Connessione, creazione, selezione del DB eseguita."."</br>");
        // creo la tabella mail
        $sql= "CREATE TABLE mail (
                id_utente INT(10) NOT NULL,
                nome_cognome CHAR(100) NOT NULL,
                mail CHAR(50) NOT NULL)";
        // Invio l'istruzione SQL a MySQL
$res = mysql_query($sql,$conn) or die ('Non riesco a creare la tabella: '.mysql_error());
        print ("La tabella mail è stata creata con successo."."</br>");
        // script per inserire i dati nella tabella mail: un nuovo indirizzo
        mysql_query("insert into mail (id_utente, nome_cognome,mail) values ('1','Mario
Rossi','mario@suosito.com')");
        // script per leggere i dati contenuti in un campo della tabella mail
        $dati = mysql_query("select * from mail");
        while ( $array = mysql_fetch_array($dati)) {
                print ("Contenuto della colonna id_utente: $array[id_utente] "."</br>");
        print ("Contenuto della colonna nome_cognome: $array[nome_cognome] "."</br>");
                print ("Contenuto della colonna mail: $array[mail] "."</br>");
                                        }
```

?>



Lo script legge una riga per volta e se nella istruzione *select* non si seleziona nulla MySQL restituirà i dati dell'ultima riga inserita, diversamente si può controllare il flusso dei dati MySQL con l'istruzione *where*.

```
<?php
        // visualizzare i dati dell'utente Mario Rossi
        $dati = mysql_query("select * from mail where nome_cognome='Mario Rossi");
        $array = mysql _fetch_array($dati);
?>
```

L'istruzione SQL risulta così organizzata.

"Seleziona (select) tutto (*) dalla tabella (from) mail dove (where) la colonna nome_cognome è uguale a Mario Rossi.

Questo script va bene nel caso si deve leggere solo una riga della tabella, nel caso in cui si vuole invece leggere tutto il contenuto della tabella si deve aggiungere un ciclo.

```
<?php
        // script per leggere i dati contenuti in tutti i campi della tabella mail
        $dati = mysql _query("select * from mail");
        while ( $array = mysql _fetch_array($dati)) {
                print "Contenuto della colonna id_utente: $array[id_utente] ";
                print "Contenuto della colonna nome_cognome: $array[nome_cognome] ";
                print "Contenuto della colonna mail: $array[mail] ";
                                                        }
?>
```

Con questo script finché la tabella *mail* non sarà vuota PHP creerà l'array associativo contenente i dati letti dal DB.

Modifica e cancellazione dei dati.

Per la modifica si usa l'istruzione *update*, ricordare sempre di specificare il *where* perché altrimenti la modifica sarà eseguita in tutti campi della tabella *mail*, la cui sintassi è.

```
update nome_tabella set nome_colonna='nuovo_valore' where
nome_colonna='identificativo_colonna';
```

```
<?php
        // script per la modifica di un dato nella tabella mail
        $dati = mysql _query ("update mail set mail='mario@tiscalinet.it' where
        nome_cognome='mario rossi'");
?>
```

Nella creazione della tabella all'inizio si era creata una colonna *id_utente* per memorizzare l'identificativo numerico del campo. Questo indice è importantissimo per le istruzioni di modifica e di cancellazione perché in questo modo ogni riga ha un numero univoco e non

si rischia di cancellare e/o modificare altre righe. Utilizzando l'identificativo si scrive.

```
<?php
        // script per la modifica di un dato nella tabella mail
        $dati = mysql _query ("update mail set mail='mario@tiscalinet.it' where
        id_utente='1'");
?>
```

Per la cancellazione, l'istruzione *delete,* permette di cancellare un intera riga dalla tabella.

```
<?php
        // script per la cancellazione di tutti i dati nella tabella mail
        $dati = mysql _query ("delete from mail");
?>
```

Con questo script si cancellano tutte le righe presenti all'interno della tabella *mail*. Nel caso si voglia cancellare una determinata riga s'inserisce nell'istruzione SQL l'istruzione *where*.

```
<?php
        // script per la cancellazione di una riga nella tabella mail
        $dati = mysql _query ("delete from mail where id_utente='1'");
?>
```

CLICKTHROUGH

Clic su un banner che conduce il navigatore alla pagina web dell'inserzionista. Questo software tiene traccia dei clicktrough mediante la creazione di un account sul sito. Per fare questo, crea un identificatore unico (ID di rintracciamento) in modo da controllare da quale sito il clic è arrivato.

Il primo passo consiste nella progettazione del DB *dbTracking* e delle tabelle *tblAccounts* e *tblClicks* che memorizzano gli account e i clicktrough.

Questo è ottenuto dalla shell MySQL con il codice seguente.

```
-- Host: localhost    Database: dbTracking
-- Table structure for table `tblAccounts`
CREATE TABLE `tblAccounts` (
  `accountID` int(11) NOT NULL auto_increment,
  `accountCompany` varchar(255) default NULL,
  `accountWebsite` varchar(255) default NULL,
  `accountGUID` varchar(50) default NULL,
  PRIMARY KEY  (`accountID`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
-- Table structure for table `tblClicks`
CREATE TABLE `tblClicks` (
  `clickID` int(11) NOT NULL auto_increment,
  `accountID` int(11) default NULL,
  `clickDate` datetime default NULL,
  PRIMARY KEY  (`clickID`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
C:\EasyPHP\mysql\bin>mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 105 to server version: 5.0.27-community-log
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> create database dbTracking;
Query OK, 1 row affected (0.00 sec)
mysql> use dbTracking
Database changed
mysql> CREATE TABLE `tblAccounts` (
    ->  `accountID` int(11) NOT NULL auto_increment,
    ->  `accountCompany` varchar(255) default NULL,
    ->  `accountWebsite` varchar(255) default NULL,
```

```
    ->  `accountGUID` varchar(50) default NULL,
    ->  PRIMARY KEY  (`accountID`)
    -> ) ENGINE=MyISAM DEFAULT CHARSET=latin1;
Query OK, 0 rows affected (0.00 sec)
mysql> CREATE TABLE `tblClicks` (
    ->  `clickID` int(11) NOT NULL auto_increment,
    ->  `accountID` int(11) default NULL,
    ->  `clickDate` datetime default NULL,
    ->  PRIMARY KEY  (`clickID`)
    -> ) ENGINE=MyISAM DEFAULT CHARSET=latin1;
Query OK, 0 rows affected (0.01 sec)
mysql>quit
Bye
C:\EasyPHP\mysql\bin>
```

Il secondo passo consiste nel progettare tre script.
Il primo script elenca tutti gli account ordinati per nome.

```
<?php require('inc/db.php');?>
<head>
<title>accounts.php</title>
<link rel="stylesheet" type="text/css" href="inc/layout.css" />
</head>
<body>
<div id="header"></div>
<div id="content">
<h1>Account</h1>
<?php if($_GET['new']) { ?>
<p><strong>Il nuovo account é stato aggiunto</strong></p>
<?php } ?>
<p>Questi sono gli account sul sistema.
Per aggiungere un nuovo account completa il <a href="nuovoaccount.php">form</a>.</p>
<?php
        $sql = "select * FROM tblAccounts order by accountCompany";
        if ($result = mysql_query($sql)) { ?>
        <table class="datatable">
                <tr>
                        <th>Compaglia</th>
                        <th>Sito Web</th>
                        <th>Tracking URL</th>
                </tr>
<?php while($row = mysql_fetch_array($result)) { ?>
                <tr>
                        <td><?php echo $row['accountCompany']; ?></td>
                        <td><?php echo $row['accountWebsite']; ?></td>
                        <td>http://www.tuodominio.com/?tid=<?php                echo
$row['accountGUID']; ?></td>
                </tr>
<?php } ?>
        </table>
<?php } ?>
</div>
</body>
</html>
```

Se c'è una variabile *new*, un nuovo account è stato aggiunto sulla pagina che si sta per creare.

Il secondo script crea un nuovo account.
Se il form è stato avviato s'inizia ad aggiungere nuovi account al DB

```
<?php
	require('inc/db.php');
	if($_POST['btnSubmit']) {
	$accountCompany = $_POST['accountCompany'];
	$accountWebsite = $_POST['accountWebsite'];
	$accountGUID =  md5(uniqid(1));
	if($accountCompany != '' and $accountWebsite != '') {
	$sql="insert into tblAccounts (accountCompany,accountWebsite,accountGUID)
values ('".$accountCompany."','".$accountWebsite."','".$accountGUID."')";
	if (mysql_query($sql, $conn)){
			header("Location: accounts.php?new=true");
				}
	} else {$err = true;}
}
?>
<head>
<title>nuovoaccount.php</title>
<link rel="stylesheet" type="text/css" href="inc/layout.css" />
</head>
<body>
<div id="header"></div>
<div id="content">
<h1>Aggiungi un nuovo account</h1>
<?php if($err == true) { ?>
<p>Devi riempire entrambi i campi per creare un nuovo account.</p>
<?php } else { ?>
<p>Aggiungi un nuovo account, ciò permetterà di inserire link al tuo sito <br>
e i loro clickthrough saranno registrati.</p>
<?php } ?>
<form method="post" action="nuovoaccount.php" id="admin-form">
<p><label for="accountCompany">Nome compagnia</label>:
<br /><input type="text" name="accountCompany" id="accountCompany" class="text"
/></p>
<p><label for="accountWebsite">Sito web</label>:
<br /><input type="text" name="accountWebsite" id="accountWebsite" class="text" /></p>
<p><input type="submit" name="btnSubmit" value="Crea Account" /></p>
</form>
</div>
</body>
</html>
```

Il terzo script è una home page, è la pagina a cui i possessori di un account accedono attraverso i clic sugli annunci pubblicitari nei loro siti.

```
<?php
	require('inc/db.php');
	if($_GET['tid']) {
	$accountGUID  = $_GET['tid'];
	$sql="select accountID from tblAccounts where accountGUID='".$accountGUID."'";
	if ($result = mysql_query($sql)) {
		$row = mysql_fetch_array($result);
		$accountID = $row['accountID'];
		$datetime = date("Y-m-d G:i:s");
$sql="insert into tblClicks (accountID,clickDate) values(".$accountID.",'".$datetime. "')";
```

```
                if(mysql_query($sql, $conn)) {
                        $tracked = true;
                } else {$tracked = false;}
        }
}
?>
<head>
<link rel="stylesheet" type="text/css" href="inc/layout.css" />
<title>index.php</title>
</head>
<body>
<div id="header">
</div>
<div id="content">
<p><?php
                if($tracked) {
                        echo 'Questo riferimento ha generato un clickthrough';
                } else {
                        echo 'Questo riferimento non é arrivato da una URL affiliata';
                }
?>
</p>
</div>
</body>
</html>
```

Se c'è una variabile *tid*, essa è un clicktrough rintracciato e quindi copiare il contenuto nella variabile account GUID.
File include nella sotto cartella *inc*.

DB.php

```
<?php
        $conn = mysql_connect("localhost","root","");
        mysql_select_db("dbTracking", $conn);
?>
```

layout.css

```
body {
        padding: 0;
        margin: 0;
        background-color: #FFFFFF;
        color: #000000;
        font-size: 1em;
}
#header {
        background-color: #2F8BBB;
        border-top: 1px solid #92C8E4;
        border-bottom: 1px solid #1C5572;
        height: 10px;
}
#content {
        padding: 1em 2em 0 2em;
        margin-right: 220px;
        font-size: 80%;
        font-family: Arial, Helvetica, sans-serif;
}
.text {
```

```
        width: 300px;
        border: 1px solid #2F8BBB;
}
.datatable {
        border-collapse: collapse;
        border: 1px solid #92C8E4;
}
.datatable th {
        background-color: #ECF6FA;
        text-align: left;
        padding: 0.2em 0.2em 0.2em 0.4em;
        border: 1px solid #92C8EF;
}
.datatable td {
        padding: 0.2em 0.2em 0.2em 0.4em;
        border: 1px solid #92C8EF;
}
```

PHP - ACCESS

INTRODUZIONE

Su Windows, PHP è in grado di lavorare anche con DB diversi da MySQL, sfruttando a pieno le caratteristiche e le interfacce richieste da DBMS differenti.
Con PHP è quindi possibile creare uno script che permetta di lavorare con un DB Access. Per accedere ad un DB Access si usano le funzioni **ODBC** (*Open DataBase Connectivity*), è un metodo di accesso standard verso un DB sviluppato dal SQL group nel 1992, di PHP; per fare questo si deve però creare un **DSN** di sistema (*Data Source Name*) in pratica un file che contiene le informazioni relative al tipo e al percorso del DB. Lo scopo di ODBC è quello di rendere il protocollo di accesso al DB indipendente dal tipo di DB utilizzato.

```
<?php
        $dsn = ‘dati.mdb';
        $connessione = odbc_connect("dati.mdb", "nomeutente","password");
        $interrogazione = "select * from elenco";
        $html_risultato ="";
        if ($risultato = odbc_exec($connessione, $interrogazione)) {
            do {
               $html_risultato .= '<tr>';
               $html_risultato .= '<td> '.odbc_result($risultato, "citta").'</td>';
               $html_risultato .= '<td> '.odbc_result($risultato, "CAP").'</td>';
               $html_risultato .= '</tr>';
           } while (odbc_fetch_row($risultato));
        }
        odbc_close($connessione);
?>
```

Molti hosting però non permettono di utilizzare un DSN; in questo caso, quindi, è possibile usare le funzioni **COM** (*Component Object Model*) di PHP.
L'interfaccia tra un qualsiasi linguaggio di programmazione con DBMS di Microsoft, ad esempio Access, è **ADO** (*ActiveX Data Object*), le informazioni di accesso al DB dovranno essere inserite in una stringa che poi si passa al metodo *Open()* dell'oggetto *ADODB.Connection*.
ADO è una libreria di casa Microsoft che permette l'interazione con DBMS di casa Microsoft stessa o con altri DBMS, ad esempio con MySQL per il suo utilizzo con ASP. ADO mette a disposizione tre oggetti fondamentali: *Connection*, *Recordset* e *Command*. In questo esempio si useranno i primi due che servono rispettivamente a gestire la connessione ed a gestire i dati. Il terzo è un oggetto più specialistico.
I programmatori ASP sono abituati ad utilizzare il carattere punto (.) per separare il nome della variabile che contiene, ad esempio la connessione, col metodo o con la proprietà da utilizzare; in PHP, per la sola interazione con ADO, si usa come separatore la forma: *variabile->metodo*.

VISUALIZZARE I DATI DELLA TABELLA ELENCO DEL DB DATI

Posizionare il DB, il file *dati.mdb*, sul server web Apache o EasyPHP, per esempio nella path indicata nella stringa di connessione che *C:\EasyPHP\www\dati.mdb*

```
<html>
<head>
<title>Visualizzazione dei dati della tabella elenco del DB dati</title>
</head>
<body>
<h1>Visualizzazione dati</h1>
```

```
<?php
	// $db contiene il percorso fisico del DB
	$db = "C:\EasyPHP\www\dati.mdb";
	// $sc contiene la stringa di connessione OLEDB al DB
	$sc = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=$db;";
	// Crea due oggetti COM contenenti gli oggetti Connection e Recordset
	$cn = new COM("ADODB.Connection");
	$rs = new COM("ADODB.Recordset");
	// Apre la Connection
	$cn->open($sc);
?>
<br></br>
<table border="1">
<tr><th>Città</th><th>CAP</th></tr>
<?php
	// Apre il Recordset per visualizzare i dati
	$rs->Open("select * from elenco", $cn);
	// Controlla che sulla tabella ci siano dati
	if ($rs->EOF) print "<p>Nessun dato trovato</p>";
	// Se ce ne sono effettua un ciclo di lettura
	else while ($rs->EOF == FALSE) {
		print("<tr>");
		print ("<td>" . $rs->Fields ['citta']->value."</td>");
		print ("<td>" . $rs->Fields ['CAP']->value."</td>");
		print("</tr>");
		$rs->MoveNext();
					}
	// Chiude il Recordset
	$rs->Close();
	// $rs->Release();
	$rs = null;
	// Chiude la Connection
	$cn->Close();
	// $cn->Release();
	$cn = null;
?>
</table>
</body>
</html>
```

Il funzionamento è semplice: al *Recordset* si passa la query in lettura e si specifica la variabile in cui è stata aperta la connessione; la proprietà EOF dell'oggetto *Recordset* (*End Of File*) legge tutti i dati in funzione della query specificata; se settata su *TRUE* significa che non ci sono dati nel DB, quindi si lancia il messaggio di notifica; se settata su *FALSE* all'interno di un ciclo iterativo, legge tutti i dati in funzione della query specificata; il metodo *MoveNext()* interrompe il ciclo e posiziona il cursore di ADO all'inizio del ciclo, evitando un loop infinito. Per la chiusura degli oggetti si usano i metodi *Close()* e *Release()* che servono rispettivamente a chiudere l'oggetto ed a distruggere un COM. In fine si setta la variabile su *NULL* in modo da svuotare la memoria della variabile.

Inserire dati nella tabella elenco del DB dati

Il criterio con cui si scrivono dei dati su un DB Access con PHP è simile al criterio di lettura dei dati; nel codice che segue non si utilizza il *Recordset,* ma si esegue una *insert sql* grazie al metodo *Execute()* dell'oggetto *Connection*.

Questa pagina HTML è utilizzata per l'inserimento di una nuova città ed un nuovo CAP nella tabella *elenco* del DB *dati.*

```
<html>
<head>
<title>Inserimento dati nella tabella elenco del DB dati</title>
</head>
<body>
<h1>Inserimento dati</h1>
<form method="get" action="inserimentoDBmdb.php">
<table>
<tr><td>Città</td>
<td><input type=text name="citta"></td></tr>
<tr><td>CAP</td>
<td><input type=text name="CAP"></td></tr>
</table>
</br>
<input type="submit" value ="Inserisci nel DB">
</body>
</html>
```

Si riceve in input la citta e il CAP dalla pagina HTML, quindi s'inserisce il tutto nella tabella *elenco* del DB *dati.*

```
<html>
<head>
<title>Inserimento dati nella tabella elenco del DB dati</title>
</head>
<body>
<h1>Inserimento dati</h1>
<?php
        $ci = $_GET['citta'];
        $ca = $_GET['CAP'];
        $db = "C:\EasyPHP\www\dati.mdb";
        $sc = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=$db;";
        $cn = new COM("ADODB.Connection");
        $cn->open($sc);
        if (trim($ci)=="") {
?>
<p>La città deve essere specificata</p>
<a href="inviarichiestaDBphpmdb.htm">Ripeti</a>
<?php
   }
   else {
                $insert="insert into elenco(citta,CAP)values ('$ci','$ca')";
                $cn -> Execute ($insert);
                print ("Inserimento riuscito");
       }
   $cn->Close();
   $cn = null;
?>
</body>
</html>
```

Ricercare dati nella tabella elenco del DB dati

Questa pagina HTML è utilizzata la ricerca di una città nella tabella *elenco* del DB *dati.*

```
<html>
```

```
<head>
<title>Ricerca dati nella tabella elenco del DB dati</title>
</head>
<body>
<h1>Ricerca dati</h1>
<form method="get" action="ricercaDBmdb.php">
<table>
<tr><td>Città</td>
<td><input type=text name="citta"></td></tr>
<tr><td>CAP</td>
<td><input type=text name="CAP"></td></tr>
</table>
</br>
<input type="submit" value ="Ricerca nel DB">
</body>
</html>
```

Si riceve in input la citta dalla pagina HTML, quindi si ricerca il tutto nella tabella *elenco* del DB *dati.*

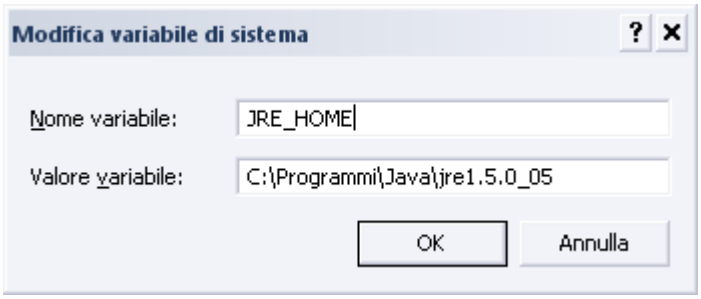
```
<html>
<head>
<title>Ricerca dati nella tabella elenco del DB dati</title>
</head>
<body>
<h1>Ricerca dati</h1>
<?php
        $ci = $_GET['citta'];
        $ca = $_GET['CAP'];
        $db = "C:\EasyPHP\www\dati.mdb";
        $sc = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=$db;";
        if (trim($ci)=="") {
?>
<p>La città deve essere specificata</p>
<a href="ricercaDBphpmdb.htm">Ripeti</a>
<?php
   }
   else {
        $cn = new COM("ADODB.Connection");
        $rs = new COM("ADODB.Recordset");
        $cn->open($sc);
        $query="select * from elenco where citta='$ci';";
?>
<br></br>
<table border="1">
<tr><th>Città</th><th>CAP</th></tr>
<?php
        if ($ci == '') $query="select * from elenco;";
                $rs->Open($query, $cn);
                if ($rs->EOF) print "<p>Nessun dato trovato</p>";
                else while ($rs->EOF == FALSE){
                        print("<tr>");
                        print ("<td>" . $rs->Fields ['citta']->value."</td>");
                        print ("<td>" . $rs->Fields ['CAP']->value."</td>");
                        print("</tr>");
                        $rs->MoveNext();
```

```
                                        }
    $rs->Close();
    $rs = null;
    $cn->Close();
    $cn = null;
    }
?>
</body>
</html>
```

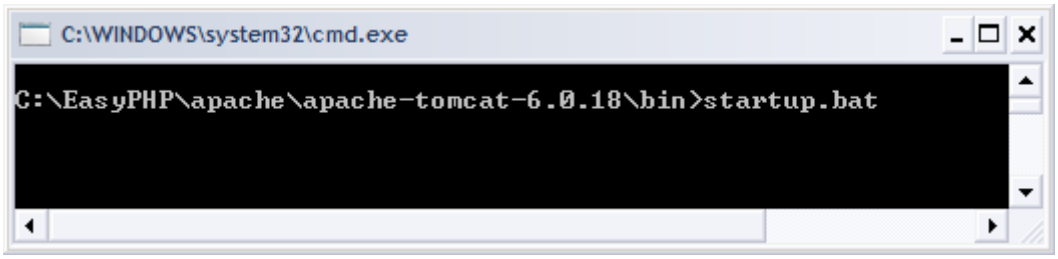
TOMCAT

INSTALLAZIONE

Procurarsi l'eseguibile d'installazione rilasciato da Apache dal sito del progetto Jakarta, *http://tomcat.apache.org.*
Tomcat richiede che sia installato sul sistema il **JRE** (*Java Runtime Enviroment*), scaricarlo dal sito Sun *http://java.sun.com/j2se* ed installarlo prima di procedere con l'installazione di Tomcat.
Indicare il pathname.
Start/Pannello di controllo/Prestazioni e manutenzione/Sistema, nella finestra *Proprietà del sistema* fare clic sulla scheda *Avanzate* e *Variabili d'ambiente*.



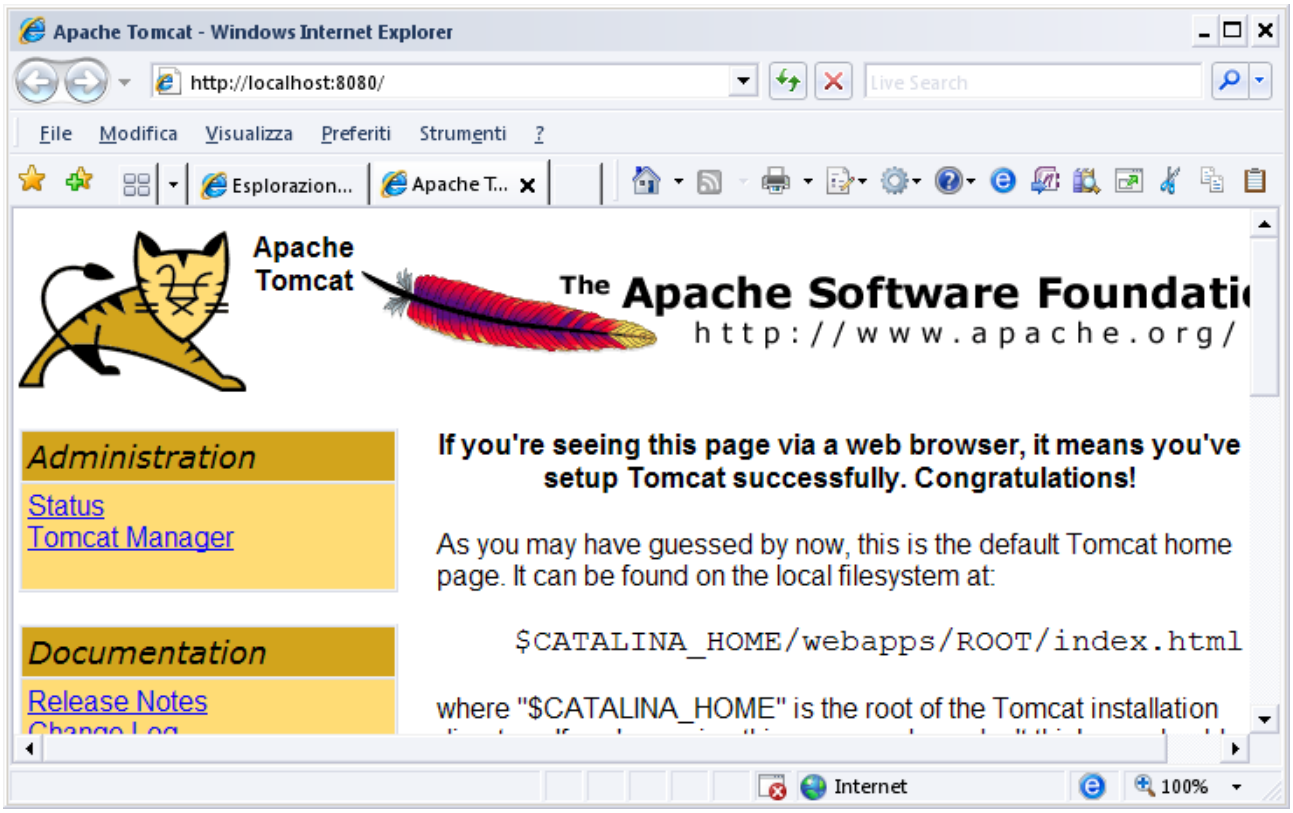
Tomcat rappresenta l'implementazione delle tecnologie Java Servlet e **JSP** (*Java Server Page*), crea l'ambiente all'interno del quale le servlet e le pagine JSP sono eseguite (server engine), s'integra con IIS e Apache ed estende le loro funzionalità per l'esecuzione delle servlet.
Il server Tomcat è associato alla porta 8080, questo significa che Tomcat può funzionare anche se è attivo IIS o Apache.
Per avviare il server.
Start/Esegui… cmd



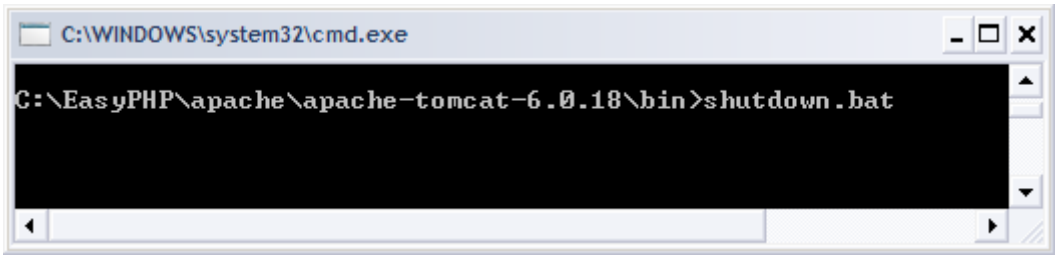
Per controllare il funzionamento del server basta inserire nella barra degli indirizzi del browser *http://localhost:8080 (http://127.0.0.1:8080)* essendo il localhost (127.0.0.1) il nome che identifica il server nel PC locale.
È aperta la pagina iniziale del server Tomcat.

- ✓ Administration
  - Status
  - Tomcat Manager gestisce le applicazioni create tramite il browser.
- ✓ Documentation
  - Release Notes
  - Change Log
  - Tomcat Documentation, la documentazione.
- ✓ Tomcat Online
- ✓ Miscellaneous

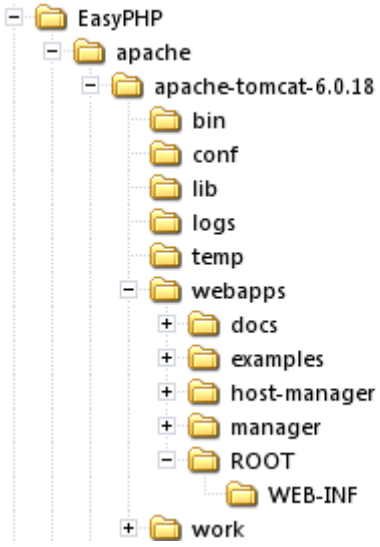
- Servlets Examples esempi di applicazioni web.
- JSP Examples esempi di JSP.



Per arrestare il server.
Start/Esegui… cmd



Per far sì che le applicazioni siano gestite correttamente da Tomcat, i file che le compongono devono essere correttamente collocati all'interno delle cartelle del server.



La cartella che contiene le applicazioni sarà la:
C:\EasyPHP\apache\apache-tomcat-6.0.18\webapps\ROOT.
All'interno della *ROOT* ci saranno tutti i file che compongono l'applicazione (file HTML, pagine JSP, immagini, suoni) opportunamente distribuiti in cartelle per una più facile gestione. Particolare importanza ricopre la cartella *WEB-INF* contenuta nella *ROOT*. Qui si ha, dentro la cartella *classes*, i file *.class* delle classi Java che comprendono il motore del sito web (tra cui le servlet) e, dentro la cartella *lib*, le opportune librerie in file di tipo jar. Le informazioni sulle applicazioni web del server sono memorizzate in un file speciale, in formato **XML** (*eXtensible Markup Language*), denominato *web.xml* che deve essere posizionato nella sotto cartella *WEB-INF*.
Il file *web.xml* si chiama deployment descriptor (descrittore del dispiegamento) ed è un descrittore dei componenti utilizzati nell'applicazione web, la sua struttura è la seguente.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--
Licensed to the Apache Software Foundation (ASF)
-->
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"version="2.5">
<display-name>Welcome to Tomcat</display-name>
<description>
Welcome to Tomcat
</description>
<servlet>
<servlet-name> Esempio di servlet</servlet-name>
<servlet-class> Prova </servlet-class>
</servlet>
<servlet-mapping>
<servlet-name> Esempio di servlet</servlet-name>
<url-pattern> Prova1 </url-pattern>
</servlet-mapping>
</web-app>
```

Dove.
<*web-app*> è il tag radice che indica l'elenco delle informazioni sulle applicazioni web.
<*servlet-name*> è il titolo identificativo della servlet.
<*servlet-class*> è il nome fisico del file *.class* contenente la servlet compilata.
<*url-pattern*> è l'URL da scrivere nella casella degli indirizzi del browser per eseguire la servlet.
Per ogni servlet bisogna inserire i due elementi: <*servlet*> e <*servlet-mapping*>
Supponendo di aver creato il file *Prova.class* e di averlo memorizzato nella sotto cartella *C:\EasyPHP\apache\apache-tomcat-6.0.18\webapps\ROOT/esempi/WEB-INF/classes* per eseguire la servlet basta inserire nella barra degli indirizzi del browser
http://localhost:8080/esempi/Prova1

JAVA SERVER PAGE

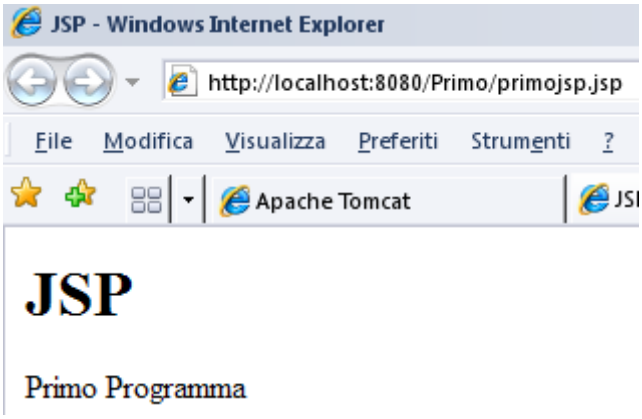
Furono presentate in occasione di Java One del 1998, una manifestazione organizzata dalla Sun a San Francisco, per la presentazione delle innovazioni legate a Java.
La tecnologia JSP è una soluzione multipiattaforma per realizzare pagine HTML dinamiche, dal lato server. Le pagine JSP sono un'evoluzione dei già collaudati servlet Java, create per separare i contenuti dalla loro presentazione: una pagina JSP si presenta, infatti, come un normale documento in linguaggio HTML, frammentato da sezioni di codice Java. Si potranno quindi modificare le parti sviluppate in Java lasciando inalterata la struttura HTML o viceversa.
L'aspetto della portabilità non deve essere visto esclusivamente sotto l'aspetto server: anche dal lato client questa tecnologia offre un completo svincolamento sia dal tipo di SO che dal tipo di browser (il linguaggio Java è infatti interpretato dal server sollevando il browser da questo compito, limitando la sua funzione a interprete di pagine HTML).
Indipendenza dalla piattaforma, le JSP permettono una vantaggiosa possibilità di riutilizzare il codice: si può infatti di includere nelle pagine sezioni di codice (java bean) che rendono molto più abbordabile l'implementazione di applicazioni anche complesse anche senza approfondite conoscenze di Java, rendendo inoltre molto più semplice la modifica e la manutenzione delle pagine stesse.
Le JSP possono essere invocate (richiamate) utilizzando due diversi metodi.

1. La richiesta può venire effettuata direttamente ad una pagina JSP, che grazie alle risorse messe a disposizione sul lato server, è in grado di elaborare i dati di ingresso per ottenere e restituire l'output voluto.
2. La richiesta può essere fatta in un primo tempo ad un servlet che, dopo l'elaborazione dei dati, incapsula adeguatamente i risultati e richiama la pagina JSP. Sarà poi quest'ultima che, in grado di prelevare i dati immagazzinati, produrrà l'output voluto.

Creare nella cartella *C:\EasyPHP\apache\apache-tomcat-6.0.18\webapps\ROOT* una cartella che conterrà il sito web e chiamarla *Primo*, infine creare il primo programma JSP.

```
<html>
<head>
<title>JSP</title>
</head>
<body>
<% out.println ("<h1>JSP</h1>");%>
<% out.println("Primo Programma");%>
</body>
</html>
```

e memorizzarlo come *primojsp.jsp*. A questo punto, digitare nel browser.



Come si può notare le sezioni racchiuse tra i tag <% e %> sono quelle che contengono le istruzioni in linguaggio Java. Da notare che i file per essere riconosciuti dal browser come pagine JSP devono essere salvati con estensione *.jsp*. La prima volta che volta che si effettua la richiesta di visualizzazione del file, quest'ultimo è compilato, creando un oggetto

servlet, che sarà archiviato in memoria (per servire le richieste successive); solo dopo questi passaggi l'output è mandato al browser, che potrà interpretarlo come fosse una semplice pagina HTML. Ad ogni richiesta successiva alla prima il server controlla se sulla pagina *.jsp* è stata effettuata qualche modifica, in caso negativo richiama il servlet già compilato, altrimenti si occupa di eseguire nuovamente la compilazione e memorizzare il nuovo servlet.
Gli script in JSP sono vere e proprie porzioni di codice inserite all'interno di pagine HTML che possono rappresentare: dichiarazioni, espressioni o scriptler.

Dichiarazioni
La sintassi per le dichiarazioni è la seguente.
<%! dichiarazione %>
Esempi.
<%//dichiarazione di una stringa %>
<% ! String stringa=new string("ciao a tutti") %>
<% // dichiarazione di una funzione che dati due numeri in ingresso
// restituisce la loro somma
%>
<% ! public int somma (int primo, int secondo){
return (primo + secondo);
}//somma
%>

Espressioni
La sintassi per le espressioni è la seguente.
<%= espressione %>
Esempi.
<%= somma (2,3)%>
Questa istruzione se inserita all'interno dei tag <BODY> e </BODY> stamperà a video l'output della funzione somma (in questo caso il numero 5, ottenuto dalla somma dei due valori in ingresso 2 e 3).

Direttive
1. *page*: modifica alcune impostazioni che influiscono sulla compilazione della pagina, attraverso la modifica degli attributi *language* (specifica il linguaggio da d utilizzare in fase di compilazione, esempio language="Java"), *import* (specifica quali package dovranno essere inclusi nella pagina, esempio: import="java.util.*"), *isThreadSafe* (indica se la pagina può servire più richieste contemporaneamente, esempio *isTreadSafe = "true*"), *info* (contiene delle informazioni riguardanti la pagina, consultabili grazie al metodo *Servlet.getServletInfo*, esempio *info="Prima prova*"), *errorPage* e *isErrorPage* (entrambi riguardanti la gestione delle pagine di errore, Esempio *<%@ page language="java" import="java.sql.*" %>*
2. *include*: modificando l'attributo file è possibile aggiungere dei file sorgenti aggiuntivi da compilare assieme alla pagina (possono per esempio essere file HTML o semplici file di testo).
3. *taglib*: permette di utilizzare una libreria di tag personalizzati (basta specificare l'URL della libreria e il prefisso da utilizzare per accedere ai tag, rispettivamente settando i parametri url e prefix).

Application
L'oggetto implicito *application* consente di accedere alle costanti dell'applicazione e di memorizzare oggetti a livello di applicazione e quindi accessibili da qualsiasi utente per un tempo che va dall'avvio del motore JSP alla sua chiusura, in pratica fino allo spegnimento

del server. Gli oggetti memorizzati nell'oggetto application come appena detto sono visibili da ogni utente e ogni pagina può modificarli.

Memorizzare i dati nell'oggetto application
Per memorizzare i dati all'interno dell'oggetto application è sufficiente utilizzare il metodo *setAttribute* specificando il nome dell'oggetto da memorizzare e una sua istanza. Per esempio, per memorizzare il numero di visite alla pagina è sufficiente fare *application.setAttribute("visite", "0").*

Leggere il contenuto di un oggetto contenuto in application
La lettura di un oggetto application precedentemente memorizzato è possibile grazie al metodo *getAttribute* che ha come unico ingresso il nome dell'oggetto application con cui si era memorizzato il dato da reperire. *application.getAttribute("visite")* restituisce l'oggetto corrispondente, in questo caso semplicemente il valore 0. Se non sono trovate corrispondenze con il nome è restituito il valore null. Anche l'oggetto *application* come *session* possiede il metodo *getAttributeNames()* che restituisce un oggetto di tipo enumerativo di stringhe contenente i nomi di tutti gli oggetti memorizzati nell'applicazione in esecuzione. Per rimuovere un oggetto si utilizza il metodo *removeAttribute("nomeoggetto").*
Per accedere alle costanti di applicazioni ci sono due metodi.
1. *getRealPath("\")* che restituisce (con quella stringa in ingresso) il percorso completo su cui è memorizzato il file.
2. *application.getServerInfo()* restituisce delle informazioni, riguardanti la versione, del motore JSP che si sta utilizzando per l'esecuzione della pagina.

La gestione degli errori nelle pagine JSP
Una delle caratteristiche di Java è quella di poter gestire le eccezioni, cioè tutti quelli eventi che non dovrebbero accadere in una situazione normale e che non sono causati da errori da parte del programmatore.

Errori al momento della compilazione
Questo tipo di errore si verifica al momento della prima richiesta, quando il codice JSP è tradotto in servlet. Generalmente sono causati da errori di compilazione ed il motore JSP, che effettua la traduzione, si arresta nel momento in cui trova l'errore ed invia al client richiedente una pagina di "Server Error" (o errore 500) con il dettaglio degli errori di compilazione.
Ecco cosa succede quando si richiede una pagina sintatticamente errata.

Errori al momento della richiesta
Sono causati da errori durante l'esecuzione della pagina e non in fase di compilazione. Si riferiscono all'esecuzione del contenuto della pagina o di qualche altro oggetto contenuto in essa. I programmatori java sono abituati ad intercettare le eccezioni innescate da alcuni tipi di errori, nelle pagine JSP questo non è più necessario perché la gestione dell'errore in caso di eccezioni è eseguita automaticamente dal servlet generato dalla pagina JSP. Il compito del programmatore si riduce al creare un file *.jsp* che si occupi di gestire l'errore e che permetta in qualche modo all'utente di tornare senza troppi problemi all'esecuzione dell'applicazione JSP.

Creazione di una pagina di errore
Una pagina di errore può essere vista come una normale pagina JSP in cui si specifica, tramite l'opportuno parametro della direttiva *page*, che si tratta del codice per gestire l'errore. Ecco un semplice esempio: *PaginaErrore.jsp*.
<HTML>
<BODY>

<%@ page isErrorPage = "true" %>
<CENTER>
*Siamo spiacenti, si è verificato un errore durante l'esecuzione:

*
<%= exception.getMessage()%>
</CENTER>
</BODY>
</HTML>
A parte la direttiva *page* il codice Java è composto da un'unica riga che utilizza l'oggetto *exception* (implicitamente contenuto in tutte le pagine di errore) richiamando l'oggetto *getMessage()* che restituisce il messaggio di errore.

Uso delle pagine di errore
Perché nelle pagine JSP sia utilizzata una determinata pagina di errore l'unica cosa da fare è inserire la seguente direttiva.
<% page errorPage = "PaginaErrore.jsp" %>
che come è facile capire specifica quale pagina di errore deve essere richiamata in caso di errore in fase di esecuzione.

Gestione delle stringhe
Java mette a disposizione degli sviluppatori una classe (inclusa nel package *java.lang*) chiamata *string*, che permette la gestione delle stringhe (da notare che da questa classe non è possibile derivare altre sottoclassi).
Gli oggetti di tipo *string* possono essere inizializzati in più modi, ma normalmente i costruttori più utilizzati sono due.
1. *public String ():* che costruisce una stringa vuota (perché sia allocata è necessario utilizzare l'istruzione *nomeStringa=new String [ ]*).
2. *public String (string value):* che costruisce una stringa contenente la stringa value.

I metodi implementati da questa classe più utilizzati sono i seguenti.
string.length (): restituisce il numero di caratteri che compongono la stringa escluso il terminatore.
string.charAt (int i): restituisce il carattere della stringa in posizione *i*.
string.concat (string str): restituisce una nuova stringa ottenuta dalla concatenazione delle stringa string con la stringa *str*.
string.equals (string str): confronta la stringa *string* con la stringa *str*, restituendo true se risultano uguali.
string.equals.IgnoreCase (string str): confronta la stringa *string* con la stringa *str* tenendo conto delle lettere minuscole e maiuscole.
string.compareTo (string str): confronta la stringa *string* con la stringa *str*, restituendo 0 se risultano uguali, un numero positivo se *str* precede in ordine alfabetico *string* e un numero negativo se *str* anticipa segue in ordine alfabetico *string*.

Gestione dei vettori
Java oltre all'implementazione di una classe per la gestione delle stringhe, ne fornisce una che permette di gestire i vettori (contenuta nel package *java.util*): la classe *Vector*. Un oggetto di tipo vettore è fornito di tre costruttori, e può quindi essere definito in altrettanti modi.
- ✓ *public Vector ():* costruisce un vettore vuoto (avente dimensione di default 10).
- ✓ *public Vector (int capacitàIniziale):* costruisce un vettore di capacità *capacitàIniziale*.
- ✓ *public Vector (int capacitàIniziale, int incremento):* costruisce un vettore di capacità iniziale *capacitàIniziale* avente come incremento di capacità pari a incremento (quando è aggiunto un nuovo elemento ad un vettore già pieno, infatti, la classe *Vector* alloca un nuovo vettore di capacità uguale alla precedente più un incremento, che di default è uguale a 1).

I metodi implementati da *Vector* più utilizzati sono i seguenti.

- ✓ *vector.elementAt (int i):* restituisce l'elemento in posizione *i.*
- ✓ *vector.firstElement ():* restituisce il primo elemento di *vector*.
- ✓ *vector.lastElement ():* restituisce l'ultimo elemento di vector.
- ✓ *vector.addElements (Object obj):* aggiunge *obj* dopo l'ultimo elemento del vettore (la classe *Object* è la radice da cui sono derivate tutte le altre classi Java).
- ✓ *vector.insertElementsAt (Object obj, int i):* inserisce nella posizione *i* di *vector* l'elemento *obj*.
- ✓ *vector.setElementsAt (Object obj, int i):* sostituisce l'elemento in posizione i con *obj*.
- ✓ *vector.removeElementAt (int i):* rimuove l'elemento in posizione i da *vector*.
- ✓ *vector.removeElement (Object obj):* se *vector* contiene l'elemento *obj*, esso sarà rimosso e il metodo restituirà il valore true, altrimenti sarà restituito il valore false.
- ✓ *vector.removeAllElements ():* rimuove tutti gli elementi del vettore.

La Classe Hashtable

La classe *Hashtable* (contenuta nel package *java.util* e derivata dalla classe astratta *Dictionary*) permette di memorizzare una collezione di oggetti (più specificatamente delle coppie oggetto-chiave) avente funzioni molto simili ai DB: le chiavi non replicabili permettono infatti un veloce accesso ai dati.
Sia le chiavi sia gli oggetti devono essere di tipo *Object*.
I metodi più importanti implementati da Hashtable sono i seguenti.

- ✓ *hashtable.size():* restituisce un intero contenente il numero di elementi inseriti in *hashtable*.
- ✓ *hashtable.isEmpty ():* restituisce un valore booleano che indica se il dizionario è vuoto o meno (il valore true segnala il dizionario vuoto).
- ✓ *hashtable.get (Object chiave):* restituisce l'oggetto cui è associata la chiave *chiave*, se la chiave non è trovata restituisce null.
- ✓ *hashtable.put (Object chiave, Object elemento):* inserisce in *hashtable* l'elemento *elemento* e gli associa la chiave *chiave*, se chiave è già stata inserita è sostituito l'oggetto a lei associato ed è restituito, altrimenti è restituito null.
- ✓ *hashtable.remove (Object chiave):* rimuove da *hashtable* l'elemento associato a chiave, se la chiave non esiste è restituito null.

Ad un oggetto di tipo *Hashtable* può essere associato un oggetto di tipo *Enumeration*, che permette di scorrere la collezione di oggetti, utilizzando i metodi seguenti.

- ✓ *enumeration.hasMoreElements ():* che restituisce il valore booleano true se ci sono ancora elementi da scorrere.
- ✓ *enumeration.nextEle():* si sposta all'elemento successivo.

FORM HTML

Il modo più semplice per recuperare i dati da un form o dalla *querystring* è quello di utilizzare l'istruzione: *Request.getParameter("nome_parametro")* che legge sia i parametri provenienti da un form con metodo *POST*, sia quelli inviati tramite *querystring* con metodo *get* o semplicemente concatenandoli all'URL della pagina. Il metodo cerca *"nome_parametro"* nell'elenco dei parametri contenuti nella richiesta e ne restituisce l'eventuale valore.
Se si riceve più di un parametro con lo stesso nome, si ricorre al metodo *getParameterValues("nome_parametro"),* dell'oggetto *request*. *GetParameterValues()* restituisce un vettore di stringhe con tutti i diversi valori del parametro di cui si specifica il nome. Per accedere al singolo valore è sufficiente aggiungere un *[n]* in questo modo: *request.getParameterValues("nome_parametro")[0]* per ottenere il primo valore *request.getParameterValues("nome_parametro")[1]* per il secondo e così via per tutti gli altri.
Se per esempio in un form chiediamo all'utente d'inserire gli indirizzi di tutti i siti che ha realizzato, è possibile ricevere informazioni di persone che hanno creato molti siti ma altri ancora agli inizi con poche pagine all'attivo. Creato il form con una quantità adeguata di

input di tipo testo, tutti con lo stesso nome, la lettura dei dati dalla pagina JSP si risolve in poche righe.

```
%< int i=0
while (request.getParameterValues("siti_realizzati")[i].length()==0)
{
   out.println(request.getParameterValues("siti_realizzati")[i]);
   i++;
}
%>
```

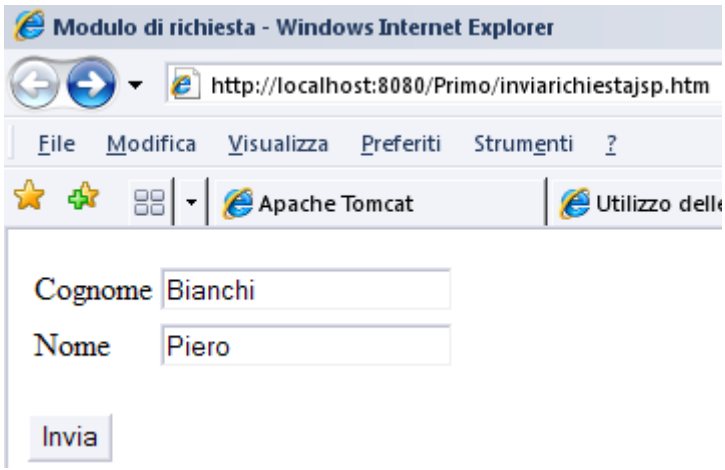
Se si vuole gestire anche la possibilità che le caselle di testo d'input non siano compilate in ordine, si deve gestire il tutto con un ciclo for.

```
<%
for (i=0; i<n; i++){
   if (request.getParameterValues("siti_realizzati")[i].length()==0)
   out.println(request.getParameterValues("siti_realizzati")[i]);
}
%>
```

dove *n* contiene il numero di caselle di testo presenti nel form.

Digitare il seguente file: *inviarichiestajsp.htm*

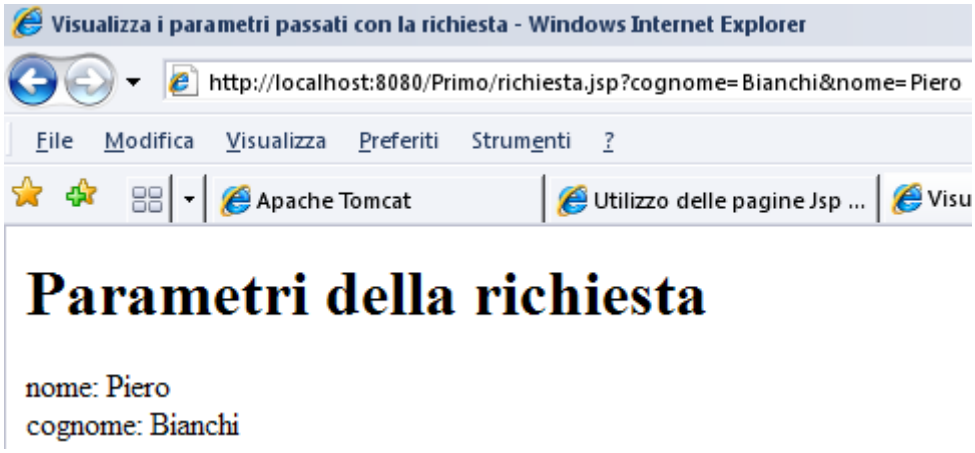
```
<html>
<head>
<title>Modulo di richiesta</title>
</head>
<body>
<form method="get" action="richiesta.jsp">
<table>
<tr><td>Cognome</td>
<td><input type=text name="cognome"></td></tr>
<tr><td>Nome</td>
<td><input type=text name="nome"></td></tr>
</table>
</br>
<input type="submit" value ="Invia">
</body>
</html>
```



Premere il pulsante *Invia*. Il browser si sposterà alla pagina *richiesta.jsp.*

```
<%@ page import="java.util.Enumeration"%>
<html>
<head>
```

```
<title>Visualizza i parametri passati con la richiesta</title>
</head>
<body>
<h1>Parametri della richiesta</h1>
<%
        Enumeration parametri=request.getParameterNames();
        while (parametri.hasMoreElements())
            {
                        String nome=(String)parametri.nextElement();
                        String valore=request.getParameter(nome);
%>
<%=nome + ":" %>
<%=valore + "</br>" %>
</tr>
<%      } %>
</body>
</html>
```



COOKIES

I cookies non sono altro che quell'insieme d'informazioni associate ad un particolare dominio che sono memorizzate sul client sotto forma di file di solo testo, in modo da poter riconoscere un particolare utente dell'applicazione. Con lo sviluppo di applicazioni sempre più complesse ed efficienti, l'utilizzo dei cookies è diventato indispensabile, come anche far capire agli utenti del sito che, se usati nel modo giusto, non fanno altro che migliorare la loro navigazione.
La prima fase è quella di scrivere un cookie, compito del metodo *addCookie()* dell'oggetto *response.addCookie(cookie)* dove il parametro d'ingresso *cookie* non è altro che una classe che ha alcuni metodi che permettono d'impostare le proprietà del cookie, ecco come.

```
Cookie mioCookie = new Cookie ("nome", "valore");
```

definisce un oggetto Cookie con il ripettivo nome e valore.

```
mioCookie.setPath("/");
//specifica il percorso che ha il privilegio di scrittura e lettura del cookie
//se omesso è inteso il percorso corrente
mioCookie.setAge(secondiDiVita);
//imposta il periodo di vita, espresso in secondi
mioCookie.setSecure(false);
//indica se il cookie va trasmesso solo su un protocollo sicuro, protetto cioè da crittografia
response.addCookie(mioCookie);
//scrive il cookie
```

A questo punto *mioCookie* è scritto sul client. Per leggerlo l'operazione è leggermente più complicata. L'oggetto *request* dispone infatti di un metodo *getCookies()* che restituisce un array di oggetti cookie trovati nel client. Una volta ottenuto l'array, è quindi necessario passare in rassegna tutti i suoi elementi fino a trovare quello che interessa.

```
Cookie mioCookie = null;
//definisce in cookie che sarà letto
Cookie[] cookiesUtente = request.getCookies();
//definisce l'array di cookie e legge quelli dell'utente
int indice = 0;
//indice per la gestione del ciclo
while (indice < cookiesUtente.length) {
//esegue il ciclo fino a quando ci sono elementi in cookieUtente
if (cookiesUtente[indice].getName().equals("nomeMioCookie");
break;
//se trova un cookie con il nome che si sta cercando esce dal ciclo
  indice++;
}//while
```

A questo punto controllando il valore dell'indice si è in grado di capire se si è trovato il cookie.

```
If (indice < cookiesUtente.length)
//il cookie è stato trovato ed è messo nell'oggetto mioCookie
   mioCookie = cookiesUtente[indice];
else
   mioCookie = null;
//il cookie non è stato trovato
```

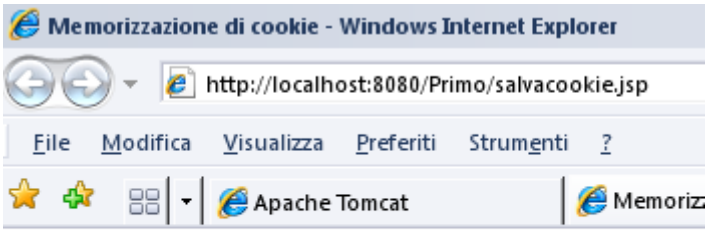
Ora, se il cookie è stato trovato, è possibile accedere ai suoi dati con i metodi dell'oggetto stesso, principalmente *getValue()* per ottenere il valore contenuto.

```
Valore = mioCookie.getValue();
```

Altri metodi sono.
getName(): restituisce il nome.
getAge(): restituisce il periodo di vita in secondi.
getPath(): restituisce il percorso che ha il permesso di lettura/scrittura.
getSecure(): restituisce un booleano che dice se il cookie è sicuro oppure no.

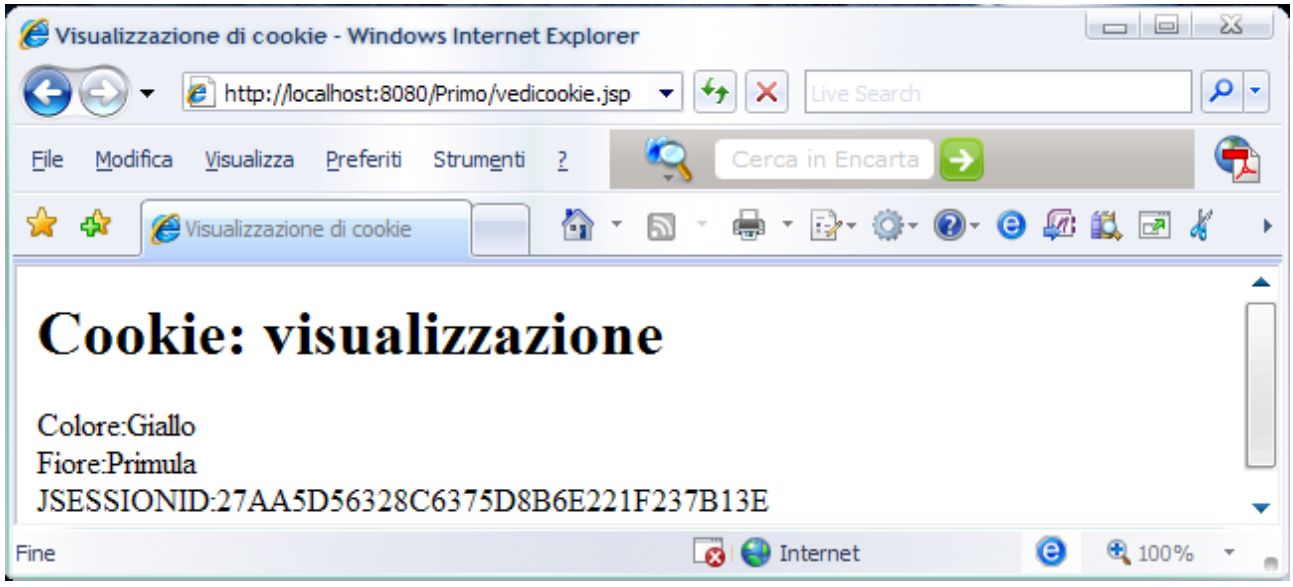
salvacookie.jsp

```
<html>
<head>
<title>Memorizzazione di cookie</title>
</head>
<body>
<h1>Cookie: memorizzazione</h1>
<%
        response.addCookie(new Cookie("Colore","Giallo"));
        response.addCookie(new Cookie("Fiore","Primula"));
%>
</body>
</html>
```



vedicookie.jsp

```
<html>
<head>
<title>Visualizzazione di cookie</title>
</head>
<body>
<h1>Cookie: visualizzazione</h1>
<%
        Cookie[]Cookies=request.getCookies();
        if (Cookies==null)
                { out.print("Non ci sono cookie"); }
        else {
                for (int i=0;i<Cookies.length;i++)
                                        {
                                        out.print(Cookies[i].getName()+ ":");
                                        out.print(Cookies[i].getValue()+ "</br>");
                                                }
                }
%<>
</body>
</html>
```



SESSIONI

Una delle funzionalità più richieste per un'applicazione web è mantenere le informazioni di un utente lungo tutto il tempo della sua visita al sito. Questo problema è risolto dall'oggetto implicito *session* che gestisce le informazioni a livello di sessione, relative ad un singolo

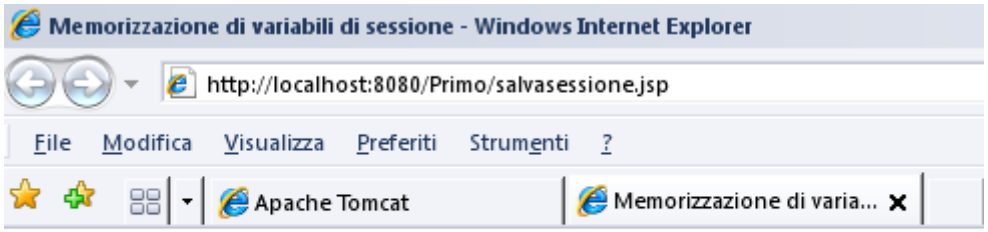
utente a partire dal suo ingresso alla sua uscita con la chiusura della finestra del browser. È possibile quindi creare applicazioni che riconoscono l'utente nelle varie pagine del sito, che tengono traccia delle sue scelte e dei suoi dati. È importante sapere che le sessioni sono memorizzate sul server e non con dei cookies che devono però essere abilitati per poter memorizzare il così detto *SessionID* che consente di riconoscere il browser e quindi l'utente nelle fasi successive. I dati di sessione sono quindi riferiti e riservati ad un utente cui è creata un'istanza dell'oggetto *session* e non possono essere utilizzati da sessioni di altri utenti.

Memorizzare i dati nell'oggetto Session
Per memorizzare i dati all'interno dell'oggetto *session* è sufficiente utilizzare il metodo *setAttribute* specificando il nome dell'oggetto da memorizzare e una sua istanza. Per esempio, per memorizzare il nome dell'utente al suo ingresso alla pagina ed averlo a disposizione i seguito e sufficiente fare *session.setAttribute("nomeUtente", nome)* a patto che nome sia un oggetto di tipo stringa che contenga il nome dell'utente.

Leggere il contenuto di una variabile di sessione
La lettura di una variabile di sessione precedentemente memorizzata è possibile grazie al metodo *getAttribute* che ha come unico ingresso il nome della variabile di sessione con cui avevamo memorizzato il dato che interessa reperire. *Session.getAttribute("nomeUtente")* restituisce il nome dell'utente memorizzato; se non sono trovate corrispondenza con il nome dato in ingresso, restituisce *null*.
getAttributeNames() restituisce un oggetto di tipo enumerativo di stringhe contenente i nomi di tutti gli oggetti memorizzati nella sessione corrente.
getCreationTime() restituisce il tempo (in millisecondi dalla mezzanotte del primo gennaio del 1970) di quando è stata creata la sessione.
getId() restituisce una stringa contenente il *sessionID* che permette d'identificare univocamente una sessione.
getLastAccesedTime() restituisce il tempo (in millisecondi dalla mezzanotte del primo gennaio del 1970) dall'utima richiesta associata alla sessione corrente.
getMaxInactiveInterval() restituisce un valore intero che corrisponde all'intervallo massimo di tempo tra una richiesta dell'utente ad un'altra della stessa sessione.
removeAttribute(nome_attributo) rimuove l'oggetto dal nome specificato dalla sessione corrente.

salvasessione.jsp

```
<html>
<head>
<title>Memorizzazione di variabili di sessione</title>
</head>
<body>
<h1>Sessione: memorizzazione variabili</h1>
<%
        String AttrNome1="cognome";
        String AttrValore1="Rossi";
        String AttrNome2="nome";
        String AttrValore2="Piero";
        session.setAttribute(AttrNome1,AttrValore1);
   session.setAttribute(AttrNome2,AttrValore2);
%>
</body>
</html>
```



vedisessione.jsp

```
<%@ page import="java.util.Enumeration"%>
<html>
<head>
<title>Visualizzazione delle variabili di sessione</title>
</head>
<body>
<h1>Sessione: visualizzazione variabili</h1>
<p>ID di sessione:
<%=request.getRequestedSessionId()%>
</p>
<p>Valori di sessione</p>
<%
	Enumeration valoriSessione=session.getAttributeNames();
	while (valoriSessione.hasMoreElements())
		{
			String nome = (String)valoriSessione.nextElement();
			String valore = (String)session.getAttribute(nome);
%>
<%=nome+ ":" %>
<%=valore+ "</br>"%>
<%}%>
</body>
</html>
```

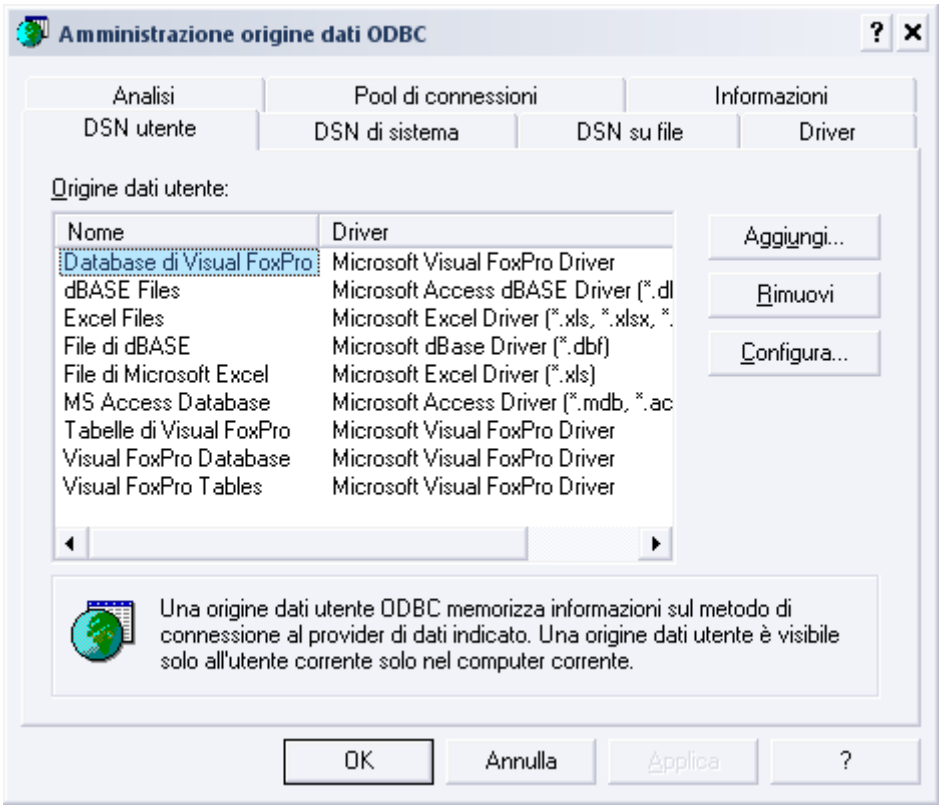
JSP E IL RDBMS ACCESS

Il driver per DB è un software standard che consente di trasferire i dati presenti in un DB: quando un'applicazione vuole accedere al DB creato con un altro applicativo, deve utilizzare il driver per quel DB.
JDBC (*Java Database Connectivity*) è un'interfaccia completamente Java utilizzata per eseguire istruzioni SQL. L'implementazione più utilizzata di questa interfaccia è quella detta bridge JDBC - ODBC. In questo caso il driver JDBC funge da ponte, per accedere al DB attraverso driver ODBC, che deve essere presente e configurato sul server. Esistono però altre implementazioni che differiscono per il modo di operare delle stesse.

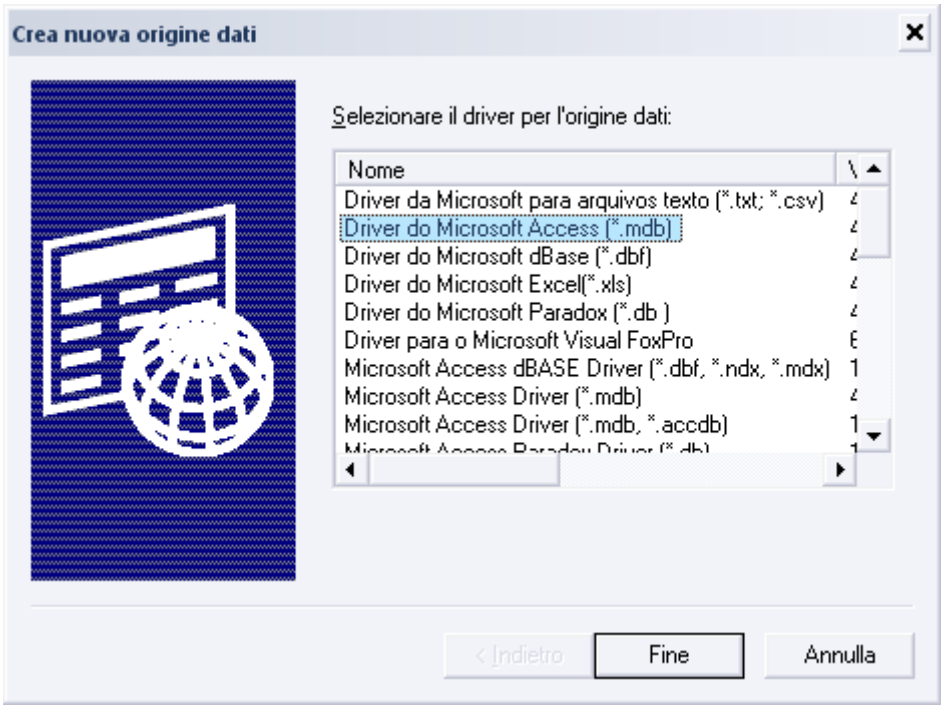
Driver API – nativa
Questa tecnica s'interfaccia direttamente con i DB commerciali, convertendo i comandi JDBC in chiamate specifiche del DBMS, il sistema di gestione di basi di dati.

Driver puro Java
È un driver puro Java che comunica direttamente con il DB, convertendo i comandi JDBC nel protocollo del motore di DB. Questa è senza dubbio la soluzione migliore perché non richiede nessuna traduzione aggiuntiva.

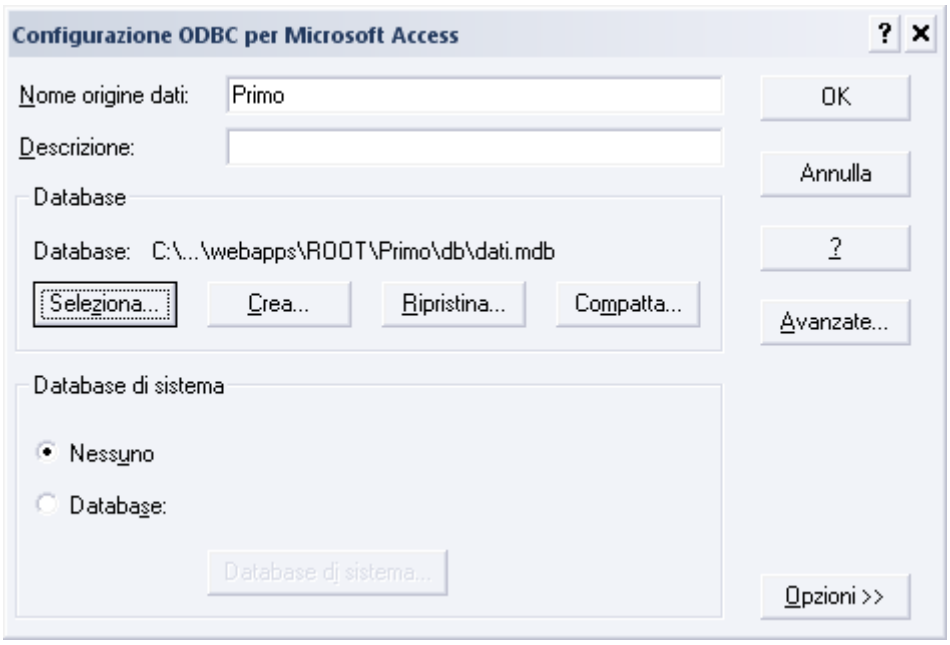
Driver bridge JDBC – ODBC
È il più diffuso e utilizzato anche nelle applicazione puramente Java, è il più semplice e soprattutto è disponibile sin dalla prima versione del JDK. Questa interfaccia necessita di un driver ODBC configurato. ODBC è l'interfaccia software standard in ambiente Windows che consente ai programmatori di scrivere in modo semplice le applicazioni per connettersi ai DB.
Per visualizzare le proprietà di ODBC in Windows XP e per fissarne le impostazioni occorre fare le seguenti operazioni.
Start/Tutti i programmi/Strumenti di amministrazione/Origine dati (ODBC), si aprirà la seguente finestra.



Definire una nuova origine dati per un driver installato, nella finestra sceglire il pulsante *Aggiungi*… Si aprirà una nuova finestra.



Selezionare *Microsoft Access Driver (*.mdb)* e clicchare su *Fine*.
Si aprirà un'ultima finestra.



Impostare il nome dell'origine dati (*Primo*), la descrizione (facoltativa) e il percorso del DB fisico a cui deve far riferimento. Per impostare quest'ultimo parametro clicchare su *Seleziona*… e selezionare il pathname corretto che porta al DB che sarà:
C:\EasyPHP\apache\apache-tomcat-6.0.18\webapps\ROOT\Primo\db\dati.mdb
Quando si clicca su *OK* il DB sarà presente nella lista delle *Origini Dati Utente*.

Una volta configurato il driver ODBC, la prima operazione è la lettura dei dati contenuti nel DB. L'operazione per leggere i dati da una tabella di un DB è suddivisa in quattro fasi.

1. Apertura connessione al DB.
2. Esecuzione istruzione.

3. Elaborazione risultato.
4. Chiusura connessione.

Ecco il codice che riporta in pratica queste fasi.

```
<%
//deve essere importata il package java.sql.* per eseguire le istruzioni SQL
@ page language="java" import="java.sql.*"
connection dbconn = null;
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
//carica il file di classe del driver per il collegamento al DB con il ponte ODBC
dbconn = DriverManager.getConnection("jdbc:odbc:miobd","admin","pwd");
//Apertura connessione con il DB "miodb"
Statement statement = dbconn.createStatement();
ResultSet rs = statement.executeQuery("SELECT dato FROM tab1");
//Esecuzione istruzione SQL aquesto punto l'oggetto rs contiene il set di risultati ottenuti
//dall'istruzione SELECT sul DB. Ora i singoli campi di ogni record possono essere letti con
//opportuni metodi.
while (rs.next()){
   //ottiene il dato
   dat = rs.getInt("dato");
   //stampa a video
   out.println(dat);
}//while
//Elaborazione risultati
dbconn.close();
//chiude la connessione
%>
```

In questo esempio sono stati prelevati dalla tabella *tab1* del DB *miodb* tutti i valori del campo dato. I metodi di lettura dei campi dell'oggetto *ResultSet* si differenziano in base al tipo campo. La tabella elenca i tipi di variabile del DB e i rispettivi metodi da utilizzare.

Metodo	Tipo del campo
getInt()	Numerico intero
getFloat()	Numerico con virgola
getByte()	Numerico byte
getlong()	Numerico lungo
getString()	Stringa
getBoolean()	Buleano (si/no)
getDate()	Data

Per muovere in avanti il cursore *resultSet* si utilizza la il metodo *next()* dell'oggetto stesso, che restituisce un valore booleano che indica se il cursore è arrivato al termine dalla tabella. È sufficiente creare un ciclo *while(rs.next())* per scorrere senza ulteriori istruzioni i records della base di dati.

Se è necessario aggiornare, inserire, modificare o eliminare records di un DB, il codice varia leggermente, infatti l'oggetto *ResultSet*, non è più necessario visto che l'operazione non restituisce più un riferimento ad una tabella creata con un'istruzione *select*. È sufficiente richiamare il metodo *executeUpdate()* dell'oggetto *Statement* precedentemente definito, e dare come ingresso la query SQL (Insert, update, create table). Questo metodo restituisce un valore intero che è uguale a 1 se tutto è stato eseguito con successo. È bene quindi effettuare un controllo sul valore restituito in modo da riuscire a capire se la modifica del DB sia effettivamente riuscita.

L'esempio riportato aggiunge al DB un record il cui campo è contenuto in una varibile dalle pagina JSP, che può quindi provenire da un form o da un'elaborazione dati precedente.

```
<html>
<head>
<title>Esempio inserimento riga nel DataBase</title>
</head><body>
<%- deve essere importata il package java.sql.* per eseguire le istruzioni SQL -%>
<%@ page language="java" import="java.sql.*" %>
<%
   int valore = 25; //valore di esempio
   int esito;       //esito aggiornamento
   Connection conn = null;
   //carica il file di classe del driver per il ponte Odbc
   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
   //crea la connessione con l'origine dati
   conn = DriverManager.getConnection("jdbc:odbc:miodb","","");
   //crea lo statement
   Statement st = conn.createStatement();
   //esegue l'aggiornameto o l'inserimento
   esito = st.executeUpdate("INSERT INTO tab1 (dati) values ("+valore+")"))
   //se esito è uguale a 1 tutto è andato bene
   if (esito == 1)
      out.println("inserimento eseguito correttamente");
   else
      out.println("inserimento non eseguito");
   rs.close();
   conn.close();
%>
</body>
</html>
```

Nell'esempio è stato inserito nella tabella *tab1*, nel campo *dati*, il valore della variabile *valore* che era stata definita e inizializzata precedentemente, ma che in pratica proverrà quasi sempre da un form di inserimento dati. Ciò è stato possibile con una semplice operazione di concatenazione di stringhe (per farlo si usa +) per creare la query SQL che si occupa dell'aggiornamento. Si capisce che le atre operazioni, l'eliminazione e l'aggiornamento, sono eseguite nello stesso modo, scrivendo le adeguate istruzioni SQL.

Visualizzare i dati della tabella elenco del DB dati

Si usa *dati.mdb*, con all'interno una tabella *elenco*, contenente due campi: *città* e *CAP*. Salvare il file come *dati.mdb* nella cartella *Primo/db.*

visualizzaDB.jsp

```
<%@ page language="java" %>
<%@ page import="java.sql.*" %>
<%@ page contentType="text/html; charset=ISO-8859-5" %>
<html>
<head>
<title>Visualizzazione dei dati della tabella elenco del DB dati</title>
</head>
<body>
<h1>Visualizzazione dati</h1>
<%
Connection conn = null;
Statement stmt = null;
ResultSet rs = null;
```

```
String url = "jdbc:odbc:Primo";
try
{
   /*chiamata al driver jdbc-odbc, apertura connessione al database*/
   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
   conn = DriverManager.getConnection(url, "", "");
   /*per operare sui DB si deve creare l'oggetto*/
   stmt = conn.createStatement();
}
catch(ClassNotFoundException ex) {out.println("Driver non trovato.");}
catch(SQLException ex) {out.println("Connessione fallita.");}
out.println ("Connessione eseguita.");
%>
<br></br>
<table border="1">
<tr><th>Citta</th><th>CAP</th></tr>
<%
/*per eseguire interrogazioni*/
String query = "SELECT * FROM elenco";
try
{
   rs=stmt.executeQuery(query);
   while (rs.next())
        {
              out.println ("<tr>");
              out.println ("<td>"+rs.getString("citta")+"</td>");
              out.println ("<td>"+rs.getString("CAP")+"</td>");
              out.println ("</tr>");
        }
   /*chiudo la connessione*/
   stmt.close();
   conn.close();
}
catch(SQLException ex) {out.println("Eccezione SQL.");}
%>
</table>
</body>
</html>
```

INSERIRE DATI NELLA TABELLA ELENCO DEL DB DATI

Questa pagina HTML è utilizzata per l'inserimento di una nuova città ed un nuovo CAP nella tabella elenco del DB *dati*.

inviarichiestaDBjsp.htm

```
<html>
<head>
<title>Inserimento dati nella tabella elenco del DB dati</title>
</head>
<body>
<h1>Inserimento dati</h1>
<form method="get" action="inserimentoDB.jsp">
<table>
<tr><td>Città</td>
<td><input type=text name="citta"></td></tr>
<tr><td>CAP</td>
```

```
<td><input type=text name="CAP"></td></tr>
</table>
</br>
<input type="submit" value ="Inserisci nel DB">
</body>
</html>
```

Si riceve in input la citta e il CAP dalla pagina HTML, quindi s'inserisce il tutto nella tabella elenco del DB *dati*.
inserimentoDB.jsp

```
<%@ page language="java" %>
<%@ page import="java.sql.*" %>
<%@ page contentType="text/html; charset=ISO-8859-5" %>
<%
/*leggo i valori ricevuti dal form HTML*/
String citta=new String (request.getParameter("citta"));
String CAP=new String (request.getParameter("CAP"));
%>
<html>
<head>
<title>Inserimento dati nella tabella elenco del DB dati</title>
</head>
<body>
<h1>Inserimento dati</h1>
<%
        if (citta.equals(""))
        {
%>
<p>La citta deve essere specificata</p>
<a href="inviarichiestaDBjsp.htm">Ripeti</a>
<%
      }
      else
        {
                Connection conn = null;
                Statement stmt = null;
                String url = "jdbc:odbc:Primo";
        try
        {
                /*chiamata al driver jdbc-odbc, apertura connessione al database*/
                Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                conn = DriverManager.getConnection(url, "", "");
                /*per operare sui DB si deve creare l'oggetto*/
                stmt = conn.createStatement();
        }
        catch(ClassNotFoundException ex) {out.println("Driver non trovato.");}
        catch(SQLException ex) {out.println("Connessione fallita.");}
        /*per eseguire l'inserimento*/
        String query = new String ("INSERT INTO elenco VALUES ('" + citta + "','" + CAP
+"')" );
        try
        {
                // esegui update del database
                stmt.executeUpdate(query);
```

```
            out.println("Nuovo citta registrata: "+citta+" "+CAP);
            /*chiudo la connessione*/
            stmt.close();
            conn.close();
        }
        catch(SQLException ex) {out.println("Eccezione SQL.");}
        }
%>
</body>
</html>
```

RICERCARE DATI NELLA TABELLA ELENCO DEL DB DATI

Questa pagina HTML è utilizzata la ricerca di una città nella tabella elenco del DB *dati*.

ricercaDBjsp.htm

```
<html>
<head>
<title>Ricerca dati nella tabella elenco del DB dati</title>
</head>
<body>
<h1>Ricerca dati</h1>
<form method="get" action="ricercaDBjsp.jsp">
<table>
<tr><td>Città</td>
<td><input type=text name="citta"></td></tr>
<tr><td>CAP</td>
<td><input type=text name="CAP"></td></tr>
</table>
</br>
<input type="submit" value ="Ricerca nel DB">
</body>
</html>
```

Si riceve in input la citta dalla pagina HTML, quindi si ricerca il tutto nella tabella elenco del DB *dati*.

ricercaDBjsp.jsp

```
<%@ page language="java" %>
<%@ page import="java.sql.*" %>
<%@ page contentType="text/html; charset=ISO-8859-5" %>
<%
/*leggo i valori ricevuti dal form HTML*/
String citta=new String (request.getParameter("citta"));
String CAP=new String (request.getParameter("CAP"));
%>
<html>
<head>
<title>Ricerca dati nella tabella elenco del DB dati</title>
</head>
<body>
<h1>Ricerca dati</h1>
<%
        if (citta.equals(""))
        {
%>
<p>La citta deve essere specificata</p>
```

```
<a href="ricercaDBjsp.htm">Ripeti</a>
<%
      }
      else
        {
              Connection conn = null;
              Statement stmt = null;
              ResultSet rs = null;
              String url = "jdbc:odbc:Primo";
        try
        {
              /*chiamata al driver jdbc-odbc, apertura connessione al database*/
              Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
              conn = DriverManager.getConnection(url, "", "");
              /*per operare sui DB si deve creare l'oggetto*/
              stmt = conn.createStatement();
        }
        catch(ClassNotFoundException ex) {out.println("Driver non trovato.");}
        catch(SQLException ex) {out.println("Connessione fallita.");}
%>
<br></br>
<table border="1">
<tr><th>Citta</th><th>CAP</th></tr>
<%
/*per eseguire interrogazioni*/
String query = "SELECT * FROM elenco WHERE citta = '"+citta+"'";
try
{
   rs=stmt.executeQuery(query);
   while (rs.next())
        {
              out.println ("<tr>");
              out.println ("<td>"+rs.getString("citta")+"</td>");
              out.println ("<td>"+rs.getString("CAP")+"</td>");
              out.println ("</tr>");
        }
   /*chiudo la connessione*/
   stmt.close();
   conn.close();
}
catch(SQLException ex) {out.println("Eccezione SQL.");}
}
%>
</table>
</body>
</html>
```

GESTIRE UN SONDAGGIO

L'applicazione pratica con jsp che è analizzata consiste nella creazione di un sondaggio, in cui gli utenti di un sito possono rispondere ad una domanda scegliendo tra le risposte proposte ed eventualmente visualizzare i risultati parziali delle votazioni. Come per tutti i sondaggi realizzati per un sito Internet è previsto il controllo del doppio voto, reso impossibile dalla memorizzazione degli indirizzi IP degli utenti che votano. L'applicazione è costituita da quattro file.

1. *sondaggio.htm*, contenente il form da cui l'utente risponde alla domanda.
2. *vota.jsp*, che esegue la votazione proveniente dal form.
3. *risultati.jsp*, che visualizza i risultati parziali delle votazioni.
4. *sondaggio.mdb*, il DB che gestisce tutto il sondaggio. Questo è costituito da due tabelle. La prima *frequenza* contiene un campo chiamato *risp* di tipo byte che indica il numero della risposta, e un campo *frequenze* di tipo int che indica il numero di volte che si è votato per la rispettiva risposta. Nell'esempio, con quattro possibili risposte al sondaggio, la tabella è stata completata inserendo nelle prime quattro righe del campo *risp* i valori 0-1-2-3 e nel campo *frequenze*, ovviamente, i valori 0-0-0-0. La seconda tabella si chiama *ip_tab* e contiene in un campo *IP* di tipo testo la lista degli indirizzi IP di tutti gli utenti che hanno effettuato il voto.

sondaggio.htm

```
<html>
<head>
<title>Sondaggio</title>
</head>
<body>
<b><big>Domanda:</big></b>
<form action="vota.jsp">
<input name="risposta" type="radio" value="0">Risposta 1<br>
<input name="risposta" type="radio" value="1">Risposta 2<br>
<input name="risposta" type="radio" value="2">Risposta 3<br>
<input name="risposta" type="radio" value="3">Risposta 4<br>
<input type="submit" value="vota">
</form>
<a href="risultati.jsp">Visualizza i risultati parziali</a>
</font>
</body>
</html>
```

Il codice contenuto nei tag body del file potrà essere copiato in qualsiasi parte di un qualsiasi documento HTML, e dovrà essere personalizzato inserendo tanti radio button quante sono le possibili risposte al quesito e incrementando sempre di uno il valore dell'input eventualmente aggiunto.

vota.jsp

Questo file esegue in pratica la votazione, aggiornando il DB dopo aver fatto il controllo sull'IP dell'utente. Non deve essere necessariamente modificato. Al limite si possono personalizzare i messaggi di eseguita o fallita votazione.

```
<html>
<head>
<title>Sondaggio</title>
</head>
<body>
<font face="verdana" color="#3300ff" size="2">
<% page errorPage = "PaginaErrore.jsp" %>
<%@ page language="java" import="java.sql.*" %>
<%
   Connection conn = null;
   //carica il file di classe del driver
   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
   //crea la connessione con l'origine dati
   conn = DriverManager.getConnection("jdbc:odbc:sondaggio","","");
   //crea lo statement
   Statement st = conn.createStatement();
   //legge il parametro contenente la risposta
```

```
   String preferenza = new String (request.getParameter("risposta"));
   String ip = new String(request.getRemoteAddr());
   ResultSet rs = st.executeQuery("SELECT ip from ip_tab WHERE ip LIKE '"+ip+"'");
   if(!rs.next()){
      //l'ip non ha mai votato
            //esito dell'aggiormanento al Data base
            int esito;
      //crea la tringa SQL per l'aggiornamento
      String stringaSql = new String ("INSERT INTO ip_tab (ip) VALUES ('"+ip+"')");
      st.executeUpdate(stringaSql);
            //crea la tringa SQL per l'aggiornamento
            stringaSql = "UPDATE frequenze SET frequenza=frequenza+1 WHERE risp
LIKE '"+preferenza+"'";
      esito = st.executeUpdate(stringaSql);
            //controlla che tutto sia andato bene
     if (esito==1)
               out.println("Votazione eseguita con successo");
           else
               out.println("Errore, non  stato possibile eseguire la votazione");
   }//if
   else{ //l'utente ha gia votato
      out.println("Spicenti, hai gia votato!<br>");
      out.println("Per ragioni di credibilità del sondaggio<br>");
      out.println("ogni visitatore può votare una sola volta<br><br>");
   }//else
   st.close();
   conn.close();
 %>
<br><br>
<a href="risultati.jsp">Visualizza i risultati parziali</a>
</font>
</body>
</html>
```

risultati.jsp

Questo è il file che esegue il compito più complicato. Deve estrarre dal DB il numero di utenti che hanno votato in totale e per ogni singola risposta, calcolare la percentuale e stampare una breve tabella di riepilogo e un grafico che riassume visivamente l'andamento del sondaggio. Nel file deve essere modificato il numero di possibili risposte (riga 20) e il valore delle stringhe contenenti le risposte (righe 25-28). Tutto il resto del codice può rimanere invariato.

```
<html>
<head>
<title>Risultati del sondaggio</title>
</head>
<body bgcolor=#FFFF99><center>
<b><big>Domanda del sondaggio</b></big><br><br>
<% page errorPage = "PaginaErrore.jsp" %>
<%@ page language="java" import="java.sql.*" %>
<%
Connection conn = null;
   //carica il file di classe del driver
   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
   //crea la connessione con l'origine dati
   conn = DriverManager.getConnection("jdbc:odbc:sondaggio","","");
```

```
    //crea lo statement
    Statement st = conn.createStatement();
    /* --- DEFINIZIONE DI VARIABILI --- */
    int numRisp = 4; //numero possibili risposte
    int votiTot; //numero totale dei voti
    float percentuale; //percentuale per ogni risposta
    int frequenze[] = new int[numRisp]; //numero di voti per ogni risposta
    String risposte[] = new String[numRisp]; //array di stringhe con le risposte
       risposte[0]="Risposta 1";
       risposte[1]="Risposta 2";
       risposte[2]="Risposta 3";
       risposte[3]="Risposta 4";
    //crea il recordset -> calcolo dei voti totali
    ResultSet rs = st.executeQuery("SELECT sum(frequenza) as tot from
                                frequenze");
    rs.next();
    votiTot = rs.getInt("tot");
    out.println("Totale dei voti: "+votiTot+"<br>");
    //calcolo dei voti per ogni singola risposta e stampa
    for (int i=0; i<numRisp; i++){
        rs=st.executeQuery("SELECT sum(frequenza) as tot from frequenze WHERE
                        risp = "+i);
        rs.next();
        frequenze[i]=rs.getInt("tot");
        out.println(risposte[i]+": "+frequenze[i]+" voti<br>");
    }//for
    %>
    <br><br>
    <table align=center border=0>
    <%//creazione del grafico dei voti
    for (int i=0; i<numRisp; i++){
        //calcolo della percentuale
        percentuale = (float)frequenze[i]/votiTot*100;
        out.println("<tr valign='center' align='left'><td>");
        out.println(risposte[i]+"</td><td>");
        out.println("<img src='blu.gif' width="+percentuale*5+" height=20>");
        out.println((int)percentuale+"%</td>");
    }//for
%>
</tr>
</table>
<%
st.close();
rs.close();
conn.close();
 %>
</body>
</html>
```

Il risultato a video della pagina *risultati.jsp* risulterà così.

JSP E IL RDBMS MYSQL

Se si volesse utilizzare un database MySQL, il context da inserire nel file *server.xml* è così strutturato.

```
<Context path="/DBTest" docBase="DBTest" debug="5" reloadable="true"
crossContext="true">
<Resource name="jdbc/TestDB" auth="Container" type="javax.sql.DataSource"/>
<ResourceParams name="jdbc/TestDB">
<parameter>
<name>factory</name>
<value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
</parameter>
<parameter>
<name>username</name>
<value>admin</value>
</parameter>
<parameter>
<name>password</name>
<value>admin</value>
</parameter>
<parameter>
<name>driverClassName</name>
<value>com.mysql.jdbc.Driver</value>
</parameter>
<parameter>
<name>url</name>
<value>jdbc:mysql://localhost:3306/javatest?autoReconnect=true</value>
</parameter>
</ResourceParams>
</Context>
```

La classe del driver per la connessione al DB è: *com.mysql.jdbc.Driver*.
L'URL del DB è definito con una diversa sintassi:
jdbc:mysql://localhost:3306/javatest?autoReconnect=true

SERVLET

Un servlet non è altro che un'applicazione Java in esecuzione su una **JVM** (*Java Virtual Machine*) residente su un server. Questa applicazione tipicamente esegue una serie di operazioni che producono in output un codice HTML che sarà poi inviato direttamente al client per essere interpretato da un qualsiasi browser che non necessita di funzionalità aggiuntive.
A differenza di uno script CGI i servlet sono caricati solo una volta, al momento della prima richiesta, e rimangono residenti in memoria pronti per servire le richieste fino a quando non sono chiusi, con ovvi vantaggi in fatto di prestazioni (risultano infatti un po' più lenti in caricamento, ma una volta "aperti" le successive richieste sono evase molto velocemente).
Fino a questo momento si è visto come è possibile inserire il codice Java all'interno dei file JSP per dare dinamicità alle pagine. Gli esempi visti sono molto semplici, ma nella pratica di tutti i giorni il codice da scrivere potrebbe essere molto più lungo e laborioso. Per evitare di riempire le pagine JSP di centinaia di righe di codice e renderle quindi difficilmente manipolabili, c'è un modo per spostare totalmente sul server la parte di calcolo. Il concetto è il seguente: quando il browser richiede di visualizzare una pagina JSP, il server, individuato il file richiesto, interpreta il codice Java contenuto nella pagina ed invia al browser solamente la pagina HTML con il risultato. Allora è possibile spostare il codice in una classe Java e farla semplicemente richiamare dalla pagina JSP. Cosicché il controllo sarebbe comunque fatto dal server, ma la progettazione risulta più semplice e scalabile. Per fare questo possiamo mettere a punto delle servlet ad hoc. Tomcat infatti è un *servlet container* ed una JavaServlet è una particolare applicazione Java che gestisce un flusso HTTP. È possibile costruire una servlet scrivendo una classe Java che erediti le caratteristiche della classe *HttpServlet*. Questa può essere richiamata da una pagina JSP per effettuare lavori come calcoli, ricerche, elaborazioni di vario genere. In pratica può essere considerata una’estensione dell'application server.

Ciclo di vita di un servlet

Il ciclo di vita di un sevlet (cioè il tempo che intercorre dal suo caricamento al suo rilascio) si concentra su tre metodi fondamentali.
Init(): questo metodo segna la vera e propria "nascita" del servlet, è chiamato una sola volta, subito dopo la sua istanziazione. Attraverso questo metodo il servlet s’instanzia tutte le risorse che utilizzerà poi per gestire le richieste.
Service(): questo metodo è incaricato di gestire tutte le richieste effettuate dal client, e ovviamente potrà iniziare il suo lavoro esclusivamente dopo la chiamata del metodo init().
Destroy(): questo metodo segna la vera e propria fine del servlet, solitamente è a questo punto che si effettuano tutti i salvataggi utili per memorizzare informazioni utili ad un prossimo caricamento del servlet.
Il servlet (*prova.jsp*) generato sarà costituito dal seguente file Java.

```
jrun_provaejspa.java
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
import allaire.jrun.jsp.JRunJSPStaticHelpers;
public class jrun__prova2ejspa extends
 allaire.jrun.jsp.HttpJSPServlet implements
 allaire.jrun.jsp.JRunJspPage {
    private ServletConfig config;
    private ServletContext application;
    private Object page = this;
    private JspFactory __jspFactory =
    JspFactory.getDefaultFactory();
    public void _jspService(HttpServletRequest request,
```

```
    HttpServletResponse response)
      throws ServletException, java.io.IOException {
      if(config == null) {
        config = getServletConfig();
        application = config.getServletContext();
      }//if
      response.setContentType("text/html; charset=iso-8859-1");
      PageContext pageContext = __jspFactory.getPageContext(this,
      request, response, null, true, 8192, true);
      JspWriter out = pageContext.getOut();
      HttpSession session = pageContext.getSession();
      try {
        out.print("<!-- PROVA.JSP
        -->\r\n\r\n<html>\r\n\r\n<body>\r\n\r\n");
        out.println ("PRIMA PROVA JSP");
        out.print("\r\n\r\n</body>\r\n\r\n</html>");
      } catch(Throwable t) {
        if(t instanceof ServletException)
          throw (ServletException) t;
        if(t instanceof java.io.IOException)
          throw (java.io.IOException) t;
        if(t instanceof RuntimeException)
          throw (RuntimeException) t;
        throw JRunJSPStaticHelpers.handleException(t,
        pageContext);
      } finally {
        __jspFactory.releasePageContext(pageContext);
      }
    }
    private static final String[] __dependencies__ =
    {"/prova.jsp",null};
    private static final long[] __times__ = {988194570000L,0L};
    public String[] __getDependencies() {
      return __dependencies__;
    }
    public long[] __getLastModifiedTimes() {
      return __times__;
    }
    public int __getTranslationVersion(){
      return 14;
    }
}
```

Analizzando il codice si può notare come il metodo che effettua il servizio vero e proprio è il metodo *_jspService*. Esso in un primo tempo verifica la configurazione del server (se non è stata effettuata se ne occupa), per poi settare in maniera adeguata i parametri che permettono la comunicazione dell'output voluto (è per esempio specificato che il testo in uscita è codice HTML). Effettuato il controllo della configurazione sono eseguite le istruzioni per cui il servlet è stato creato (in questo caso la stampa a video del messaggio), tenendo in considerazione gli eventuali errori, creando delle adeguate eccezioni. Con l'ultima istruzione il servlet viene rilasciato. Gli altri metodi servono per fornire alcune informazioni riguardanti il servlet: quando è stato modificato l'ultima volta, la versione della traduzione, il nome del file contenente la pagina JSP a cui fa riferimento.

Azioni
Questi oggetti sono finalizzati ad un migliore incapsulamento del codice, generalmente inteso come inclusione e utilizzo di Java Bean.
Le azioni standard per le pagine JSP sono:
<jsp:useBean>: permette di utilizzare i metodi implementati all'interno di un JavaBean.
<jsp:setProperty>: permette d'impostare il valore di un parametro di un metodo di un Bean.
<jsp:getProperty>: permette di acquisire il valore di un parametro di un Bean.
<jsp:param>: permette di dichiarare ed inizializzare dei parametri all'interno della pagina. Sintassi.
<jsp:params>
<jsp:param name="nomeParametro" value="valore">
</jsp:params>
dove i parametri indicano:
name: nome del parametro dichiarato.
value: valore del parametro appena dichiarato.
<jsp:include>: permette d'includere risorse aggiuntive di tipo sia statico che dinamico. Il risultato è quindi quello di visualizzare la risorsa, oltre alle informazioni già inviate al client. Sintassi.
<jsp:include page="URLRisorsa" flush="true | false" />
dove i parametri indicano:
page: URL della risorsa fa includere.
flush: attributo booleano che indica se il buffer deve essere svuotato o meno. Questa azione può venire completata con la dichiarazione di eventuali parametri legati agli oggetti inclusi.
<jsp:include page="URLRisorsa flush="true|false">
{<jsp:param ... />}
</jsp:include>.
<jsp:foward>: consente di eseguire una richiesta di una risorsa statica, un servlet o un'altra pagina JSP interrompendo il flusso in uscita. Il risultato è quindi la visualizzazione della sola risorsa specificata.
Sintassi.
<jsp:forward page="URLRisorsa"/>
dove *page* specifica l'URL della risorsa cui eseguire la richiesta del servizio. Ovviamente questa azione può anch'essa essere completata con la dichiarazione di parametri.
<jsp:forward page="URLRisorsa>
{<jsp:param ... />}
</jsp:forward>.
Gli oggetti impliciti sono disponibili per la stesura del codice senza particolari inclusioni, per per usufruire delle loro funzionalità è quindi sufficiente usare la tipica sintassi *nomeOggetto.nomeMetodo*.

Request
L'oggetto request permette di accedere alle informazioni d'intestazione specifiche del protocollo HTTP. Al momento della richiesta questo metodo incapsula le informazioni sulla richiesta del client e le rende disponibili attraverso alcuni suoi metodi. L'uso più comune è quello di accedere ai parametri inviati (i dati provenienti da un form per esempio) con il metodo *getParameter*("nomeParametro"), che restituisce una stringa con il valore del parametro specificato. Alto metodo molto importante è *getCookies(),* che restituisce un array di cookies).
Gli altri metodi, i più importanti, sono i seguenti.
getAttributeNames() restituisce una variabile di tipo *Enumeration* contenente i nomi di tutti gli attributi coinvolti nella richiesta.

getContentLength() restituisce un intero che corrisponde alla lunghezza in byte dei dati richiesti.
getContentType() restituisce il tipo MINE della richiesta, cioè il tipo di codifica dei dati.
getInputStream() restituisce un flusso di byte che corrisponde ai dati binari della richiesta, utile per funzioni di upload di file da client a server.
getParameter(String) restituisce una stringa con il valore del parametro richiesto.
getParameterNames() restituisce una variabile di tipo *Enumeration* contente i nomi dei parametri della richiesta.
getParameterValues(String) restituisce un array contenente tutti i valori del parametro specificato (nel caso ci siano più parametri con lo stesso nome).
getProtocol() corrisponde alla variabile *CGI SERVER_PROTOCOL*, rappresenta il protocollo e la versione della richiesta.
getRemoteHost() corrisponde alla variabile *CGI REMOTE_HOST*.
getServerName() corrisponde alla variabile *CGI SERVER_NAME* e restituisce l'indirizzo IP del server.
getServerPort() corrisponde alla variabile *CGI SERVER_PORT*, la porta alla quale in server è in ascolto.
getRemoteAddr() corrisponde alla variabile *CGI REMOTE_ADDR*. Restituisce in pratica l'indirizzo IP del visitatore.
getRemoteHost() corrisponde alla variabile *CGI REMOTE_HOST*.
getRemoteUser() come per la variabile *CGI REMOTE_USER* restituisce lo username della macchina richiedente.
getMethod() come per la variabile *CGI REQUEST_METHOD* e restituisce GET o POST.
getPathInfo() restituisce informazioni extra sul path, corrisponde alla varibile *CGI PATH_INFO*.
getQueryString() restituisce una stringa contenete l'intera *querySting* (tutti i caratteri dopo il punto di domanda), come per la variabile *CGI QUERY_STRING*.
request.getServletPath() restituisce il percorso relativo della pagina jsp.
Response questo oggetto fornisce tutti gli strumenti per inviare i risultati dell'esecuzione della pagina jsp.
setBufferSize(int) imposta la dimensione in byte del buffer per il corpo della risposta, escuse quindi le intestazioni.
setContentType("tipo") imposta il tipo MINE per la risposta al client.
flushBuffer() forza l'invio dei dati contenuti nel buffer al client.
Out questo oggetto ha principalmente funzionalità di stampa di contenuti. Con il metodo *print*(oggetto/variabile) è possibile stampare qualsiasi tipo di dato, come anche per *println*() che a differenza del precedente termina la riga andando a capo. Si capisce comunque che l'andare o meno a capo nella stampa dei contenuti serve solo a migliorare la leggibilità del codice HTML.
PageContext questo oggetto rappresenta in pratica il contesto di esecuzione del servlet creato dalla pagina JSP. In pratica è poco utilizzato dai programmatori di pagine JSP.
Config permette di gestire tramite i suoi metodi lo startup del servlet associato alla pagina JSP, di accedere quindi a parametri di inizializzazione e di ottenere riferimenti e informazioni sul contesto di esecuzione del servlet stesso.
Exception questo oggetto è accessibile solo dalle pagine di errore (dove la direttiva *isErrorPage* è impostata a *true*), contiene le informazioni relative all'eccezione sollevata in una pagina in cui il file è stato specificato come pagina di errore, il metodo principale è *getMessage()* che restituisce una stringa con la descrizione dell'errore.

Utilizzare una servlet che effettui il controllo sulla stringa digitata dall'utente e richiami la pagina corretta.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;
```

```
public class Main extends HttpServlet {
  public void doGet (HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException
  {
    String categoria = request.getParameter("categoria");
    String jspPage = "";
    if ( categoria.equals("politica") )
    {
      jspPage = "pages/politica.jsp";
    }
    else if ( categoria.equals("finanza") )
    {
      jspPage = "pages/finanza.jsp";
    }
    else if ( categoria.equals("sport") )
    {
      jspPage = "pages/sport.jsp";
    }
    else
      jspPage = "pages/error.jsp";
    request.getRequestDispatcher( jspPage ).forward(request,response);
  }
}
```

La servlet recupera dalla *request* il parametro *categoria*, che corrisponde al campo input del form della pagina *index.html* e in base al suo valore richiama una pagina JSP differente.

Il comando chiave della classe sopra esposta è:

request.getRequestDispatcher(jspPage).forward(request,response)

Tramite questo comando infatti si reindirizza il browser alla pagina JSP voluta. Una volta scritta e compilata bisognerà registrarla questa classe come servlet nell'applicazione. Per fare ciò si crea, nella cartella *Primo* dell'applicazione, una cartella *WEB-INF* ed all'interno una cartella *classes*. Dentro la cartella *WEB-INF* andrà creato il file *web.xml*.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
“http://java.sun.com/dtd/Web-app_2_3.dtd">
<web-app>
<servlet>
<servlet-name>MainServlet</servlet-name>
<servlet-class>Main</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>MainServlet</servlet-name>
<url-pattern>/MainUrl</url-pattern>
</servlet-mapping>
<session-config>
<session-timeout>30</session-timeout>
</session-config>
</web-app>
```

JSP e Java Beans

Le JSP sono estensioni dei servlet Java, di conseguenza consentono di ottenere tutti i risultati di questi ultimi. Lo svantaggio dei Servlet è però rappresentato, come per tutti gli script CGI, dalla difficoltà di manutenzione del codice HTML delle applicazioni eccessivamente complesse, in quanto il più delle volte contengono una quantità di codice scriptler eccessiva per un programmatore non esperto in Java.
Soluzione a questo problema è rappresentata dai **Java Beans**, componenti software contenenti una classe Java, che possono essere inclusi in una pagina JSP, permettendo quindi un ottimo incapsulamento del codice, peraltro riutilizzabile. Al programmatore quindi sarà pressochè invisibile la sezione di codice puro, sostituito da richiami ai metodi delle classi incluse.
I Bean sono costituiti esclusivamente da codice Java e possono essere realizzati con un semplice editor di testo, salvati con estensione *.java* e compilati generando un file di estensione *.class*. Sarà il file *.class* che dovrà essere incluso nella pagina JSP. Da ricordare che in Java i file devono avere lo stesso nome della classe, lettere maiuscole e minuscole comprese.

Aggiunta di JavaBean
Esistono tre azioni standard per facilitare l'integrazione dei JavaBean nelle pagine JSP.
<jsp:useBean> permette di associare un'istanza di un JavaBean (associata ad un determinato ID) ad una variabile script dichiarata con lo stesso ID. In pratica offre la possibilità di associare la classe contenuta nel JavaBean ad un oggetto visibile all'interno della pagina, in modo da poter richiamare i suoi metodi senza dover ogni volta far riferimento al file di origine.
Attributi:
Id: identità dell'istanza dell'oggetto nell'ambito specificato.
Scope: ambito dell'oggetto, le opzioni sono:
- **page**: gli oggetti con questo ambito sono accessibili solo all'interno della pagina in cui sono stati creati, in pratica possono essere paragonati alle variabili locali di un qualsiasi linguaggio di programmazione, sono distrutte alla chiusura della pagina e i dati saranno persi;
- **request**: gli oggetti con questo ambito sono accessibili esclusivamente nelle pagine che elaborano la stessa richiesta di quella in cui è stato creato l'oggetto, quest'ultimo inoltre rimane nell'ambito anche se la richiesta è inoltrata ad un'altra risorsa;
- **session**: gli oggetti definiti in quest'ambito sono accessibili solo alle pagine che elaborano richieste all'interno della stessa sessione di quella in cui l'oggetto è stato creato per poi essere rilasciati alla chiusura della sessione cui si riferiscono, in pratica restano visibili in tutte le pagine aperte nella stessa istanza (finestra) del browser, fino alla sua chiusura. Solitamente i bean istanziati in questo modo sono utilizzati per mantere le informazioni di un utente di un sito;
- **application**: gli oggetti definiti in quest'ambito sono accessibili alle pagine che elaborano richieste relative alla stessa applicazione, in pratica sono validi dalla prima richieta di una pagina al server fino al suo shotdown.
class: nome della classe che definisce l'implementazione dell'oggetto beanName: contiene il nome del bean che, deve coincidere con il nome del file *.class* (senza estensione).
Esempio.
<jsp:useBean id="nomeBean" scope="session" class="classe" />
Crea un'istanza della classe *classe* con ambito *session* richiamabile attraverso l'id NomeBean: da questo momento sarà possibile accedere a metodi e variabili (pubbliche) attraverso la sintassi *nomeBean.nomeMetodo* e *nomeBean.nomeVariabile*, rispettivamente per metodi e variabili.
<jsp:setProperty> permette d'impostare il valore di una delle proprietà di un bean.
Attributi:

name: nome dell'istanza di bean definita in un'azione;
property: rappresenta la proprietà di cui impostare il valore;
param: nome del parametro di richiesta il cui valore si vuole impostare;
value: valore assegnato alla proprietà specificata;
Esempio.

```
<jsp:setProperty name="nome_bean" property="prop"
  param="nome_parametro" />
```

Permette di assegnare il valore del parametro *nome_parametro* alla proprietà *prop* del bean di nome *nome_bean*.
<jsp:getProperty> prende il valore di una proprietà di una data istanza di bean e lo inserisce nell'oggetto out implicito (in pratica lo stampa a video).
Attributi:
name: nome dell'istanza di bean da cui proviene la proprietà definita da un'azione;
property: rappresenta la proprietà del bean di cui si vuole ottenere il valore.
Esempio di bean, realizzato per contenere le informazioni di un utente durante la sua permanenza nel sito.
InfoUtente.java

```
public class InfoUtente {
   private String nome = null;
   private String email = null;
   private int pagineViste;
   public InfoUtente() {
      pagineViste=0;
   }
   public aggiornaPV(){
      pagineViste++;
   }
   public int getPagineViste(){
      return pagineViste;
   }
   public void setNome(String value) {
      nome = value;
   }
   public String getNome() {
      return nome;
   }
   public void setEmail(String value) {
      email = value;
   }
   public String getEmail() {
      return email;
   }
   public String riassunto(){
      String riassunto = null;
      riassunto = "Il nome dell'utente è"+nome+",";
      riassunto+= "il suo indirizzo e-mail è: "+email;
      riassunto+=" e ha visitato "+pagineViste+" del sito";
      return riassunto;
   }

}//InfoUtente
```

Come è facile capire, questo bean contiene il nome dell'utente ed i metodi per modificarlo e restituirlo, il suo indirizzo e-mail con i relativi metodi, il numero di pagine viste dall'utente e un metodo che restituisce un riassunto schematico dei dati dell'utente.

Ecco come utilizzarli.
html>
<head><title>Utilizzo del Bean</title></head>
<body>
<jsp:useBean id="utente" scope="session" class="InfoUtente"/>
È creata un'istanza del bean *InfoUtente* con ambito *session*, necessario per questo tipo di funzione.
jsp:setProperty name="utente" property="nome" value="Zina&Tram"/>
La proprietà del bean possono essere impostate con l'azione *setProperty* o agendo direttamente con i metodi creati appositamente.
<%
utente.setNome("Zina&Tram");
utente.setEmail("ciao@dom.it");
%>
Lo stesso vale per la lettura dei bean che puo essere fatta con l'azione.
<jsp:getProperty name="utente" property="nome"/>
O agendo sui metodi creati.
<% out.println(utente.getNome());
out.println(utente.riassunto());
%>
Per incrementare il numero di pagine viste è sufficiente richiamare il metodo *aggiornaPV()* e per ottenere il valore *getPagineViste().*
<% utente.aggiornaPV();
out.println(utente.getPagineViste());
%>

Gestire un carrello della spesa

Il più delle volte per comprare un qualsiasi tipo di oggetto sul web, si utilizza quello che è chiamato carrello della spesa elettronico, che permette di memorizzare gli oggetti che s'intende acquistare ed eventualmente annullarne l'ordinazione. Come esempio si è supposto di dover creare un carrello della spesa per un negozio di CD musicali. Il progetto che implementa tale servizio è formato da tre file.

1. *carrello.jsp*: rappresenta il catalogo dei CD disponibili, con la possibilità di ordinazion.
2. *visCarr.jsp*: visualizza il contenuto attuale del carrello, permettendo l'annullamento di un eventuale acquisto.
3. *carrello.java*: bean che contiene tutte le funzioni di memorizzazione, eliminazione e quant'altro, utili allo sviluppo delle pagine JSP. Per prima cosa è necessario esaminare il bean, affinché risultino chiare le funzioni utilizzate poi nelle due JSP.

carrello.java

```
import java.lang.String;
import java.lang.Integer;
import java.lang.Float;
import java.util.Hashtable;
import java.util.Enumeration;
public class Carrello{
   //variabile di tipo Hashtable utilizzata per emorizzare gli elementi presenti nel carrello
   protected Hashtable cd = new Hashtable();
   //varibile intera utilizzata per la memorizzazione del numero di CD attualmente nel
carrello
   private int numCd;
   //costruttore
   public Carrello() {
      numCd=0;
   }//Carrello
```

```
    //metodo per ottenere il numero di CD attualmente nel carrello
    public int totCD(){
        return numCd;
    }//numCD
    // Inserimento di cd nel carrello
    // I: id: indice chiave di identificazione del CD
    //    art: nome dell'artista
    //    tit: titolo dell'album
    //    prezzo: prezzo del CD
    public void aggiungiCd (String id, String art, String tit, float prezzo){
        //inserimento in un vettore di stringhe degli attributi del CD appena inserito nel
carrello
        //è inserito inoltre il numero di CD di quel tipo ordinati
        String[] dati_cd= {art, tit, Float.toString(prezzo) , "1", id};
        numCd++;
        //se il CD è non già stato ordinato
        if (!cd.containsKey(id)){
            //inserisci il CD nel carrello
                cd.put(id, dati_cd);
        }//if
        //altrimenti
        else {
        //incrementa il numero di CD di quel tipo ordinati
            int tmp;
            String[] datiTemp = (String[])cd.get(id);
            tmp = Integer.parseInt(datiTemp[3]);
                tmp++;
            datiTemp[3]=Integer.toString(tmp);
                cd.put(id, datiTemp);
        }//else
    }//aggiungiCd
    //Metodo di rimozione dal carrello di un CD
    //I: id: indice del CD da rimuovere
    public void rimuoviCd (String id){
        //se il CD è presente nel carrello
        if (cd.containsKey(id)) {
            //decrementa il numero di CD
            numCd--;
                String[] temp = (String[])cd.get(id);
                //se il CD è presente in copia singola
            if (Integer.parseInt(temp[3]) == 1)
                    //elimina il CD dal carrello
                    cd.remove(id);
                //altrimenti
            else {
                //decrementa il numero di copie di quel CD ordinate
                    int tmp = 0;
                String[] datiTemp = (String[])cd.get(id);
                tmp = Integer.parseInt(datiTemp[3]);
                    tmp--;
                datiTemp[3]=Integer.toString(tmp);
                    cd.put(id, datiTemp);
            }//else
        }//if
```

```
    }//rimuoviCd
    //Metodo per la creazione di un oggetto Enumeration utilizzato per scorrere
velocemente
    //gli elementi nel carrello
    public Enumeration enum() {
        return cd.elements();
    }//enum
    //Metodo che restituisce il prezzo totale dei CD inseriti nel carrello fino adesso
    public float spesaTot(){
        float tot = 0.00f;
              String[] temp;
        //crea l'oggetto Enumeration
        Enumeration i = this.enum();
        //finchè ci sono elementi nel carrello
        while (i.hasMoreElements()){
            temp = (String[])i.nextElement();
            //incrementa la spesa totale del prezzo del CD corrente
            tot += (Float.parseFloat(temp[2]) * Integer.parseInt(temp[3]));
              }//while
        return tot;
    }//spesaTot
    //Metodo che restituisce il prezzo di un CD moltiplicato per il numero delle copie
ordinate
    //I: id:indice del CD
    public float prezzo(String id){
        String[] temp;
        float tmp=0.0f;
        temp = (String[])cd.get(id);
        tmp = Float.parseFloat(temp[2]) * Integer.parseInt(temp[3]);
        return tmp;
    }//prezzo

}
```

Il codice non presenta particolare difficoltà d'interpretazione, è interessante notare come la classe *Hashtable* renda molto semplice la memorizzazione e l'accesso alle informazioni, rendendole molto simili per certi aspetti ad un DB.
Grazie alla creazione di questo bean la pagina *carrello.jsp* presenta un buon incapsulamento del codice, facilitando sia l'interpretazione (è fatto prevalentemente a codice HTML) sia la modifica.

```
<HTML>
<HEAD>
<TITLE>Vendita CD On-line</TITLE>
</HEAD>
<BODY BGCOLOR="#33CCCC" text="#000099" vlink="#990099" alink="#000099">
<!-istanziazione del bean Carrello.jsp, con ambito session -->
<jsp:useBean id="car" scope="session" class="Carrello" />
<%
String id = request.getParameter("id");
//se è stato aggiunto un CD al carrello
if (id != null){
    //acquisisci i dati del CD
    String artista=request.getParameter("artista");
    String titolo=request.getParameter("titolo");
    float prezzo = Float.parseFloat(request.getParameter("prezzo"));
```

```
   //e aggiungilo al carrello
   car.aggiungiCd (id, artista, titolo, prezzo);
}//if
%>
<font face="Verdana" size="6">
<center>
CATALOGO COMPACT DISC A DISPOSIZIONE <br><br>
</center>
</font>
<font face="Verdana">
<a href="visCarr.jsp"> Quantità attualmente nel carrello: </a> <%= car.totCD();%>
<br><br>
<center>
<table width="60%" border="1" align="center" bordercolorlight="#990099"
bordercolordark="#990099">
   <tr>
      <th>Artista</th>
          <th>Titolo</th>
          <th>Prezzo</th>
   </tr>
   <tr>
      <form action="Carrello.jsp" method="post">
          <td>Dream Theater</td>
          <td>Awake</td>
          <td>L.25000</td>
          <td><center><input type="submit" name="add" name="acquista" value="Aggiungi
al carrello"></center></td>
          <input type="hidden" name="id" value="1">
          <input type="hidden" name="artista" value="Dream Theater">
          <input type="hidden" name="titolo" value="Awake">
          <input type="hidden" name="prezzo" value="25000">
          </form>
      <td>
   </tr>
   <tr>
      <form action="Carrello.jsp" method="post">
          <td>Iced Earth</td>
          <td>The Dark Saga</td>
          <td>L.35000</td>
          <td><center><input type="submit" name="add" name="acquista" value="Aggiungi
al  carrello"></center></td>
          <input type="hidden" name="id" value="2">
          <input type="hidden" name="artista" value="Iced Earth">
          <input type="hidden" name="titolo" value="The Dark Saga">
          <input type="hidden" name="prezzo" value="35000">
          </form>
      <td>
   </tr>
   <tr>
      <form action="Carrello.jsp" method="post">
          <td>Queen</td>
          <td>Innuendo</td>
          <td>L.35000</td>
```

```
        <td><center><input type="submit" name="add" name="acquista" value="Aggiungi
al carrello"></center></td>
        <input type="hidden" name="id" value="3">
        <input type="hidden" name="artista" value="Queen">
        <input type="hidden" name="titolo" value="Innuendo">
        <input type="hidden" name="prezzo" value="35000">
        </form>
      <td>
   </tr>
</table>
</font>
</center>
</BODY>
</HTML>
```

Il bean è stato istanziato con ambito *session*, in modo da non permettere la perdita d'informazioni se non dopo la chiusura del browser.

visCar.jsp

```
<!-- inclusione dei package per l'utilizzo di alcune funzioni standard -->
<%@ page import="java.util.*" %>
<%@ page import="java.lang.*" %>
<!-- istanziazione del bean Carrello.jsp con ambito session -->
<jsp:useBean id="car" scope="session" class="Carrello" />
<HTML>
<HEAD>
<TITLE>Carrello della spesa</TITLE>
</HEAD>
<BODY BGCOLOR="#33CCCC" text="#000099" vlink="#990099" alink="#000099">
<%
String idtmp = request.getParameter("id");
//se è stato eliminato un CD dal carello
if (idtmp != null){
   car.rimuoviCd(idtmp);
}//if
%>
<font face="Verdana" size="6">
<center>
CARRELLO
</center>
</font><br><br>
<font face="Verdana">
Quantità attualmente nel carrello: <%= car.totCD();%>
<br><br>
<center>
<table width="60%" border="1" align="center" bordercolorlight="#990099"
bordercolordark="#990099">
   <tr>
      <th>Artista</th>
         <th>Titolo</th>
         <th>Prezzo (L.)</th>
      <th>Quantità</th>
   </tr>
<%
//creazione di un oggetto Enumeration
Enumeration ogg = car.enum();
```

```
String[] temp;
//finchè ci sono elementi nel carrello
while (ogg.hasMoreElements()){
   temp= (String[])ogg.nextElement();
%>
   <tr>
          <td><%= temp[0] %></td>
          <td><%= temp[1] %></td>
          <td>L.<%= car.prezzo(temp[4])%></td>
      <td><center><%= Integer.parseInt(temp[3]) %></center></td>
          <td><center><form action="visCarr.jsp" method="post"><input type="submit"
name="rem"
                name="togli" value="Togli"></center></td>
      <input type="hidden" name="id" value= <%= temp[4]%>>
          </form>
      <td>
   </tr>
<%
}//while
%>
   <tr>
      <td></td>
      <td>Tot</td>
      <td>L.<%= car.spesaTot()%></td>
          <td><center><%= car.totCD()%></center></td>
   </tr>
</table>
<font size="2">
[ <a href="Carrello.jsp">Indietro</a> ]
</font>
</center>
</BODY>
</HTML>
```

UBERTINI MASSIMO
http://www.ubertini.it
massimo@ubertini.it

Dip. Informatica Industriale
I.T.I.S. "Giacomo Fauser"
Via Ricci, 14
28100 Novara Italy
tel. +39 0321482411
fax +39 0321482444
http://www.fauser.edu
massimo@fauser.edu