

PHP e MySQL

Rev 5.2 del 05/05/2020

Introduzione ai web server	2
Sintassi base di PHP	3
Funzioni	5
Array Enumerativi	6
Array Associativi	7
Matrici	8
La gestione delle date	9
Variabili superglobali	10
Incorporazione di altri file: include e require	10
Il pulsante di submit e la gestione delle richieste HTTP	11
Lettura lato server dei Parametri Get e Post	12
Sintassi base di MySQL	13
Tipi di dati	13
Comandi DDL	16
Comandi DML	17
Comandi SQL	18
Indici	20
Principali Operatori e Funzioni di MySQL	21
Connessione tra PHP e MySQL	23
L'oggetto mysqli	23
Il problema del SQL injection	25
Redirect	26
La funzione md5()	26
Creazione di un web service tramite xampp	27
Creazione e Serializzazione di uno stream JSON	28
Creazione e Serializzazione di uno stream XML	30
Gestione delle variabili SESSION	31
Impostazione dei cookies	34
Principali funzioni per la gestione di un file	35
Gestione dell'upload di un file	36
Lettura dei parametri passati con i metodi PUT e DELETE	38
Invio di una mail	39

Introduzione ai web server

Un **web server** è una applicazione che, a fronte di una precisa richiesta, invia al client la risorsa richiesta che può essere

- un **semplice file** (html, js, css)
- un **insieme di dati** (in formato xml o json)

Inoltre il web server, prima di inviare la risposta, può eseguire delle **elaborazioni** sia sulle pagine sia sui dati. Il protocollo di comunicazione utilizzato per lo scambio di dati con un web server è il protocollo **http**

Quando il client riceve dal server una pagina html, provvede a 'leggerla' e a richiedere al server, una alla volta, tutte le ulteriori risorse utilizzate da quella pagina: uno o più file .css o .js, immagini, video inglobati nella pagina o file di caratteri .ttf o altro ancora.

Pagine Dinamiche

Molto spesso le pagine html attuali devono essere **dinamiche**, nel senso che devono aggiornare il contenuto più o meno frequentemente. Si pensi ad esempio a:

- una pagina di previsioni meteo, in cui la pagina è sempre la stessa ma i dati cambiano
- un catalogo prodotti, in cui i dati cambiano meno frequentemente ma devono essere aggiornati
- una pagina contenente le quotazioni di borsa, in cui i dati cambiano in modo continuo.

In tutti questi casi il web server, prima di inviare la pagina, deve aggiornare i dati.

A tal fine si sono succeduti storicamente due approcci differenti:

- **Web Form** Il server, in corrispondenza di ogni richiesta, ricostruisce dinamicamente l'intera pagina andando a leggere i dati aggiornati in un db ed inserendoli all'interno della pagina prima di inviarla.
- **Ajax** Il server invia inizialmente una pagina statica priva di dati. In corrispondenza del ricevimento della pagina il client richiede i dati tramite una chiamata Ajax che verrà ripetuta ad ogni richiesta di aggiornamento. La pagina html non viene però più ricaricata. Approccio molto più snello.

I principali web server

Web server	Azienda detentrici del marchio	Linguaggio di programmazione	Database utilizzato
IIS (internet information server)	Microsoft	Aspx / C#	SQL Server
Apache	Apache Foundation GNU License (GNU's Not Unix)	PHP	MySQL marchio MySQL acquistato da Oracle
Tomcat	Apache Foundation GNU License (GNU's Not Unix)	JSP / JAVA	Oracle
Node.js	Node.js Foundation	Java Script	mongoDB

xampp

xampp è una distribuzione contenente più pacchetti nata in ambiente Linux (con il nome **Lampp**) e resa successivamente disponibile anche in ambiente windows e scaricabile gratuitamente dal sito xampp.org

X = Windows
A = Apache (web server)
M = MySQL (database di tipo client server)
P = PERL (linguaggio server alternativo a PHP)
P = PHP

L'installer provvede ad installare l'intero pacchetto all'interno della cartella c:\xampp.
La sottocartella **htdocs** rappresenta la cartella home per le applicazioni utente

Avvio di Apache

C:\xampp\xampp-control.exe

Avviare **Apache**.

Per modificare la porta di ascolto aprire il file **Config / apache**, cercare 80 e sostituirlo con la porta desiderata (2 sostituzioni). Oppure **pulsante in alto a destra Config / Service and Port Settings**

Clickando su **Admin** si apre la pagina informativa C:\xampp\htdocs\dashboard\index.html

Sintassi base di PHP

Php Hypertext PreProcessor (ricorsivo)

<http://php.net>

guide: <http://www.html.it/guide/guida-php-di-base>, <http://www.w3schools.com/php/default.asp>

Scopo di PHP è quello di costruire dinamicamente una pagina html da inviare al client.

Le pagine con estensione .html vengono inviate dal server al client così come sono.

Le pagine con estensione .php vengono prima elaborate dal server e solo dopo inviate al client.

Il server provvede ad eseguire le istruzioni php contenute nella pagina e sostituirle con opportuni tag html che verranno poi interpretati dal browser.

Sintassi Base

Case Sensitive. Non accetta caratteri speciali al di fuori dell'underscore

Ogni istruzione **deve** essere terminata da ; esattamente come in C

Virgolette doppie e semplici possono essere utilizzate quasi indifferentemente (Le virgolette singole non accettano variabili al loro interno).

E' disponibile un vasta libreria di funzioni quasi coincidenti con le librerie classiche dell' Ansi C

```
/* Commento multi linea */
// Commento su singola riga
# Altro tipo di commento, poco utilizzato
```

Gli script php possono essere inseriti in qualsiasi punto di una pagina html, che **però** deve necessariamente avere estensione **.php**.

echo() con o senza parentesi tonde, consente di scrivere sulla pagina html prima che questa venga inviata al client. Accetta al suo interno una stringa contenente qualsiasi tag HTML.

Non va a capo automaticamente. Per andare a capo non riconosce il \n, ma occorre utilizzare **"
"** ;

print() è identica a echo ma in più restituisce true / false a seconda che l'operazione sia andata a buon fine oppure no. In pratica restituisce sempre true, ma il fatto di ritornare un valore consente di utilizzarla all'interno dell'operatore condizionale ternario: `$b ? print "true" : print "false";`

Per contro, restituendo un valore, è più lenta di echo().

print_r() serializza qualunque variabile visualizzandola in un formato human-readable.

Ad esempio serializza automaticamente vettori sia enumerativi che associativi

die("msg") Analogo a **echo** però termina l'elaborazione dello script. Comodo per segnalare ad esempio parametri mancanti. L'utente premerà indietro e ritornerà alla compilazione della pagina precedente.

Nel caso di **die()** le parentesi tonde sono obbligatorie.

Dichiarazione ed utilizzo delle Variabili

Non sono tipizzate e non è richiesta la dichiarazione. Vengono allocate automaticamente al primo utilizzo. Devono iniziare con il carattere \$. Ad esempio: \$n = 3.4; \$NumeroEsa = 0x1B

Le variabili dichiarate all'interno di uno script sono visibili in tutti gli altri script della pagina, ma non sono visibili all'interno di eventuali funzioni utente (come avviene ad esempio in jQuery).

Per vedere se una variabile esiste : `if(isset($var))`

Per vedere se una variabile è stata valorizzata : `if($var != "undefined")`

Costanti

Vengono dichiarate mediante la funzione **define**("nome", "valore")

```
define ("NOME", "Mario Rossi");
```

```
$nome = NOME;
```

Concatenamento di stringhe

Operatore di concatenamento `.` `$s = "mario"."rossi";`

Il puntino può essere scritto indifferentemente con spazi laterali o senza spazi. E' ammesso anche `.=`

Utilizzo di variabili all'interno di una stringa

Una interessante possibilità di PHP è quella di poter utilizzare le variabili semplici all'interno delle stringhe;

```
$s = "Salve $nome !"; // Salve Pippo !.
```

- Funziona solo all'interno delle **virgolette doppie** (le single quote sono utilizzate per rappresentare la stringa così com'è)
- Sono ammesse anche variabili composite come ad esempio i **vettori associativi**, nel qual caso però occorre NON utilizzare altri apici per specificare la chiave. Ad esempio:

```
$msg ="Benvenuto $_GET[txtNome]"
```

Caratteri di escape

Utilizzabili nelle stringhe. Sono gli stessi identici del C

```
\n \t \\ \" \\\$
```

Questi caratteri non sono però interpretati dall'HTML, per cui non possono essere utilizzati quando si invia una risposta ad un browser. Possono invece essere utilizzati, ad esempio, quando si sta salvando una qualche informazione su file.

Operatori

A differenza del concatenamento, sono gli stessi identici del C

Aritmetici `+` `-` `*` `/` `%` (resto della divisione fra interi)

Assegnazione `=` `+=` `-=` `*=` `/=`

Di Confronto `==` `!=` `>` `>=` `<` `<=`

Logici `&&` `||` `!`

Unari `++` `--`

Creazione di variabili a partire dal contenuto di una stringa

```
$var="txtNome";
```

```
$var="pippo"; // crea una nuova variabile di nome txtNome
```

```
echo($txtNome); // pippo
```

`$$var` crea una nuova variabile avente come nome `$txtNome` (cioè il contenuto della variabile `$var`)

Istruzioni Base

Stesse identiche del C

if - for - do...while - while - switch.

Switch

```
switch ($colore) {  
    case 'blu' : echo ("Il colore selezionato è blu"); break;  
    case 'giallo': echo ("Il colore selezionato è giallo"); break;  
    default : echo ("Nessun colore selezionato"); break;  
}
```

Esattamente come in C, una volta individuato un case valido, l'elaborazione procede fino al termine dello switch, a meno che non si incontri un break.

Funzioni

L'utilizzo delle funzioni è esattamente come in Java Script. E' possibile richiamare una funzione dall'interno di una altra funzione, a patto che essa sia 'visibile' al chiamante, cioè **deve essere scritta prima della funzione da cui viene richiamata**.

Dallo script principale è invece possibile richiamare funzioni anche scritte dopo

Passaggio dei parametri

I parametri vengono passati di default per valore.

Per passare un parametro **per indirizzo** occorre anteporre l'operatore **&** nella definizione della funzione (al contrario dell' Ansi C).

```
elabora($val1, &$val2); // il secondo parametro è passato per indirizzo
```

Per passare un **vettore**, all'interno della firma della funzione si può anteporre facoltativamente **array**.

La parola chiave global

Le funzioni utente **non 'vedono'** le variabili 'globali' dichiarate all'interno degli script (nemmeno quelle dichiarate all'interno dello stesso script). Una funzione può comunque accedere alle variabili dello script anteponendo al nome della variabile il termine **global**, che può però essere usato **soltanto** in fase dichiarativa. Tecnica alternativa rispetto al passaggio dei parametri, ma meno strutturata.

```
global $contatoreSomme = 0; // qui global potrebbe anche essere scritto o omesso  
function somma($a, $b) {  
    $somma = $a + $b;  
    global $contatoreSomme;  
    $contatoreSomme++;  
    return $somma;  
}
```

La prima pagina PHP

```
<?php  
$nome = "pippo";  
echo ("Il mio nome &grave; $nome <br>");  
visualizza ($nome);  
function visualizza($nome) {  
    echo ("<p style='font-weight:bold'>Il mio nome &grave; $nome </p>");  
}  
?>
```

Funzioni di libreria

`isset($var)` restituisce **true** se la variabile esiste (cioè se è già stata dichiarata o utilizzata)

matematiche

`$val = rand(A, B)` genera un numero intero random compreso tra A e B entrambi gli estremi inclusi
`$ris = pow(2, 3)` Esegue l'elevamento a potenza 2^3
`$ris = round(num, nCifre)` Arrotonda a n cifre dopo la virgola
`is_numeric($str)` Restituisce true se la stringa è convertibile in numero

stringhe

`$n = intval($str)` Converta la `str` in numero
`$str = strtoupper($str)` Converta in maiuscolo
`$str = strtolower($str)` Converta in minuscolo
`$str = ucwords($str)` - Uppercase the first character of each word in a string
`$s = str_replace($string, $s1, $s2)` Sostituisce `s1` con `s2` all'interno di `string`
`$s = substr($string, $start [, int $length])` Estrae n caratteri a partire dalla posiz `start`
`$pos = strpos($string, $substring, [$start])`
 Restituisce la prima occorrenza di `substring` all'interno di `string`. Se non la trova restituisce false
`$pos = strrpos($string, $substring, [$start])`
 Restituisce l'ultima occorrenza di `substring` all'interno di `string`. Se non la trova restituisce false
`stripos()` e `strripos()` Uguali alle precedenti ma eseguono una ricerca case un sensitive
`$str = htmlentities($str)` Converta in `htmlentity` (`>`; `<`; `"`;...) tutti i caratteri che hanno un corrispondente carattere di tipo `htmlentity`
`$str = htmlspecialchars($str)` Simile alla precedente ma non converte tutti i caratteri che dispongono di un `htmlentity`, ma solo i caratteri speciali. **Non** converte ad esempio le lettere accentate.
`$str = nl2br($str)` Converta il `\n` in `
`

Le funzioni implode() e explode()

implode() converte un vettore enumerativo in stringa. Non funziona con i vettori associativi
 Dato `$vect`, non è possibile fare `echo($vect)`; ma occorre fare un ciclo oppure convertirlo in stringa:

```
$s = implode("<br>", $vect); echo($s);
```

Il primo parametro di `implode` rappresenta il separatore da utilizzare all'interno della nuova stringa. Nel caso di vettori associativi restituisce l'elenco dei valori ma senza indicare le chiavi.

explode() rappresenta l'inverso della precedente. Converta una stringa in un vettore enumerativo.

```
$vect = explode($delimiter, $str);
```

Le funzioni parse_str() e http_build_query()

Analoghe alle precedenti ma utilizzabili per i vettori associativi.

parse_str() converte una stringa urlencoded in un vettore associativo. `parse_str($string, $vect)`;
http_build_query() converte un vettore associativo in str urlencoded. `$s=http_build_query($vect)`

Altre funzioni

`exit()` **funzione** che termina l'elaborazione della pagina corrente
 0 rappresenta il valore default, per cui può essere omesso.

sleep(\$nSec) Mette lo script in attesa per n secondi
 usleep(\$nSec) Mette lo script in attesa per n micro secondi

Array Enumerativi (o Indexed Array, Array ad indice)

```
$vect = array();            // senza il new !!  

$vect = [];  

$vect = array('a', 'b', 'c', 'd');  

echo ($vect [3]);          // d
```

Funzioni sui vettori enumerativi

```
count($vect);            // Numero di elementi contenuti nel vettore  

array_push($vect, "e", "f", "g" );    // Consente di aggiungere nuove voci al vettore.  

Le voci possono essere variabili primitive o anche object, ma il vettore deve essere enumerativo e  

NON associativo  

$vect [] = "e";          // Analogo al precedente. Accetta anche più voci in solo colpo  

array_sum($vect);        // Somma gli elementi di un vettore contenente numeri
```

Array Associativi

Per creare un array associativo (o Literal Object singolo) non esiste la semplice notazione java script

```
var obj = { 'Peter' : 35, 'Ben' : 37, 'Joe' : 43 }
```

ma occorre utilizzare una delle seguenti possibilità:

1)

```
$age = array(); // dichiarazione, non obbligatoria  

$age['Peter'] = 35;  

$age['Ben'] = 37;  

$age['Joe'] = 43;  

echo "Peter is " . $age['Peter'] . " years old.";
```

2)

```
$age = array("Peter"=>35, "Ben"=>37, "Joe"=>43);
```

L'operatore => (**double arrow operator**), utilizzato all'interno del costruttore array(), consente di creare un nuovo vettore associativo composto da una sequenza di campi di tipo chiave – valore, in cui il valore di sinistra rappresenta la **chiave**, mentre il valore di destra rappresenta il **valore**.

Per accedere alle celle di un vettore associativo non è ammessa la sintassi javascript

```
$peterAge = $age.Peter;
```

ma è riconosciuta soltanto la seguente sintassi:

```
$peterAge = $age['Peter'];
```

Aggiunta di nuovi valori ad un vettore associativo

```
$age = array_merge($age, array("Jack"=>29, "foo"=>13));
```

Concatena i due vettori ricevuti come parametro e restituisce come risultato il vettore concatenato.

In alternativa a array_merge() è anche possibile utilizzare semplicemente l'operatore +

```
$age = $age + array("Jack"=>29, "foo"=>13);
```

Ovviamente è possibile creare un literal object come stringa, che potrà eventualmente essere trasmessa in rete senza nessuna serializzazione : `$age = '{"Peter":35, "Ben":37, "Joe":43}';`

Scansione tramite ciclo for each di un vettore enumerativo

```
$somma = 0;
$array = array(1, 2, 3, 4, 5);
foreach ($array as $valore) $somma += $valore;
```

Scansione tramite ciclo for each di un vettore associativo

L'operatore => è utilizzabile anche all'interno di un ciclo foreach per scandire gli elementi di un vettore associativo. In tal caso consente di accedere al valore corrispondente ad una certa chiave.

```
$vect = array(
    'Simone' => 29,    'Josephine' => 30,
    'Giuseppe' => 23,  'Renato' => 26,
);

foreach ($vect as $nome => $eta) {
    echo "L'utente " . $nome . " ha " . $eta . " anni\n";
    $somma += $eta;
}

foreach ($vect as $nome => $eta)
    $eta++; // incremento di 1 l'età di ogni persona
```

Nel ciclo foreach è obbligatorio impostare sia la chiave sia il valore. In caso contrario viene generato un errore di sintassi: `foreach ($vect as $nome) // errore !`

Matrici

```
$cars = array (
    array("Volvo",22,18),
    array("BMW",15,13),
    array("Saab",5,2),);

echo $cars[0][0]

for ($row = 0; $row < count($cars); $row++) {
    echo "<ul>";
    for ($col = 0; $col < count($cars[$row]); $col++)
        echo "<li>".$cars[$row][$col]."</li>";
    echo "</ul>"
}
```

Try and Catch

```
try {

}
catch (Exception $ex){
    echo($ex.getMessage());
}
```


La gestione delle date

time() restituisce il timestamp unix corrente espresso in **sec** (numero di secondi trascorsi a partire dalla data zero dei sistemi unix, 1/1/1970)

microtime(true) Uguaale alla precedente ma restituisce un float contenente anche i microsecondi. Impostando il parametro **true** il risultato viene restituito come float con i microsecondi aggiunti dopo la virgola. Impostando il parametro **false** (default) il risultato viene restituito come stringa nel formato "usec sec" con un semplice spazio come separatore

date(format, time) Converte un timestamp in una data di tipo stringa nel formato indicato **time**, facoltativo, rappresenta il timestamp da convertire in stringa. Se omesso viene utilizzato il time() corrente **format** è una stringa che rappresenta il formato in cui deve essere restituita la data. I formattatori utilizzabili sono in realtà moltissimi. Esempi:
 'd/m/Y' Y=anno a 4 cifre: y=anno a 2 cifre 'd-m-y'
 'H:i:s' H=0-24; h=0-12
 'Y-m-d H:i:s'

strtotime(str) inversa rispetto alla precedente. Riceve come parametro una data memorizzata sotto forma di **stringa** in the English date format (Y/m/d) e la restituisce sotto forma di **timestamp unix**.

date_create_from_format('d/m/Y:H:i:s', str);
 Simile alla precedente, consente però di specificare il formato in cui è memorizzata all'interno di **str** la data da convertire.
 Restituisce però la data sotto forma di oggetto datetime. Per ottenere il timestamp unix occorre applicare il metodo \$date->getTimestamp()

```
$date = date("Y-m-d", strtotime($myStringDate)); // anno da 2 a 4 cifre
```

La differenza fra due date

```
$date1 = new DateTime("2007-03-24 10:15:20");
$date2 = new DateTime("2009-06-26 11:16:10");
// L'ordine con cui si utilizzano le date è irrilevante !
$interval = $date1->diff($date2);
$years=$interval->y;
$months= $interval->m;
$days=$interval->d;
$hours=$interval->h;
$minutes=$interval->i;
$seconds=$interval->s;

// oppure
echo $interval->format('%h ore, %i minuti');
```

\$interval->days restituisce il numero complessivo di giorni che intercorrono fra le due date, senza suddividerlo in anni, mesi e giorni
days è però l'unica property di questo tipo. Non esistono ad es hours o minutes

Sintassi equivalente in forma procedurale:

```
$date1 = date_create("2007-03-24 10:15:20") //oppure new DateTime(".....")
$date2 = date_create("2009-06-26 11:16:10") //oppure new DateTime(".....")
$interval = date_diff($date1, $date2);
```

Per trovare la differenza in secondi fra due date sono sufficienti le seguenti istruzioni:

```
$time1 = strtotime('data1'); $time2 = strtotime('data2');
$differenceInSeconds = $time2 - $time1;
```

Variabili superglobali

Le variabili superglobali sono variabili predefinite direttamente accessibili in qualsiasi script sempre in forma di **vettori associativi** chiave – valore. Le variabili superglobali sono

\$_SERVER che contiene tutte variabili di sistema del server

```
foreach ($_SERVER as $nome => $valore)
    echo ($nome . " : " . $valore . "<br>");
```

\$_ENV variabili d'ambiente,

Il caricamento di questo vettore deve però essere esplicitamente abilitato all'intero del file php.ini

\$_POST, **\$_GET**, **\$_REQUEST**, **\$_COOKIE**, **\$_SESSION**

\$GLOBALS (senza underscore). Elenco di tutte le variabili definite all'interno dell'applicazione (sia quelle utente sia alcune di quelle superglobali di sistema).

La funzione **phpinfo()** consente di visualizzare un lunghissimo elenco di variabili relative alla configurazione di PHP, del server Apache, e degli altri server. Quelle utilizzate più frequentemente sono: **\$REQUEST_METHOD** metodo utilizzato per inviare la richiesta HTTP.

\$HTTP_USER_AGENT nome del browser utilizzato dal client che ha effettuato la richiesta.

\$PHP_SELF nome del file relativo alla pagina corrente. Comodo per pagine che si autorichiamano.

Incorporazione di file: include e require

E' possibile includere alla pagina web un file esterno di funzioni php utilizzando le funzioni **include** o **require**. Le due funzioni sono praticamente identiche. L'unica differenza è che, in caso di file mancante:

- **require()** will produce a fatal error and stop the script. ES require ("mioFile.php");
- **include()** will only produce a warning and the script will continue

Il require in caso di file non trovato interrompe lo script generando un errore che però può essere intercettato soltanto tramite un **try - catch**

require once() fa in modo che il file venga incluso una sola volta

E' possibile includere file con qualunque estensione (typ php, html, txt).

Il Pulsante di Submit e la gestione delle richieste HTTP

Il pulsante **<input type="submit">** fa parte dello standard HTML fin dalle primissime versioni. Il suo scopo è quello di essere utilizzato all'interno di un modulo dati (**web form**) **per inviare una richiesta al server trasmettendo automaticamente come parametri il contenuto di TUTTI i controlli posizionati all'interno del modulo stesso**. I parametri vengono passati nel cosiddetto formato URL_Encoded, cioè in formato **NomeControllo = Valore** separati tra loro da una &. (Attenzione : Nome, non ID).

- L'attributo **ACTION** del tag FORM indica la risorsa da richiedere al server in corrispondenza del SUBMIT (tipicamente una pagina HTML statica o dinamica).
In assenza dell'attributo ACTION il browser richiama automaticamente la pagina stessa.
- L'attributo **METHOD** del tag FORM indica il tipo di richiesta da inviare al server e può assumere i valori **GET** oppure **POST**. In assenza dell'attributo METHOD la modalità di default è GET

Richieste HTTP GET e HTTP POST

Quando si invia una richiesta al server digitando la URL nella barra degli indirizzi, il browser invia al server una richiesta **HTTP GET**, di solito priva di parametri che possono eventualmente essere accodati alla url direttamente all'interno della barra di navigazione.

Anche il tag `<a>` consente di inviare al server una richiesta HTTP GET.

Quando invece si invia una richiesta tramite un pulsante di **submit**, il browser invia al server una richiesta **HTTP GET** oppure **HTTP POST** a seconda del contenuto dell'attributo **METHOD** del tag form.

In caso di richiesta **HTTP GET** il contenuto di tutti i controlli presenti all'interno della form viene inviato al server concatenandolo tramite un punto interrogativo alla url indicata nell'attributo ACTION. Dunque in questo caso i parametri risulteranno visibili sulla barra di navigazione accodati all'indirizzo.

Questa tecnica va bene per richiamare direttamente dalla barra di navigazione una pagina web che prevede dei parametri e per copia / incollare una URL completa

Ad esempio: `pagina2.html?txtNome=mario&txtCognome=rossi&txtEta=16`

In caso di richiesta **HTTP POST** il contenuto di tutti i controlli presenti all'interno della form viene inviato al server inserendoli all'interno del body della HTTP REQUEST, sempre in formato URL_Encoded (NomeControllo = Valore) ma non visibili sulla barra di navigazione. Maggiore riservatezza.

Note sul passaggio dei parametri da parte del browser al server

- Tutti i controlli devono essere inseriti all'interno di un tag **form** che è **obbligatorio**.
In assenza del tag form il submit NON viene eseguito !!
- Se un controllo è **privo del name** o **disabilitato** NON viene passato al server.
- Nel caso del **ListBox**, omettendo i values viene automaticamente inviato al server il valore della option. E' comunque buona regola inserire l'ID dentro il value e visualizzare la descrizione.
- Se più controlli presentano lo stesso name (es **checkbox**) il browser accoda i valori alla queryString come parametri indipendenti: Esempio: `&hobbies=calcio&hobbies=cinema&hobbies=musica`. Quando il server va a leggere un parametro definito più volte viene letto soltanto l'ultimo valore che in pratica sovrascrive i precedenti. In questo caso, affinché il server possa leggere tutti i valori, occorre aggiungere nell'html, al termine del name, una aperta e chiusa quadra es **hobbies[]**. In questo modo i vari hobbies verranno sempre trasmessi in modo accodato come prima,
`chkHobbies[]=sport&chkHobbies[]=cinema&chkHobbies[]=musica`
però i parametri il cui nome termina con `[]` vengono trattati dal server come un vettore enumerativo all'interno del quale verranno inseriti tutti i vari valori (anche nel caso in cui il valore sia uno solo). Le parentesi quadre sono indispensabili nel caso di più controlli con lo stesso name

In alternativa si possono usare nomi diversi oppure accodare il manualmente con i campi separati da virgola: "hobbies=sport,cinema,musica". Il server potrà poi fare una explode()

- Gli attributi **action** e **method** della form possono essere eventualmente scritti all'interno come attributi del **pulsante di submit** nel modo seguente:
`<button type='submit' formmethod='get' formaction='pag2.html'> </button>`
- Quando si definisce **method='post'** (indifferentemente nella form oppure nel pulsante di submit) diventa possibile **aggiungere manualmente dei parametri GET in coda alla ACTION**. In questo modo i controlli della form verranno passati in modalità POST, mentre quelli concatenati alla URL verranno passati in modalità GET. Se si imposta invece **method='get'** i parametri della form sovrascrivono i parametri concatenati manualmente per cui l'obiettivo decade.

Esecuzione del Submit tramite procedura javascript

Volendo validare i valori inseriti prima di eseguire il submit, è possibile sostituire il pulsante di submit con un **normale button** che richiami una procedura java script la quale potrà eseguire tutti i controlli del caso e, in caso di esito soddisfacente, eseguire il submit al server nel modo seguente:

```
var frm = document.getElementById("form1");
frm.method="post";
frm.action="inserisci.php?op=salva";
frm.submit();
```

Esecuzione del Submit tramite procedura jQuery

```
var _form=$(this).parent("form"); // genitore diretto
_form.prop("method", "post");
_form.prop("action", "pagina2.php");
_form.submit();
```

Annullamento del submit

La form dispone di un evento .on("submit") richiamato in corrispondenza del submit all'interno del quale si possono scrivere delle azioni javascript da eseguire prima del submit ed eventualmente anche abbattere il submit mediante un eventuale **return false** da scriversi come ultima istruzione.

Un'altra opportunità è quella di far inviare alla procedura javascript, in parallelo, una richiesta ajax in modo da eseguire una azione aggiuntiva in corrispondenza del submit

Lettura lato server dei Parametri Get e Post

Il vettore associativo **\$_GET** contiene l'elenco dei parametri ricevuti con metodo get.

Il vettore associativo **\$_POST** contiene l'elenco dei parametri ricevuti con metodo post.

Il vettore associativo **\$_REQUEST** contiene l'elenco dei parametri sia GET che POST.

Per conoscere il metodo utilizzato dal client si può utilizzare la variabile globale

\$_SERVER["REQUEST_METHOD"] con valore automaticamente convertito in maiuscolo, es "GET".

```
if($_SERVER["REQUEST_METHOD"] == "GET" )
```

Lato server in caso di parametri, la **PRIMA cosa da fare è sempre** verificare se i parametri sono settati

```
if (isset($_GET['txtNome']))
    $nome = strtolower($_GET['nome']);
```

Se i parametri non sono settati si segnala l'errore e si termina.

Applicazioni web forms e Applicazioni web services based

Le applicazioni web di tipo client-server erano inizialmente di tipo **Web Form**, in cui una pagina in **corrispondenza del submit, richiama una'altra pagina** che presentava all'utente dei dati di maggior dettaglio. Ogni volta il server rimandava al client tutte le informazioni necessarie.

Cambiare pagina in corrispondenza di ogni richiesta delle volte può essere sveniente. D'altro lato ricaricare la stessa pagina in corrispondenza di richieste successive, oltre ad essere molto oneroso dal punto di vista del tempo, è anche molto complicato da gestire perché bisogna tenere memoria del contenuto dei controlli già compilati. Per risolvere questo problema sono nati in un certo momento i **web services** basati sulla tecnologia **Ajax** con il mantenimento dello stato decentrato sul client. Tramite Ajax la pagina richiede al server SOLTANTO i dati che gli servono, senza dover ricaricare sempre l'intera pagina. Con l'avvento di Ajax la tendenza è diventata sempre più quella di sviluppare applicazioni **Single Page**

Vantaggi: Facili da programmare, Navigazione molto fluida

Svantaggi: Lo svantaggio principale è legato all'individuazione da parte dei motori di ricerca. In una applicazione multi page quando si ricerca una certa informazione, i motori rimandano direttamente alla pagina contenente quella informazione. In una SPA questo è molto più complicato.

Sintassi Base di MySQL

Caratteristiche

DBMS client-server veloce, multiutente, affidabile, estrema sicurezza con possibilità di gestire le SSL Secure Cockets Layer per proteggere i dati durante il passaggio tra server e client. Gestisce una laborazione **multi-threading**, che permette a più client di connettersi contemporaneamente al database e di eseguire contemporaneamente interrogazioni diverse. Versione attuale 5.7 (settembre 2016).

Usa un linguaggio SQL conforme allo standard ANSI SQL92.

Formati di memorizzazione delle informazioni

Il formato **MyISAM** utilizzato inizialmente per la gestione delle tabelle non gestiva i comandi di transazione COMMIT e ROLLBACK. Dalla versione 5 in avanti è stato incluso un nuovo gestore denominato **InnoDB** in grado di gestire le transazioni con i comandi di COMMIT e ROLLBACK. L'attributo **TYPE=InnoDB** di CREATE TABLE consente di creare tabelle nel formato InnoDB anziché nel formato standard MyISAM.

Riguardo alla codifica dei caratteri i formati normalmente utilizzati sono **utf8_general** o **utf8_unicode**

Il valore di default è **latin1 / latin1_swedish_ci**

Note Operative

- Le istruzioni SQL sono completamente **case insensitive**. Sono infatti case insensitive :
 - I comandi SQL (select, from, where, etc)
 - I nomi dei campi
 - Il contenuto delle stringhe passate al where
- Sono invece **case sensitive** le chiavi ed i valori del vettore associativo restituito da fetch-assoc

Nel caso di query che ritornano campi con lo stesso nome, MySQL non ritorna MAI nomi composti (filiali.id), ma **l'ultimo campo con lo stesso nome maschera i precedenti**. E' possibile procedere in due modi:

- Estrarre dalle tabelle secondarie solo i campi interessati che hanno comunque nomi univoci:
SELECT concerti.*, citta.nomeCitta, generi.nomeGenere
- Utilizzare degli ALIAS nel caso in cui sia necessario restituire un campo avente lo stesso nome in 2 tabelle:
SELECT concerti.*, citta.nome As nomeCitta, generi.nome AS nomeGenere

Tipi di Dati

Attributi utilizzabili per tutti i tipi di campi:

NOT NULL	Indica che nel campo non sono ammessi valori NULL
DEFAULT	Indica il valore da assegnare per default al campo. Se non si specifica nulla, il DEFAULT è automaticamente NULL, indipendentemente dal tipo del campo.

I Dati Numerici

- TINYINT** – 1 byte - Valore numerico compreso tra -128 e 127. **TINYINT UNSIGNED** 0 e 255
- SMALLINT** – 2 byte - Valore numerico tra -32768 e 32767. **SMALLINT UNSIGNED** 0 e 65535
- MEDIUMINT** – 3 byte Valore numerico tra -8.388.608 e 8.388.607. **MEDIUM UNSIGNED** 0 e 16.777.215
- INT** o **INTEGER** – 4 byte Valore numerico tra -2.147.483.648 e 2.147.483.647. **INT UNSIGNED** 0 e 4Mld
- BIGINT** – 8 byte Valore numerico compreso tra $6 \cdot 2^{63}$ **BIGINT UNSIGNED** 0 e 2^{64}
- FLOAT** (m,d) – 4 byte - Floating point tra -3.402823466E+38 e -1.175494351E-38.
- DOUBLE**(m,d) o **REAL**(m,d) – 8 byte - Floating point tra -1.7976931348623157E+308 e -2.2250738585072014E-308
- DECIMAL** (m,d) o **NUMERIC**(m,d) – Valore numerico con decimali memorizzato come serie di caratteri.

Attributi per i dati numerici:

- UNSIGNED** numero senza segno. Per tutti il default è SIGNED, che può essere omissso.
- ZEROFILL** riempie con zeri l'ampiezza indicata.
 Es N INT (5) ZEROFILL . Se N = 46 verrà visualizzato come 00046.
 N può comunque contenere numeri a 6, 7 cifre.
 Es N FLOAT (6,3) ZEROFILL (6 chr complessivi, compresa la virgola e 3 chr dopo la virgola).
 Se N = 3,58 verrà visualizzato come 03,580
- AUTO_INCREMENT** Per tutti i tipi interi si può specificare l'attributo AUTO_INCREMENT che trasforma il campo in un contatore. In ogni tabella ammesso un solo campo AUTO_INCREMENT. Il campo AUTO_INCREMENT deve necessariamente essere NOT NULL ed avere associato un indice (PRIMARY KEY o UNIQUE). Come in Access, i valori utilizzati da un contatore non sono riutilizzabili, tranne nel caso in cui si svuoti completamente la tabella. L' AUTO_INCREMENT parte normalmente da 1, ma si può forzare un qualunque valore iniziale: Esempio:
 CREATE TABLE Libri
 (CodiceLibro ID INT AUTO_INCREMENT NOT NULL PRIMARY KEY,
 Autore CHAR(30), Titolo CHAR(30))
 AUTO_INCREMENT = 1000 // i codici partono da 1000
 Il campo AUTO_INCREMENT deve essere introdotto come NULL
 INSERT INTO Libri VALUES (NULL, "Pirandello", "Il fu Mattia Pascal"),

Le Stringhe

- **CHAR** (M) – Stringa a lunghezza fissa pari a M (max 255). Il campo viene completato con degli spazi. Quando si legge il contenuto del campo, gli spazi finali vengono rimossi, per cui è come se non ci fossero. CHAR senza dimensione equivale a CHAR(1)
- **VARCHAR** (M) – E' identico a CHAR, cambia solo la modalità di registrazione. Vengono memorizzati soltanto i caratteri effettivamente presenti (max 255) più un byte per la lunghezza. Eventuali caratteri eccedenti M vengono eliminati. I campi definiti CHAR con lunghezza superiore a 4 vengono automaticamente convertiti in VARCHAR.
 L'attributo **BINARY**, utilizzabile sui campi CHAR e VARCHAR, trasforma il campo in *case sensitive*. Normalmente le stringhe sono gestite come *case unsensitive*
- **TINYTEXT** – Lunghezza variabile senza il parametro M. Max 255 caratteri + 1 byte per la lunghezza
- **TEXT** – Lunghezza variabile. Max 65.535 caratteri + 2 byte per la lunghezza
- **MEDIUMTEXT** - Lunghezza variabile. Max 16.777.215 + 3 byte per la lunghezza
- **LONGTEXT** - Lunghezza variabile. Max 4.294.967.295 + 4 byte per la lunghezza
- **TINYBLOB BLOB MEDIUMBLOB LONGBLOB** – Come TEXT ma case sensitive (BLOB = Binary Large Object)
- **ENUM** ("Francia", "Germania", ...) – Campo enumerativo simile a quello di Access. Max 65.565 voci
- **SET** ("Francia", "Germania", ...) – Campo enumerativo a valori multipli Max 64 voci

Esempio

```
CREATE TABLE Nazioni (
  Nome ENUM ("Italia", "Francia", "Germania") NULL )
```

NULL indica che il campo accetta i valori NULL. Il default è NOT NULL.

Alla voce "Italia" viene automaticamente associato l'indice 1, alla voce "Francia" l'indice 2 e così via.

Al valore NULL viene associato l'indice NULL. Qualunque voce non compresa fra quelle impostate viene comunque memorizzata nel database con indice 0 (compresa la stringa vuota).

Esempio

Supponendo di partire dalla tabella creata al punto precedente, il seguente comando

```
INSERT INTO Nazioni VALUES ("Germania"), (""), ("Italia"), (NULL), ("Spagna"), (2), ("Italia")
```

Inserisce nel database **7 record** aventi i seguenti indici: 3, 0, 1, NULL, 0, 2, 1

Quando si inserisce un valore numerico, questo viene automaticamente considerato come un indice.

Per accedere agli indici da SQL si utilizza il nome del campo seguito da +0.

```
SELECT Nome+0, Nome FROM Nazioni // Visualizzo gli indici con a fianco i nomi delle nazioni.
```

Un eventuale ORDER BY esegue un ordinamento sugli indici: NULL, 0, 1, 2, 3

L'ENUM si presta bene per la costruzione di un Boolean : 1=False, 2=True

Il SET, a differenza di ENUM, consente di inserire contemporaneamente all'interno del campo più valori enumerativi, al limite anche tutti.

```
CREATE TABLE Nazioni (  
  Nome SET ("Italia", "Francia", "Germania") NULL)
```

```
INSERT INTO Nazioni VALUES ("Germania"), (""), ("Italia, Francia"), (NULL), ("Spagna")
```

Inserisce nel database 5 record aventi i seguenti indici: 3, 0, 12, NULL, 0

Dove il 12 realtà è un 1 concatenato ad un 2

DATA e ORA

- DATE – Data in formato "AAAA-MM-GG" Come separatore è ammesso ogni tipo di punteggiatura
Il tipo DATE è su 3 byte e va dal 01/01/1000 al 31/12/9999
- TIME – Ora nel formato: "hh:mm:ss" Come separatore è ammesso ogni tipo di punteggiatura
- DATETIME – Data e Ora, in formato 'AAAA-MM-GG hh:mm:ss'
- YEAR – Anno nel formato "AAAA" . Occupa un solo byte

Se si cerca di immettere in un campo DATE o TIME una informazione non valida, questa viene memorizzata come 00:00:00. Purtroppo è possibile confondere questo valore di data non valida con l'ora valida 00:00:00. E' l'applicazione client che dovrà preoccuparsi di non immettere dati non validi.

E' possibile copiare direttamente dati da un tipo ad un altro. Se copiamo ad esempio un DATETIME in un DATE l'informazione relativa all'ora viene rimossa. Se copiamo un DATE in un DATETIME l'ora viene settata a 00:00:00

Comandi DDLCreazione, Cancellazione, Selezione di un database

CREATE DATABASE [IF NOT EXISTS] biblioteca; //Crea la cartella corrispondente nel percorso /Data

DROP DATABASE biblioteca; // Cancella tutte le tabelle e tutti i dati contenuti

SHOW DATABASES; // Mostra l'elenco dei database esistenti

Creazione, Cancellazione, Selezione di una tabella

```
CREATE TABLE [IF NOT EXISTS] libri
( CodiceLibro INT NOT NULL PRIMARY KEY,
  Autore TEXT NOT NULL,
  Titolo TEXT NOT NULL,
  Prezzo FLOAT,
  Disponibile TINYINT DEFAULT 1 ); // Non esiste il Boolean
```

Si può andare a capo in qualsiasi momento. Andando a capo MySQL utilizza il prompt -> che significa che il comando non è ancora terminato ed il sistema è in attesa di continuazione del comando. In corrispondenza del punto e virgola il comando sarà considerato chiuso e verrà inviato in esecuzione.

DROP TABLE libri; // Cancella la tabella libri e tutti i dati contenuti

SHOW TABLES; // Mostra l'elenco delle tabelle contenute nel database corrente

DESCRIBE Libri; // Mostra la struttura della tabella Libri (Tipo, Not Null, Indici, Default). Ottimo.

```
ALTER TABLE libri // Modifica la struttura della tabella libri
ADD [COLUMN] CasaEditrice TEXT [FIRST | AFTER NomeColonna],
DROP [COLUMN] Prezzo,
MODIFY [COLUMN] Titolo VARCHAR(100),
CHANGE [COLUMN] Autore Scrittore VARCHAR(100), // Cambia nome e tipo
ALTER [COLUMN] Disponibile { SET DEFAULT 2 | DROP DEFAULT },
RENAME NuovoNomeTabella,
ADD PRIMARY KEY (CodiceLibro), // Trasforma CodiceLibro in chiave primaria, se ancora non lo era
ADD INDEX IndiceAutore (Autore), // Crea un IndiceAutore non univoco sul campo Autore
ADD UNIQUE IndiceTitolo (Titolo), // Crea un IndiceTitolo univoco (chiave) sul campo Titolo
DROP PRIMARY KEY
DROP INDEX IndiceAutore
DROP UNIQUE IndiceTitolo;
```

SHOW INDEXES FROM libri; // Mostra l'elenco degli indici della tabella Libri

SHOW KEYS FROM libri; // Analogo.

Creazione di tabelle temporanee

Come detto MySQL non supporta le interrogazioni annidate. Supporta però una interessante variante dell'istruzione SQL standard **SELECT INTO** che consente di memorizzare il risultato di una query all'interno di una tabella temporanea che verrà rimossa al termine della connessione corrente al database.

```
CREATE TEMPORARY TABLE tmp
SELECT codicelibro As codice, autore, titolo
FROM libri
WHERE casaeditrice = "einaudi";
```

La tabella temporanea tmp potrà essere utilizzata nelle query al pari di qualunque altra tabella reale.

Comandi DML

INSERT INTO (Query di Accodamento)

Consente di aggiungere nuovi record al fondo di una tabella esistente

```
INSERT INTO libri
VALUES
  ( 101, "Pirandello", "Il fu Mattia Pascal", 7.30, 0) ,
  ( 102, "Pirandello", "Uno Nessuno Centomila", 7.30, 0) ,
  ( 103, "Wilde", "Il ritratto di Dorian Gray", 7.30, 0);
```

Ai campi di AutoIncrement (contatori) occorre assegnare il valore NULL. Idem per i campi da lasciare vuoti. Occorre però assolutamente specificare un valore per ogni campo esistente.

Volendo specificare soltanto qualche campo, si può utilizzare la seguente sintassi, in cui l'AutoIncrement viene gestito automaticamente.

```
INSERT INTO libri (Autore, Titolo)
VALUES
  ("Pirandello", "Il fu Mattia Pascal") ,
  ("Wilde", "$myTitle"); // Anche in questo caso servono gli apici !
```

Anziché indicare valori diretti, è possibile leggerli da un'altra tabella mediante una istruzione SELECT che deve restituire tipi corrispondenti a quelli attesi da Insert Into

```
INSERT INTO Libri (CodiceLibro, Autore, Titolo)
SELECT CodiceLibro, Autore, Titolo
FROM LibriOld
WHERE Prezzo Is Not Null;
```

REPLACE (Query di Accodamento)

Molto simile a INSERT INTO, con la stessa sintassi. L'unica differenza è che se si cerca di inserire un record avente chiave primaria già esistente (o chiave UNIQUE già esistente), mentre INSERT INTO dà un errore, REPLACE cancella il vecchio record sostituendolo con quello nuovo.

UPDATE (Query di Aggiornamento)

Consente di modificare selettivamente i singoli campi di uno o più record

```
UPDATE Libri
SET Prezzo = Prezzo * 1.1,
titolo = UCASE(titolo)
[WHERE CasaEditrice = 'Einaudi'];
```

All'interno di SET si può utilizzare Null per azzerare un campo: SET Prezzo = Null;

DELETE (Query di Eliminazione)

La sintassi del comando DELETE è la stessa identica del comando SELECT. DELETE provvede a cancellare i record estratti. A differenza di Access, dopo DELETE si può facoltativamente omettere l' *.

```
DELETE FROM Libri WHERE Prezzo Is Null;
```

Se non si specifica il WHERE vengono rimossi tutti i record della tabella.

Comandi SQL

```
SELECT [DISTINCT] elenco_campi  
[FROM elenco_tabelle]  
[WHERE filtro_recordset [AND | OR | BETWEEN | LIKE | IN | IS NULL] ]  
[GROUP BY nome_colonna]  
[HAVING filtro_sui_gruppi]  
[ORDER BY nome_colonna [ASC | DESC] ]  
[LIMIT [posizione,] numero_righe];
```

DISTINCT fa in modo che vengano restituite soltanto righe *univoche*.
LIMIT è la clausola SQL standard equivalente al predicato TOP di Access. Posizione indica il numero di record da ignorare prima di iniziare la visualizzazione.

Nello standard SQL anche la clausola FROM è opzionale. E' infatti possibile utilizzare l'istruzione SELECT semplicemente per valutare un'espressione o il risultato di una funzione.

```
SELECT NOW ( );
```

Utilizzo della funzione count nello standard SQL

L'asterisco utilizzato con count chiede alla query di considerare *tutte* le righe esistenti nel gruppo e di contarle. In questo modo non si dà importanza al valore delle righe, ma solo al fatto che siano presenti o meno. Access non supporta altre opzioni. Lo standard SQL consente invece di specificare all'interno delle parentesi di COUNT il nome di un campo. In tal caso il conteggio riguarderà soltanto le righe in cui quel campo presenta un valore diverso da NULL. Inoltre all'interno delle parentesi di COUNT è possibile utilizzare anche il predicato DISTINCT, in modo da conteggiare soltanto quei record che presentano valore differente sul campo indicato

```
SELECT COUNT(DISTINCT campo) FROM tabella
```

SELECT INTO OUTFILE

Il SELECT produce una tabella virtuale che non viene memorizzata su disco. SELECT INTO consente di memorizzare il risultato di una query all'interno di un file di testo che non deve essere già esistente. La sintassi è la stessa di LOAD DATA INFILE

```
SELECT *  
  INTO OUTFILE 'FileDati.txt'  
  FIELDS TERMINATED BY ';' ENCLOSED BY '\"'  
  LINES TERMINATED BY '\n'  
FROM libri;
```

Operatore IN

Le query annidate non sono consentite. L'operatore IN consente soltanto la scelta fra un certo numero di valori indicati direttamente:

```
SELECT * FROM tabella WHERE campo IN ('valore1', 'valore2');
```

Query combinate mediante l'operatore UNION

Consente di combinare tra loro i recordset prodotti da 2 query indipendenti, recordset che devono essere chiaramente dello stesso tipo. L'operatore UNION che dispone dei due predicati ALL, che aggiunge indistintamente tutti i record del secondo recordset in coda al primo recordset, e DISTINCT, che aggiunge in coda solo i record del secondo recordset che non hanno corrispondenza all'interno del primo recordset.

Query combinate mediante l'operatore INTERSECT

Se connettiamo due comandi SELECT con un operatore INTERSECT, si ottengono soltanto quei record che sono presenti in entrambi i recordset in questione. Esempio:

```
SELECT * FROM tabella WHERE a = 3
INTERSECT
SELECT * FROM tabella2 WHERE b = 5;
```

Query combinate mediante l'operatore MINUS

Se connettiamo due comandi SELECT con un operatore MINUS, si ottengono soltanto quei record che sono presenti nel primo recordset, ma non nel secondo. Esempio:

```
SELECT * FROM tabella WHERE a = 3
MINUS
SELECT * FROM tabella2 WHERE b = 5;
```

Motori di ricerca: MATCH AGAINST

Funzionalità non standard introdotta a partire dalla versione 4.0. Si supponga di dover trovare un titolo di cui non si ricorda esattamente il nome. Per poter utilizzare MATCH AGAINST occorre innanzitutto indicizzare il campo Titolo mediante un indice FULLTEXT: `ALTER TABLE libri ADD FULLTEXT (Titolo);`

```
SELECT titolo FROM libri
WHERE MATCH (titolo) AGAINST ('Mattia Pascal');
```

L'algoritmo utilizzato per ricercare le corrispondenze è il cosiddetto algoritmo *fuzzy* che ordina i record restituiti in base al numero di parole della frase "Mattia Pascal" che hanno corrispondenza all'interno del campo titolo, e, a parità di ricorrenze, porta in alto quelle in cui la ricorrenza costituisce la parte proporzionalmente maggiore rispetto alla lunghezza del campo. Il confronto è chiaramente *case insensitive*. Inoltre non ricerca:

- Le parole di lunghezza minore uguale a 3 caratteri
- Le parole "comuni" sulla base di una apposita stop-word-list
- Le parole che compaiono in più del 50% delle righe della tabella

E' possibile anche utilizzare gli operatori + e - per ricercare solo quei record che hanno match su entrambi i nomi oppure quei record che hanno match su un nome ma non sull'altro.

```
SELECT titolo FROM libri
WHERE MATCH (titolo) AGAINST ('+Mattia +Pascal' IN BOOLEAN MODE);
WHERE MATCH (titolo) AGAINST ('-Mattia +Pascal' IN BOOLEAN MODE); // Pascal ma non Mattia
```

JOIN

Sono supportate sia la sintassi SQL standard, che definisce la condizione di join all'interno della clausola where,

```
SELECT Piloti.Nome
FROM Piloti, Nazioni
WHERE Piloti.CodiceNazione = Nazioni.CodiceNazione AND Nazioni.Nome = 'Italia';
```

sia la nuova sintassi INNER JOIN ON che consente anche gli outer join.

```
SELECT Piloti.Nome
FROM Piloti INNER JOIN Nazioni ON Piloti.CodiceNazione = Nazioni.CodiceNazione
WHERE Nazioni.Nome = 'Italia';
```

Il **Left Join** (outer join) restituisce tutti i record della tabella di sinistra.

Il **Right Join** (outer join) restituisce tutti i record della tabella di destra.

INDICI

Velocizzano le ricerche, perché le ricerche stesse vengono eseguite su tabelle ordinate.

Se l'indice è univoco, le prestazioni sono ancora migliori perché il DBMS sa che c'è un solo record da recuperare.

E' bene creare un indice per tutti quei campi su cui probabilmente verranno eseguite ricerche frequenti.

Tutte le chiavi esterne devono essere sempre indicizzate.

E' del tutto inutile indicizzare campi che contengono al loro interno pochi valori (ad esempio il campo genere M/F)

E' inutile indicizzare campi su tabelle contenenti pochi record, sui quali si può eseguire una scansione sequenziale.

Un indice può essere applicato a un singolo campo oppure a campi multipli.

MySQL supporta 4 tipi di indici:

PRIMARY KEY (UNIQUE + NOT NULL)

UNIQUE

INDEX o KEY

FULLTEXT

Su ciascuna tabella è ammesso un solo indice di tipo PRIMARY KEY (che è un indice univoco su un campo necessariamente definito NOT NULL). INDEX | KEY è un indice generico non univoco. Gli indici UNIQUE e INDEX accettano il valore NULL all'interno del campo. L'indice FULLTEXT serve per eseguire ricerche con le istruzioni MATCH AGAINST. Per ogni tabella è ammesso un limite massimo di 16 indici.

Gli indici possono essere definiti al momento del CREATE TABLE oppure aggiunti / rimossi in un secondo tempo mediante le istruzioni ALTER TABLE ADD / DROP. All'interno dell'istruzione CREATE TABLE gli indici vengono dichiarati al termine della dichiarazione dai campi, mediante le seguente sintassi:

TIPO [NomeIndice] (Elenco_colonne)

Se NomeIndice non viene specificato, l'indice assume automaticamente lo stesso nome del campo.

```
.....,  
PRIMARY KEY (CodiceLibro),  
INDEX (Autore),  
UNIQUE IndiceTitolo (Titolo, CasaEditrice);
```

Il PRIMARY KEY può anche essere specificato a fianco del campo.

Gli indici a campi multipli vanno bene quando si pensa di dover eseguire spesso delle interrogazioni che agiscono contemporaneamente su quei campi:

```
WHERE Titolo = "PIPPO" AND CasaEditrice = "PLUTO"
```

Per aggiungere l'indice in un secondo tempo, oltre alle già viste istruzioni ALTER TABLE ADD / DROP INDEX, MySQL supporta anche la sintassi non standard CREATE PRIMARY KEY, CREATE UNIQUE, CREATE INDEX utilizzata da molti database fra cui Access.

Gli indici non sono applicabili ai campi di tipo TEXT, ma solo sui campi di tipo CHAR e VARCHAR.

Per i campi TEXT è riconosciuto soltanto l'indice FULLTEXT.

Principali Operatori e Funzioni di MySQL

ASCII	Restituisce il valore ASCII di un carattere. Sintassi: ASCII('a');
CONCAT	Concatena due stringhe. Sintassi: CONCAT (stringa1, stringa2);
LENGTH	Restituisce la lunghezza di una stringa. Sintassi: LENGHT(stringa); Esempio: SELECT LENGHT('prova'); restituirà 5. Se per esempio vogliamo selezionare tutti i record che hanno Campo1 di lunghezza maggiore o uguale a 3 faremo: SELECT (*) FROM tabella WHERE LENGHT(Campo1) >= 3;
LIKE	E' una condizione del WHERE. Seleziona dei dati da un database dove il campo è simile ad uno da noi specificato. Il carattere jolly per indicare infiniti caratteri è il simbolo di percento (%), mentre il carattere jolly per indicare un singolo carattere è l'underscore (_). Il primo esempio troverà ad esempio panettone, panettiere, appannato, mentre il secondo esempio troverà pala, para, paca, etc... Primo esempio: SELECT * FROM tabella WHERE campo LIKE '%pan%'; Secondo esempio: SELECT * FROM tabella WHERE campo LIKE 'pa_a';
LOWER	Riduce in minuscolo tutti i caratteri di una stringa. Sintassi: LOWER(stringa); Esempio: SELECT LOWER('PROVA'); restituirà: prova.
UPPER	Trasforma in maiuscolo tutti i caratteri di una stringa. Sintassi: UPPER(stringa); Esempio: SELECT UPPER('prova'); restituirà: PROVA.
LEFT	Restituisce una parte della stringa partendo da sinistra. Sintassi: LEFT(stringa,numero_caratteri); Esempio: SELECT LEFT('prova',3); restituirà 'pro'.
NOW	Restituisce data e ora corrente. Se il risultato viene copiato in un campo DATE viene mantenuta soltanto l'informazione relativa alla data. Se viene copiato in un campo TIME viene mantenuta soltanto l'informazione relativa all'ora.
RIGHT	Restituisce una parte della stringa partendo da destra. Sintassi: RIGHT(stringa,numero_caratteri); Esempio: SELECT RIGHT('prova',3); restituirà 'ova'.
SUBSTRING	Seleziona un determinato numero di caratteri da una stringa Sintassi: SUBSTRING(stringa,numero_carattere_d'inizio,numero_caratteri); Esempio: SELECT SUBSTRING('prova',2,3); restituirà 'rov'.
REPLACE	Sostituisce una serie di caratteri da una stringa con degli altri caratteri. Sintassi: REPLACE(stringa, carattere_da_trovare, sostituto); Esempio: REPLACE ('prova','ov','a'); Restituirà 'praa'.
REVERSE	Restituisce la stringa scritta al contrario. Sintasi: REPLACE(stringa); Esempio: SELECT REPLACE('stringa di prova'); Restituirà: 'avorp id agnirts'
TRIM	Elimina gli spazi a sinistra e a destra di una stringa. Sintassi: TRIM(stringa):
LTRIM	Elimina gli spazi a sinistra di una stringa. Sintassi: LTRIM(stringa)
RTRIM	Elimina gli spazi a destra di una stringa. Vedere anche LTRIM e TRIM. Sintassi: RTRIM(stringa):

RAND	Genera un numero casuale di tipo float compreso tra 0 e 1.
MOD	Restituisce il resto della divisione tra due valori. Sintassi: MOD(5,2); Esempio: SELECT MOD(5,2); darà come risultato 1.
ROUND	Restituisce il valore arrotondato con un certo numero di decimali. Sintassi: ROUND(valore, numero_decimali); Esempio: SELECT ROUND(3.444245,2); Restituirà 3.44 SELECT ROUND(3.5154654,2); Restituirà 3.52
FLOOR	Restituisce il valore integer maggiore che risulta minore o uguale all'espressione numerica specificata. Sintassi: FLOOR(valore); Esempio: SELECT FLOOR(123.45), FLOOR(-123.45); Restituirà nell'ordine 123 e -124.
CEILING	Restituisce il valore integer minore che risulta maggiore o uguale all'espressione numerica specificata. Sintassi: CEILING(valore); Esempio: SELECT CEILING(123.45), CEILING(-123.45); Restituirà 124.00 e -123.00
SQRT	Esegue la radice quadrata di un valore. Sintassi: SQRT(valore);
POWER	Restituisce un valore dato in input elevato ad un determinata potenza. Sintassi: POWER(valore,potenza); Esempio: SELECT POWER(3,2); Restituisce 9.
SIN	Restituisce il seno di un valore espresso in radianti. Sintassi: SIN(valore);
COS	Funzione matematica che restituisce il coseno di un angolo espresso in radianti. Sintassi: COS(valore)
TAN	Restituisce la tangente di una espressione. Sintassi: TAN(valore);
ATAN	Calcola l'arcotangente del valore tra parentesi; Sintassi: ATAN(valore);
COT	Funzione matematica che restituisce la cotangente di un angolo in radianti Sintassi: COT(valore);
LOG E LOG10	Restituisce il logaritmo di un numero (LOG10 lo stesso, ma in base 10). Sintassi: LOG(valore); LOG10(valore);
PI	Restituisce il valore di pigreco con 6 decimali. Sintassi: PI();
RADIANS	Converte un valore da gradi in radianti. Sintassi: RADIANS(valore);

Connessione tra PHP e MySQL

L'oggetto mysqli

Le cosiddette **mysqli functions** sono estensioni fornite nativamente fin dalle prime distribuzioni di php, ma oggi non più mantenute e sconsigliate. PHP è un linguaggio **procedurale** nato molto prima rispetto all'avvento della OOP. Nel momento in cui sono risultati evidenti i vantaggi della OOP (soprattutto in sede di sviluppo collaborativo), invece di riscrivere da zero l'architettura di PHP, come è stato fatto ad esempio da Microsoft con dot NET, nel caso di php sono state aggiunte semplicemente delle patch in modo da far 'sembrare' OOP ciò che in realtà continua ad essere procedurale.

In quest'ottica le **mysqli functions** sono state sostituite dalla libreria **MySQLi** che significa **MySQL Improved** e che presenta una doppia interfaccia, sia **procedurale** che **object oriented**.

- l'approccio procedurale utilizza una sintassi molto simile alle mysqli functions originali.
- l'approccio object oriented utilizza un costruttore che restituisce un oggetto \$connection dotato di proprietà e metodi. Per accedere a proprietà e metodi dell'oggetto si utilizza l'operatore -> (come in C++) per il semplice motivo che si è voluto mantenere il puntino come operatore di concatenamento.

A livello di prestazioni non ci sono differenze fra la programmazione procedurale ed object oriented, ed è quindi possibile scegliere liberamente l'uno o l'altro. Permangono comunque i vantaggi tipici della programmazione *object oriented*: codice più facilmente riutilizzabile e più facilmente comprensibile.

In seguito sono state aggiunte anche le **PDO** (Php Data Object), completamente OOP ed in grado di interfacciare non solo MySQL ma ben 12 dbms (compreso SQL Server).

Proprietà e Metodi dell'oggetto mysqli (versione procedurale)

\$con = mysqli_connect(); stabilisce la connessione con MySQL. Restituisce false se fallisce (ad es se il dbms non risponde, oppure il database indicato non è presente, o se non ho i diritti di accesso)

\$con = mysqli_pconnect(); stabilisce una connessione persistente che non viene chiusa né alla chiusura dello script né dal metodo close(). Viene abbattuta dal MySQL server e seguito di un tempo di inattività dato dalla variabile di mysql *wait_timeout*

mysqli_select_db(\$database [, \$con]); seleziona il database indicato (uno solo per ogni connessione)

\$rs = mysqli_query(\$con, \$sql); Esegue il comando SQL e restituisce il recordset corrispondente. (che potrebbe eventualmente avere un num_rows=0). In caso di errore restituisce false. Se il comando è di tipo DDL o DML restituisce true se ok oppure false in caso di errore). \$con è la connessione da usare
mysqli_errno(\$con); Codice di errore relativo all'ultima istruzione mysqli eseguita
mysqli_error(\$con); Messaggio di errore relativo all'ultima istruzione mysqli eseguita
\$qta = mysqli_num_rows(\$rs); restituisce il numero di record appartenenti al recordset.

\$rs = mysqli_multi_query(\$con, \$sql); Consente di eseguire query multiple memorizzate all'interno di \$sql e separate da punto e virgola

\$record= mysqli_fetch_row(\$rs); restituisce i campi del record corrente in un vettore **enumerativo**
\$record = mysqli_fetch_assoc(\$rs); restituisce i campi del record corrente in un vettore **associativo**
\$record = mysqli_fetch_array(\$rs); restituisce i campi del record corrente nel vettore accessibile sia come vettore enumerativo sia come vettore associativo `record['nomeCampo']`. In realtà per ogni campo restituisce due property, una chiave "0" e l'altra con chiave "nomeCampo", per cui se il risultato deve essere restituito tramite uno stream json crea soltanto problemi. Molto meglio fetch_assoc().
\$json = mysqli_fetch_all(\$rs, MYSQLI_ASSOC); restituisce l'intero recordset sotto forma di vettore enumerativo di object. Estremamente comodo.

mysqli_result(\$rs, \$i, "Autore"); Restituisce il campo indicato all'interno dell'i-esimo record

`$num=mysqli_affected_rows()`; restituisce il numero di record modificati da un comando DML
`$id=mysqli_insert_id()`; restituisce il codice assegnato ad un contatore dopo un'istruzione INSERT
`mysqli_host_info($con)` Informazioni relativa alla connessione corrente

Esempio procedurale

```
$con=mysqli_connect('localhost', 'root', '', 'dbName');
if (mysqli_connect_errno($con))
    die ("Errore Connessione:". mysqli_connect_errno($con) . mysqli_connect_error($con));
else
    set_charset($con, "utf8");
$sql = "select * from libri";
$rs = mysqli_query($con, $sql);
if(!$rs)
    die("Errore query: " . mysqli_errno($con) . "-" . mysqli_error($con));
$qta = mysqli_num_rows($rs);
while($riga = mysqli_fetch_assoc($rs)) {
    echo("<TR> <TD>" . $riga["Autore"] . "</TD> </TR>");
}
mysqli_close($con);
```

Esempio Object Oriented

```
define('DBHOST', 'localhost');
define('DBUSER', 'root');
define('DBPASS', '');
define('DBNAME', 'db_dischi');
$con = new mysqli(DBHOST, DBUSER, DBPASS, DBNAME);
if ($con->connect_errno)
    die("Errore Connessione: " . $con->connect_errno . $con->connect_error);
else
    $con->set_charset("utf8");

$sql = "select * from libri";
$rs = $con->query($sql);
if(!$rs)
    die("Errore query: " . $con->errno . "-" . $con->error);

$qta = $rs->num_rows; // property
while ($riga = $rs->fetch_assoc()) {
    echo("<TR>");
    echo("<TD>".$riga["Autore"]."</TD> <TD>".$riga["Titolo"]."</TD>");
    echo("</TR>");
}
$con->close();
```

Try and Catch

Il try and catch è un costrutto nato con la programmazione OOP. L'oggetto **mysqli**, pur presentando una interfaccia OOP, rimane di base procedurale per cui, ad esempio, nel caso di errore nella creazione di una nuova connessione, il metodo `mysqli_connect()` non genera una eccezione, ma si limita a settare le proprietà `connect_errno` e `connect_error`. Allo stesso modo il metodo `mysqli_query()`, in caso di errore, non genera una eccezione ma restituisce **null**.

Per fare in modo che i vari metodi **mysqli** generino una eccezione in caso di errore, occorre eseguire preventivamente la seguente dichiarazione:


```
mysqli_report(MYSQLI_REPORT_ERROR | MYSQLI_REPORT_STRICT);
try{
    $con = new mysqli('localhost', 'root', '', 'dbName');
}
catch (mysqli_sql_exception $ex) {
    die ("Errore connessione db: <br>" . $ex->getMessage());
}
```

Gestione Caratteri accentati e apici singoli/doppi

Due le operazioni da eseguire:

- All'interno della sezione head inserire la seguente riga (**case sensitive**, **SOLO per i servizi**)
`<?php header('Content-type: text/html; charset=utf-8'); ?>`
 Nel caso di pagine php va inserito PRIMA del tag `<!DOCTYPE html>`
- Dopo aver stabilito la connessione con il database (prima di lanciare le query) inserire una delle seguenti righe del tutto equivalenti :
`$con->set_charset("utf8");`
`$con->query("set names 'utf8'");`

Le righe precedenti risolvono il problema della lettura dei dati da database, ma rimane ancora un problema. Quando un valore, letto correttamente da database, viene utilizzato come **valore di un qualunque attributo html** (ad esempio come value di un textbox) eventuali apici o doppi apici interni alla stringa possono interferire con l'apice del value medesimo e vengono interpretati come identificatore di chiusura della stringa interna. Es: `<input type='text' value="Gigi d'Alessio" >`

Questa situazione può essere risolta usando le funzioni `htmlentities` oppure `htmlspecialchars` che convertono i caratteri speciali in htmlEntities. Convertono `>` in `>`; apice in `'`; etc
 L'opzione `ENT_QUOTES` fa sì che vengano convertiti sia apici singoli che apici doppi.

```
$value= htmlentities($value, ENT_QUOTES);
$value= htmlspecialchars($value, ENT_QUOTES);
echo ("

```

Il problema del SQL injection

SQL injection è una tecnica di code injection, usata per attaccare applicazioni di gestione dati, con la quale vengono inseriti delle stringhe di codice SQL malevole all'interno di campi di input in modo da modificare il significato della stringa sql. Si supponga che all'interno di un TextBox un utente inserisca `user = nn`

L'applicazione potrebbe impostare una query del tipo:

```
select * from utenti where user = '$txtUser';
```

che diventa

```
select * from utenti where user = 'nn';
```

Se l'utente invece di `nn` imposta come username `xx' or 'A' = 'A' --`

-- significa che tutto ciò che segue è commentato. Il comando sql precedente diventerebbe:

```
select * from utenti where user = 'xx' or 'A' = 'A' -- ';
```

La query andrebbe sempre a buon fine.

Non solo, ma se l'utente inserisse come username `xx'; DROP TABLE utenti; --`

Il comando sql precedente diventerebbe

```
select * from utenti where user = 'xx'; DROP TABLE utenti; --';
```

Accorgimenti per evitare SQL injection

- Impedire l'utilizzo di apici e doppi apici all'interno dei Text Box. Non sempre possibile. Se il mio nome è Gigi D'Alessandro come posso fare ?
- utilizzare **per ogni parametro** la seguente funzione la quale **antepone un backslash davanti a: apice, doppio apice backslash**

```
$s = mysqli_real_escape_string($con, $s); // procedurale  
$s = $con->real_escape_string($s);      // object oriented
```

Opera solo in presenza di connessione. In assenza di connessione restituisce stringa vuota.

Questa operazione deve essere eseguita SEMPRE su tutti i parametri, prima di passarli alla query

Nel caso in cui la stringa intercettata tramite `$con->real_escape_string` dovesse essere visualizzata, occorre rimuovere il carattere \ introdotto da `$con->real_escape_string`. Per fare ciò:

```
$s=stripslashes($s);
```

Redirect

Il redirect lato server è una operazione che consente al server di restituire una pagina differente rispetto a quella richiesta. Molto utilizzato ad esempio in fase di login (se il login va a buon fine -> redirect alla prima pagina reale del sito). In php per eseguire il redirect si usa la funzione `header()` che, tramite l'utilizzo dell'oggetto `location`, richiede al browser il caricamento di una pagina diversa rispetto a quella attuale:

Come path si può passare o un **percorso assoluto** (che deve iniziare con uno `/`), oppure un **percorso relativo** a partire dalla posizione della pagina corrente.

```
header("location: pagina2.php?codice=5"); // percorso relativo  
header("location: /4B/ese03/pagina2.php?codice=5"); // percorso assoluto  
header("location: http://www.example.com") // collegamento esterno
```

Notare che:

- La pagina viene comunque inviata al client che leggerà la header ricevuta e richiederà la nuova pagina al server
- le `echo()` antecedenti al `redirect` non vengono visualizzate dal browser.
- **Il redirect non termina l'elaborazione dello script in corso** per cui se più avanti nel codice c'è un'altra istruzione di redirect, la successiva copre la precedente, che dunque NON verrà eseguita

La funzione md5()

Tecnica utilizzata per evitare che estranei possano intercettare una password attraverso la rete. Invece di trasmettere la password si calcola la sua impronta MD5 (irreversibile) e si trasmette in rete e si memorizza all'interno del DB l'impronta MD5. L'impronta md5 è una sequenza di **16 bytes binari**, generalmente restituiti come stringa di 32 caratteri esadecimali-

```
string md5(string $str)
```

```
$str = 'apple';  
if (md5($str) == '1f3870be274f6c49b3e31a0c6728957f') .....
```

Altra funzione di hash: sha256

```
$password = hash ("sha256" , $password, false); // true=return binary data
```

Il simbolo @

Anteponendo il simbolo `@` davanti ad una qualunque espressione/ funzione PHP, gli eventuali messaggi di errore prodotti da quell'espressione vengono soppressi:

```
$conID = @mysql_connect()
```

Creazione di un web service tramite PHP

Apache è nato per inviare pagine html/php. Volendo creare un semplice web service è sufficiente utilizzare adeguatamente la funzione `echo` in modo che vada a scrivere dentro la pagina NON tag html ma uno stream json che sarà restituito al termine del servizio.

Siccome però il rendering finale viene inviato sempre come “pagina”, la pagina dovrà **contenere un UNICO SCRIPT PHP** facendo molta attenzione a non lasciare spazi prima e dopo dello script, spazi interpretati come caratteri html che verrebbero aggiunti alla stringa di risposta.

La prima riga del servizio deve essere sempre il set di una intestazione che specifichi **content-type** e **charset** della risposta. Prestare attenzione anche al fatto che il file sia salvato in UTF-8 **senza BOM**

```
<?php
header('Content-Type: application/json; charset=utf-8'); // oppure
header('Content-Type: application/xml; charset=utf-8'); // oppure
header('Content-Type: text/plain; charset=utf-8'); // utf-8 case sensitive
```

Esempio di servizio PHP

```
<?php
header('Content-type: application/json; charset=utf-8');
// 1. controllo parametri
if (!isset($_GET["codStudente"]) || !is_numeric($_GET["codStudente"])){
    http_response_code(400);
    die("Parametro mancante o non valido: codice studente");
}

// 2. Connessione e lettura parametri
$dbName = "myDatabase";
$con=creaConnessione($dbName);
$codStudente=$con->real_escape_string($_GET["codStudente"]);

// 3. costruzione della risposta
$sql = "select * from studenti where cStudente=$codStudente";
$data = eseguiQuery($con, $sql);
echo(json_encode($data));

// 4. Chiusura connessione
$con->close();
?>
```

Considerazioni sulla PHP header

La **header** iniziale serve ad avvisare il browser sul formato della stringa che riceverà. Nel caso di \$.ajax, impostando il campo **DataType="json"**, essa provvederà automaticamente ad eseguire il parsing della stringa json ricevuta, indipendentemente dal fatto che l'intestazione http contenga text/plain oppure application/json. Pertanto, in questo caso, l'intestazione http diventa poco significativa.

Creazione e serializzazione di uno stream JSON

La funzione `json_encode($obj)` serializza un JSON in stringa.

La funzione `json_decode($str, true)` parsifica una stringa JSON e restituisce l'object php corrispondente. Il secondo parametro `true` fa sì che l'object restituito venga trasformato in vettore associativo. Restituisce `null` in caso di stringa json NON valida.

Esempi:

Vettore enumerativo di stringhe

```
$vect = array("Peter", "Ben", "Joe");  
$s = json_encode($vect);  
echo ($s); // '["Peter", "Ben", "Joe"]'
```

Vettore associativo (object singolo)

```
$vect = array("Peter"=>35, "Ben"=>37, "Joe"=>43);  
$s = json_encode($vect);  
echo ($s); // '{"Peter":35, "Ben":37, "Joe":43}'
```

Oggetto singolo costruito passo passo

```
$json = array();  
$json["ok"] = true;  
echo json_encode($json);
```

Vettore di objects

```
$vect = array(array("Peter"=>35), array("Ben"=>37), array("Joe"=>32));  
$s = json_encode($vect);  
echo ($s); // '[{"Peter":35}, {"Ben":37}, {"Joe":43}]'
```

Restituisce il vettore di stringhe serializzato : `"['{"Peter":35}', '{"Ben":37}', '{"Joe":43}']"`

Serializzazione di un recordset restituito da mySqli

Il metodo `$rs->fetch_assoc()` restituisce una alla volta le righe del recordset sotto forma di vettore associativo contenente i vari campi che costituiscono la riga :

```
{ "campo1" : valore1, "campo2" : valore2 }
```

Volendo trasmettere un **singolo** record come stringa json, la soluzione più semplice è quella di serializzare direttamente il vettore associativo restituito da `fetch_assoc()` :

```
if ($rs->num_rows == 1)  
    echo(json_encode( $rs->fetch_assoc() ));
```

In caso di record multipli è sufficiente costruire un **vettore enumerativo** di record come indicato di seguito

```
[  
    { "campo1" : valore1, "campo2" : valore2 },  
    { "campo1" : valore1, "campo2" : valore2 },  
    { "campo1" : valore1, "campo2" : valore2 },  
    { "campo1" : valore1, "campo2" : valore2 } ]
```

e poi serializzarlo:

```
$vect = array(); // vettore enumerativo di record
while($riga = $rs->fetch_assoc())
    array_push($vect, $riga);
echo(json_encode($vect));
```

In alternativa è possibile serializzare direttamente il vettore enumerativo restituito da

```
echo(json_encode($mysqli_fetch_all($rs, MYSQLI_ASSOC));
```

Costruzione degli oggetti post e posts

Un approccio più strutturato potrebbe essere quello di incapsulare ogni singolo record all'interno di un oggetto JSON avente come nome ad esempio **post** e come valore l'intero record in formato json.

Tutti i vari oggetti **post** { 'post' : {} } potrebbero poi essere inseriti all'interno di un vettore **enumerativo**

```
// $riga = { 'nome': 'pippo', 'residenza': 'fossano', eta: 16 }

$vect = array(); // vettore enumerativo
while($riga = $rs->fetch_assoc()) {
    // creo un object con un unico campo post, cioè { 'post' : {} }
    $obj = array('post' => $riga);
    // accodo l'object precedente all'interno di un vettore enumerativo
    array_push($vect, $obj);
}
```

\$vect è un vettore enumerativo avente il seguente formato :

```
[ { "post": { "campo1" : valore1, "campo2" : valore2 } },
  { "post": { "campo1" : valore1, "campo2" : valore2 } },
  { "post": { "campo1" : valore1, "campo2" : valore2 } } ]
```

Al termine il vettore enumerativo potrebbe a sua volta essere incapsulato all'interno di un ulteriore oggetto JSON avente come nome **posts** e come valore il contenuto dell'intero vettore.

```
echo json_encode(array('posts' => $vect));
```

viene cioè creato il seguente JSON object:

```
{ "posts" : [...] };
```

Lettura Java Script dell'oggetto posts

```
for (i=0; i<json.posts.length; i++) {
    var post = json.posts[i].post;
    option = document.createElement("option");
    option.setAttribute("value", post.ID);
    option.innerHTML = post.Nome;
    myList.appendChild(option);
}
```

Nota: Passaggio dei parametri in formato json

Lato PHP i parametri passati come stringa JSON non sono più accessibili all'interno dei vettori **\$_POST** e **\$_REQUEST**, ma occorre andarli a leggere manualmente nel body della http request:

```
$params = (file_get_contents('php://input')); // stringa
$params = json_decode($params, true);
for($i=0; $i<count($params); $i++) // vengono aggiunti a Request
    $_REQUEST = array_merge($_REQUEST, $params[$i]);
```

Creazione e serializzazione di uno stream XML

Per inviare come risposta uno stream xml, il modo più semplice è quello di creare una semplice **stringa xml** con concatenamenti successivi.

Volendo invece lavorare con gli oggetti, fino alla versione PHP 4, per il parsing di documenti XML si utilizzava la libreria **DOM** praticamente identica a quella di java script. Principali metodi :

```
$xmlDoc = new DOMDocument();  
$xmlDoc->load("file.xml");  
$xmlDoc->loadXML(xmlString); // parsing della stringa xml  
xmlString = $xmlDoc->saveXML(); // serializzazione dell'oggetto xmlDoc
```

In PHP 5 è stato aggiunto un nuovo strumento basato su un'interfaccia ad oggetti, SimpleXML, forse più semplice ma con un'interfaccia diversa rispetto al classico DOM.

Esempio basato su DOM library

```
<?php  
$xmlDoc = new DOMDocument();  
// creo una stringa XML da trasformare poi in object  
$xml = "<dischi>";  
$xml .= "<disco><autore>Battiato</autore><titolo>Centro</titolo></disco>";  
$xml .= "<disco><autore>Zuccherò</autore><titolo>Baila </titolo></disco>";  
$xml .= "</dischi>";  
$xmlDoc->loadXML($xml);  
$root = $xmlDoc->documentElement;  
  
// Esempio 1  
foreach ($root->childNodes AS $disco)  
    foreach ($disco->childNodes AS $item)  
        echo($item->nodeName . " = " . $item->textContent . "<br>");  
  
// Esempio 2  
$disco = $root->firstChild;  
echo($disco->nodeName . "<br>"); // disco  
$item = $disco->firstChild; // autore  
echo($item->nodeName . " = " . $item->textContent . "<br>");  
$item = $item->nextSibling; // titolo  
echo($item->nodeName . " = " . $item->textContent . "<br>");  
  
// Esempio 3  
$dischi = $root->childNodes;  
$items = $dischi[1]->childNodes; // disco di zucchero  
echo($items[0]->nodeName . " = " . $items[0]->textContent . "<br>");  
echo($items[1]->nodeName . " = " . $items[1]->textContent . "<br>");  
  
// Esempio 4 : creazione di un nuovo disco  
$disco = $xmlDoc->createElement('disco');  
$disco->setAttribute('tipo', 'nuovo');  
$root->appendChild($disco);  
$autore = $xmlDoc->createElement('autore', "Lucio Dalla");  
$disco->appendChild($autore);  
$titolo = $xmlDoc->createElement('titolo', "Piazza Grande");  
foreach ($root->childNodes AS $disco) {  
    echo($disco->getAttribute("tipo") . "<br>");  
    foreach ($disco->childNodes AS $item)  
        echo($item->nodeName . " = " . $item->textContent . "<br>");  
}
```

Principali status code ritornati dalle http response

Se si richiede un dominio inesistente, il browser risponde con “**Impossibile raggiungere il sito**”
Viceversa se il dominio esiste e non si verificano errori, il web server restituisce automaticamente uno status_code = **200** che significa OK. In caso di errore può utilizzare il comando

```
http_response_code(500);
```

per restituire un apposito codice di errore. I codici di errore più comuni sono i seguenti:

- **Error 404 Page Not Found** Significa dominio esistente, ma risorsa non trovata
- **Error 401 Unauthorized** Significa credenziali non valide
- **Error 403 Forbidden** (credenziali ok ma accesso alla risorsa non consentito ad esempio in caso di sessione o token non validi o scaduti)
- **Error 500 Internal Server Error** Significa errore nel codice lato server
- **Error 503 Database Connection Error**
- **Error 400 Bad Request** Il server non riesce ad interpretare la richiesta.
Ad esempio per un **errore sintattico** nei parametri, a causa di un **parametro mancante** oppure di un valore atteso numerico ma in realtà **non numerico**
- **Error 422 Unprocessable Entity** The server understands the content type of the request entity and the syntax of the request entity is correct (thus a 400 (Bad Request) status code is inappropriate) but was unable to process the contained instructions. Ad esempio per un **errore semantico** nei parametri come ad esempio un parametro fuori valore, ad esempio eta < 18

L'oggetto SESSION

Una sessione di navigazione è una sequenza di richieste HTTP logicamente correlate tra loro, provenienti da uno stesso client e dirette verso uno stesso server

L'oggetto **Session** consente di memorizzare lato server alcune informazioni specifiche di riconoscimento del client, in modo che, in corrispondenza delle prossime connessioni, il server possa riconoscere il client.

All'interno dell'oggetto **Session** di ogni singolo client, si possono definire della **variabili di sessione** relative a qual client.

Meccanismo di riconoscimento di un client

- Quando il server riceve una nuova richiesta, completamente sconosciuta e anonima, viene creato un nuovo **oggetto Session** memorizzato lato server ed accessibile attraverso un **SessionID univoco**.
- Assieme all'oggetto sessione, il server crea anche un cookie, in cui scrive il **SessionID** relativo alla sessione appena creata. Questo cookie viene inviato al browser assieme alla pagina richiesta.
- Il browser, nel momento in cui riceve la pagina, riceve e memorizza anche il cookie. Da questo momento in poi il browser spedisce il cookie al server tutte le volte che verrà fatta una nuova richiesta a quel server.
- Attraverso il cookie, ad ogni richiesta il server è in grado di recuperare l'oggetto Session relativo a quell'utente e contenente tutte le informazioni associate alla sessione.
- Come tutti i cookies, anche il cookie relativo al Session ID ha un **timeout di default pari a 0**, che significa che il cookie verrà mantenuto nella memoria del browser senza limiti temporali, ma solo fino alla chiusura del browser. Nel momento in cui il browser viene chiuso il cookie non viene salvato su disco ma semplicemente rimosso. Per cui se si chiude e riapre il browser la sessione viene terminata.

Creazione e Accesso all'oggetto Session

Il metodo `session_start()`; crea un oggetto SESSION associato all'indirizzo IP del client (se ancora non esiste), altrimenti accede all'oggetto SESSION esistente.

I metodi `session_name()` `session_id()` consentono di accedere in lettura / scrittura, lato server, a Nome e Id della sessione corrente. Il Session_Name è la chiave utilizzata per la creazione del cookie, mentre il Session_Id sarà il valore memorizzato all'interno del cookie.

Browser con cookies disabilitati

Se il browser ha i cookie disabilitati come si fa a gestire la sessione sul server? • Si deve fare in modo che tutti i percorsi utilizzati dal browser appendano all'url l'identificativo di sessione mediante una tecnica detta di **URL rewriting** o **cookies munching** (sbriciolamento dei cookies). A tal fine, prima di generare l'oggetto Session, occorre eseguire le seguenti impostazioni:

```
ini_set("session.use_cookies", 0);
ini_set("session.use_only_cookies", 0);
ini_set("session.use_trans_sid", 1);
ini_set("session.cache_limiter", "");
```

Con queste impostazioni tutte le URL interne verranno automaticamente completate aggiungendo in coda il session ID corrente opportunamente codificato. Se la navigazione dell'utente procederà attraverso gli hyperlink la sessione verrà mantenuta: il client riproporrà al server l'identificatore di sessione in ogni richiesta. La sessione viene invece persa se:

- l'utente scrive l'url sulla barra degli indirizzi del browser o utilizza un bookmark
- l'utente usa il tasto indietro ritornando alla pagina precedente senza riscrivere il session_id.

Il problema però è capire lato server se i cookie sono abilitato oppure no (how to check cookies enabled)

Gestione delle variabili session

Lato server, all'interno dell'oggetto Session il programmatore può definire delle **variabili Session** specifiche per quell'utente. Le variabili di sessione mantengono il valore tra una chiamata e l'altra e mantengono il loro valore fino a quando la sessione rimane attiva, cioè fino a quando:

- La sessione viene esplicitamente **terminata** dal server
- Scatta un certo **timeout** di inattività

Dopo aver acceduto all'oggetto SESSION tramite il metodo `session_start()`; è possibile creare / leggere variabili di sessione con la solita sintassi dei vettori associativi:

```
$_SESSION["nome"] = "pippo";
$nome = $_SESSION["nome"];
```

Terminazione esplicita di una Sessione

Per terminare forzatamente una sessione si può scrivere:

```
session_start();      // accede all'oggetto session
session_unset();      // rimuove tutte le variabili session
session_destroy();    // rimuove l'oggetto session
```

Prima del destroy è raccomandata anche la rimozione dell'eventuale cookie di sessione:

```
if (isset($_COOKIE[session_name()]))
    setcookie(session_name(), '', time()-42000, '/');
```

Quando si esegue un nuovo login, per essere sicuri di creare una nuova session, si può usare il metodo `session_regenerate_id(true)`; che di fatto termina una eventuale sessione ancora aperta.

Durata di una Sessione

La sessione deve avere un tempo massimo di vita e deve essere automaticamente terminata dal server dopo un certo timeout di inattività da parte del client. La property `session.gc_maxlifetime` consente di impostare un tempo **espresso in secondi** scaduto il quale la sessione verrà rimossa dalla memoria.

```
ini_set('session.gc_maxlifetime', 600); // 10 min
```

In realtà allo scadere di questo tempo la sessione verrà semplicemente contrassegnata come “garbage” ma non viene subito rimossa, per cui la durata effettiva è sostanzialmente non deterministica.

Questa impostazione garantisce comunque un tempo di vita minimo della sessione. Per quanto riguarda invece l'upper bound, di solito è preferibile gestire manualmente il timeout.

All'inizio di ogni servizio, si può aggiungere il seguente codice in cui, se l'ultima richiesta è stata fatta da più di 10 minuti, la sessione viene automaticamente chiusa

```
session_start();

if ( !isset($_SESSION['codUtente']) ) {
    http_response_code(403);
    die("Sessione non riconosciuta");
}

else if ( !isset($_SESSION['scadenza']) || time() > $_SESSION['scadenza'] ) {
    session_unset();
    session_destroy();
    http_response_code(403);
    die("Sessione scaduta");
}

else {
    $_SESSION['scadenza'] = time() + 600; // 10 min
    setcookie(session_name(), session_id(), $_SESSION['scadenza'], "/"); }
```

Validazione tramite session (server based authentication)

La validazione tramite session è stata molto utilizzata nelle applicazioni web form ma è poco adatta ad essere utilizzata nelle applicazioni basate su ajax. Infatti va contro i principi base dei servizi REST, secondo i quali il server non deve mantenere memoria degli accessi precedenti (servizi stateless).

Volendo comunque utilizzare la validazione tramite session anche con Ajax si può procedere in questo modo:

- la pagina **login.html**, in corrispondenza del submit, richiama un servizio **login.php** che, in caso di credenziali valide, imposta la session e restituisce un messaggio di “OK”
- Ogni servizio lato server richiama una procedura **controllaSession()** che esegue il codice precedente verificando l'esistenza e la validità della Session
 - in caso di credenziali valide, aggiorna la session e restituisce i dati richiesti
 - in caso di credenziali non valide il servizio restituirà al client un codice di errore **403** di sessione scaduta.
- In caso di codice 403 la pagina corrente provvede a richiamare la pagina login.html

L'unico neo è che all'avvio per un attimo viene visualizzata index.html e subito dopo viene visualizzata login.html. Questo problema può essere risolto semplicemente utilizzando una specie di splash screen che rimane visualizzato finché non arriva la risposta del server. Solo in caso di “ok” si provvederà a visualizzare il contenuto della pagina html.

Impostazione dei cookies

Per la scrittura dei cookies in php è possibile utilizzare la seguente sintassi :

```
session_start(); // da fare PRIMA di setcookie()  
setcookie(session_name(),session_id(),time()+3600, "/" );
```

L'aggiornamento del cookie di sessione è importante in modo da poter impostare la sua scadenza con lo stesso valore della scadenza della session. Diversamente l'impostazione di default per la scadenza del cookie sarebbe zero, ed il cookie verrebbe automaticamente rimosso in corrispondenza della chiusura del browser.

Impostazione dei cookies di sessione tramite la funzione **session_set_cookie_params()**

Questa funzione deve essere utilizzata immediatamente prima di **session_start()** e serve a modificare i parametri di default utilizzati dal server per l'invio dei cookies di sessione, **in modo particolare il tempo di vita del cookie**.

Presenta gli stessi parametri del metodo **setcookie()**. Il tempo di vita di un cookie per default vale **zero**, per cui quando l'utente chiude il browser il cookie viene rimosso e la sessione viene chiusa. Impostando invece un tempo diverso da 0, questo fa sì che, alla chiusura del browser, i cookies vengano salvati su disco con relativo mantenimento della sessione anche in corrispondenza della chiusura e riapertura del browser. Il tempo di vita del cookie può anche essere diverso rispetto alla durata della session:

- Se il tempo di vita del cookie è **maggiore** rispetto alla session non è un problema. Anche se il cookie rimane 'vivo' sul browser ma la sessione è scaduta, quando il cookie relativo al session ID verrà inviato dal browser al server, il session_id non verrà più riconosciuto come valido.
- Se il tempo di vita del cookie è **minore** rispetto alla session, in corrispondenza della prossima chiamata il browser NON invierà più il cookie ed il server, in corrispondenza di una richiesta priva di cookie, non riuscirà ad agganciare la session che risulterà pertanto undefined.

Esempio

```
session_set_cookie_params(600);  
session_set_cookie_params(600, "/", "localhost", false, true);  
session_start();
```

La funzione **session_set_cookie_params** ha effetto soltanto all'interno della pagina in cui si trova.

Lettura delle impostazioni dei cookies di sessione

Per leggere le impostazioni correnti relative ai cookies di sessione è disponibile anche il metodo:

```
$currentCookieParams = session_get_cookie_params();  
session_set_cookie_params(  
    $currentCookieParams["lifetime"], // o altro valore  
    $currentCookieParams["path"],     // o altro valore  
    $currentCookieParams["domain"],    // o altro valore  
    $currentCookieParams["secure"],    // o altro valore  
    $currentCookieParams["httponly"] ); // o altro valore
```

Principali funzioni per la gestione dei files

`__FILE__` contiene il path completo assoluto sul server del file corrente (a partire da `htdocs`)
`__DIR__` contiene il path completo assoluto sul server della cartella corrente (a partire da `htdocs`)
`getcwd()`; uguale a `__DIR__`
`$dirname = dirname ($filepath)`
Restituisce il path completo assoluto sul server del file indicato, eliminando il nome contenuto dopo l'ultimo slash
`$filename = basename ($filepath)`
Restituisce il nome del file (con estensione) contenuto dopo l'ultimo slash, eliminando in pratica il path antecedente. Elimina anche l'eventuale slash iniziale.
`file_exists ($filepath)`
Verifica se il file esiste all'interno del percorso relativo indicato (a partire dalla cartella corrente).
`$ext = pathinfo ($file, PATHINFO_EXTENSION);`
Estrae l'estensione del file ricevuto come parametro

Lettura e Scrittura di un file di testo

```
$handle = fopen (string $filename, string $mode)
    Apre il file nella modalità indicata che può essere: r = sola lettura, r+ = lettura e scrittura,
w = sola scrittura (se il file non esiste lo crea automaticamente), w+ = lettura e scrittura cancellando
    completamente il vecchio contenuto, a = append in coda al file
fgets ($handle, $n); // Legge una riga alla volta il contenuto del file. Il secondo parametro n indica
    quanti caratteri leggere a partire dalla posizione corrente. Se omissso viene letta l'intera riga
fwrite ($handle, $line); // Accoda una riga al file di testo
fclose ($handle);
fseek ($handle, 10, SEEK_SET); // Mi posiziono al 10° carattere dalla posizione corrente

$fh = fopen('filename.txt','r');
while ($line = fgets($fh)) {
    echo($line);
}
fclose($fh);

$fp = fopen('lidn.txt', 'w');
    fwrite($fp, 'writing a line ...');
fclose($fp);
```

Altre funzioni

`$string file_get_contents ($filename);` // Legge l'intero **file testuale** indicato e lo restituisce all'interno di una stringa.

`array file ($filename);` // Legge il **file testuale** indicato e lo restituisce all'interno di un array, ogni riga in una cella successiva. Il `\n` viene mantenuto al termine della riga

Queste ultime due funzioni non hanno bisogno che il file sia aperto ma provvedono automaticamente ad aprirlo e chiuderlo.

`$img = getimagesize ($file)`

Legge il contenuto del file e restituisce sotto forma di **vettore associativo** tutte le varie caratteristiche dell'immagine (dimensione, mime/type, etc). Riconosce quasi tutti i principali formati relativi alle immagini. Se il file non contiene una immagine restituisce **false**.

Gestione dell'Upload di un file

Indipendentemente dal fatto che l'upload lato client sia stato eseguito mediante pulsante di submit oppure tramite Ajax con relativo caricamento delle informazioni all'interno di un oggetto **FormData**, la procedura di lettura lato server rimane la stessa identica.

Il server PHP (apache) intercetta i parametri ricevuti in formato FormData e:

- I campi di tipo stringa vengono riconvertiti in url-encoded e resi disponibili all'interno dei soliti vettori **\$_POST** e **\$_REQUEST**.
- I campi di tipo object sono resi disponibili all'interno di un vettore associativo **\$_FILES** il quale però non sarà restituito come vettore enumerativo di oggetti **File**, ma **un unico oggetto in cui ogni chiave viene trasformata in un vettore enumerativo di valori**. Cioè in pratica la struttura dati restituita da PHP sarà la seguente

```
{
  name: ["nome1", "nome2", "nome3", "nome4"],
  size: ["size1", "size2", "size3", "size4"],
  type: ["type1", "type2", "type3", "type4"],
  tmp_name: ["tmp_name1", "tmp_name2", "tmp_name3", "tmp_name4"]
}
```

Nota: Express invece restituisce un normalissimo vettore enumerativo di oggetti File.

Gestione di un File singolo

In assenza dell'attributo **multiple** l'utente può selezionare ed inviare un unico file. In tal caso :

- Se il **txtName** trasmesso dal client **NON** ha le parentesi quadre, le varie informazioni saranno disponibili in forma scalare, come indicato nel seguente esempio

```
$fileRicevuto = $_FILES["txtFiles"];
$filename=basename($fileRicevuto["name"]);
$size=$fileRicevuto["size"];
$mimeType=$fileRicevuto["type"];
$ext = pathinfo($filename,PATHINFO_EXTENSION);
$sourceFile=$fileRicevuto["tmp_name"];
$targetFile = "uploads/$filename";
move_uploaded_file($sourceFile, $targetFile);
```

La funzione **move_uploaded_file** consente di salvare il file ricevuto all'interno della web directory del server. Restituisce **true** in caso di successo oppure **false** in caso di errore.

In caso di file già esistente lo sovrascrive senza nessuna segnalazione.

Eventualmente si può controllare manualmente se il file già esiste, evitando di sovrascriverlo:

```
if (file_exists($target_file))
    echo ("Sorry, file already exists");
else{
    .....
}
```

- Se invece il **txtName** trasmesso dal client **ha** le parentesi quadre, le varie informazioni saranno disponibili in forma di vettore enumerative con lunghezza 1. Ad esempio:

```
$fileRicevuto = $_FILES["txtFiles"]; // senza le quadre !
$filename=basename($fileRicevuto["name"][0]);
```

Files multipli

Se sul tag `<input type="file">` se si imposta l'attributo `multiple` occorre necessariamente impostare come `name` del controllo client un nome vettoriale del tipo `txtFiles[]`

```
<form action="upload.php" method="post" enctype="multipart/form-data">
    Select file to upload:
    <input type="file" name="txtFiles[]" multiple>
    <input type="submit" value="Upload">
</form>
```

Anche in questo caso apache restituirà un **unico Object** in cui ogni campo viene trasformato in **vettore enumerativo contenente i diversi values**

Lato server per scorrere i files si può utilizzare il ciclo indicato di seguito:

```
$fileRicevuti = $_FILES["txtFiles"]; // senza le quadre
for ($i = 0; $i < count($fileRicevuti["name"]); $i++){
    $filename=basename($fileRicevuti["name"][$i]);
    $size=$fileRicevuti["size"][$i];
    $mimeType=$fileRicevuti["type"][$i];
    $ext = pathinfo($filename,PATHINFO_EXTENSION);
    $target_file = "uploads/$filename";
    move_uploaded_file($fileRicevuti["tmp_name"][$i], $target_file);
}
```

Notare che l'accesso alla chiave `txtFile` deve essere sempre scritto SENZA le parentesi quadre obbligatorie invece nel client:

```
$fileRicevuti = $_FILES["txtFiles[]"];
```

Lettura di parametri passati con i metodi PUT e DELETE

Se sulla pagina client si imposta come method il valore PUT, eventuali parametri vengono inseriti all'interno del body della HTTP Request, esattamente come avviene con POST, però questa volta il server non provvede più a leggerli 'automaticamente' restituendoli all'interno di una apposita struttura dati, ma occorre andare a leggerli manualmente all'interno del body della HTTP Request.

Lettura lato PHP dei parametri urlencoded trasmessi all'interno del body della http-request

Si può utilizzare la seguente procedura basata sulla funzione **file_get_contents** che restituisce sotto forma di stringa l'intero "file testuale" ricevuto sullo stream di input.

```
function _parseParams() {  
    $params = Array();  
    $method = $_SERVER['REQUEST_METHOD'];  
    if ($method == "PUT" || $method == "DELETE") {  
        parse_str(file_get_contents('php://input'), $params);  
        $GLOBALS["_{$method}"] = $params;  
        $_REQUEST = array_merge($params, $_REQUEST);  
        // oppure: $_REQUEST = $params + $_REQUEST;  
    }  
}
```

parse_str esegue uno split della stringa sulla base del carattere &, restituendo un vettore associativo (\$params) contenente tutti i vari parametri in formato chiave:valore.

`$GLOBALS["_{$method}"] = $params;` crea una nuova variabile globale avente come nome il contenuto della variabile method, cioè `_PUT` oppure `_DELETE`

`$_REQUEST = array_merge($params, $_REQUEST)` copia i parametri anche dentro il vettore `_REQUEST` preposto a contenere tutti i parametri HTTP ricevuti insieme alla chiamata

Upload di un file in modalità PUT tramite Ajax

Il **Lato Client** è lo stesso identico che si usa per una richiesta POST.

Lato Server occorre andare a leggere manualmente i dati all'interno del body della HTTP Request. Trattandosi non di un file testuale ma di un file binario non è possibile utilizzare **file_get_contents**. Il file può essere letto sullo STDIN stream nel modo seguente:

```
<?php  
$fp1 = fopen("php://input", "r");  
$fp2 = fopen("newFile.ext", "w");  
/* Read the data 1 KB at a time */  
while ($data = fread($fp1, 1024))  
    fwrite($fp2, $data);  
fclose($fp1); fclose($fp2);  
?>
```

In realtà, prima della lettura, occorre ancora agire sul file eliminando le intestazioni relative alla codifica multipart, cosa fatta automaticamente da PHP, in caso di trasmissione POST, nel momento in cui intercetta il file e lo salva all'interno del vettore associativo `$_FILES`.

Invio di una mail

La funzione `mail()` nativa di PHP rappresenta la strada più semplice, ma necessita di un mail server locale. Comoda per creare una form di invio mail al server che ospita la pagina corrente (alter vista)

In alternativa è possibile utilizzare la classe `PHPMailer` disponibile su github, estremamente stabile ed universalmente utilizzata. Consente di:

- utilizzare in modo trasparente qualsiasi server SMTP
- formattare automaticamente il contenuto della mail
- aggiungere allegati che vengono automaticamente convertiti in formato base64

Impostare lato client un timeout di almeno 10 secondi e un dataType : "text",

Utilizzo della libreria PHPMailer

Il modo più veloce è richiamare il file `PHPMailerAutoload.php` che richiama tutti i file necessari

```
require 'PHPMailer/PHPMailerAutoload.php';
```

In realtà i files indispensabili sono soltanto due: `phpmailer.php` e `smtp.php`

Se si vuole interagire via POP con il mail server esterno, occorre utilizzare anche il file `pop3.php`

```
require 'phpmailer.php';  
require 'smtp.php';
```

```
$mail = new PHPMailer;  
// Indica a PHPMailer di utilizzare la classe SMTP  
$mail->isSMTP();
```

Abilitazione alla visualizzazione dei messaggi

```
// 0 = off (for production use)  
// 1 = Abilita la visualizzazione dei soli messaggi inviati dal client  
// 2 = visualizzazione dei messaggi inviati sia dal client sia dal server  
$mail->SMTPDebug = 2;  
$mail->Debugoutput = 'html'; // formato di visualizzazione dell'output
```

url da contattare

```
$mail->Host = 'smtp.gmail.com';
```

Protocollo da utilizzare per l'invio

E' possibile utilizzare uno fra i seguenti tre protocolli :

SMTP porta **25**. Utilizzato normalmente tra mail server e mail server.

Accetta anche mail prive di autenticazione

SMTPs porta **465** Si tratta di SMTP basato su SSL (deprecato)

SMTP con cifratura TLS porta **587** (RFC4409). Preferibile a SSL nel caso della posta in cui presenta alcune opzioni e garanzie aggiuntive.

```
$mail->SMTPSecure = 'tls'; // Options: '', 'ssl' or 'tls'  
$mail->Port = 587; // Options: 25, 465 or 587  
// si potrebbe direttamente scrivere  
// $mail->Host = "tls://smtp.gmail.com:587";
```

Disabilitazione al controllo del certificato. (peer= certificato)

```
indipendentemente dal protocollo in uso
$mail->SMTPOptions = array(
    'ssl' => array(
        'verify_peer' => false,          // peer = certificato
        'verify_peer_name' => false,     // inutile
        'allow_self_signed' => true      // inutile
    )
);
```

Impostazione delle Credenziali di accesso all'SMTP Server

```
$mail->SMTPAuth = true; // abilitazione al controllo delle credenziali
$mail->Username = "roberto.mana@vallauri.edu";
$mail->Password = "*****";
```

Mittente

```
$mail->setFrom("roberto.mana@vallauri.edu", 'pippo.pluto@vallauri.edu');
```

// Il primo parametro rappresenta l'indirizzo del mittente. Qualunque cosa si imposti viene comunque sovrascritto dallo username impostato sopra. Tanto vale replicarlo
// Il secondo parametro rappresenta il nome del mittente. Nome che, all'interno del client di posta del destinatario, "maschera" l'indirizzo mail del mittente. Nel caso di una form di invio mail al server può essere utilizzato per 'memorizzare' l'indirizzo mail di colui che ha visitato la pagina e spedito la mail

Destinatario

```
$mail->addAddress('ing.mana@tiscali.it');
// $mail->addCC('ing.mana@tiscali.it');
// $mail->addBCC('ing.mana@tiscali.it');
```

Impostazione della Mail

```
$mail->Subject = 'Here is the subject';
$mail->Body = 'This is the HTML message body <b>in bold!</b>';
$mail->AltBody = 'This is the body in plain text for non-HTML mail clients';
$mail->isHTML(true); // Set email format to HTML
$mail->addAttachment('tmp/myfile.txt');
$mail->addAttachment('tmp/Desert.jpg', "nome Allegato");
```

Il primo parametro di `addAttachment()` rappresenta il nome del file da allegare.

Il second parametro rappresenta il nome da assegnare all'allegato (nome che 'maschera' il nome del file)

Invio

```
if (!$mail->send())
    echo "Mailer Error -> " . $mail->ErrorInfo;
else
    echo "Message sent!";
```