PHP per la ***propagazione e persistenza dei dati***

Introduzione

Quando si ha a che fare con la programmazione ***server-side*** in generale, non si avvertono da subito alcune esigenze, che diventano poi delle vere e proprie necessità con applicazioni più complesse.

Molti di voi si saranno sicuramente già scontrati col problema di dover far transitare dei dati da una pagina all'altra o meglio da uno script all'altro, o anche di stipare dei dati per un periodo di tempo più o meno lungo; dati che devono essere necessariamente associati al determinato utente che in quel momento sta fruendo del nostro servizio.

Per risolvere questa situazione, ci sono diverse soluzioni che vanno attentamente studiate in base poi alle esigenze del momento; non sempre sarà possibile usare le sessioni, così come non sempre sarà possibile usare i cookies e così via. In questo articolo vi spiegherò come utilizzare i vari metodi presenti in **PHP** per la ***propagazione e persistenza dei dati***, vi illustrerò i vantaggi dell'una o dell'altra metodologia, eventuali note sulla sicurezza con un occhio alle varie versioni di PHP (sarà molto utile la lettura del mio articolo del num. 6), cercando come sempre di proporre numerosi esempi pratici di vita quotidiana.

Le metodologie diffuse

Vediamo brevemente i metodi (non intesi come nella OOP) che possiamo sfruttare con PHP e MySQL per ottenere ciò che ci siamo prefissati:

- **Cookies**
- **Sessioni**
- **Shared Memory**
- **MySQL Memory Tables**

†Cookies

I famosi "biscottini", sono dei semplici file di testo che risiedono sul computer dell'utente, possono stipare un gran numero di informazioni e sono facilmente invocabili dagli script php.

La limitazione consiste nel fatto che l'utente potrebbe aver disabilitato la gestione degli stessi dal browser e quindi non possiamo attivarli ed usarli. Uno dei maggiori vantaggi che offrono è la possibilità di stipare informazioni per un lungo periodo di tempo (fino a quando il rispettivo file non viene cancellato e comunque la data di scadenza viene decisa dal programmatore).

†Sessioni

Le sessioni sono una versione evoluta dei cookies, in pratica le informazioni vengono memorizzate sul server e non più sul computer dell'utente. Grosso vantaggio è che funzionano (almeno dovrebbero) anche quando l'utente hai i cookies disabilitati, lo svantaggio è che hanno vita breve, ovvero vengono cancellate nell'istante in cui l'utente chiude il browser oppure passa troppo tempo dall'ultimo refresh dello script (per default 1440 secondi ovvero *25 minuti*).

†Shared Memory

Uno strumento sicuramente molto potente ma al contempo anche molto delicato e complesso da gestire. Abbiamo la possibilità in questo caso di stipare tutte le informazioni che vogliamo all'interno della **memoria** (***RAM***) **del server**. Grosso vantaggio è il fatto che queste informazioni non hanno una scadenza in termini di tempo (a meno che non riavviate il server), al contrario invece lo svantaggio principale è che sono molto difficili da gestire (o meglio bisogna fare molta attenzione) e vanno ad intaccare in maniera importante sulle risorse del sistema. Funzionano egregiamente su sistemi Unix/Linux, su sistemi Windows c'è bisogno di abilitare una particolare estensione (php_shmop.dll) ed avere almeno un sistema Windows 2000.

†MySQL Memory Tables

Altro potente mezzo per stipare un bel pò di informazioni senza preoccuparsi delle scadenze, funziona un pò come la shared memory che abbiamo visto poco fa. In pratica sono delle **tabelle di MySQL che vengono allocate in memoria principale** (***RAM***) invece che sul disco (come accade normalmente per le tabelle MyIsam o InnoDB ecc..). Il vantaggio è che sono persistenti nel tempo ed hanno una velocità di interrogazione molto elevata (rispetto alle sorelle memorizzate su disco), svantaggio principale è la quantità di memoria allocabile e quindi l'ammontare dei dati che si possono memorizzare. Le **Memory** Tables soffrono ovviamente di alcune limitazioni sul tipo di dati archiviabili e sugli indici, questo per ovvi motivi di performance e risorse.

Fatta la dovuta carrellata andremo, ora, in dettaglio ad analizzare ognuno dei metodi con relative spiegazioni ed esempi d'uso.

Cookies

Prima usati da tutti a sproposito, poi completamente odiati dagli utenti e quindi disattivati, ora stiamo ritornando ad una situazione di normalità.

I cookies spesso sono associati ad azioni di "spionaggio" sul pc dell'utente, ma in effetti non è così (almeno nella maggior parte dei casi); non c'è alcun motivo per aver paura di attivare tale funzionalità del browser che per la maggior parte dei siti si traduce nella possibilità di dare maggiori servizi agli utenti. La loro funzione è molto semplice. Nel momento in cui ne viene chiesta l'attivazione, viene creato sul computer dell'utente un file di testo (si trovano di solito in una dir dal nome "Cookies" almeno su sistemi Windows).

Il file contiene tutte le informazioni che il programmatore vuol far memorizzare al suo interno, oltre ad un identificatore (in modo da poter richiamare quelle informazioni quando servono), al tempo di vita del cookie, e al nome del sito in cui è stato attivato. Ogni cookie ha una durata variabile e settabile direttamente dal programmatore, al momento della sua scadenza verrà automaticamente cancellato (fisicamente) dal browser.

Per esempio possiamo usare un cookie per memorizzare l'accesso di un utente in un'area protetta del nostro sito, in modo che alle successive visite l'utente non debba più autenticarsi. Oppure possiamo memorizzare nel cookie impostazioni sullo stile grafico scelto dall'utente, in modo da fargli vedere il sito sempre con le impostazioni da lui scelte. Caso più semplice e banale è invece la memorizzazione dei dati personali in modo, per esempio, di dare un saluto personalizzato all'utente. Vediamo in dettaglio quest'ultimo caso, costruendo una piccola pagina di autenticazione, ed una pagina che visualizza il saluto prendendo il nome dell'utente dal cookie. Nel caso in cui la password sia sbagliata o il cookie non impostato allora faremo visualizzare la pagina di autenticazione.

Iniziamo subito con la **pagina di autenticazione**, che conterrà semplicemente un form che invia i dati allo script “login.php”.

```
// PAGINA DI AUTENTICAZIONE: index.html

<form action=”login.php” method=”post” />
    Nominativo: <input type=”text” name=”nominativo” /><br />
    Username: <input type=”text” name=”username” /><br />
    Password:  <input type=”password” name=”pwd” /><br /><br />
   <input type=”submit” value=”Accedi” />
</form>
```

Questo è lo script, invece, che si occupa di controllare username e password. In caso di esattezza dei dati allora setterà il cookie ed invierà l’utente alla pagina protetta.

```
// PAGINA DI ELABORAZIONE DATI DI LOGIN: login.php


// Controllo se l’utente ha inviato dei dati tramite il metodo POST del form

if (isset($_POST["username"]) && isset($_POST["pwd"]) )
{
  if( $_POST["username"]==’""|| $_POST["pwd"]==’"")
  {
     // L’utente ha fatto il submit non mettendo alcunché nel form quindi
     // lo rimando a index.html con la funzione Header();

     Header("Location: index.html’";

     exit;
  }
  // … to be continue …
}
```

sarebbe interessante in questo caso, poter mandare l’utente ad una pagina di errore, o meglio ancora settare un cookie che memorizza lo stato d’errore e relativo messaggio e poi lo facciamo visualizzare nella index. Questo è sicuramente un utile esercizio che vi consiglio di realizzare, così come è un utile esercizio aggiungere controlli di sicurezza per evitare che eventuali pirati possano “bucare” il server. Ma ora andiamo avanti con la nostra applicazione…

```
// … to be continue …
else
{
   // settiamo come username e password la parola “pippo”
   if($_POST["username"]!="pippo" || $_POST["pwd"]!="pippo")
   {
      // come prima, l’utente ha sbagliato,
      //quindi lo rimandiamo all’autenticazione

      Header("Location: index.html");
      exit;
   }
   else
   {
      // Questo è il punto più importante, dove settiamo il cookie
      // se è presente il nominativo impostiamo quello come saluto
      // altrimenti usiamo la username.
```

```
     $dato_da_stipare_nel_cookie='';
      if( isset($_POST["nominativo"]) && $_POST["nominativo"]!="")
      {
         // Se il nominativo è presente allora setto questo
         $dato_da_stipare_nel_cookie=$_POST["nominativo"];
      }
      else
      {
         // Altrimenti uso la username
         $dato_da_stipare_nel_cookie=$_POST["username"];
      }

      // tempo di scadenza espresso in secondi
      // a partire dal momento attuale.
      // in questo caso 3600 secondi, ovvero 1 ora
      // oltre l'orario attuale.
      $scadenza = 60 * 60 + time();

      // SETTAGGIO COOKIE
      setcookie("utente",$dato_da_stipare_nel_cookie,$scadenza);
      Header("Location: protetta.php");
      exit;
   }
}
```

Ora bisogna fare una doverosa precisazione, la funzione setcookie() può essere interpretata correttamente solo nel caso in cui non sia stato inviato al browser dell'utente alcun carattere (quindi nessun echo, header, print, tag html ecc..), dopo aver invocato tale funzione sarà possibile produrre tutto l'output desiderato. Inoltre notate come il primo parametro passato alla funzione sia 'utente', tale infatti sarà il nome di identificazione del nostro cookie, e col quale potremo invocarlo.

La funzione in esame accetta anche altri parametri di facile comprensione e che potrete trovare sul manuale di php oppure all'url: http://it.php.net/setcookie.

Costruiamo ora la semplice paginetta 'protetta.php', che controllerà la presenza del cookie e stamperà a video il saluto personalizzato, nel caso in cui il cookie non sia settato allora rimanderà l'utente alla index. Notate che a seconda della versione del php che utilizzate, dovrete usare una differente variabile

```
// PAGINA PROTETTA: protetta.php
// se avete php > 4.1.0 utilizzare la variabile superglobal $_COOKIE
// altrimenti la variabile $HTTP_COOKIE_VARS
if ( isset($_COOKIE["utente"]) )
{
  // Il cookie esiste quindi faccio uscire il saluto
  echo "Ciao, <b>".$_COOKIE["utente"]."</b>";
}
else
{
   // Il cookie non esiste quindi ritorno alla pagina di login.
   Header("Location: index.html");
   exit;
}
```

Come vedete il codice è molto semplice. Notate come il nostro cookie è semplicemente un elemento di un array associativo, la chiave non è altro che il nome assegnato al cookie con la funzione setcookie(), e come valore il contenuto della var $dato_da_stipare_nel_cookie, usata nello script login.php.

Sessioni

Come ho già introdotto qualche riga fa, le sessioni non sono altro che un particolare tipo di cookie, ovvero il file che contiene tutte le informazioni dell'utente viene stipato sul server e non sul pc del client; nonostante possiamo comunque decidere a priori il lifetime (il tempo di vita) della sessione, questa verrà automaticamente distrutta nel momento in cui l'utente decide di chiudere il browser. Molti sono i vantaggi di usare le sessioni al posto dei cookie.

Primo fra tutti il non dover essere vincolati dai settaggi del browser e, sicuramente la maggiore facilità di gestione e l'estrema usabilità in fase di programmazione. Purtroppo l'uso delle sessioni cambia a seconda della versione di PHP che usate e se avete ancora la sfortuna di dover lavorare con una versione antecedente la 4 allora non troverete il supporto nativo per la gestione delle stesse. Facciamo ora una breve panoramica su come si istanzia una sessione, si registrano in essa i dati, e si richiamano nello script, vedendo le differenze fra le versioni di php dalla 4 alla 4.1.0 e da quest'ultima in avanti; non tratteremo tutte le tematiche connesse alle sessioni, così come non tratteremo la gestione avanzata delle stesse, forse in un prossimo articolo.

Nel momento in cui si crea una nuova sessione (invocando esplicitamente una funzione particolare), il sistema crea un file nella dir di stoccaggio indicata nel php.ini ed assegna alla sessione un identificativo chiamato SID (Session ID), che verrà propagato attraverso le varie pagine e script tramite un meccanismo automatico che solitamente è attivo nel php (--enable-trans-sid e session.use_trans_sid), nel caso non fosse così si dovrà provvedere manualmente alla trasmissione di questo parametro. Potete tranquillamente sostituire il codice che vi spiegherò più avanti al posto del precedente codice per i cookie dal punto in cui vi è il commento //SETTAGGIO COOKIE.

Creiamo la nostra prima sessione

```
session_start();
```

Ora dobbiamo stipare i dati che ci servono, rifacendoci all'esempio precedente dobbiamo chiamare la var di sessione 'utente' e dobbiamo stipare il contenuto dalla var $dato_da_stipare_nel_cookie. Mettiamo queste istruzioni al posto dell'invocazione alla funzione setcookie().
Se abbiamo PHP < 4.1.0

```
//registriamo utente in sessione
session_register("utente");

// ora assegnamo alla var utente
// il contenuto del saluto
// se avete register_globals a off usate questa:
$HTTP_SESSION_VARS["utente"]=$dato_da_stipare_nel_cookie;

// altrimenti basta:
// $utente=$dato_da_stipare_nel_cookie;

// Ora vado a protetta.php passando nella get string anche il SID
Header("Location: protetta.php?".session_id());
```

Come potete vedere è molto semplice da usare…, notate come passo nella get string dell'url il SID in modo da evitare problemi nel caso il trans_sid fosse disabilitato. Nel caso in cui non vi funzioni provate a sostituire la stringa session_id() con la stringa session_name().'='.session_id() che vi da maggiore sicurezza sulla propagazione del SID.

Ora vediamo invece il caso in cui abbiamo PHP >= 4.1.0

```
//registriamo utente in sessione

$_SESSION["utente"]=$dato_da_stipare_nel_cookie;

// Ora vado a protetta.php passando nella get string anche il SID

Header("Location: protetta.php?".session_id());
```

con le nuove versioni è molto più semplice memorizzare dati in sessione. Ricordate che a differenza dei cookie (e questo è un altro grosso vantaggio) le sessioni possono essere definite in qualsiasi punto dello script anche se c'è stato precedentemente un output.

Ora andiamo a vedere come modificare la pagina protetta.php in modo da adattarla alle sessioni:

```
// In pratica tutti i riferimenti a $_COOKIE e $HTTP_COOKIE_VARS vanno cambiati

// nei rispettivi $_SESSION e $HTTP_SESSION_VARS


// Facciamo partire la sessione

session_start();
```

per PHP < 4.1.0

```
// Controlliamo se la var di sessione

// è settata

if( session_is_registered("utente") )
{
   // La sessione esiste quindi stampo il saluto

   echo "Ciao, <b>".$HTTP_SESSION_VARS["utente"]."</b>";

}
 else
 {
    // la sessione non esiste quindi ritorno alla pagina di login.

    Header("Location: index.html");

    exit;
 }
```

notate l'uso dell'apposita funzione session_is_registered() che controlla appunto se una var di sessione col nome utente è settata nella sessione corrente, in caso positivo semplicemente fa riferimento alla sessione come elemento dell'array $HTTP_SESSION_VARS; se register_globals fosse settato a on allora potremo far riferimento a quella sessione anche semplicemente con la var $utente. Nel momento in cui invochiamo la funzione session_start() questa automaticamente rintraccia il SID ed apre il file di sessione giusto.

Invece per PHP >= 4.1.0 diventa

```
// Controlliamo se la var di sessione
// è settata

if( isset($_SESSION["utente"]) )
{
   // La sessione esiste.
   echo "Ciao, <b>".$_SESSION["utente"]."</b>";
}
 else
 {
    // la sessione non esiste quindi ritorno alla pagina di login.
    Header("Location: index.html");
    exit;
 }
```

Shared Memory

La gestione della *memoria condivisa* è un argomento molto delicato, sicuramente non riducibile in poche righe; PHP alla stregua dell'ANSI C o C++ ci da la possibilità di memorizzare i nostri dati all'interno di **segmenti di memoria (*RAM)* liberi**, chi di voi ha giocato con le funzioni malloc_*() e calloc_*() di C sicuramente sa quanto sono comode ma anche molto inclini a far crashare il PC.

In PHP non abbiamo la stessa libertà di movimento e quindi abbiamo meno possibilità di combinare qualche grosso guaio, infatti è lo stesso parser ad occuparsi di trovare il segmento di memoria libera e di allocare e riservare per noi lo spazio; poche sono le funzioni a disposizione ma fanno il loro dovere egregiamente. Ora vi farò un piccolo esempio giusto per farvi capire come funziona tutto il processo, ma ricordate che il limite qui è la memoria del PC, più dati stipate al suo interno e più le risorse del server calano drasticamente.

Riprendendo la situazione affrontata precedentemente con i cookies e le sessioni, possiamo vedere come sostituire i vari processi in modo da stipare i dati in shared memory e propagare così la nostra pseudo-sessione. Ecco il segmento di codice da inserire nel punto del commento //SETTAGGIO COOKIE:

```
// creiamo un nuovo segmento di memoria
// con id: 0xa (numero esadecimale)

$seg_id = shmop_open(0xa,"c",0644,strlen($dato_da_stipare_nel_cookie));
```

con la funzione shmop_open() andiamo a creare un nuovo segmento di memoria, notate i parametri passati alla funzione:

- 0xa: ID da assegnare al segmento
- 'c': flag che indica appunto la creazione di un nuovo segmento
- 0644: come nei file indica i permessi sul segmento
- strlen…: indica la dimensione in byte del segmento (in questo caso calcolata sulla stringa da stipare)

dopo aver invocato la precedente funzione, bisogna controllare se la creazione del segmento è andata a buon fine:

```
// Controlliamo se il segmento è stato creato con successo

if( !$seg_id )
{
   // Creazione fallita, ritorno all'autenticazione
   Header("Location: index.html");
   exit;
}
```

ora scriviamo nel nostro blocco i dati che ci interessano:

```
$check=shmop_write($seg_id, $dato_da_stipare_nel_cookie, 0);

// Controllo se la scrittura ha avuto successo

if ( $check != strlen($dato_da_stipare_nel_cookie) )
{
   // Scrittura fallita ritorno alla index
   Header("Location: index.html");
   exit;
}
else
{
   // Abbiamo avuto successo, quindi
   // andiamo alla pagina protetta passando
   // l'id del segmento

   Header("Location: protetta.php?seg_id=0xa");
  exit;
}
```

notate come ad ogni passettino che facciamo, dobbiamo necessariamente effettuare dei controlli, questo per evitare errori grossolani o peggio ancora crash di sistema. La funzione usata per scrivere i dati è shmop_write(), alla quale passiamo come primo parametro l'id del segmento generato prima, la stringa da stipare, ed infine l'offset (il punto di memoria) da cui iniziare a scrivere.

L'ultimo passo è quello di richiamare nella pagina protetta.php il segmento di memoria tramite l'id passato sulla get string dell'url e controllare se è valido e se contiene dati:

```
// Controlliamo se l'id di segmento è stato inviato
if( isset($_GET["seg_id"]) )
{
   // L'id è stato passato, ora controlliamo
   // se il segmento esiste e contiene dati
   $seg_id = shmop_open($_GET["seg_id"],"ac",0,0);
   if( !$seg_id )
   {
      // Il segmento non esiste o non è valido
      Header("Location: index.html");
      exit;
   }
   $stringa = shmop_read($seg_id, 0, shmop_size($seg_id));
   if( !$stringa )
   {
      // La lettura è fallita quindi esco
      Header("Location: index.html");
      exit;
   }

   echo "Ciao, <b>".$stringa."</b>";
```

```
   // per coerenza, liberiamo la memoria
   // cancellando il segmento
   @shmop_delete($seg_id);
   @shmop_close($seg_id);
}
 else
 {
    // L'id del segmento non è presente quindi esco
    Header("Location: index.html");
    exit;
 }
```

In questa fase, utilizziamo sempre la funzione shmop_open() ma passando i parametri in maniera diversa, siccome dobbiamo solo verificare se il segmento esiste ed eventualmente aprirlo, non dobbiamo passare valori di permessi e dimensioni (infatti sono a zero), mentre come attributo di apertura utilizziamo la combinazione 'ac' che è l'insieme dei flag 'a' ? access e 'c' ? create per una non precisata ragione l'uso del solo flag 'a' non è sufficiente quindi bisogna usare la combinazione indicata. Nel passo successivo, in caso di apertura con successo del segmento, andiamo a leggerne il contenuto ed utilizziamo la funzione shmop_read() che come primo parametro vuole l'id del segmento, a seguire l'offset da cui iniziare a leggere e il totale di byte da leggere, siccome iniziamo a leggere dalla posizione zero (vedete il segmento di memoria come un array di caratteri), gli diamo la dimensione totale del segmento con la funzione apposita shmop_size().

Dopo i soliti controlli dobbiamo avere l'accortezza di distruggere il segmento (se non serve più ovviamente) per liberare ovviamente memoria, siccome potrebbero accadere dei problemi ho anteposto alle due funzioni usate il simbolo '@' che come ben sapete evita l'output di eventuali errori, warning, ecc. Ovviamente la questione è molto più complessa di quella che vi ho presentato, ma già potete farvi un'idea, sarebbe una buona idea leggersi il manuale del php alla voce "Shared memory Functions". Inoltre voglio ricordarvi che se usate la versione 4.0.3 di php, per un qualche strano motivo che non conosco, le funzioni usate non hanno il prefisso shmop ma semplicemente shm.

L'uso di **memoria condivisa** è anche un modo alternativo per **Comunicare tra script PHP.**

MySQL Memory Tables

Le **Memory** Tables di MySQL fanno testo a parte, rispetto ai metodi visti fin'ora.

In pratica possiamo avere la potenzialità e la robustezza di un DBMS (interrogazione con query SQL in primis) con la flessibilità ma anche le pecche della memoria centrale del sistema. Infatti i nostri dati potranno essere stipati temporaneamente in una tabella che verrà allocata in memoria invece che su disco, potremo quindi utilizzare su di essa quasi la maggior parte (ma non tutte) delle istruzioni SQL per estrarre i dati che ci servono.

Uno dei grossi vantaggi è caratterizzato dal poter stipare dati importanti (occhio sempre alla quantità di RAM disponibile) e poterli estrarre molto facilmente; non userei le **Memory** Tables per l'esempio di cui sopra, in quanto sarebbero inutili, ma sicuramente potrebbero tornare utili per stipare risultati di ricerca, oppure elenchi di prodotti (per un sito di ecommerce ad esempio).

Creare una **Memory** table è molto semplice, basta usare lo statement di creazione tabelle classico di MySQL e poi specificare il tipo ad esempio:

```
CREATE TABLE elenco_prodotti (
id_prodotto INT(6) NOT NULL,
nome_prodotto VARCHAR(50) NOT NULL,
prezzo FLOAT(10) NOT NULL DEFAULT 0.0) ENGINE=MEMORY;
```

ma un utilizzo molto più interessante è la creazione della tabella partendo dal result set di una query di select, in questo modo non dobbiamo preoccuparci della struttura della tabella che sarà automaticamente creata:

```
CREATE TABLE elenco_prodotti ENGINE=MEMORY
SELECT id_prodotto,nome_prodotto,prezzo FROM prodotti ORDER BY nome_prodotto
```

Purtroppo sull'uso delle **Memory** Tables ci sono svariate limitazioni, ma credo che siano molto interessanti da usare nonostante tutto.

Ad esempio non è possibile stipare dati di tipo BLOB/TEXT, oppure non è possibile avere campi con attributo auto_increment.

Fate sempre e comunque attenzione all'utilizzo di memoria, per ragioni di sicurezza il DBA (DataBase Administrator) setta tramite una variabile la grandezza massima che una **Memory** table potrebbe avere.

Non vi resta che fare tante prove e test sul vostro sistema.

Un consiglio, dopo aver utilizzato la tabella vi conviene cancellarla o troncarla per liberare memoria e risorse con una delle seguenti istruzioni:

1. DELETE FROM elenco_prodotti
2. TRUNCATE elenco_prodotti
3. DROP TABLE elenco_prodotti

Conclusioni

In questo articolo ho[1] fatto una carrellata molto veloce sull'argomento Persistenza Dati, che sicuramente meriterebbe altri approfondimenti. Di certo un buon punto di partenza; con gli esempi e le notizie che vi ho dato potrete facilmente fare delle prove e strada facendo, le idee non tarderanno ad arrivare.

Vi rinnovo comunque l'invito a dare la massima attenzione e priorità alla sicurezza e alla pulizia del codice. Un codice pulito e robusto ci farà perdere un poco di tempo in fase di progettazione e coding ma sicuramente di consentirà di avere nel futuro meno problemi sia in termini di attacchi informatici sia in termini di manutenibilità ed aggiornamento.

Approfondimenti su sicurezza, cookie e sessioni per PHP
di Simone Carletti

Lunedì 11 Giugno 2007 - 12:05

http://blog.html.it/archivi/2007/06/11/approfondimenti-su-sicurezza-cookie-e-sessioni-per-php.php

[1] Autore Tommaso D'argenio

Soluzione con uso DB

Le **applicazioni web complesse** hanno bisogno di un **supporto** alla **persistenza dei dati**.

Si è visto come i **cookies** e le **sessioni** realizzino una forma di **mantenimento dello stato**, tra pagine delle stessa sessione o tra visite successive ma la **quantità di dati** che **si può gestire** con sessioni e cookie è **molto limitata** ed alcuni tipi di dati, in **applicazioni web complesse**, devono avere una persistenza che va oltre quella di cookie e sessioni, potendo anche essere **aggiornati** dai **gestori del sito**

Si progetta allora l'**interazione con DB**.