Interazione linguaggio PHP con DB *MySQL* estensione PDO

Nelle prime versioni, fin da PHP 2.0, che non prevedevano l'uso di oggetti, l'***estensiopne nativa per MySQL*** rendeva disponibili ***funzioni*** (da **PHP 7** non più supportate) per realizzare tutte le fasi necesarie nell'interagire con tale RDBMS; a partire da PHP 5.0 si usa l'***estensione MySQLi*** *(MySQL improved)* che può essere programmata tramite un **approccio Object Oriented** sebbene mantenga la possibilità di poter essere utilizzata anche con un approccio **procedurale**.

Una terza possibilità, dalla versione 5.1, è data dal ricorso all'***estensione*** PHP Data Objects (***PDO***), completamente orientata agli oggetti, e che non può quindi essere utilizzata con un approccio procedurale. La caratteristica principale[1] che differenzia questa estensione rispetto a MySQLi è che essa dispone di driver per diversi DBMS, non limitandosi esclusivamente a MySQL. Solo alcune delle funzioni dell'estenzione nativa sono oggi deprecate.

Guide in italiano:

Mr.Webmaster

- Estensioni PHP per l'interazione con i database (2014)
  - per connettersi con estensione PDO (eventuale set dell'ambiente)
  - per creare una tabella *personalizzando l'*esempio *su hosting free altervista*
  - per inserire
  - per estrarre
  - per modificare (aggiornare o cancellare)
  - uso di ***prepared statement*** *(estrazione)*
  - per creare un database

HTML.it

- Introduzione a PDO (2017)
  - PDO vs MSQLi
  - Set-up dell'ambiente di sviluppo
  - per connettersi
  - per scrivere (creazione tabella e INSERT) anche con ***prepared statements***
  - per estrarre
  - transaction

Guide in inglese:

***w3Schools.com*: confonto usando estensioni MSQLi procedurale, orientata ad oggetti, PDO**

- ***Connessione*** (PHP 5 e successivi)
- ***Creazione DB***
- ***Creazione tabella***
- ***Inserimento (singolo o multiplo)***
- ***Estrazione (run example)***
- ***Cancellazione***
- ***Aggiornamento***

***Prepared statements* - sito *w3Schools* (*inserimento** personalizzando l'*esempio *su hosting free altervista)*

***Prepared statements* -** *Manual*

1 Svantaggi: leggermente meno performante di MySQLi e non ha accesso diretto ad alcune caratteristiche delle ultime versioni di MySQL come i *multiple statements*.

Connessione al database

Innanzi tutto vedremo le operazioni necessarie per la **connessione al database**.
Prima di poter comunicare con un DB abbiamo infatti bisogno di creare un "collegamento" fra lo script e MySQL.

Avremo bisogno di alcune informazioni relative all'accesso al database: l'**host** da cui si può raggiungere MySQL (generalmente è *localhost*); **username** e **password** per l'accesso al database; il **nome** del database. Questi quattro parametri vengono forniti dall'amministratore del nostro spazio web e prevediamo di memorizzare in altrettante variabili tali dati relativi alla configurazione dello script che salveremo con estensione **.php** per motivi di sicurezza.

Se qualcuno cercasse, infatti, di visualizzare questa pagina con il browser, vedrebbe solo una pagina vuota.

Il webserver, infatti, grazie a quest'estensione, prima di passare la pagina al browser, la farà elaborare dal modulo Php. Visto che non è previsto nessun output, sul browser verrà visualizzata solo una pagina bianca.

```
// parametri di connessione
$db_host = "localhost";                    // il server che si trova sulla macchina locale
$db_user = "";
$db_password = "";

// parametri del database
$db_name = "";
```

I valori da inserire solitamente sono dati dal server dove risiede il database (in fase di registrazione utente dovete inserire la password ed il login).

Per **connettersi**, allora, ad un database ***MySQL*** remoto con ***estensione PDO***

```
<?
// parametri del database
$db_host = "localhost";
$db_user = "new345";
$db_password = "";   // opzionale
$db_name = "my_new345";

/*
  blocco try/catch di gestione delle eccezioni
*/
try {
  // stringa di connessione al DBMS
  $connessione = new PDO ("mysql:host=$db_host;dbname=$db_name", $db_user, $db_password);
  // impostazione dell'attributo per il report degli errori
  $connessione->setAttribute (PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  //// codice con esecuzione di query
 $connessione = null;                    // risorsa deallocata
}
catch (PDOException $e)
{
  // notifica in caso di errore nel tentativo di connessione
  echo "Errore: " . $e->getMessage();
  die();    // uscita
}
?>
```

Se la connessione ha buon fine restituisce un *identificatore (handle)* alla connessione che si memorizza nella variabile **$connessione**. Useremo questa variabile ogni volta che vorremo fare un'operazione sul database.

Se la connessione non dovesse andare a buon fine (per esempio se uno dei parametri fosse sbagliato) verrebbe generata un'eccezione ed il costrutto **try-catch verificare il buon esito della connessione o notifica in caso contrario**.

setAttribute() è utilizzato per gestire eventuali eccezioni all'interno del blocco **try/catch** specificando l'*attributo PDO::ATTR_ERRMODE* ed il *valore PDO::ERRMODE_EXCEPTION*;

Esecuzione di una query

Si è arrivati alla parte fondamentale del colloquio con un database, cioè l'esecuzione di una query.

Tale query sarà inserita nel controllo **try-catch**.

Estrazione

- Se si tratta di una **query di interrogazione** (**SELECT**, SHOW, EXPLAIN, DESCRIBE), il metodo **query ()** restituisce un **identificativo del risultato** (cioè un'altra variabile di tipo resource a cui si può assegnare una ***nome***), che ci servirà successivamente, se la query è andata a buon fine. Nell'esempio $row ci servirà successivamente per **leggere** le righe restituite al volo.

```
// selezione e visualizzazione dei dati estratti dalla tabella contatti
 foreach ($connessione->query("SELECT nome, cognome FROM contatti") as $row)
 {
  echo $row['nome'] ." ". $row['cognome'] . "<br />";
 }
 // chiusura della connessione
$connessione = null;                    // risorsa deallocata
```

In alternativa, si può usare il metodo **fetch()** o **fetchAll()** che svolge la stessa funzione di fetch(), però a differenza di quest'ultimo che elabora un risultato alla volta, fetchAll() restituisce un array contenente tutti i dati della nostra query. È più veloce ma consuma più memoria rispetto a fetch().

```
// estrazione di record con il metodo fetch
$query = $connessione->query("SELECT nome, cognome FROM contatti");
while($row = $query->fetch(PDO::FETCH_ASSOC)) {
  echo $row['nome'] ." ". $row['cognome'] . "<br />";
}
```

Tale metodo recupera un record da un set di risultati associato ad un oggetto PDOStatement; il metodo accetta un parametro, detto ***fetch_style***, che a sua volta determinerà le modalità di restituzione del record recuperato da parte di PDO.

Il valore predefinito del ***fetch_style*** è uguale a **PDO::FETCH_BOTH** che consente di ottenere un **array** nel quale ogni elemento costitutivo è presente sia con chiave letterale (i nomi dei campi) che numerica (un indice da "0" a "*n*"); **PDO::FETCH_ASSOC** è invece un parametro più specifico che restituisce un array indicizzato tramite i nomi dei campi presenti in tabella.

Da notare come, nell'esempio proposto, tale array sia stato ciclato tramite un'istruzione basata su **while** (che proseguirà nella sua iterazione sino a quando il recordset non sarà stato "svuotato"), mentre il codice da iterare sarà quello della stampa a video dei dati recuperati

Lo stesso risultato dell'esempio visto qui sopra avrebbe potuto raggiungersi anche con uso del metodo **setFetchMode** in questo modo:

```
// estrazione di record con il metodo setFetchMode
$query = $connessione->query("SELECT nome, cognome FROM contatti");
$query->setFetchMode(PDO::FETCH_ASSOC);
while($row = $query->fetch()){
  echo $row['nome'] ." ". $row['cognome'] . "<br />";
}
```

setFetchMode() ha in pratica il compito di impostare la modalità, "*mode*", di recupero dei dati per l'istruzione SELECT passata al metodo query(); nel caso del nostro esempio il "*mode*" scelto sarà il già noto PDO::FETCH_ASSOC, mentre il metodo fetch() consentirà di ottenere i valori da recuperare tramite la loro associazione ad un array nel ciclo while.

Per gestire in **modo sicuro**:

```
// quoting del parametro passato alla clausola WHERE
$sql = "SELECT cognome FROM contatti
        WHERE nome = '". $connessione->quote($_POST['nome']) . "'";
foreach ($connessione->query($sql) as $row)
{
  echo $row['nome'] ." ". $row['cognome'] . "<br />";
}
```

Nel caso in cui si operi con query generate *dinamicamente*, ad esempio mediante l'inclusione di parametri generati da input degli utenti (ad esempio attraverso un form o una querystring) è consigliabile fare ricorso al metodo **quote()** per effettuare l'*escape* dei parametri, come accade nell'esempio precedente dove l'istruzione SQL prevede l'adozione della clausola WHERE, recuperando da un form una stringa da ricerare all'interno di uno specifico campo della nostra tabella. In questo modo sarà possibile garantire una maggiore protezione da tentativi di attacco come per esempio quelli basati sulla **SQL Injection**; PDO offre però strumenti ancora più avanzati per la protezione contro azioni malevole quali l'uso di *prepared statements*

Esempio di estrazione in **ordine cronologico** di **sole 3 righe**:

```
<?
$query = "SELECT id,data,titolo FROM news ORDER BY data DESC LIMIT 0,3";
?>
```

NB: analogamente, si potranno realizzare estrazioni da più tabelle (*join*)

con vari criteri di selezione

e ancora si potranno deallocazione tabelle, indici, database

Modifica: aggiornamento, inserimnto,cancellazione

- Se invece si tratta di una **query di modifica** (INSERT, UPDATE, DELETE e tutte le altre diverse da quelle viste prima), il metodo restituirà in ogni caso un valore booleano, ad indicare se l'esecuzione è andata a buon fine oppure no.

Aggiornamento (UPDATE) dei record mediante PDO

Se per esempio volessimo aggiornare il contenuto del campo "nome" della tabella "contatti" corrispondente all'"id = 1" con il nuovo valore "Anita":

```
// istruzione per l'aggiornamento
$aggiorna_dati = $connessione->exec("UPDATE contatti SET nome = 'Anita' WHERE id = 1");
  // chiusura della connessione
 $connessione = null;                    // risorsa deallocata
```

Il metodo **exec()** è simile a **query()**, dato che esegue una query in una sola chiamata, ma invece di restituire i valori di quest'ultima, restituisce il numero di righe coinvolte. Quindi non va utilizzato con il comando SELECT, dato che non restituirebbe nulla, bensì con UPDATE e DELETE, ad esempio per vedere quanti elementi sono stati eliminati.

Si noti che il metodo **exec()** può essere accompagnato con la clausola di salvaguardia **or die()** al fine di interrompere l'esecuzione del codice qualora la query sia "ineseguibile" (una query sbagliata).

Nel caso specifico, però, è necessario fare una puntualizzazione: le istruzioni per l'aggiornamento dei dati (ma il medesimo discorso vale anche per le procedure di cancellazione) potrebbero non avere effetto su alcun record (si supponga, ad esempio, che nel nostro esempio l'ID 1 non esista !) in questo caso la query restituirà "0" (FALSE) ma non si tratterebbe certo di un comando "ineseguibile", motivo per il quale si adotta una sintassi che prevede l'utilizzo dell'operatore "===" per verificare il valore di ritorno restituito dall'esecuzione, in questo modo:

```
// istruzione per l'aggiornamento
$aggiorna_dati = $connessione->exec("UPDATE contatti SET nome = 'Anita' WHERE id = 1");
if ($aggiorna_dati === FALSE) {
  die("Nessuna modifica eseguita.");
}
// chiusura della connessione
 $connessione = null;                    // risorsa deallocata
```

Cancellazione (DELETE) di record con PDO

```
// istruzione per la cancellazione
  $cancella_dati = $connessione->exec("DELETE FROM contatti WHERE id = 1");
// chiusura della connessione
 $connessione = null;                    // risorsa deallocata
```

INSERIMENTO

```
// popolo la tabella contatti con due record
 $inserisci_dati = $connessione->exec("INSERT INTO contatti (nome, cognome)
                                       VALUES ('Giuseppe', 'Garibaldi')");
 $inserisci_dati_2 = $connessione->exec("INSERT INTO contatti (nome, cognome)
                                         VALUES ('Silvio', 'Pellico')");
 // visualizza l'identificatore univoco dell'ultimo inserimento effettuato
 // in questo caso stampa la cifra "2"
 echo $connessione->lastInsertId();
 // chiusura della connessione
$connessione = null;                    // deallocata risorsa
```

lastInsertId() senza la necessità del passaggio di alcun parametro, consentirà di visualizzare l'identificatore dell'ultimo inserimento effettuato tramite istruzione per verificare che il campo "id", essendo un auto-incrementale, si aggiorna tramite incremento numerico.

Si ricordi che il metodo **exec()** può essere accompagnato con la clausola di salvaguardia **or die()** al fine di interrompere l'esecuzione del codice qualora la query sia "ineseguibile" (una query sbagliata).

Creazione di una tabella

Per creare una tabella e controllare la riuscita dell'operazione:

http://new345.altervista.org/PHP/db/OO/install_contattiPDO.php

```
<?
include("config.inc.php");

/*  blocco try/catch di gestione delle eccezioni */
try {
  // stringa di connessione al DBMS
  $connessione = new PDO("mysql:host=$host;dbname=$db", $user, $password);
  // impostazione dell'attributo per il report degli errori
  $connessione->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  // creazione per popolamento della tabella contatti
  $crea_tb = $connessione->exec("CREATE TABLE IF NOT EXISTS contatti (id int(4) NOT NULL
                                  AUTO_INCREMENT,  nome varchar(30) NOT NULL,
                                 cognome varchar(40) NOT NULL,  PRIMARY KEY (id)  )");
  // chiusura della connessione
  $connessione = null;                        // risorsa deallocata
}
catch(PDOException $e)
{
  // notifica in caso di errore nel tentativo di connessione
  echo $e->getMessage();
}
?>
```

nb: in **PHP 7 non sono più incluse le** ***mysql functions*** che non vanno più considerate un'alternativa valida. Se si usa l'estensione **MySQLi** (*MySQL improved*), si sfrutta la caratteristica peculiare di questa estensione: quella di disporre di una doppia interfaccia: procedurale e **ad oggetti**.

Esempio di connessione a MySQL con ***estensione mysqli*** nell'***approccio ad oggetti***:

```
$mysqli = new mysqli('localhost', 'username', 'password', 'nome_database');
```

Prepared statements

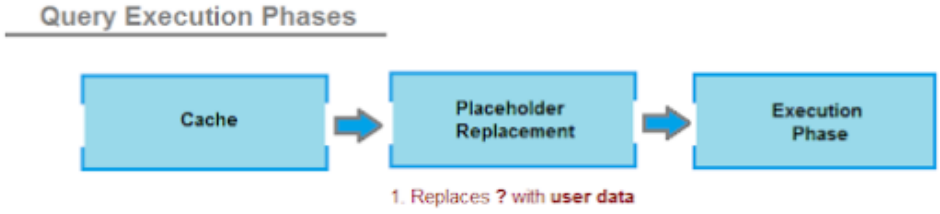
Uno degli ambiti di utilizzo pratico più importanti per i *Prepared Statements* è quello che prevede l'invio di parametri per le query tramite **campi form** compilabili dagli utenti o **querystring** presenti nella URL di una pagina web (e pertanto facilmente alterabili dall'esterno e quindi potenziamente pericolosi in quanto utilizzabili per veicolare istruzioni malevole): con l'uso dei *prepared statements*, invece, gli input non vengono determinati a priori dallo sviluppatore e il loro **contenuto rimane sconosciuto all'applicazione fino al momento dell'elaborazione**.

Un vantaggio di tipo sintattico dato da PDO è la possibilità di **assegnare dei nomi** ai parametri indicati nei *prepared statements* (come vedremo più avanti), senza dover utilizzare il formalismo meno intuitivo di MySQLi basato sui segnaposto (*placeholders*).

Prepared statements – *estensione MSQLi con approccio OO*

http://new345.altervista.org/PHP/db/OO/index_MySQLi_OO.php

```
<?
include("top_foot.inc.php");
include("config.inc.php");
top();

// connessione usando l'estensione MySQLi con approccio agli oggetti
$db = new mysqli($db_host, $db_user, $db_password);
if ($db ->connect_error)
  die ("Errore connessione: " . $db->connect_error);

// test dell'esistenza del database
if($db->select_db($db_name) == 0)
{
  echo("Il database non esiste");
  exit;
}

/* preparazione della query con parametri "?"  segnaposto (placeholder) */
$stmt = $db -> prepare (" SELECT id,data,titolo FROM news WHERE mail =?");
/* binding dei parametri */
$stmt -> bind_param ('s', $posta);                    // "s"  per stringa

/* set dei parametri ed esecuzione della query */
$posta = "infcol10@gmail.com";
$stmt-> execute ();

// estrazione dei risultati - prepared statement
 $ris = $stmt->get_result();

if ($ris->num_rows > 0)
{
   while ($row = $ris->fetch_assoc())
   {
      echo "<a href=\"view.php?id=$row[id]\">" . date("j/n/y", $row[data]) . " - $row[titolo]</a><br>";
   }
} else
        {
           echo "0 risultati";
        }
$stmt->close();
$db->close();
foot();
?>
```



Prepared statements – estensione PHP Data Objects

Estrazione

da HTML.it

```
$sql = 'SELECT nome, cognome, FROM utenti';
$stmt = $db->prepare($sql);
$stmt->execute();

$sql = 'SELECT nome, cognome, FROM utenti WHERE email like "%:email%"';
$stmt = $db->prepare($sql);
$stmt->bindParam(':email', $email, PDO::FETCH_ASSOC);
$stmt->execute();
```

Per conoscere il numero di record estratti: **$totale = $stmt->rowCount();**
Per **estrazione dei risultati :** **$ris = $stmt->get_result();**

Inserimento - Query parametriche

```
$sql = "INSERT INTO utenti(nome, cognome, email)
        VALUES(':nome',':cognome',':email',':anno')");
```

Si noti la sostituzione delle variabili, per esempio **$anno** viene rappresentato dal parametro **:anno**.

```
$stmt = $db->prepare($sql);
$stmt->bindParam(':nome', $nome, PDO::PARAM_STR);
$stmt->execute();
```

Il metodo **bindParam** accetta 3 argomenti, il nome del parametro, il valore del parametro e infine il tipo di dato, parametro opzionale

L'istruzione SQL e i parametri vengono spediti separatamente al database server (motivo per il quale non sarà possibile stampare la query "completa") e solo dopo la query verrà finalmente eseguita. Questo modo di procedere è quello che ci fornisce una maggiore **sicurezza** rispetto ai pericoli derivanti dall'iniezione di codice SQL.

bindParam vs. bindValue

bindValue associa il valore della variabile al parametro, **bindParam** referenzia invece la variabile agganciandola al parametro, in poche parole con bindValue non possiamo modificare il valore associato, mentre usando bindParam verrà usato il nuovo valore se quello della variabile cambia tra il *bind* e l'*execute*.

il codice seguente inserirà il valore "Lorenzo"	il codice seguente inserirà il valore "Pippo"
$sql = "INSERT INTO *utenti*(nome) VALUES(':nome'); $stmt = $db->prepare($sql); $nome = 'Lorenzo'; $stmt->**bindValue**(':nome', $nome, PDO::PARAM_STR); $stmt->execute();	$sql = "INSERT INTO *utenti*(nome) VALUES(':nome'); $stmt = $db->prepare($sql); $nome = 'Lorenzo'; $stmt->**bindParam**(':nome', $nome, PDO::PARAM_STR); **$nome = 'Pippo';** $stmt->execute();

Altri esempi da Mr. Webmaster

```
// definizione delle variabili per la query
$contatto_cognome = "Pellico";
$contatto_anni_vissuti = 64;

// preparazione della query SQL
$sql = $connessione->prepare("SELECT nome FROM contatti
                              WHERE cognome = :contatto_cognome
                              AND anni_vissuti = :contatto_anni_vissuti");
```

Ora si potrà passare al vincolo tra variabili e segnaposto (*binding*), ciò potrà avvenire per ciascun valore passando all'apposito metodo **bindParam()** i parametri relativi al segnaposto utilizzato nel template, alla variabile che contiene il valore e al tipo di dato da associare a tale valore (nel nostro caso abbiamo un intero e una stringa, per quest'ultima è stata indicata anche la lunghezza, pari a 7 caratteri, tramite un ulteriore parametro).

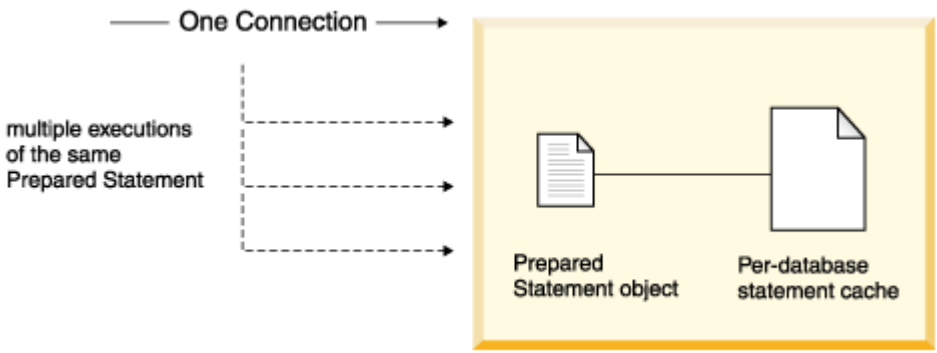
```
// bind dei parametri
$sql->bindParam(':contatto_cognome', $contatto_cognome, PDO::PARAM_STR, 7);
$sql->bindParam(':contatto_anni_vissuti', $contatto_anni_vissuti, PDO::PARAM_INT);
```

PDO::PARAM_STR e PDO::PARAM_INT sono delle costanti di classe utilizzate da PDO per PHP a partire dalla versione 5.1 di quest'ultimo per la rappresentazione di un data type SQL. A tal proposito potremo adottare le seguenti costanti:

1.PDO::PARAM_BOOL: rappresenta un tipo di dato booleano;
2.PDO::PARAM_NULL: rappresenta un tipo di dato SQL NULL (assenza di valore);
3.PDO::PARAM_INT: rappresenta un tipo di dato numerico intero (SQL INTEGER);
4.PDO::PARAM_STR: rappresenta un tipo di dato stringa (SQL CHAR, VARCHAR, TEXT..);
5.PDO::PARAM_LOB: rappresenta un tipo di dato large object di SQL.

Il *binding* dei parametri consentirà quindi di eseguire il ***Prepared statement*** tramite il metodo execute()

```
$sql->execute();
```



USO pratico

il form

```
<form action="ps_pdo.php" method="POST">
Il cognome dell'autore di "Le mie prigioni"?:<br />
<input type="text" name="cognome"><br/>
<input type="submit" value="Invia">
</form>
```

pagina *ps_pdo.php*

```
<?
if( (isset($_POST['submit'])) || (isset($_POST['cognome'])) ){
 /*    blocco dei parametri di connessione  */
 // nome di host
 $host = "localhost";
 // nome del database
 $db = "my_new345";
 // username dell'utente in connessione
 $user = "new345";
 // password dell'utente
 $password = "";        // opzionale

 /*    blocco try/catch di gestione delle eccezioni  */
 try {
  // stringa di connessione al DBMS
  $connessione = new PDO("mysql:host=$host;dbname=$db", $user, $password);
  // imposto l'attributo per il report degli errori
  $connessione->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
   // definizione delle variabili per la query
  $contatto_cognome = $_POST['cognome'];
  // preparazione della query SQL
  $sql = $connessione->prepare("SELECT id FROM contatti WHERE cognome = :contatto_cognome");
  // bind dei parametri
  $sql->bindParam(':contatto_cognome', $contatto_cognome, PDO::PARAM_STR, 7);
  // esecuzione del prepared statement
  $sql->execute();
  // conteggio dei record coinvolti dalla query
  if($sql->rowCount() > 0){
  // creazione di un'array contenente il risultato
    $result = $sql->fetchAll();       // array che contiene tutte le t-ple
    // ciclo dei risultati
    foreach($result as $row)
    {
     echo $row['id'] . "<br />";
    }
  }else{
   echo "Nessun record corrispondente alla richiesta.";
  }
  // chiusura della connessione
  $connessione = null;
 }
 catch(PDOException $e)  {    // notifica in caso di errore nel tentativo di connessione
                              echo $e->getMessage();
                            }
}else{  echo "Nessun parametro inviato dal form."; }
?>
```

Appendice

codice **config.inc.php**

```
<?
// parametri per connessione e selezione del database
$db_host = "localhost";
$db_user = "new345";
$db_password = "";   // opzionale
$db_name = "my_new345";
?>
```

codice **top_foot.inc.php**

```
<?
function top()
{
?>
<html>
<head>
<meta name=generator content="Script di corso INFORMATICO">
<style>
        body{background-image:url("sf_mail.gif");
        background-repeat:repeat-x;
        margin:2px auto 2px auto;
        width:760px;
        text-align:center;
        }
        div {text-align:left;}
</style>
</head>
<body>
       <h1>Interazione con DB remoto</h1>
       <br/>
<?
}
function foot()
{
?>
</body>
</html>
<?
}
?>
```