La connessione php-MySQL con MySQLi

Premessa

Lo scenario che si intende alla base di questo capitolo è di disporre di un ambiente php-mysql rappresentato nel seguente schema:

L'applicazione php viene eseguita dal server web, per accedere a un database deve essere attivata una connessione con il server MySql.

Presentiamo qui l'estensione MySQLi (MySQL Improved Extension) che quindi è una estensione 'migliorata' rispetto alla precedente 'MySQL Extention'.

Obiettivo di queste enstensioni è di fornire strumenti di accesso a DataBase ad alto livello, senza cioè dover implementare il protocollo di comunicazione fra client (php) e server (MySQL).

La release MySQLi qui presentata offre strumenti orientati allo sviluppo ad oggetti che invece erano assenti nella precedente.

Per garantire compatibilità con la grossa mole di software già esistenti sviluppati secondo il paradigma procedurale in realtà anche MySQLi offre delle funzioni di tipo procedurale, pertanto nelle pagine di manuale si trovano sempre doppia indicazione:

- object oriented syle: l'implementazione orientata agli oggetti, a cui qui facciamo riferimento
- procedural style: l'implementazione procedurale che qui tralasciamo.

Inoltre segnalo che in rete si trovano moltissimi esempi di script php che accedono a data base la maggior parte dei quali sono sviluppati seguendo il modello procedurale, per cui raccomando attenzione per non perdersi in ibridi procedural-oggetto che sconsiglio.

La classe MySQLi

Gli oggetti di questa classe hanno il compito di rappresentare la connessione fra php e un database MySQL.

Connessione al database

La funzione di connessione è associata all'istanza dell'oggetto, quindi è compito del costruttore della classe l'attivazione della connessione (vedi http://it.php.net/manual/en/mysqli.construct.php).

Il costruttore ha dunque bisogno di quattro informazioni basilari:

- host in cui risiede il server MySQL a cui connettersi
- username
- password
- database identifier

che devono essere passate come parametro all'istanziazione dell'oggetto.

Esempio:

```
$db=new MySQLi('localhost','root','','ilmiodb');
```

Come si vede i quattro parametri sono stringhe, nell'esempio si indica che il server mysql risiede sullo stesso host del server web-php, le credenziali di accesso sono per utente root privo di password (cosa frequente in macchine di sviluppo, ma totalmente errate su un server in produzione!) e il database ha identificatore 'ilmiodb'.

Classe di connessione personalizzata

In base a quanto visto l'istanza di un oggetto di classe MySQLi produce la connessione al database. In una applicazione web tipicamente sono presenti molti script php ognuno dei quali avrà necessità di accedere al database, pertanto in ognuno verrà istanziato un oggetto per la connessione fornendo ogni volta i parametri adeguati.

Punto debole di questa situazione è che se c'è bisogno di modificare un parametro di connessione (ad esempio la password) sarà necessario effettuare la modifica in tutte le occasioni in cui è prevista la connessione. Per questo motivo invece che specificare i parametri ad ogni esigenza di istanza di connessione può essere più efficace definire una propria classe di connessione che estende MySQLi su cui si ridefinisce il costruttore con i parametri impostati, ad esempio:

```
//file ConnessioneDb.php
class ConnessioneDb extends MySQLi{
  function __construct() {
    parent::__construct('localhost','root','','ilmiodb');
  }
}
```

I vari script dell'applicazione potranno ottenere la connessione istanziando un oggetto di classe ConnessioneDb senza specificare parametri. Qualora uno dei parametri dovesse essere modificato andrà ritoccato il solo costruttore della classe ConnessioneDb e tutti gli script dell'applicazione saranno implicitamente aggiornati.

Query

Il metodo query consente l'invio al dbms di un comando, con la conseguente esecuzione e ottenimento del risultato.

```
mixed query ( string $query [, int $resultmode =
MYSQLI_STORE_RESULT ] )
```

Il parametro stringa $query rappresenta il comando da inviare al dbms, sarà in generale un comando SQL.

Il secondo parametro opzionale al momento lo tralasciamo (assumerà quindi il suo valore di default)

Il valore di ritorno del metodo sarà:

- in caso di fallimento della query viene restituito il valore false

- in caso di successo della query viene restituito:
  - se la query è del tipo SELECT, SHOW, DESCRIBE, EXPLAIN un oggetto di classe MySQLi_Result
  - negli altri casi valore true

Si rimanda al paragrafo seguente il dettaglio sulla classe mysqli_result

Esempio di uso:

```
        $db=new ConnessioneDb();
        $ris = $db->query("SELECT * FROM rubrica");
```

La classe MySQLi_Result

Un oggetto di questa classe rappresenta la tabella risultato ottenuta da una query. Su questa tabella l'oggetto tiene un riferimento (cursore) ad una particolare riga, che chiamiamo riga corrente. All'istanza dell'oggetto la riga corrente è la prima.

La riga indirizzata dal cursore può essere accessibile con più di una modalità, qui per ora ne prendiamo in esame una.

Metodo fetch_assoc()

Questo metodo restituisce la riga corrente del risultato come array associativo in cui le chiavi sono costituite dagli identificatori di colonna della tabella risultato. Inoltre il cursore vine spostato alla riga seguente (se disponibile). Se il cursore è a fine tabella il metodo retituisce NULL.

Esempio

```
$db=new ConnessioneDb();
$ris = $db->query("SELECT * FROM rubrica");
$riga = $ris->fetch_assoc();
// $riga è un array che contiene:
// 'nome'=>'Mario', 'cognome'=>'Rossi', 'telefono'=>'0512345678'
```

Frequentemente si ha interesse a scandire l'intera tabella risultato, pertanto questa istruzione viene inserita in un ciclo controllato dal confronto con il valore NULL che indica il raggiungimento di fine tabella.

Esempio:

```
while ( ($riga=$ris->fetch_assoc()) !== NULL ) {
  // elaboro $riga secondo la finalità specifica dell'applicazione
}
```

Attributo num_rows

Questo attributo contiene il numero di righe della tabella risultato

Esempio: visualizzazione dei dati di una tabella

Ecco un esempio di come possa essere eseguita la connessione, query e visualizzazione dei dati del risultato.

```
require_once "ConnessioneDb.php";
$db=new ConnessioneDb();
$ris = $db->query("SELECT * FROM rubrica");
echo "La rubrica contiene {$ris->num_rows} contatti<br>\n";
while ( ($riga=$ris->fetch_assoc())!=NULL) {
 echo "{$riga['nome']} {$riga['cognome']} {$riga['telefono']}
<br>\n";
}
```

Gestione degli errori

Partiamo dell'esempio del paragrafo precedente, in particolare le due righe che eseguono connessione/query:

```
$db=new ConnessioneDb();
$ris = $db->query("SELECT * FROM rubrica");
```

e poniamoci l'obiettivo di verificare che ognuna di queste due abbia avuto successo. Se infatti per qualche motivo fallisce la connessione non ha senso tentare di eseguire una query. In modo simile se fallisce l'esecuzione di una query non ha senso trattare il risultato.

Errore di connessione

La classe MySQLi rende disponibile l'attributo

```
connect_error
```

che contiene:

- in caso di successo nella connessione il valore NULL
- in caso di errore una stringa con messaggio che descrive l'errore riconosciuto

Quindi ecco una possibile applicazione:

```
$db=new ConnessioneDb();
if ($db->connect_error){
  //connect_error non ha valore null, quindi c'è stato errore di
connessione
  die('Connection failed: '.$db->connect_error);
}
```

Si noti l'uso del die che interrompe l'esecuzione dello script, infatti avevamo già osservato che in assenza di connessione non ha senso proseguire.

Errore su query

La classe MySQLi rende disponibile l'attributo

```
error
```

In modo simile a quanto visto prima questo attributo riporta informazioni su condizioni di errore, in particolare si riferisce all'esito della più recente esecuzione di un metodo della classe (che non sia il costruttore).

L'atttributo contiene:

- in caso di successo una stringa vuota
- in caso di errore una stringa con messaggio che descrive l'errore riconosciuto

Quindi ecco una possibile applicazione:

```
$ris = $db->query("SELECT * FROM rubrica");
if ($db->error){
  //error non ha stringa vuota, quindi c'è stato errore di query
  die('Query failed: '.$db->error);
}
```

Anche in questo caso il riconoscimento dell'errore viene trattato interrompendo lo script.

Classe di connessione con gestione dell'errore

Nella maggior parte delle applicazioni che prevedono l'accesso a database si ripetono in vari punti le istruzioni di connessione e/o esecuzione di query, per ognuna di queste è bene effettuare la verifica .

Possiamo quindi pensare di portare la verifica all'interno della classe di gestione della connessione e di fatto liberare l'applicazione dall'onere di effettuare la verifica ogni volta.

```
class ConnessioneDb extends MySQLi{
  function __construct() {
    parent::__construct('localhost','root','','rubrica');
    if ($this->connect_error){
      die('Connection failed: '.$this->connect_error);
    }
  }
  function query($query) {
    $ris=parent::query($query);
    if ($this->error){
      die('Query failed: '.$this->error." query='$query'");
    }
    return $ris;
  }
}
```