Integrating MySQL with PHP

Phillip C.S.R. Kilgore, M.S.

Mar. 11, 2014
We now know about MySQL and PHP.

But how do the two integrate?
Databases are interesting, but if we can’t pull the two together, what’s the point?

Fortunately, PHP comes with the *PHP Data Objects* (PDO) API.

- An object-oriented interface for accessing a variety of databases.
- The database is connected to using a *driver* supplied by the database vendor.
- You must have MySQL support compiled into PHP to access MySQL.
PDO is a recent development and databases used to be accessed through a variety of database specific interfaces.

- Each database has its own set of functions which perform various primitive tasks.
- e.g., `mysql_connect()`, `mysql_query()`, etc.

As of PHP 5.5.0, these extensions are mostly *deprecated*.

- No consistency: different RBDMSes could have different interfaces.
- Represents a security risk (we’ll see how in a bit).
The SQL Injection Attack

Randall Munroe, XKCD - Exploits of a Mom
Licensed under Creative Commons Attrib-NonCommercial 2.5
The SQL Injection Attack

- The old API encouraged people to construct queries by hand.
  - e.g., $query = "SELECT * FROM User WHERE Name='\$value';";
  - Normally, $value is in domain.
  - However, if $value comes from a form, then it is possible for it to take on a bad value.

- By manipulating query construction, it is possible to inject code into a query.
  - $value = "; UPDATE User SET Name='Loser'; --";
  - After inserting a quote to terminate the string literal, the attacker has inserted arbitrary code of his own, and then used ‘–’ to set the remaining text as a comment.
  - This is a very frequent attack on databases.
PDO is *much* better because it provides one common interface for a variety of databases.

It is also more secure because statements can be parameterized.

- When you want to execute a statement, you *prepare* it first.
- Statements can then be executed by supplying the parameters.
- Each parameter is then automatically escaped to prevent the SQL injection attack.
- Don’t be tempted to build up statements by hand with PDO.
You must have a *Data Source Name* (DSN) to connect to the database.

- A DSN is either a URI specifying the connection or a predefined name.
- The DSN URI usually consists of a protocol handler prefix (e.g., 'mysql:') and a number of database-specific parameters.
- If you have access to php.ini, you can specify variables named `pdo.dsn.name` to create named connections.

Construct a new PDO object by supplying the DSN, username, and password.

- e.g., `$db = new PDO('mysql:host=localhost;dbname=foo', 'username', 'bar');`
- This constructor can accept additional driver-specific options in an array following the password.
PDO objects have the `exec()` and `query()` methods, but these should be avoided since they process raw queries.

Instead, use the `prepare()` function to create a prepared statement.

- Only required parameter is the query.
- You can use question marks for positional parameters (e.g., `SELECT * FROM Foo WHERE ID=?`);
- You can also use named parameters, (e.g., `SELECT * FROM Foo WHERE ID=:fooId`);
Statements must be executed after preparation to work.

- You can call `execute()`, optionally with an array of parameters.
- If named parameters are used, they are in key-value form.
- You can also use `bindParam()` in order to set parameters manually.

When you are finished with a statement, you should call `closeCursor()`.

- No further statements can be executed until the cursor is closed.
- This also allows the statement to be reused.
PDOStatement::execute() will return a boolean value indicating whether or not the statement was executed successfully.

If it has, then you can use PDOStatement::fetch() to grab each record.

- Accepts a single *fetch style*. Two examples include:
  - PDO::FETCH_NUM for an numerically-indexed array.
  - PDO::FETCH_ASSOC for a key-value array.

- Thus, you can scroll through the data set with a while loop.
- When there are no more records (or no records at all), false is returned and the loop terminates.
Configuring the database becomes a lot simpler if you have a separate configuration script.

- DB credentials are global and should affect every script.
- It may be OK to see the rest of the scripts; the database credentials should not be public.

A PHP script containing configuration variables works nicely.

- No special parsers needed, just include into your script.
- The webserver just show an empty page for the configuration if accessed.
- The script can be made visible only to the webserver and the owner.
- Note: proper security management is your responsibility.
```php
<?php

    // Ask for a MySQL database called 'database' on localhost
    $dsn = "mysql:host=localhost;dbname=database";

    // And supply our user credentials.
    $db_username = 'the_user';
    $db_password = 'the_password';

?>
```
<!DOCTYPE html>
<html>
<head>
  <title>Show Accounts</title>
</head>
<body>
<?php

/* Only access database when we have a query */
if (isset($_GET['query'])) {

  /* Load our database configuration. Note: banker-mysql.sql */
  /* must have been run first. */
  require('config.php');

  // Now create a database connection
  $db = new PDO($dsn, $db_username, $db_password);
  $stmt = null;
  try {
    /* Create the select statement */
Fetching the Banker Query II

22 $stmt = $db->prepare('SELECT ID, Name, Balance '  
23 . "FROM Account WHERE Name LIKE ?");  

25 /* Check to see if the statement executed */  
26 if (!$stmt->execute(array($_GET['query']))))  
27 throw new Exception(  
28 'Couldn\'t execute statement: '  
29 . $stmt->errorCode());

31 /* Iterate over the results */  
32 echo ' <table><tr><th>ID</th><th>Holder</th>'  
33 . ' <th>Balance</th></tr>';  
34 while ($row = $stmt->fetch(PDO::FETCH_NUM)) {  
35 /* Iterate over row cells */  
36 echo ' <tr>';  
37 foreach ($row as $cell) {  
38 /* Make sure to escape report */  
39 $cell = htmlspecialchars($cell);  
40 echo "<td>$cell</td>";  
41 }  
42 echo ' </tr>';  
43 }
```php
    echo '</table>);

    /* Figure out what went wrong */
    catch (Exception $ex) {
        echo '<p><b>Error:</b> ' . $ex->getMessage() . '</p>);
    }

    /* Now do some cleanup */
    if ($stmt !== NULL)
        $stmt->closeCursor();
    }

    ?></form>
    <form action='banker-accounts.php' method='get'>
        <label for='query'>Query:</label>
        <input type='text' name='query'/>
        <input type='submit' value='Search'/>
    </form>
</body>
</html>
```
Working with Non-Query Statements

- Not all statements are able to be fetched from.
  - e.g., INSERT, UPDATE, DELETE.
  - You can use PDOStatement::rowCount() to find out how many rows were affected.
  - You can also use PDO::lastInsertId() to get the value of an AUTO_INCREMENT column during an INSERT statement.
<!DOCTYPE html>
<html>
<head>
  <title>Open a New Account</title>
</head>
<body>
<?php
  /* Load our database configuration. Note: banker-mysql.sql
   * must have been run first. */
  require('config.php');

  /* Necessary to make sure our balance is valid */
  function validateBalance($balance) {
    /* Empty string treated as default */
    if (empty($balance)) {
      return 0.0;
    }
    /* Must be numeric */
    else if (!is_numeric($balance))

Phillip C.S.R. Kilgore, M.S.
CSC435: SQL + PHP
Using INSERT to Enroll Accounts II

```php
return false;
/* Must be positive */
else if ($balance < 0)
    return false;
else
    return $balance;
}
$db = new PDO($dsn, $db_username, $db_password);

// Check to see if we have an update.
if (isset($_POST['name'])) {
    // See if we want to bind the parameters */
    $okToExec = true;
    $balance = 0;
    /* User supplied a balance */
    if (isset($_POST['balance'])) {
        $balance = validateBalance($_POST['balance']);
        if ($balance !== false)
            $okToExec = false;
    }
    /* Assume 0.0 balance */
```
else {
    $balance = 0;
}

/* Now create the new item */
if ($okToExec) {
    $stmt = $db->prepare(
        'INSERT INTO Account(Name, Balance) '
        . 'VALUES(:name, :balance);');

    // Bind parameters (or we can use a key-value array)
    $stmt->bindParam('name', $_POST['name']);
    $stmt->bindParam('balance', $balance);

    /* Execute the result */
    if ($stmt->execute()) {
        $affected = $stmt->rowCount();
        $newId = $db->lastInsertId();
        /* Display how the DB was affected. */
        echo "<p>Added account $newId "
            . "($affected rows affected)</p>";
    }
}
Using INSERT to Enroll Accounts IV

```php
else
    echo '<p><b>Error:</b> ' . $stmt->errorCode();
    $stmt->closeCursor();
}
/* An error occurred and we need to report it. */
else {
    echo '<p>Error: ' . $stmt->errorCode();
    $stmt->closeCursor();
}
/* An error occurred and we need to report it. */
```

```php
<h1>Present Accounts</h1><hr/>
<table><tr><th>ID</th><th>Name</th><th>Balance</th></tr>
<<?
// Dump the current table.
$stmt = $db->prepare('SELECT ID, Name, Balance FROM Account');
if ($stmt->execute()) {
    while ($row = $stmt->fetch(PDO::FETCH_NUM)) {
        echo '<tr>';
```

Phillip C.S.R. Kilgore, M.S.  
CSC435: SQL + PHP
Using INSERT to Enroll Accounts

```php
foreach ($row as $cell) {
    $cell = htmlspecialchars($cell);
    echo "<td>$cell</td>";
}
echo '</tr>';
}
$stmt->closeCursor();
?>
</table>
<h1>Open a New Account</h1><hr/>
<form action='banker-enroll.php' method='post'>
<p><label for='name'>Account Name:</label><input type='text' name='name'/></p>
<p><label for='name'>Opening Balance:</label><input type='text' name='balance'/></p>
<input type='submit' value='Enroll'/>
</form>
</body>
</html>
```
In most cases, you want only one copy of the database open.
- And it should be globally available if possible.
- We can use the static keyword to ensure that only one database connection is open during the script lifetime.

Create a function with a static variable that is initially null.
- When the function first runs, a new database connection is created and set as the value of the static variable.
- The configuration script is included locally; no trace of the variables will remain.
- Subsequent calls will use the connection.
In most cases, it makes sense to provide constrained input to the user.

- To prevent invalid data from being entered.
- Should still enforce integrity constraints on database whenever possible.

May include a variety of input mechanisms:

- Radio sets or combo boxes for mutually exclusive options.
- Checkboxes for mutually selectable options.
Difficult, because not all databases support stored procedures, and conventions can be slightly different.

- For MySQL, use `CALL` with a parameterized, prepared statement.
- Behaves much like a select statement.
  - May possibly have a result set.
  - But could (in fact) have multiple such sets.
  - Use `PDOStatement::nextResult()` to use the next available result set.
<?php

function dbConnection() {
    // Will remain throughout the lifetime of the script
    static $db = null;

    // Force load on first call.
    if ($db === null) {
        require ('.config.php');
        /* Now that the static variable is set, this object will be returned
        * on subsequent invocations.
        */
        $db = new PDO($dsn, $db_username, $db_password);
    }

    return $db;
}
?>
<!DOCTYPE html>
<html>
<head>
    <title>Update Balances</title>
</head>
<body>
<?php
ini_set('display_errors', true);

/* Necessary to make sure our balance in valid */
function validateBalance($balance) {
    /* Empty string treated as default */
    if (empty($balance)) {
        return 0.0;
    }
    /* Must be numeric */
    else if (!is_numeric($balance))
        return false;
    /* Must be positive */
    else if ($balance < 0)
        return false;
</body>
</html>
return false;
else
    return $balance;
}

/* Now using the singleton pattern. */
require('DB.php');
$db = dbConnection();

/* Now perform validation */
if (isset($_POST['account']) && isset($_POST['action'])) {
    $amount = 0.0;
    $query;

    // Determine an appropriate action to perform
    switch($_POST['action']) {
        case 'debit':
            $query = 'CALL debit(:id,:amount);';
            break;
        case 'credit':
            $query = 'CALL credit(:id,:amount);';
            break;
    }
default:
  die('Invalid action specified: ' .
  htmlspecialchars($_POST['action']));
}

/* Validate the balance */
if (isset($_POST['amount'])) {
  $amount = validateBalance($_POST['amount']);
}

/* Otherwise, perform the selected action */
if ($amount !== 0.0) {
  $stmt = $db->prepare($query);
  /* Using key-value array this time */
  if ($stmt->execute(array('id' => $_POST['account'],
                           'amount' => $amount))) {

    /* Obtain the return value */
    $result = $stmt->fetch(PDO::FETCH_NUM);
    $result = $result[0];
    if ($result === null)
Using INSERT to Enroll Accounts IV

```php
<?php

    if ($result > 0)
    {
        echo '<p>Warning: invalid account specified.</p>);
        }

        else if ($result < 0)
        {
            echo '<p>Warning: account would be overdrafted ' . "by $result</p>");
            }

            $stmt->closeCursor();
        }

    ?>

<h1>Present Accounts</h1><hr/>
<table><tr><th>ID</th><th>Name</th><th>Balance</th></tr>

    atas = array();
    $stmt = $db->prepare('SELECT ID, Name, Balance FROM Account');
    if ($stmt->execute()) {
        while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
            // Add available account
            $accounts[$row['Name']] = $row['ID'];
```
// Now emit account row.
echo '<tr>';
foreach ($row as $key => $value) {
    echo '<td>$value</td>'';
}
echo '</td>$value</td>'';
}

$stmt->closeCursor();
?>
</table>
<h1>Update Balances</h1><hr/>
<form action='banker-update.php' method='post'>
<p><label for='account'>Account:</label>
    <!-- Begin constrained input for account -->
    <select name='account'/><?
        foreach ($accounts as $name => $id) {
            $name = htmlspecialchars($name);
            echo '<option value='$id'>$name</option>'';

        }
    </select>
</p><label for='account'>Account:</label>
</form>


```html
}?
</select></p>

<!-- Radio group for action in question. -->
<p><label for='action'>Action:</label>
<input type="radio" checked="checked" name="action" value="credit"/>Credit
<input type="radio" name="action" value="debit"/>Debit
</p>
<p><label for='amount'>Amount:</label>
<input type='text' name='amount'/></p>
<input type='submit' value='Enroll'/>
</form>
</body>
</html>
```
Because PHP supports Object-Oriented Programming, the Model-View-Controller pattern can be leveraged.

- The *model* is a PHP object that represents the entity in question and is backed by the database.
- The *view* is responsible for displaying the entity.
- The *controller* is code responsible for handling actions performed on the entity.

MVC greatly simplifies writing forms.
- Because most operations are specified in terms of the model.
- Need a new controller? No problem, work through your model code.
Controller Scripts

- AKA, processing scripts or handlers.
  - Sole purpose is to perform some action on the data, and nothing else.
  - Controllers are usually very simple: they usually just process one form.
- It is useful to use the magic header redirect to return to the view.
  - The $SERVER[’HTTP_REFERER’] variable contains the source URL (if the browser reported it).
  - If this is not set, you should have a fallback value.
-- Create the Guestbook table
CREATE TABLE GuestbookEntry (ID int NOT NULL AUTO_INCREMENT, 
    Subject VARCHAR(64) NOT NULL, 
    Author VARCHAR(64) NOT NULL, 
    Contents TEXT NOT NULL, 
    PostedOn DATETIME NOT NULL, 
    PRIMARY KEY (ID) );
<?php

// Make sure we have the require database wrapper
require_once('DB.php');

class GuestbookEntry {
    private $m_id;
    private $m_subject;
    private $m_author;
    private $m_date;
    private $m_contents;

    public function __construct($subject, $author, $contents, $id = -1, $date = null) {
        $this->m_id = $id;
        $this->m_subject = $subject;
        $this->m_author = $author;
        $this->m_date = $date;
        $this->m_contents = $contents;
    }
}
/* Permanently delete this entry. */
public function remove() {
    $db = dbConnection();

    /* Cache the statement for reuse */
    static $stmt = null;
    if ($stmt === null)
        $stmt = $db->prepare('DELETE FROM GuestbookEntry
                        WHERE ID=?');

    /* Now delete the statement */
    $ret = $stmt->execute(array($this->m_id));
    $stmt->closeCursor();
    return $ret;
}

/* Add this guestbook entry as if it were a prototype. The
 * ID is set on success. */
public function add() {
    $db = dbConnection();
/* Cache the statement for reuse */
static $stmt = null;
if ($stmt === null)
    $stmt = $db->prepare('INSERT INTO GuestbookEntry
        (Subject, Author, Contents, PostedOn)
        VALUES(:subject, :author, :contents, ').*NOW()');
*/
/* Bind our parameters */
$stmt->bindParam('subject', $this->m_subject);
$stmt->bindParam('author', $this->m_author);
$stmt->bindParam('contents', $this->m_contents);
/* Now run the statement */
$ret = $stmt->execute();
if ($ret)
    $this->m_id = $db->lastInsertId();
$stmt->closeCursor();
return $ret;
public static function fetch() {
    $ret = array();
    $db = dbConnection();

    /* Cache the statement for reuse */
    static $stmt = null;
    if ($stmt === null)
        $stmt = $db->prepare('SELECT ID, Subject, Author, ' . 'Contents, PostedOn FROM GuestbookEntry ' . 'ORDER BY PostedOn;');

    /* Build up a list of database objects */
    if ($stmt->execute()) {
        while($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
            array_push($ret, new GuestbookEntry($row['Subject'], $row['Author'], $row['Contents'], $row['ID'], $row['PostedOn']));
        }
    }
}
87 /* Now execute the statement */
88 return $ret;
89 }
90
91
92 /* Accessors */
93
94 public function id() {
95 return $this->m_id;
96 }
97
98 public function subject() {
99 return $this->m_subject;
100 }
101
102 public function date() {
103 return $this->m_date;
104 }
105
106 public function contents() {
107 return $this->m_contents;
108 }
public function author() {
    return $this->m_author;
}
?>
<!DOCTYPE html>
<!-- View for the Guestbook -->
<html>
<head><title>My Guestbook</title></head>
<body>
<h1>Leave a message</h1><hr/>
<!-- Allow us to add to the guestbook. -->
<form action='guestbook-add.php' method='post'>
<p><label for="subject">Subject: </label><input type="text" name="subject"/></p>
<p><label for="author">Author: </label><input type="text" name="author"/></p>
<p><label for="body">Body: </label><textarea name="body"></textarea></p>
<input type="submit"/><input type="reset"/>
</form>
<h1>View Past Messages</h1><hr/>
<form action='guestbook-delete.php' method='post'>
<?php
Phillip C.S.R. Kilgore, M.S.
CSC435: SQL + PHP
LSUS Research
```php
require('guestbook-model.php');

foreach (GuestbookEntry::fetch() as $record) {
    printf('<p><input type="checkbox" name="cand[]" value="%d"/>
        <b>%s - %s</b></p><p>by %s</p></b>
        . '<p>%s</p>',
        $record->id(),
        htmlspecialchars($record->subject()),
        htmlspecialchars($record->date()),
        htmlspecialchars($record->author()),
        htmlspecialchars($record->subject())
    );
}

<input type='submit' value='Delete Selected'/>
</form></body>
</html>
```
<?php

/* Pull in required model */
require_once('guestbook-model.php');

/* We must have everything set or else */
if (!isset($_POST['subject']) && isset($_POST['author'])
    && isset($_POST['body'])) {
    print_r($_POST);
    die('<p>Not all requirements are set. Click here to go back.</p>');
}

/* Now create the Guestbook entry */
$gbe = new GuestbookEntry($_POST['subject'], $_POST['author'],
    $_POST['body']);

if (!$gbe->add()) {
    die("<p>Can’t add post for some reason.</p>");
}

/* Try to obtain the referer from the post */
$referer;

if (isset($_SERVER['HTTP_REFERER']))
    $referer = $_SERVER['HTTP_REFERER'];
else
    $referer = 'guestbook.php';

/* Perform magic header redirect. */
header("Location: $referer");
?>
<?php

/* Pull in required model */
require_once('guestbook-model.php');

/* We must have an array */
if (!isset($_POST['cand']) && is_array($_POST['cand'])) {
    print_r($_POST);
    die('<p>Not all requirements are set. Click here to go back.</p>');</n
// Delete candidate records
foreach (GuestbookEntry::fetch() as $record) {
    if (in_array($record->id(), $_POST['cand'])) {
        $record->remove();
    }
}

/* Try to obtain the referer from the post */
$referer;
if (isset($_SERVER['HTTP_REFERER'])) {
$referer = $_SERVER[‘HTTP_REFERER’];

else
    $referer = ‘guestbook.php’;

/* Perform magic header redirect. */
header("Location: $referer");
?>