LAB 11

WORKING WITH DATABASES

What You Will Learn

- How to install and manage a MySQL database
- How to use SQL queries in you PHP code
- How to integrate user inputs into SQL queries
- How to manage files inside of a MySQL database

Approximate Time

The exercises in this lab should take approximately 120 minutes to complete.

Fundamentals of Web Development

Randy Connolly and Ricardo Hoar

Textbook by Pearson
http://www.funwebdev.com

INTRODUCING MYSQL

PREPARING DIRECTORIES

1. If you haven't done so already, create a folder in your personal drive for all the labs for this book. This lab (and the next ones as well) requires a functioning webserver and MySQL installation. This lab assumes you are using easyPHP.
2. From the main `labs` folder (either downloaded from the textbook's web site using the code provided with the textbook or in a common location provided by your instructor), copy the folder titled `lab11` to your course folder created in step one.

If you have finished lab 8, you will likely already have MySQL installed locally on your development machine, set up on a laboratory web server, or set up on your web host's server.

If you are using easyPHP, MySQL will be installed but may not yet be running. The next exercise will show you how to start MySQL in the easyPHP environment. If you are using a different installation environment, you will need to follow the MySQL start up instructions for that environment.

EXERCISE 11.1 — STARTING MYSQL IN EASYPHP

1. Start the easyPHP administration window if it is not already running.
2. In the Administration browser window, you can access the MySQL administration via the Modules section (see Figure 11.1).
3. You can do this via the icon next to the text MYSQL at the top of the easyPHP Administration window.

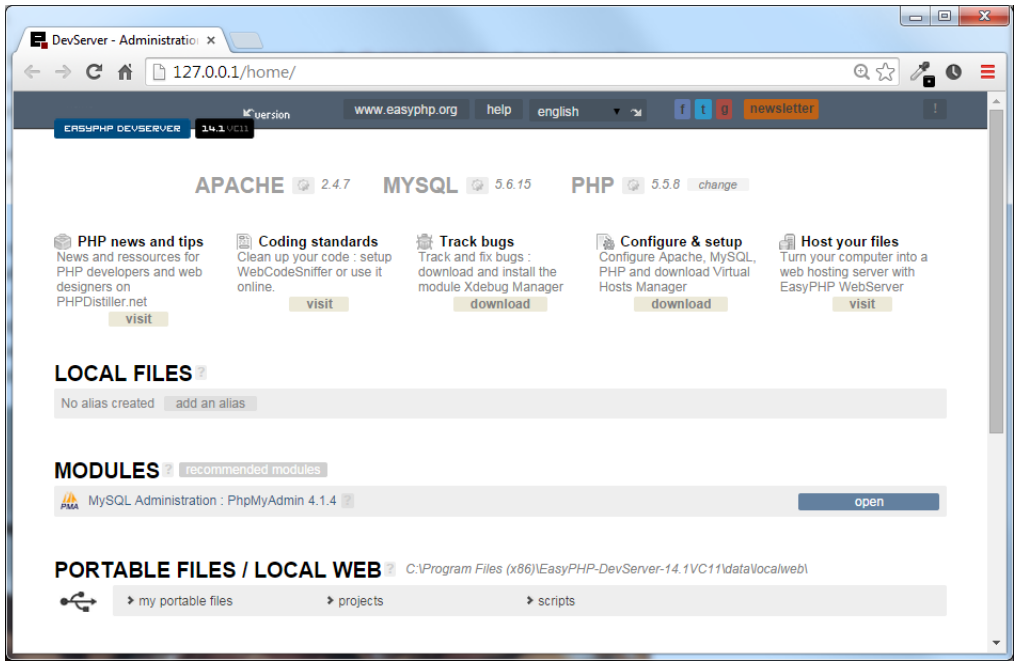

Figure 11.1 – Accessing MySQL in easyPHP

Although you will be able to manipulate the database from your PHP code, there are some routine maintenance operations that typically do not warrant writing custom PHP code. For these types of tasks, we will use a MySQL management tool. This could be a command line interface (if you are using Linux and have access to the command line) or the popular web-based front-end phpMyAdmin.

EXERCISE 11.2 — MANAGEMENT TOOLS

1 Start phpAdmin. In easyPHP, you can do this via the Administration window. If you are asked to login in, jump to step X.

If you are not asked to login in, then you should see the phpMyAdmin panel as shown in Figure 11.2. If this is your first time using phpAdmin on your machine, you will likely need to do steps 2-Y.

2 If this is the first time using MySQL on your machine, you should specify a password for the MySQL **root** user before proceeding. To do so, click on the Users link/tab at the top of the phpAdmin page.

3 From the list of users, click the Edit Privileges link for user root, host 127.0.0.1.

You will see a list of privileges and other information for this user.

4 Scroll down to the login information. And specify a password for the root user. Be sure to remember yours!! Click the Go button when ready. Return to the main phpMyAdmin

page by clicking on the logo and then jumping to step 6.

You should see a message indicating that the password was changed successfully.

5 Login using your MySQL credentials.

6 The left side of phpMyAdmin displays the existing databases in MySQL. A default installation of MySQL contains a variety of system tables used by MySQL (which depending on your installation may or may not be visible here).

Check if your installation of MySQL already has the art, books, and travels databases installed (as shown in Figure 11.2). If not, jump to Exercises 11.3 and then return to step 7.

7 Click on the **art** database.

This will display the tables in the art database.

8 Click the browse link for the **Artist** table.

This will display the first set of records in this table with edit/copy/delete links for each record.

9 Click on the Structure tab.

This will display the definitions for the fields in the current table, with links for modifying the structure.

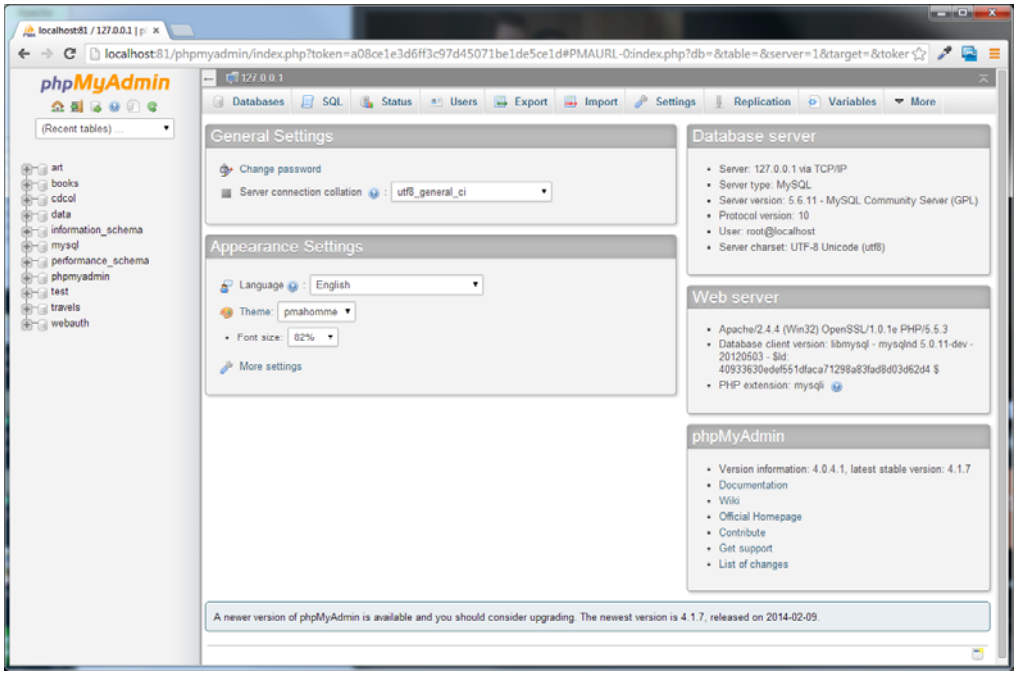

Figure 11.2 – phpMyAdmin

EXERCISE 11.3 — INSTALLING DATABASE IN PHPMYADMIN

1 Examine art.sql in a text editor. When done, close the file.

These import scripts contain the necessary DDL statements to define the database tables as well as the INSERT statements to populate the tables with data.

2 In phpMyAdmin, click on the Databases tab.

3 Create a database called **art**.

When it is complete, the art database will be visible in left-side of phpAdmin window.

4 Click on art database on left-side of window.

Currently there are no tables in this database. You can manually create a new table here, or using the art.sql script to populate the database.

5 Click on the Import tab.

6 Use the Choose File button to select the art.sql file examined in step 1. Then click the Go button.

If import works successfully, then you should be able to examine the tables in the database.

7 Repeat steps 3-6 for **travels** database (use travels.sql).

8 Repeat steps 3-6 for **books** database (use bookcrm.sql).

9 If everything worked correctly, your database will have new tables that are populated with data. One of the more common problems that can occur with importing is the size restriction on uploaded files in PHP. You can examine and change this setting by examining the PHP configuration settings (see Figure 11.3).

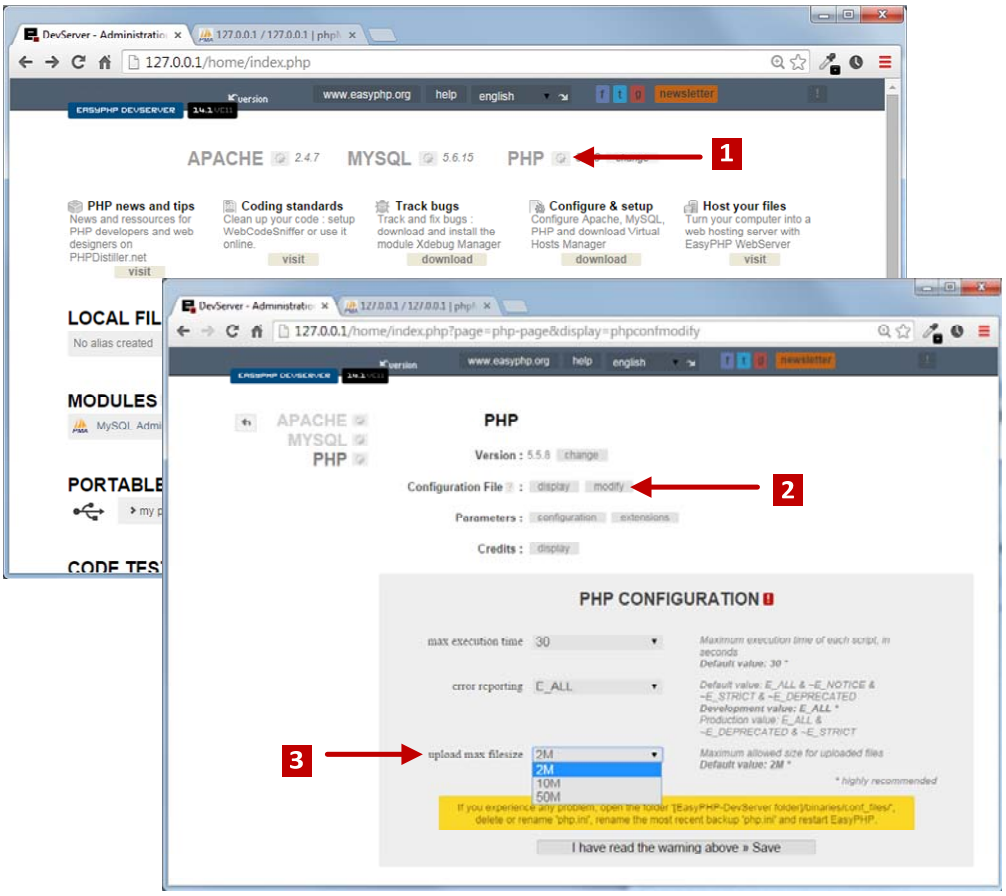

Figure 11.3 – Changing maximum upload size

SQL

MySQL, like other relational databases, uses Structured Query Language or, as it is more commonly called, SQL (pronounced sequel) as the mechanism for storing and manipulating data. Later in the lab you will use PHP to execute SQL statements. However you will often find it helpful to run SQL statements directly in MySQL, especially when debugging.

The following exercises assume that your databases have been created and populated. They also use phpMyAdmin to run the SQL commands. Depending on your installation, you may instead be using the command line.

EXERCISE 11.4 — QUERYING A DATABASE

1 In phpMyAdmin, click on the art database, then click on the **SQL** tab.

You should see the message "Run SQL query/queries on database art:"

2 In the blank SQL window, enter the following.

```
select * from artists
```

Remember that SQL is not case sensitive. So while the SQL examples in the textbook uses uppercase for SQL reserved words, that is purely a convention for readability. You will likely find it easier to simply type the SQL in all lowercase (as shown here).

3 Now press the Go button.

This will run the query. Notice that only the first 30 records are retrieved. This limit is appended to each query for performance reasons (you likely will not want all million records in a given table for instance). If you wish to see all the records retrieved from a query, there is a Show All link at the bottom of the retrieved records.

4 Return to the SQL window, enter the following new query, and then press Go.

```
select artworkid, title, yearofwork from artworks
where yearofwork < 1600
```

This will display just the art works prior to 1600. Notice that in MySQL, a query can be spread across multiple lines.

5 Modify the query (you can click the **Show query box** link) as follows and test.

```
select artworkid, title, yearofwork from artworks
where yearofwork < 1600 order by yearofwork
```

6 Modify the query as follows and test.

```
SELECT Artists.ArtistID, Title, YearOfWork, LastName FROM Artists
INNER JOIN ArtWorks ON Artists.ArtistID = ArtWorks.ArtistID
```

This query contains a join since it queries information from two tables. Notice that you must preface ArtistId with the table name since both joined tables contain a field called ArtistId..

7 Modify the query as follows and test.

```
SELECT Nationality, Count(ArtistID) AS NumArtists
FROM Artists
GROUP BY Nationality
```

This query contains an aggregate function as well as a grouping command.

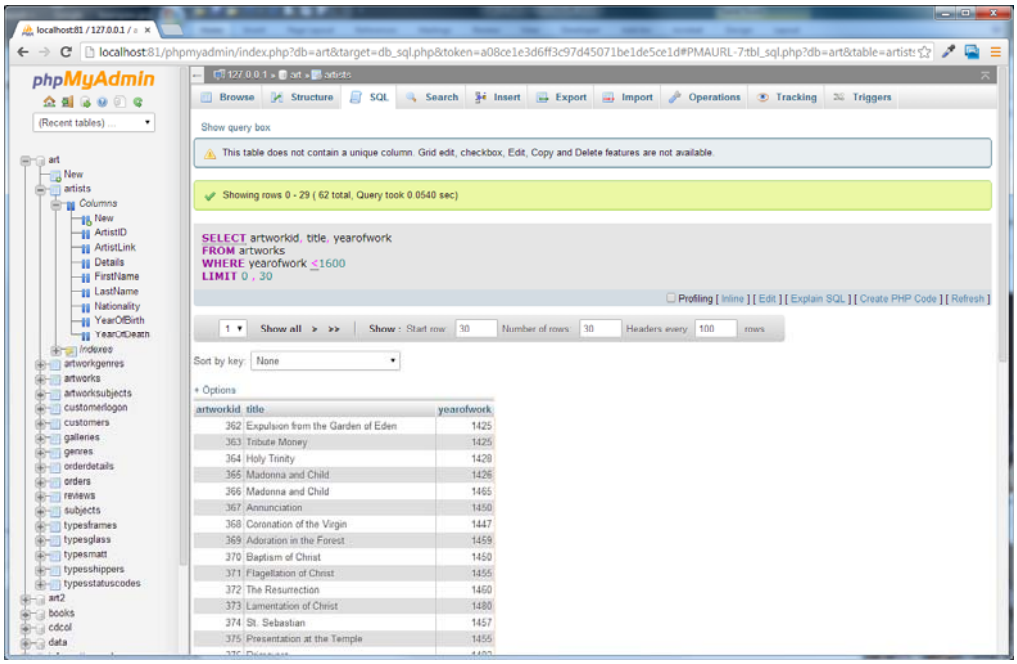

Figure 11.4 – Exercise 11.4

EXERCISE 11.5 — MODIFYING RECORDS

1 In phpMyAdmin, click on the art database, then click on the SQL tab.

You should see the message "Run SQL query/queries on database art:"

2 In the blank SQL window, enter the following.

```
insert into artists (firstname, lastname, nationality, yearofbirth,
yearofdeath) values ('Palma','Vecchio','Italy',1480,1528)
```

3 Press the Go button.

You should see message about one row being inserted.

4 Examine the just-inserted record by running the following query.

```
select * from artists where lastname = 'Vecchio'
```

Notice that `ArtistId` value has been auto-generated by MySQL. This has happened because this key field has the auto-increment property set to true.

5 Run the following new query:

```
update artists
set details='Palmo Vecchio was an Italian painter of the Venetian school'
```

6 Verify the record was updated (i.e, by running the query from step 4).

7 Run the following new query:

```
delete from artists
```

```
where lastname = 'Vecchio'
```

7 Verify the delete worked by running the following query:

```
SELECT * FROM artists WHERE nationality = 'Italy'
```

One of the key benefits of databases is that the data they store can be accessed by queries. This allows us to search a database for a particular pattern and have a resulting set of matching elements returned quickly. In large sets of data, searching for a particular record can take a long time. To speed retrieval times, a special data structure called an index is used. A database table can contain one or more indexes.

EXERCISE 11.6 — BUILD AN INDEX

1 In phpMyAdmin, click on the art database, click on the artworks table, and then click on the **Structure** tab.

2 In the Structure page, click on the **Indexes** link.

You will see a list of already-existing indexes. The import script for this database already has created indexes for the primary key, the foreign keys, and a two other fields.

3 Click the Go button in the section that begins Create an index ...

4 In the pop-up window, name the index YearOfWork, the index type INDEX, and the column YearOfWork as shown in Figure 11.4. Click Go.

There will be a short delay as MySQL creates the index.

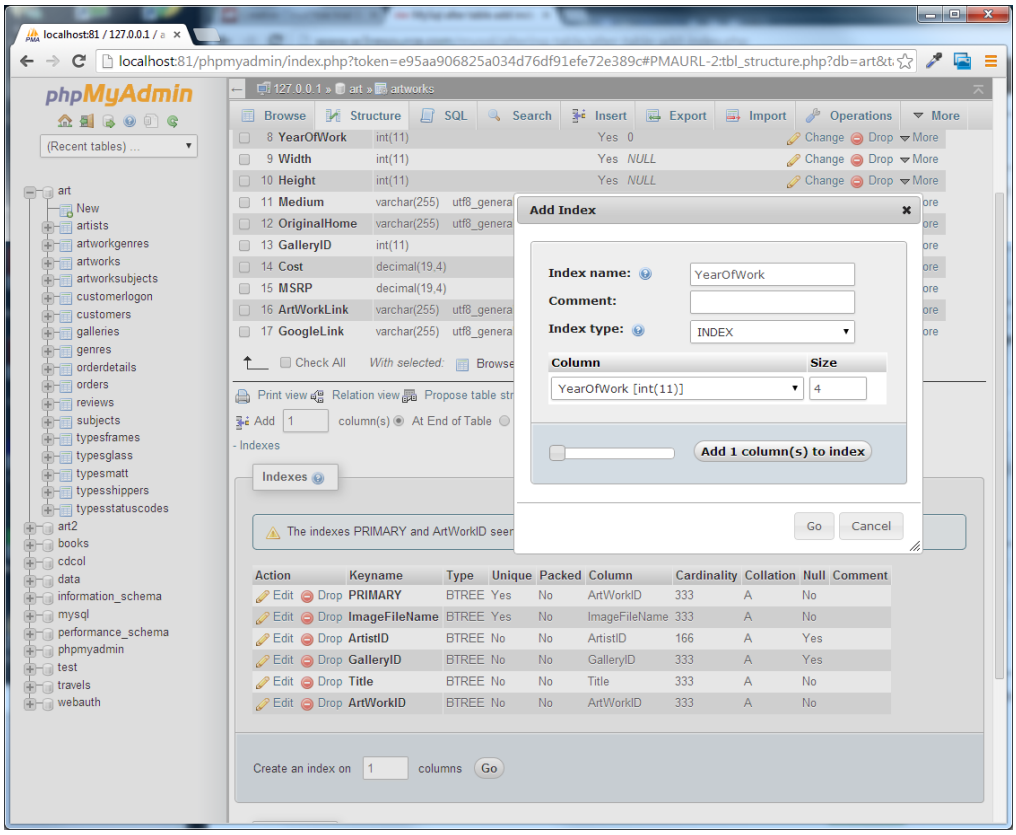

Figure 11.5 – Creating an index in phpMyAdmin

EXERCISE 11.7 — CREATING USERS IN PHPADMIN

1 In phpMyAdmin, click on the art database, and then click on the **Privileges** tab.

This will display the users who currently have access to this database. Notice the root user. This root user has special privileges within MySQL: indeed, you very well may have logged into phpMyAdmin using the root account.

You are going to create a new user which you will use for subsequent examples in this lab.

2 Click the **Add user** link.

This will display the Add user page (see Figure 11.5).

3 In the Add user page, enter the following into the Login information section:

```
User name (use text filed): testuser
Host (Local): localhost
Password (use text filed): mypassword
Re-Type: mypassword
```

You are of course welcome to enter a different user name and password. If you do, you

will need to substitute future references to testuser and password.

4 In the **Database for user** section, check the **Grant all privileges on database "art"** checkbox.

5 In the **Global privileges** section, check the five Data privileges (select, insert, update, delete, and file).

6 Click the Go button.

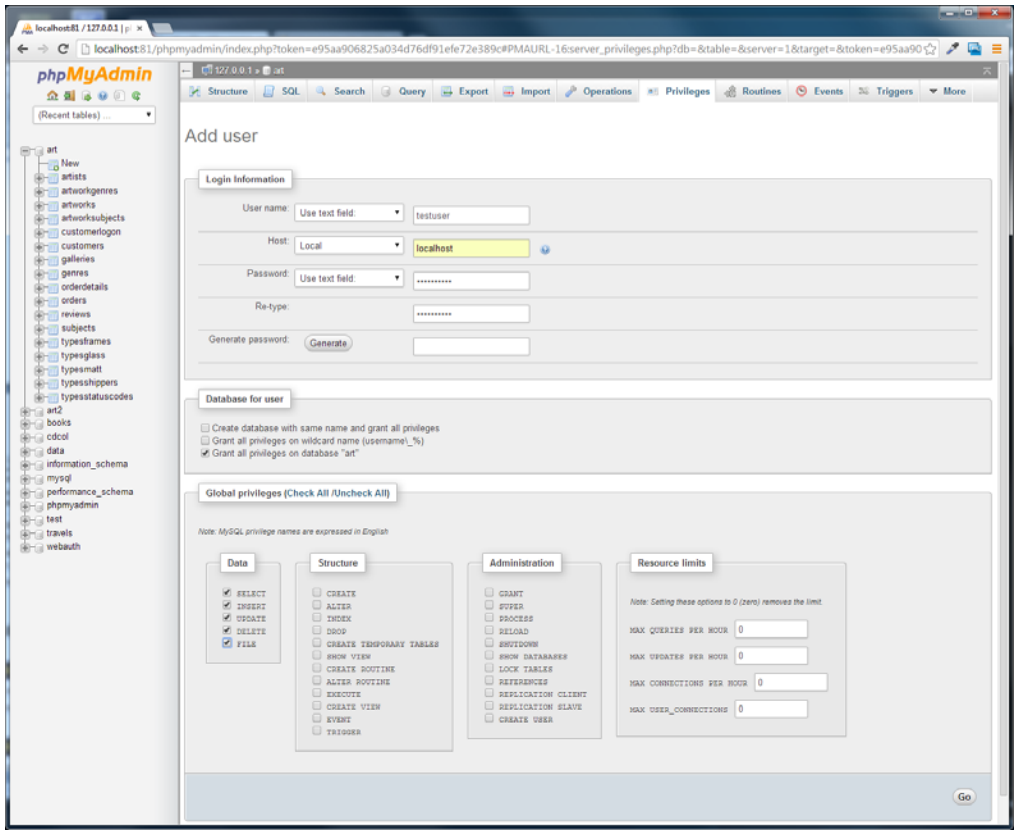

Figure 11.6 – Creating a user

ACCESSING MYSQL IN PHP

EXERCISE 11.8 — MYSQL THROUGH PHP

1 Open config.php and modify the file as follows (if you used different user name and password in Exercise 11.7, then you will have to change what you enter here).

```
<?php
define('DBHOST', 'localhost');
define('DBNAME', 'art');
```

```
define('DBUSER', 'testuser');
define('DBPASS', 'mypassword');
define('DBCONNSTRING','mysql:host=localhost;dbname=art');
?>
```

2 Open lab11-exercise08-pdo.php and modify as follows:

```
<?php require_once('config.php'); ?>
<!DOCTYPE html>
<html>
<body>
<h1>Database Tester (PDO)</h1>
<?php
try {
   $pdo = new PDO(DBCONNSTRING,DBUSER,DBPASS);
   $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
   $sql = "select * from Artists order by LastName";
   $result = $pdo->query($sql);
   while ($row = $result->fetch()) {
      echo $row['ArtistID'] . " - " . $row['LastName'] . "<br/>";
   }
   $pdo = null;
}
catch (PDOException $e) {
   die( $e->getMessage() );
}
?>
</body>
</html>
```

This uses the object-oriented PDO API for accessing databases.

3 Save and test.

This should display list of artists. Notice

4 Open lab11-exercise08-mysqli.php and modify as follows:

```
<?php require_once('config.php'); ?>
<!DOCTYPE html>
<html>
<body>
<h1>Database Tester (mysqli)</h1>
Genre:
<select>
<?php

$connection = mysqli_connect(DBHOST, DBUSER, DBPASS, DBNAME);
if ( mysqli_connect_errno() ) {
   die( mysqli_connect_error() );
}

$sql = "select * from Genres order by GenreName";

if ($result = mysqli_query($connection, $sql)) {

   // loop through the data
   while($row = mysqli_fetch_assoc($result))
    {
```

```
            echo '<option value="' . $row['GenreID'] . '">';
            echo $row['GenreName'];
            echo "</option>";
        }
        // release the memory used by the result set
        mysqli_free_result($result);
    }

    // close the database connection
    mysqli_close($connection);

    ?>
    </select>
    </body>
    </html>
```

This uses the procedural mysqli API for accessing databases.

EXERCISE 11.9 — INTEGRATING USER INPUTS (MYSQLI)

1 Open and examine lab11-exercise09.php.

This page already contains the code for displaying a select list containing all the galleries in the gallery table. You will be adding the code to display a list of art works whose GalleryId foreign key matches the selected gallery.

2 Add the following code and test.

```
<div class="container">
   <div class="row">

   <?php
   if ($_SERVER["REQUEST_METHOD"] == "GET") {
      if (isset($_GET['gallery']) && $_GET['gallery'] > 0) {

         $sql = 'select * from ArtWorks where GalleryId=' .
                   $_GET['gallery'];

         if ($result = mysqli_query($connection, $sql)) {

            // loop through the data
            while($row = mysqli_fetch_assoc($result))
            {
   ?>
               <div class="col-md-3">
                   <div class="thumbnail">
                      <img src="images/art/works/square-medium/
                         <?php echo $row['ImageFileName']; ?>.jpg"
                           title="<?php echo $row['Title']; ?>"
                           alt="<?php echo $row['Title']; ?>"
                          class="img-thumbnail img-responsive">
                       <div class="caption">
                          <?php echo $row['Title']; ?>
                       </div>
                    </div>
```

```
                </div>
    <?php
            } // end while

            // release the memory used by the result set
            mysqli_free_result($result);

        } // end if ($result

    } // end if (isset

} // end if ($_SERVER

// close the database connection
mysqli_close($connection);

?>

    </div>
</div>
```

The result should look similar to that shown in Figure 11.6. Notice that this type of coding, in which markup and PHP programming is interspersed, can be quite messy and tricky to follow. The next exercise will use PHP functions which will minimize the amount of code that is injected into your markup.

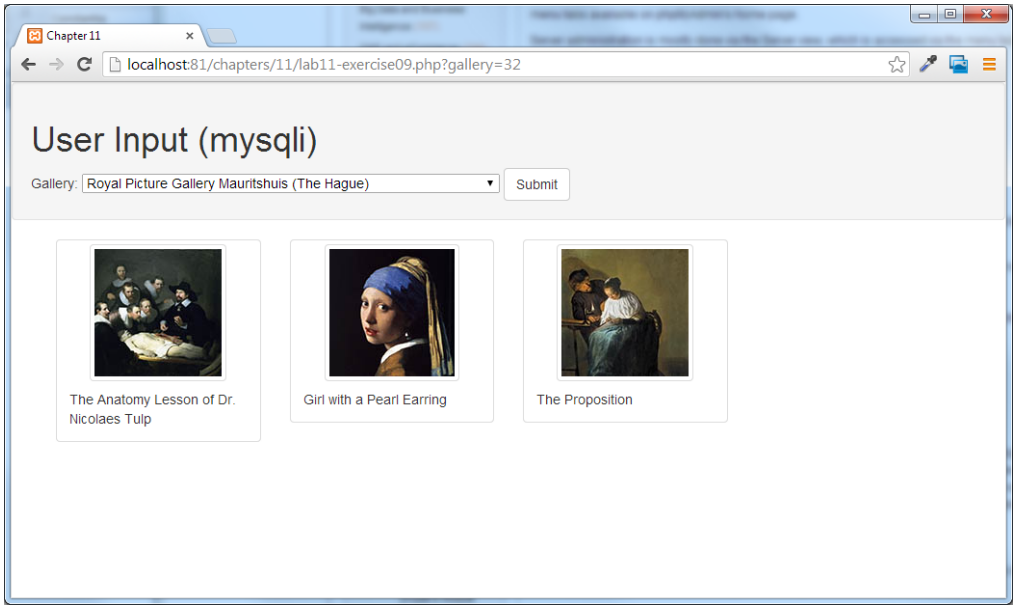

Figure 11.7 – Exercise 11.9 complete

EXERCISE 11.10 — INTEGRATING USER INPUTS (PDO)

1 Open and examine lab11-exercise10.php.

This page already contains the code for displaying a list of artist names. You will be

adding the code to display a list of art works whose ArtistId foreign key matches the selected artist.

2 Add the following code to the markup.

```
<div class="container">
   <div class="row">

      <div class="col-md-3">
         <div class="list-group">
            <?php outputArtists(); ?>
         </div>
      </div>
      <div class="col-md-9">
         <?php outputPaintings(); ?>
      </div>

   </div>
</div>
```

Notice that unlike the previous exercise, which had a lot of PHP code interspersed within the markup, this one simplifies the markup by moving most of the PHP coding into PHP functions.

3 Modify the following functions to the top of the document (i.e., after the implementation of `outputArtists()`.

```
/*
 Displays the list of paintings for the artist id specified in the id
 query string
*/
function outputPaintings() {
   try {
      if (isset($_GET['id']) && $_GET['id'] > 0) {
         $pdo = new PDO(DBCONNSTRING,DBUSER,DBPASS);
         $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

         $sql = 'select * from ArtWorks where ArtistId=' . $_GET['id'];
         $result = $pdo->query($sql);
         while ($row = $result->fetch()) {
            outputSinglePainting($row);
         }
         $pdo = null;
      }
   }
   catch (PDOException $e) {
      die( $e->getMessage() );
   }
}

/*
 Displays a single painting
*/
function outputSinglePainting($row) {
   echo '<div class="media">';
   echo '<a class="pull-left" href="#">';
   echo '<img class="media-object" src="images/art/works/square-medium/'
```

```
            . $row['ImageFileName'] .'.jpg">';
    echo '</a>';
    echo '<div class="media-body">';
    echo '<h4 class="media-heading">';
    echo htmlentities($row['Title'], ENT_IGNORE | ENT_HTML5, "ISO-8859-1");
    echo '</h4>';
    echo $row['Description'];
    echo '</div>';
    echo '</div>';
}
```

4 Test in browser. The result should be similar to that shown in Figure 11.7.

Note, not all paintings contain a description.

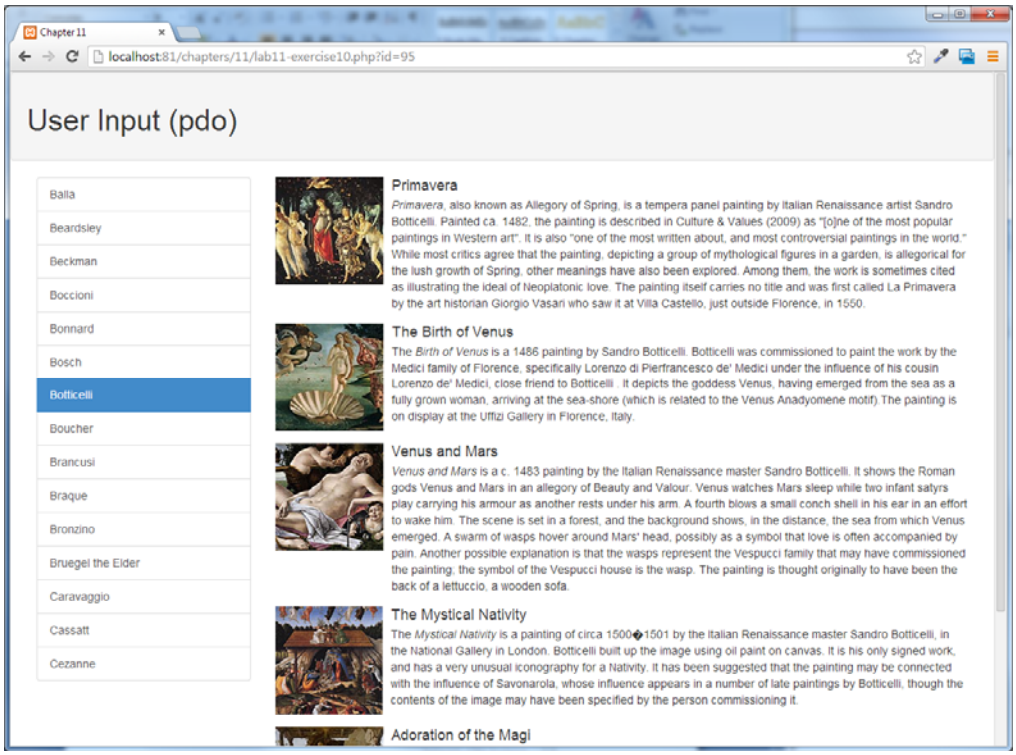

Figure 11.8 – Exercise 11.10 complete

EXERCISE 11.11 — SANITIZE INPUTS

1 Open and examine lab11-exercise11.php.

This file is the same as the finished version of exercise 9.

2 Edit the following code and test.

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "GET") {
   if (isset($_GET['gallery']) && $_GET['gallery'] > 0) {
```

```
$gallery = $mysqli->real_escape_string($_GET['gallery']);
$sql = 'select * from ArtWorks where GalleryId=' . $gallery;

if ($result = mysqli_query($connection, $sql)) {
```

*This provides **some** protection against injection attacks and other security risks. The next exercise illustrates a better approach to protecting your sites against SQL injection attacks.*

EXERCISE 11.12 — PREPARED STATEMENTS

1 Open and examine lab11-exercise12.php.

This file is the same as the finished version of exercise 10.

2 Edit the following code and test.

```
function outputPaintings() {
   try {
      if (isset($_GET['id']) && $_GET['id'] > 0) {
         $pdo = new PDO(DBCONNSTRING,DBUSER,DBPASS);
         $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

         $sql = 'select * from ArtWorks where ArtistId=:id';
         $id =  $_GET['id'];
         $statement = $pdo->prepare($sql);
         $statement->bindValue(':id', $id);
         $statement->execute();

         while ($row = $statement->fetch()) {
            outputSinglePainting($row);
         }
         $pdo = null;
      }
   }
   catch (PDOException $e) {
      die( $e->getMessage() );
   }
}
```

SAMPLE DATABASE TECHNIQUES

EXERCISE 11.13 — HTML LIST FROM A DATABASE QUERY RESULT

1 Open and examine lab11-exercise13.php.

In this example, you will be creating a list of links to a genre display page. Each link will contain a unique query string value.

2 Modify the following function.

```
/*
 Displays a list of genres
*/
function outputGenres() {
   try {
      $pdo = new PDO(DBCONNSTRING,DBUSER,DBPASS);
      $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

      $sql = 'select GenreId, GenreName, Description from Genres
              Order By Era';
      $result = $pdo->query($sql);
      while ($row = $result->fetch()) {
         outputSingleGenre($row);
      }
      $pdo = null;
   }
   catch (PDOException $e) {
      die( $e->getMessage() );
   }
}
```

3 Define the following two functions.

```
/*
 Displays a single genre
*/
function outputSingleGenre($row) {
   echo '<div class="col-md-3">';
   echo '<div class="thumbnail">';
   $img = '<img src="images/art/genres/square-medium/'
                  . $row['GenreId'] .'.jpg">';
   echo constructGenreLink($row['GenreId'], $img);
   echo '<div class="caption">';
   echo '<h4>';
   echo constructGenreLink($row['GenreId'], $row['GenreName']);
   echo '</h4>';
   echo '</div>'; // end caption div
   echo '</div>'; // end thumbnail div
   echo '</div>'; // end col-md-3 div
}

/*
  Constructs a link given the genre id and a label (which could
  be a name or even an image tag
*/
function constructGenreLink($id, $label) {
   $link = '<a href="genre.php?id=' . $id . '">';
   $link .= $label;
   $link .= '</a>';
   return $link;
}
```

Notice that in this example, the outputSingleGenre() function delegates the actual construction of the link markup to another function (constructGenreLink). This simplifies our code and makes it more self-documenting.

4 Test in browser. The result should look similar to that shown in Figure 11.7. Test the

links. They should display the appropriate genre page.

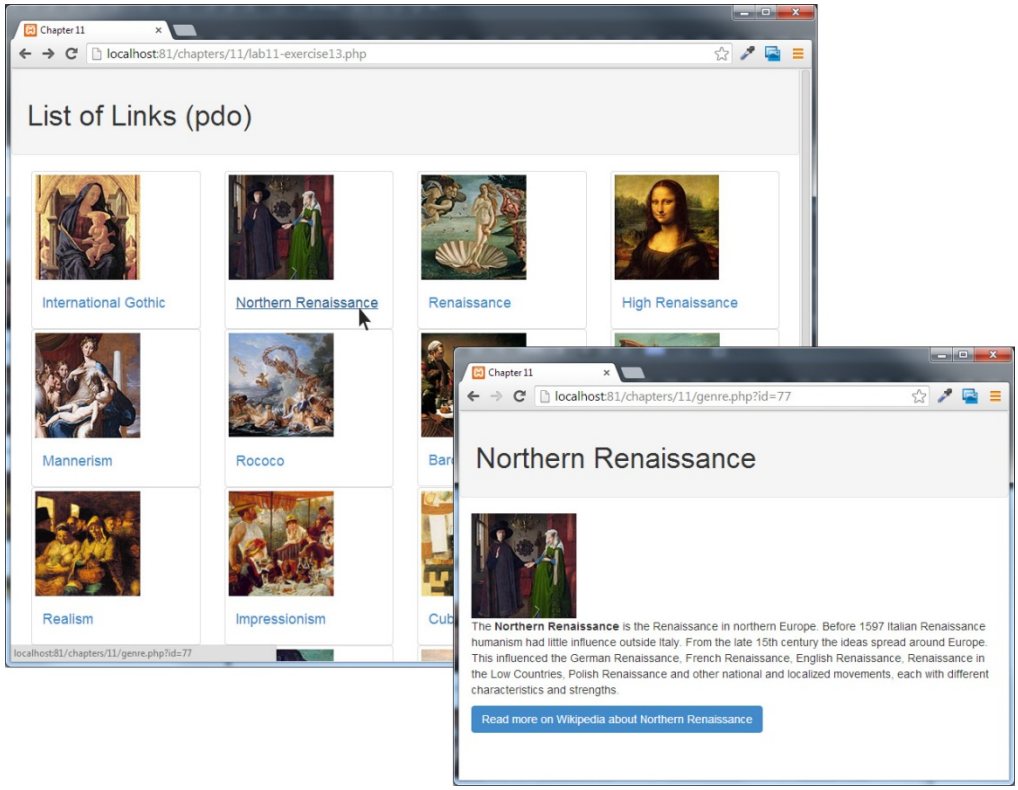

Figure 11.9 – Exercise X.Y complete

The final exercise in this lab is more complicated. It makes use of the ability within HTML to upload files to a server. In this case, the uploaded files will then be saved within a BLOB field in a database table and rendered to the user through a PHP script.

Note: This example assumes that you have already created the travel database and imported travel.sql.

EXERCISE 11.14 — READING AND STORING BLOB DATA

1 Open, examine, and view in the browser `lab11-exercise14.php`.

You will now modify this file to display the form upload, and process the upload into a BLOB. Later you will also have to display the photo from the database using another script.

Begin by adding the conditional test to detect a POST as follows:

```
if ($_SERVER["REQUEST_METHOD"] == "POST") {
   //If there was a post, we should process it.
   processFileUpload();
```

```
}
```

2 Now we will write the function `processFileUpload()`, which will draw from the solution you wrote for Exercise 9.8, where we saved the uploaded files to a location on the server. This time we will use a SQL query to upload the temporary file as a BLOB into the database.

```
function processFileUpload(){
  $pdo = new PDO(DBCONNSTRING,DBUSER,DBPASS);
  $validExt = array("jpg", "png");
  $validMime = array("image/jpeg","image/png");
  //  echo "<pre>";print_r($_FILES);echo "</pre>";
  if ($_FILES["inputFile"]["error"] == UPLOAD_ERR_OK) {
       $name = $_FILES["inputFile"]["name"];
       $extension = end(explode(".", $name));
       if (in_array($_FILES["inputFile"]["type"],$validMime) &&
           in_array($extension, $validExt)) {
         //INSERT INTO BLOB
         $fileContent =
         file_get_contents($_FILES["inputFile"]['tmp_name']);
         $sql = "INSERT INTO travelimage (ImageContent) VALUES(:data)";
         $statement = $pdo->prepare($sql);
         $statement->bindParam(':data', $fileContent, PDO::PARAM_LOB);
         $statement->execute();
        echo "file uploaded succesfully";
       }
       else{
         echo "That file extension is not supported";
       }
  }
}
```

3 Now you will demonstrate that the file uploaded correctly by adding a link to the newly uploaded file using an <img> element right after executing the SQL statement.

```
echo "file uploaded succesfully<br>";
$insertID = $pdo->lastInsertId();

//use the ID of the image to create a link to the uploaded image.
echo "<img src='viewImage.php?id=".$insertID."' />";
```

4 Nice that our image is pointing to a file `viewImage.php`. You will now create that file and write code to display the correct image from the database. Create the file `viewImage.php` and paste in the code as follows:

```
<?php

require_once('includes/travel-config.php');
$pdo = new PDO(DBCONNSTRING,DBUSER,DBPASS);

$sql = "SELECT * FROM travelimage WHERE ImageID=:id" ;
$statement = $pdo->prepare($sql);
```

```
$statement->bindValue(':id', $_GET['id']);
$statement->execute();
if($row = $statement->fetch()) {
  // Output the MIME header
  header("Content-type: image/png");
  // Output the image
  echo ($row["ImageContent"]);
}

?>
```

Save this code, so that now when you upload a file, that image will be immediately displayed back to you as shown in Figure 11.9. All database images can now be accessed so long as we know their ID, using the viewImage.php file.

As an exercise for the reader: To manage multiple types of image files you have to make some improvements. You would have to change your input filter, add a field to the table to store the mime type uploaded and then use that field when displaying the image.

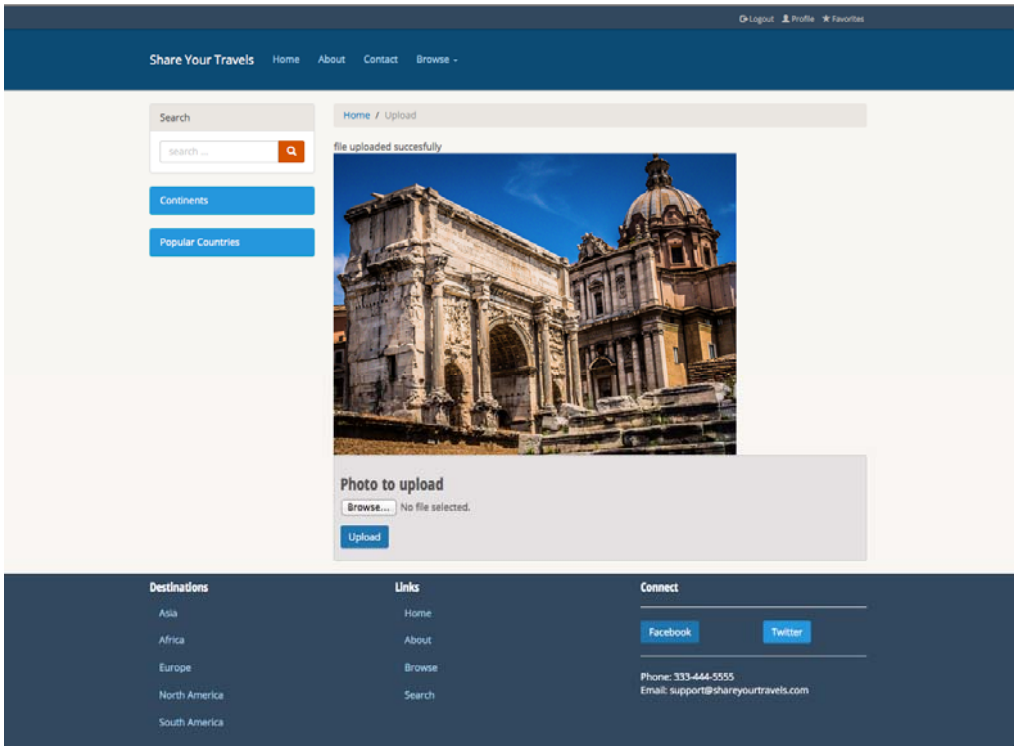

Figure 11.10 Screenshot of the completed form, which uploads and displays files.