LAB 14

WORKING WITH DATABASES

What You Will Learn

- How to install and manage a MySQL database
- Hot to use SQL queries in your PHP code
- How to integrate user inputs into SQL queries

Approximate Time

The exercises in this lab should take approximately 90 minutes to complete.

Fundamentals of Web Development, 3rd Ed

Randy Connolly and Ricardo Hoar

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

XAMPP Version, Date Last Revised: Nov 4, 2019

NTRODUCNG MySQL

Preparing Directories

1 Like the previous PHP labs, this lab requires both a functioning webserver and an installation of MySQL. XAMPP currently uses MariaDB, which is a compatible fork of the open-source MySQL project that is independent of Oracle (which acquired the rights to the MySQL name when it purchased Sun in 2010).

This lab assumes you have access to MySQL/MariaDB via XAMPP.

2 Copy the folder titled `lab14` to your PHP development location on your development machine (if using XAMPP, this might be `c:/xampp/htdocs`). This `lab14` folder could be provided by your instructor, or you could clone it from GitHub repo for this lab.

If you have finished lab 11 using XAMPPP, you will already have MySQL/MariaDB installed locally on your development machine.

Exercise 14.1 — Running MySQL in the Shell

1 To perform the next several exercises, you will be using the MariaDB shell. This shell is available on other platforms as well.

Run the XAMPP control panel and make sure both Apache and MySQL modules are started.

2 If MySQL is running, you can run SQL queries in MySQL from within the XAMPP shell. Click the Shell button in the XAMPP Control Panel. In the shell, type the following command:

```
mysql -h localhost -u root
```

You will notice that the Shell prompt has changed to MariaDB>. You can now type in different MySQL commands.

3 Type in the following command:

```
show databases;
```

This will display the already installed databases.

4 Type in the following command:

```
use mysql;
```

This tells MySQL to use the database named mysql for subsequent query commands. This particular database was created by the install script and is used by MySQL itself. You will likely never manipulate this database. We are doing so here purely for illustration purposes.

5 Type in the following command:

```
show tables;
```

This will display the tables within the current database.

6 Type in the following commands:

```
create database travel;
use travel;
```

This will create a new database named travel and make it the current database.

7 Type in the following command:

```
source c:/xampp/htdocs/lab14/travel-3rd.sql;
```

This will run the SQL statements in the specified file (if you examine this file you will notice it begins with the same two commands you just entered in step 6). If your XAMPP is installed in another location, you will have to change the path above.

8 Type in the following command:

```
show tables;
```

This should display several table names.

9 Type in the following command:

```
select * from users;
```

This should display the content of the specified table. Because MySql maps table names to file system files and because Linux-based operating systems are case sensitive, MySQL table names on Linux environments will also be case sensitive.

10 Type in the following command:

```
exit
```

This will exit the MySQL shell and return to XAMPP shell.

11 Type in the following command:

```
exit
```

This will exit the XAMPP shell.

Although you will be able to manipulate the database from your PHP code, there are some maintenance operations (such as creating tables, importing data, etc) that typically do not warrant writing custom PHP code. For these types of tasks, you will use either the MySQL command line (as shown in previous exercise) or by using some type of MySQL management tool, such as the popular web-based front-end `phpMyAdmin`.

EXERCISE 14.2 — PHPMYADMIN

1 Start `phpAdmin`. In XAMMP Control Panel, you do this by clicking on the Admin button.

Eventually you should see the phpMyAdmin panel as shown in Figure 14.1

Because MySQL has a blank password by default and phpMyAdmin uses the same credentials as MySQL, phpMyAdmin has a blank password by default as well, which makes setup significantly easier for users. Since developers generally don't (and often shouldn't) put sensitive/important data in a development environment like XAMPP, having no password on phpMyAdmin is rarely an issue.

3 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 14.1). If not, jump to Exercises 14.3 and then return to step 4.

4 Click on the `travel` database.

This will display the tables in the travel database.

5 Click the browse link for the `Countries` table.

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

6 Click on the Structure tab.

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

7 When done exploring, close the preview tab with PHPMyAdmin.

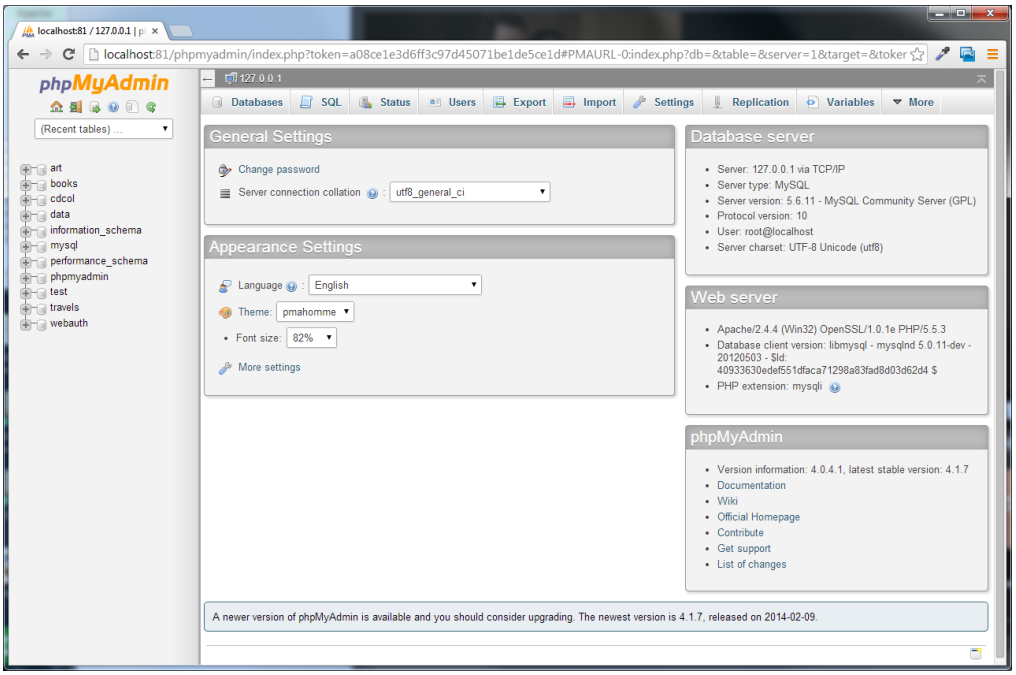

Figure 14.1 – phpMyAdmin

Test Your Knowledge #1

1 Using either the MySQL shell or PHPMyAdmin (via the Import tab … but be warned larger import files can timeout in PHPMyAdmin), perform the following actions.

Create a new database named `art` and populate its data using the `art-3rd.sql` file.

Create a new database named `books` and populate its data using the `book-small.sql` file.

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 (i.e., you have completed Test Your Knowledge #1).

Exercise 14.3 — Querying a Database

1 Using the MySQL command shell, enter the following commands.

```
use  art;
select * from Artists;
```

2 Using the MySQL command shell, enter the following command.

```
select paintingid, title, yearofwork from Paintings
where yearofwork < 1600;
```

In MySQL, database names correspond to operating system directories while tables correspond to one or more operating system files. Because of this correspondence, table and database names ARE case sensitive on non-Windows operating systems (though field names are not case sensititve).

3 Modify the query as follows and test.

```
select paintingid, title, yearofwork from Paintings
where yearofwork < 1600 order by yearofwork;
```

4 Modify the query as follows and test.

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

Why are some of the SQL words uppercase now? No reason … just wanted to illustrate that upper or lower case can be used with the SQL keywords.

5 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.

EXERCISE 14.4 — MODIFYING RECORDS

1 Using the MySQL command shell, enter the following commands.

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

You should see message about one row being affected (i.e., one record has been inserted).

2 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.

3 Run the following new query:

```
update Artists
set details='Palmo Vecchio was a painter of the Venetian school'
where lastname = 'Vecchio';
```

4 Verify the record was updated (i.e, by running the query from step 2).

5 Run the following new query:

```
delete from Artists
where lastname = 'Vecchio';
```

6 Verify the delete worked by running the query from step 2).

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 14.5 — BUILD AN INDEX

1 Using the MySQL command shell, enter the following command.

```
show indexes in Paintings;
```

You will notice that there are several indexes already defined. Where did they come from?

2 Examine the file `art-3rd.sql`. You will notice that indexes were pre-defined in the `Create Table` statements.

3 Using the MySQL command shell, enter the following command.

```
create index idxYears on Paintings (YearOfWork);
```

If you are using PHPMyAdmin, you will not need to create any separate MySQL users. If you are using a different platform (or you just want to try it), you will need to do the next exercise (**if using Cloud9 you can skip this exercise**).

EXERCISE 14.6 — 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. For development-only environments, using the root user will likely be okay. Nonetheless, we are going to create a new user which you will use for subsequent examples in this lab.

2 Click the **Add user** account link.

This will display the Add user page (see Figure 14.5).

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

```
User name (use text filed): testuser
Host (Local):
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. Also, depending on the environment you are using, you may need to enter something different in the Host field (perhaps 'localhost' or '127.0.0.1')

4 In the **Database for user account** section, ensure the **Grant all privileges on database "art"** checkbox is checked.

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

6 Click the Go button.

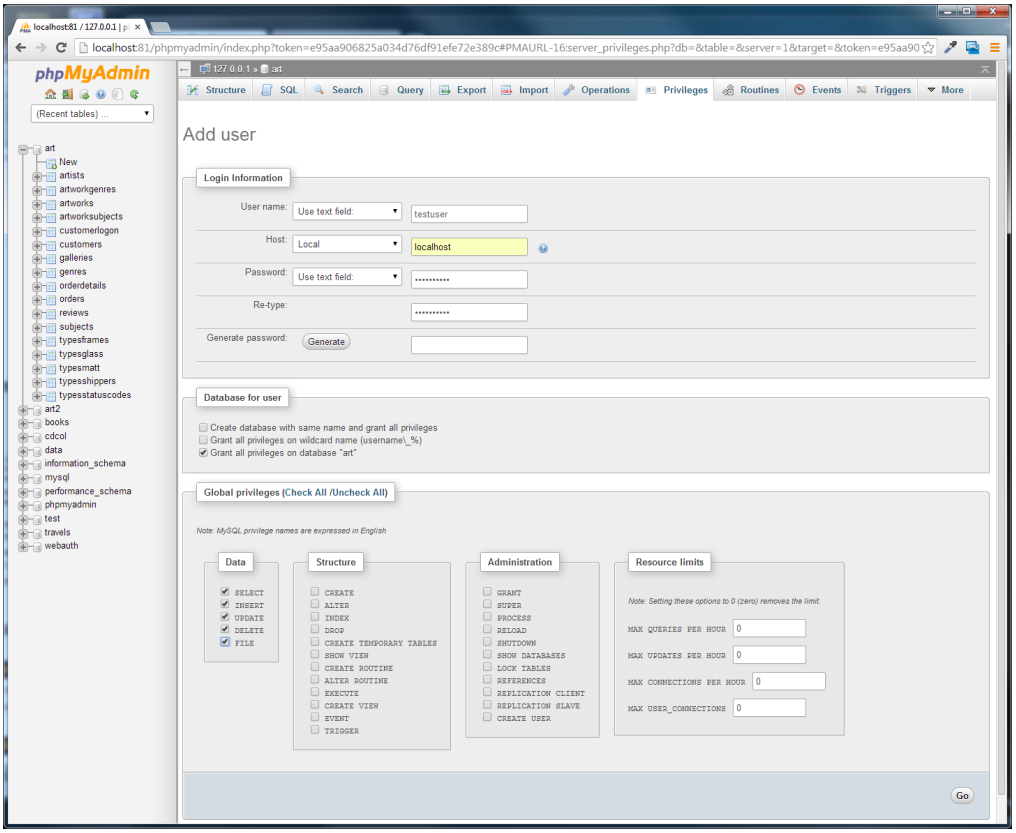

Figure 14.2 – Creating a user

Accessng MySQL n PHP

As covered in the book, there are different database APIs depending on the database you are using.

EXERCISE 14.7 — MySQL Through PHP

1 Open `config.inc.php` and modify the file as shown below.

```
<?php
define('DBHOST', 'localhost');
define('DBNAME', 'art');
define('DBUSER', 'root');
define('DBPASS', '');
define('DBCONNSTRING',"mysql:host=" . DBHOST . ";dbname=" . DBNAME .
          ";charset=utf8mb4;");
?>
```

This is the setup for XAMPP. If you are using a different environment, you will likely need to change the DBUSER, DBPASS, and DBHOST values.

The `charset` specified here in our connection string is optional. It helps ensure that the database API handles our UTF8 encoded data (i.e., foreign character sets).

2 Open `lab14-ex07-pdo.php` and modify as follows:

```
<?php require_once('config.inc.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.

4 Open `lab14-ex07-mysqli.php` and modify as follows:

```
<?php require_once('config.inc.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. Older, legacy PHP code tends to use the mysqli API.

EXERCISE 14.8 — INTEGRATING USER INPUTS

1 Open and examine `lab14-ex08.php`.

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

2 Add the following code and test.

```
<div class="ui segment">
   <div class="ui six cards">
      <?php
      // only display painting cards if one has been selected
      if ($_SERVER["REQUEST_METHOD"] == "GET") {
         if (isset($_GET['gallery']) && $_GET['gallery'] > 0) {
            $sql = 'select * from Paintings where GalleryId=' .
                   $_GET['gallery'];
            $result = $pdo->query($sql);
            while ($row = $result->fetch()) {
      ?>
               <div class="card">
                  <div class="image">
                     <img src="images/art/works/square-medium/<?php
echo $row['ImageFileName']; ?>.jpg"
                        title="<?php echo $row['Title']; ?>"
                        alt="<?php echo $row['Title']; ?>" >
                  </div>
                  <div class="extra"><?php echo $row['Title'];
?></div>
               </div>  <!-- end class=card-->
         <?php
               } // end while
            } // end if (isset
         } // end if ($_SERVER
         ?>
   </div>  <!-- end class=four cards-->
</div>  <!-- end class=segment-->
```

The result should look similar to that shown in Figure 14.3. 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.

Also, do note that this example, which constructs an SQL statement by concatenating user input with SQL, is susceptible to SQL Injection Attacks. We look at how to protect our pages from this type of security vulnerability later in the lab. The approach used here however has the advantage of being simpler to understand, which is valuable when you are first learning how to use PDO.

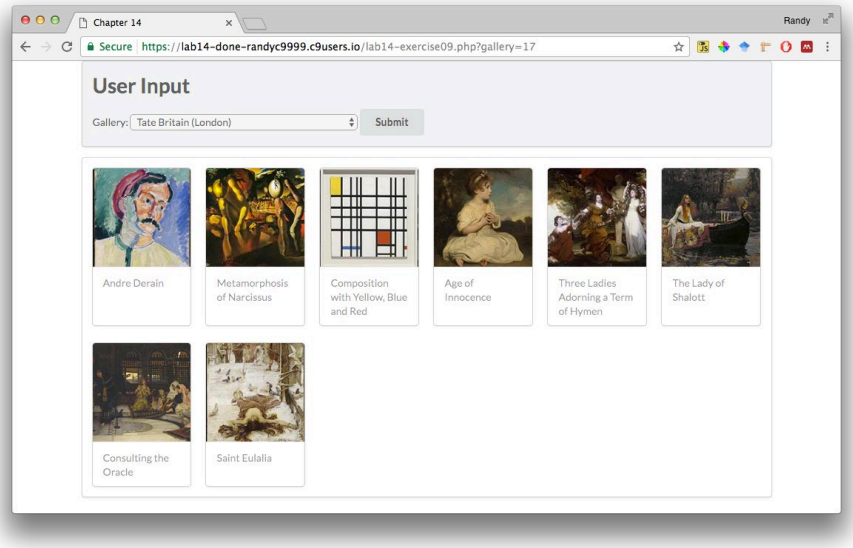

Figure 14.3 – EXERCISE 14.9 complete

EXERCISE 14.9 — INTEGRATING USER INPUTS WITH FUNCTIONS

1 Open and examine `lab14-ex09.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 paintings whose `ArtistId` foreign key matches the selected artist.

2 Add the following code to the markup.

```
<main class="ui container">
   <div class="ui secondary segment">
      <h1>User Input</h1>
   </div>
   <div class="ui segment">
      <div class="ui grid">
        <div class="four wide column">
           <div class="ui link list">
              <?php outputArtists(); ?>
           </div>
        </div>
        <div class="twelve wide column">
           <div class="ui items">
              <?php outputPaintings(); ?>
           </div>
        </div>
      </div>
   </div>
</main>
```

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 at 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 Paintings 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="item">';
   echo '<div class="image">';
   echo '<img src="images/art/works/square-medium/' .
            $row['ImageFileName'] .'.jpg">';
   echo '</div>';
   echo '<div class="content">';
   echo '<h4 class="header">';
   echo $row['Title'];
   echo '</h4>';
   echo '<p class="description">';
   echo $row['Excerpt'];
   echo '</p>';
   echo '</div>';  // end class=content
   echo '</div>';  // end class=item
}
```

4 Test in browser. The result should be similar to that shown in Figure 14.4.

Note, not all paintings contain an excerpt.

5 Modify the while loop in `outputPaintings()` as follows and test.

```
$result = $pdo->query($sql);
foreach($result as $row) {
   outputSinglePainting($row);
}
$pdo = null;
```

The foreach loop in PHP can also be used to iterate a PDO result set. This approach is especially nice since it removes the PDO call to the fetch() method within your loop.

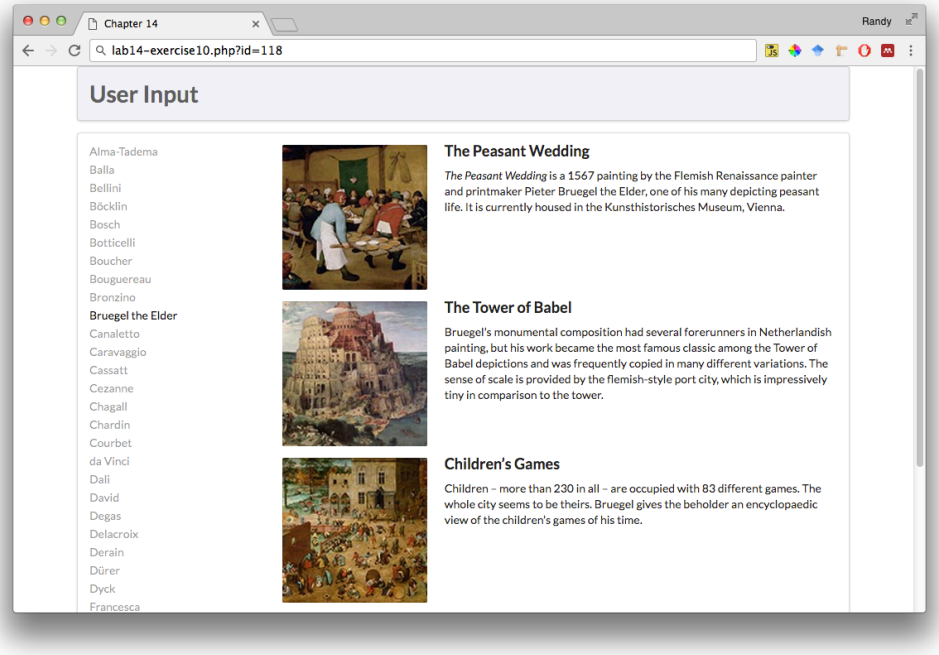

Figure 14.4 – EXERCISE 14.9 complete

EXERCISE 14.10 — PREPARED STATEMENTS

1 Open and examine `lab14-ex10.php`.

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

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 Paintings 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 TECHNQUES

EXERCISE 14.11 — HTML LIST AND RESULTS

1 Open and examine `lab14-db-functions.inc.php`.

This file contains two functions that encapsulate the functionality needed to create a connection and to query a database.

2 Open and examine `lab14-ex11.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.

3 Open and examine `lab14-ex11-helpers.inc.php`.

This file will contain various helper functions used in this exercise.

4 Modify the following functions in `lab14-ex11-helpers.inc.php`.

```
/*
 Return a result set containing all the genres
*/
function getAllGenres() {
   try {
      $connection = setConnectionInfo(DBCONNSTRING,DBUSER,DBPASS);
      $sql = 'select GenreId, GenreName, Description from Genres
         Order By EraID';

      $result = runQuery($connection, $sql, null);
      return $result;
   }
   catch (PDOException $e) {
      die( $e->getMessage() );
   }
}
/*
  Return a row containing a single genre
*/
function getSingleGenre($id) {
    try {
        $connection=setConnectionInfo(DBCONNSTRING,DBUSER,DBPASS);
        $sql = 'select GenreId, GenreName, Description, Link from
                  Genres where GenreId=?';
        $statement = runQuery($connection, $sql, array($id));
        $row = $statement->fetch(PDO::FETCH_ASSOC);
        $connection = null;
        return $row;
    }
    catch (PDOException $e) {
       die( $e->getMessage() );
    }
}
```

Notice that this code has removed almost all of the PDO-related code; the PDO-specific code now is mainly isolated to the lab14-db-functions.inc.php file.

5 Modify the following two functions in the same file.

```
function outputGenres() {
   $genres = getAllGenres();
   foreach ($genres as $g) {
      outputSingleGenre($g);
   }
}
function outputSingleGenre($genre) {
   echo '<div class="ui fluid card">';
      echo '<div class="ui fluid image">';
         $img = '<img src="images/art/genres/square-medium/' .
                  $genre['GenreId'] .'.jpg">';
         echo constructGenreLink($genre['GenreId'], $img);
      echo '</div>';
      echo '<div class="extra">';
         echo '<h4>';
         echo constructGenreLink($genre['GenreId'],
                     $genre['GenreName']);
         echo '</h4>';
      echo '</div>';
   echo '</div>';
}
```

With the PDO-related code removed, notice that the function outputGenres() is more self-documenting and simplified: it can work with genres instead of result sets or rows.

6 Add the following code to the top of `genre.php`

```
<?php

require_once('config.inc.php');
require_once('lab14-db-functions.inc.php');
require_once('lab14-ex11-helpers.inc.php');

if (isset($_GET["id"]))
  $id = $_GET["id"];
else
  $id = 78;   // set a default id if its missing

$genre = getSingleGenre($id);

?>
```

7 Add the following to the markup:

```
<main class="ui container">
    <div class="ui secondary segment">
         <h1><?php echo $genre['GenreName']; ?></h1>
    </div>
    <div class="ui segment">
        <div class="ui grid">
           <div class="three wide column">
                <img src="images/art/genres/square-medium/<?php echo
$genre['GenreId']; ?>.jpg" >
           </div>
           <div class="thirteen wide column">
                <p><?php echo $genre['Description']; ?></p>
                <p>
                <a class="ui labeled icon primary button"
                     href="<?php echo $genre['Link']; ?>">
                  <i class="external icon"></i>
                  Read more on Wikipedia about
                    <?php echo $genre['GenreName']; ?>
                </a>
                </p>
           </div>
       </div>
    </div>
</main>
```

Again, notice how our code is more self-documenting because we have extracted out the PDO-related code.

8 Test in browser. The result should look similar to that shown in Figure 14.5. Test the links. They should display the appropriate genre page.

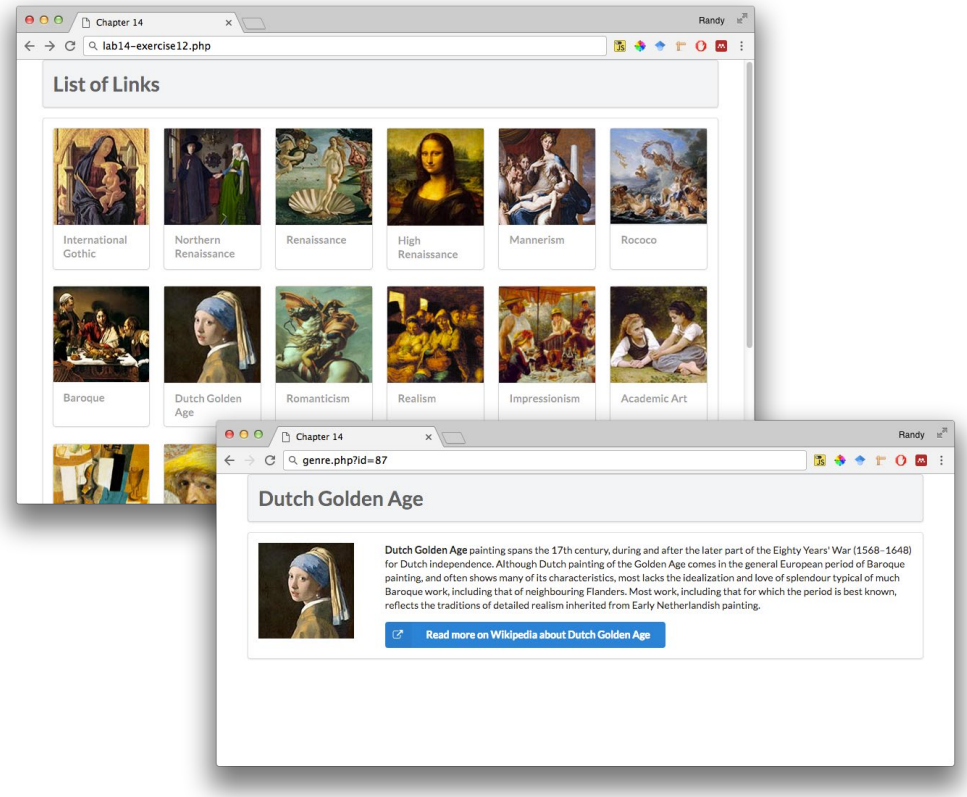

Figure 14.5 – Exercise 14.11 complete

TEST YOUR KNOWLEDGE #2

1 You have been provided with the markup for the next exercise in the file `lab14-test02.php`. You have also been provided with some helper functions in `lab14-test02-helpers.inc.php`, including two that return the necessary SQL for this exercise.

   Fill the select list with a list of sorted museums from the Galleries table. The helper function `getGallerySQL` returns the appropriate SQL for this table.

2 Display the top 20 paintings. The file `lab14-test02.php` has the sample markup for a single painting. The helper function `getPaintingSQL` returns the appropriate SQL for this task. The `addSortAndLimit` function can be used to append the sort and limit to your SQL.

3 When the user selects from the museum list (remember we are not using JavaScript so the user will have to click the filter button which rerequests the page), display just the paintings from the selected museum/gallery. This will require adding the WHERE clause `GalleryID=?`

   There is one gotcha … in SQL, the ORDER BY and LIMIT must **appear** after the WHERE.

4 The result should look similar to that shown in Figure 14.6.

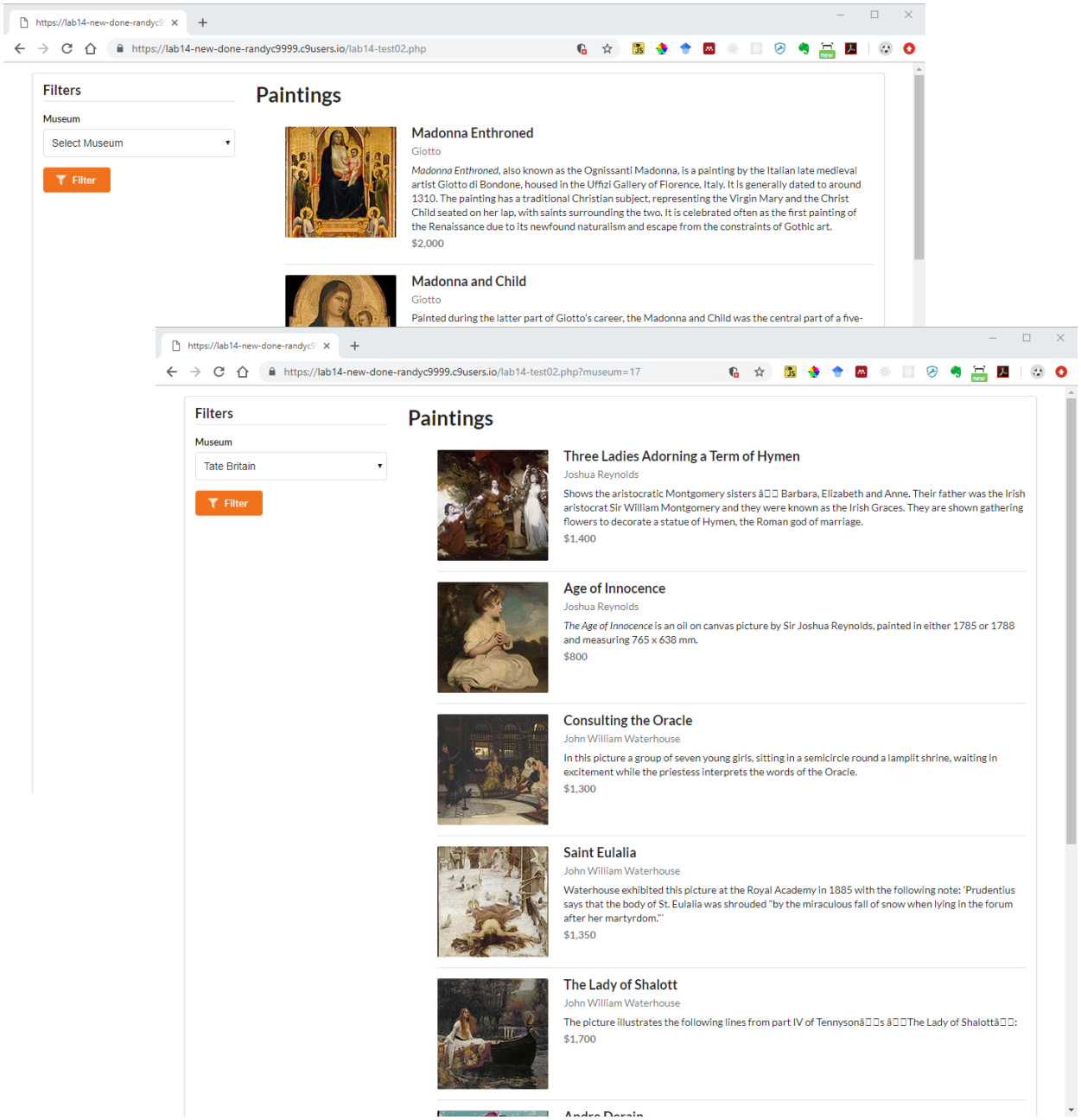

Figure 14.6 – Test Your Knowledge #2