LAB 14

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.

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.
- 2 From the main labs folder, copy the folder titled Lab014-Material to your course folder created in step one.

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 some type of MySQL management tool. You can use the popular web-based front-end phpMyAdmin.

EXERCISE 14.1 — PHPMYADMIN

- You may need to install XAMPP or PHPMyAdmin for your environment.
- 2 Start phpMyAdmin from XAMPP control panel by clicking Admin under MySQL actions.

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, 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 Exercise 14.2 and then return to step 4.

4 Click on the art database.

This will display the tables in the art database.

5 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.

6 Click on the Structure tab.

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

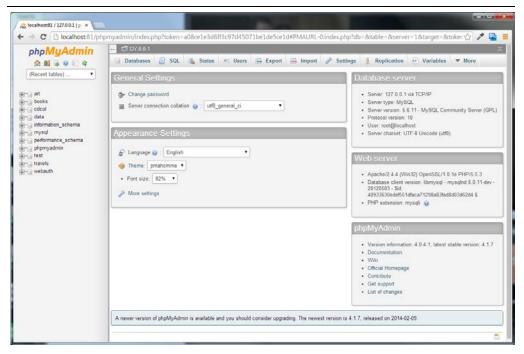


Figure 14.1 – phpMyAdmin

EXERCISE 14.2 — INSTALLING DATABASE IN PHPMYADMIN

- 1 Examine art-small.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-small.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.

Probably one of the most common problems encountered by students is that a timeout error occurs or that a file exceeds the maximum upload size. Fixing this may require modifying your php.ini configuration file or using the MySQL Monitor instead to import the data. Ask your instructor (or look online) for guidance.

- 7 Create a database named **travel** and then import **travels-small.sql**. Verify that it creates and populates a variety of tables.
- **8** Create a database named **book** and then import books-small.sql. Verify that it creates and populates a variety of tables.

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 14.3 — 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
```

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.

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 PaintingId, Title, YearOfWork FROM Paintings WHERE YearOfWork < 1600
```

This will display just the paintings prior to 1600. Notice that in MySQL, a query can be spread across multiple lines. SQL in general is not case sensitive, which means you do not have to worry *about the case of field names*. However, remember the comment in the above step 2: in MySQL, table names are case sensitive on non-Windows environments.

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

SELECT PaintingId, Title, YearOfWork FROM Paintings WHERE YearOfWork < 1600 ORDER BY YearOfWork

6 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 Artistld with the table name since both joined tables contain a field called Artistld.

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.

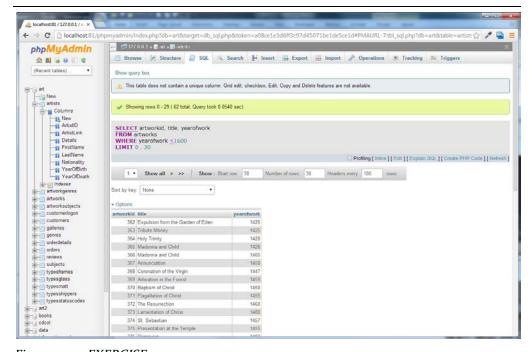


Figure 14.2 – EXERCISE 14.3

EXERCISE 14.4 — 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'
WHERE lastname = 'Vecchio'
```

- 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 14.5 — 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 link.

This will display the Add user page (see Figure 14.3).

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 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.

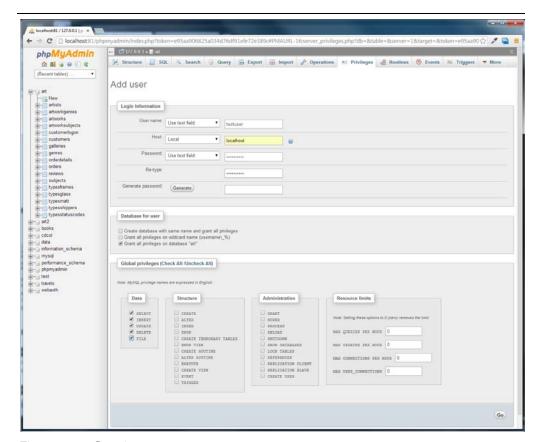


Figure 14.3 – Creating a user

ACCESSING MYSQL IN PHP

EXERCISE 14.8 - MYSQL THROUGH PHP

1 Open config.php using Sublime Text editor and modify the file as follows (if you used different user name and password in EXERCISE 14.5, then you will have to change what you enter here).

```
<?php
define('DBHOST', '');

define('DBNAME', 'art');
define('DBUSER', 'testuser');
define('DBPASS', 'mypassword');
define('DBCONNSTRING', 'mysql:dbname=art;charset=utf8mb4;);
?>
```

Depending on your environment, the connection string value for DBCONNSTRING may need to be altered. You might need to add a host value (e.g., host=Localhost;) or a port value (e.g., port=3306;). Similarly, you may also need to change the DBHOST value.

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-exerciseo8-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 = $pdo->query($sq1);
  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 lab14-exerciseo8-mysqli.php and modify as follows:

```
<?php require_once('config.php'); ?>
<!DOCTYPE html>
<html>
<body>
<h1>Database Tester (mysqli)</h1>
<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 14.9 — INTEGRATING USER INPUTS (MYSQLI)

1 Open and examine lab14-exerciseog.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.

```
if (isset($_GET['gallery']) && $_GET['gallery'] > 0) {
            $sql = 'select * from Paintings where GalleryId=' .
                   $_GET['gallery'];
            $result = $pdo->query($sq1);
            while ($row = $result->fetch()) {
      ?>
              <div class="card">
                 <div class="image">
                    <img src="images/art/works/square-medium/<?php echo</pre>
$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 14.6. 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.

This example, which constructs an SQL statement by concatenating form input, 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.

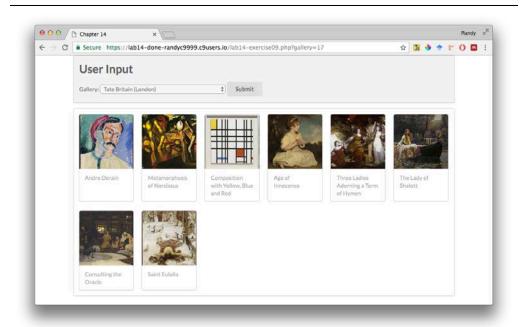


Figure 14.3 – EXERCISE 14.9 complete

EXERCISE 14.10 — INTEGRATING USER INPUTS (PDO)

1 Open and examine lab14-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 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 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 = $pdo->query($sq1);
        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 '';
   echo $row['Excerpt'];
  echo '';
  echo '</div>'; // end class=content
  echo '</div>'; // end class=item
}
```

4 Test in browser. The result should be similar to that shown in Figure 14.7. *Note, not all paintings contain an excerpt.*

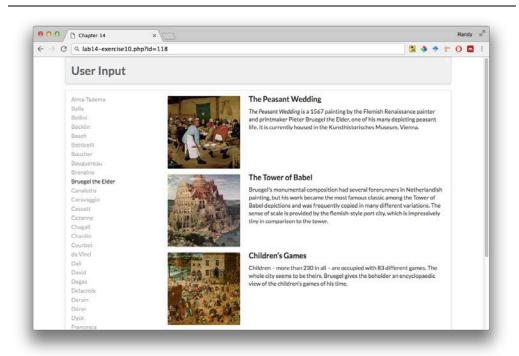


Figure 14.4 - EXERCISE 14.10 complete

EXERCISE 14.11 — PREPARED STATEMENTS

- 1 Open and examine lab14-exercise11.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 Paintings where ArtistId=:id';
         $id = $_GET['id'];
         $statement = $pdo->prepare($sq1);
         $statement->bindValue(':id', $id);
        $statement->execute();
        while ($row = $statement->fetch()) {
            outputSinglePainting($row);
         pdo = null;
     }
```

```
}
catch (PDOException $e) {
    die( $e->getMessage() );
}
```

SAMPLE DATABASE TECHNIQUES

EXERCISE 14.12 — HTML LIST AND RESULTS

- 1 Open and examine lab14-exercise12.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.

3 Define the following two functions.

```
echo '</div>'; // end class=extra
  echo '</div>'; // end class=card
}
 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 14.7. Test the links. They should display the appropriate genre page.

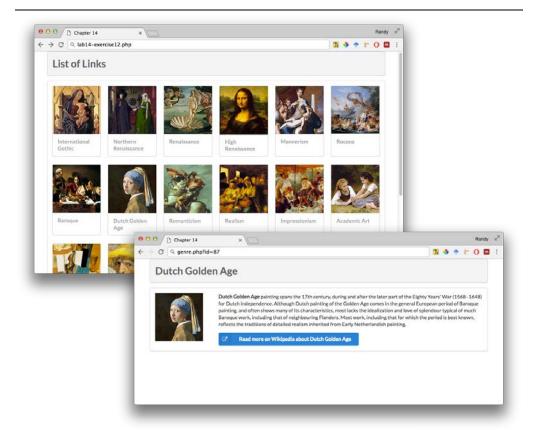


Figure 14.5 – Exercise 14.12 complete

SUBMIT YOUR COMPLETED LAB FILES

After completing all the exercises above, compress your completed files inside your lab14 folder into *lab14_yourname.zip* and submit the zip file via dropbox on elearning under lab 14 Code Submission before Sunday 11:00PM. Each lab is worth 1.5% of your total grade.