

LAB 14

WORKING WITH DATABASES

What You Will Learn

- How to install and manage a MySQL database
- How to use SQL queries in your 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, 2nd Ed

Randy Connolly and Ricardo Hoar

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

Terminal/Shell Version, Date Last Revised: Jan 25, 2018

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 have access to MySQL via a terminal or command-line interface. This could mean some type of cloud-based environment such as Cloud9 or CodeAnywhere. In this lab, we will be using Cloud9, but it should be easy to adopt it to a different environment.

- 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 Lab14 to your course folder created in step one.

If you have finished lab 11, 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, for instance, you are using a Mac and you are not using a pre-constructed LAMP stack such as MAMP, you will need to first download and install MySQL. To do so, you will need to follow the online instructions from the MySQL web site.

If you are using a Windows-based machine, then you will not have access to a Linux shell. We recommend that all web developers gain some experience with this shell. One way that a Windows developer can easily do so is to make use of an online-based development environment such as Cloud9 or CodeAnywhere.

This lab cannot provide setup instructions for all the different possible student setup configurations and environments. Instead, it will provide setup instructions for the Cloud9 environment since it provides easy access to a Linux Shell regardless of whether a student is using a Windows or Mac machine. **The instructor may thus need to provide additional or alternate setup instructions for the student if they are using a different development and configuration environment.**

Once MySQL is installed, it will likely not yet be running. The next exercise will show you how to start MySQL within the Cloud9 development environment. If you are using a different installation environment, you will need to follow the MySQL start up instructions for that environment.

EXERCISE 14.1 — STARTING MYSQL IN THE SHELL

- 1 To perform the next several exercises, you will be using Linux shell commands. On a Mac, this will require using the Terminal application. Cloud-based development environments usually provide access to Bash, the standard Linux terminal or shell. Within it, you can type in almost any standard Linux command.

The first time you run MySQL, you may likely need to run some initiation commands to set up MySQL. This lab cannot provide instructions for all the different possible student setup configurations and environments. Instead, it will make use of Cloud9, which will be similar, but not identical, to other Shell-based environments.

The Cloud9 environment installs a separate MySQL instance in each workspace, and has encapsulated the initiation steps into a simple script. In the Bash shell, type in the following command:

```
mysql-ctl start
```

This configures a default MySQL user and creates an empty starting database. This script also provides information about this default user which you should record.

- 2 Now that MySQL is installed, you can run the MySQL Monitor or interactive shell and run SQL queries from within it. For the Cloud9 environment, type in the following:

```
mysql-ctl cli
```

You will notice that the Shell prompt has changed to `mysql>`. 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 travel-small.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).

- 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 such as Cloud9 are also case sensitive.

- 10 Type in the following command:

```
exit
```

This will exit the MySQL shell and return to Linux 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 You may need to install PHPMyAdmin for you environment. For Cloud9, enter the following command in the Linux Shell:

```
phpmyadmin-ctl install
```

Notice that this install script provides you with a URL for accessing PHPMyAdmin as well as the default or root user name. Be sure to record this information.

- 2 Start phpAdmin. In Cloud9, this will require navigating to the URL provided by the PHPMyAdmin install script. If you are asked to login in, use the username provided in step 1 and a blank password

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 Cloud9, 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 **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.

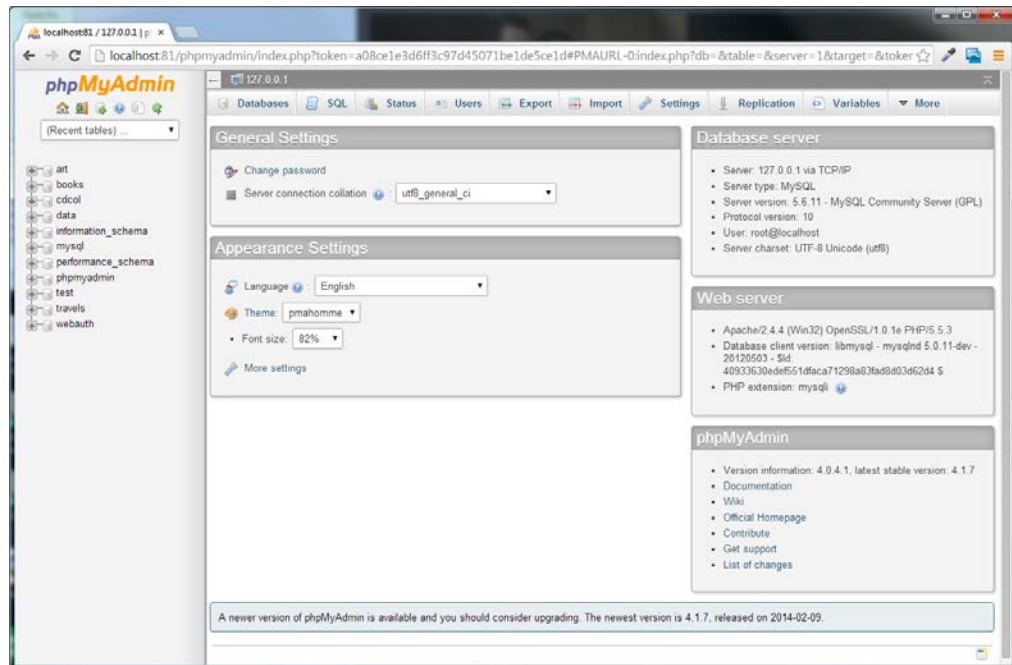


Figure 14.1 – phpMyAdmin

EXERCISE 14.3 — 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 command line instead to import the data. Ask your instructor (or look online) for guidance.

- 7 If you don't already have a database named **travel**, create it 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 14.4 — 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
```

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 that in MySQL, table names are case sensitive on non-Windows environments.

- 5 Modify the query (you can click the **Show query 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 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.

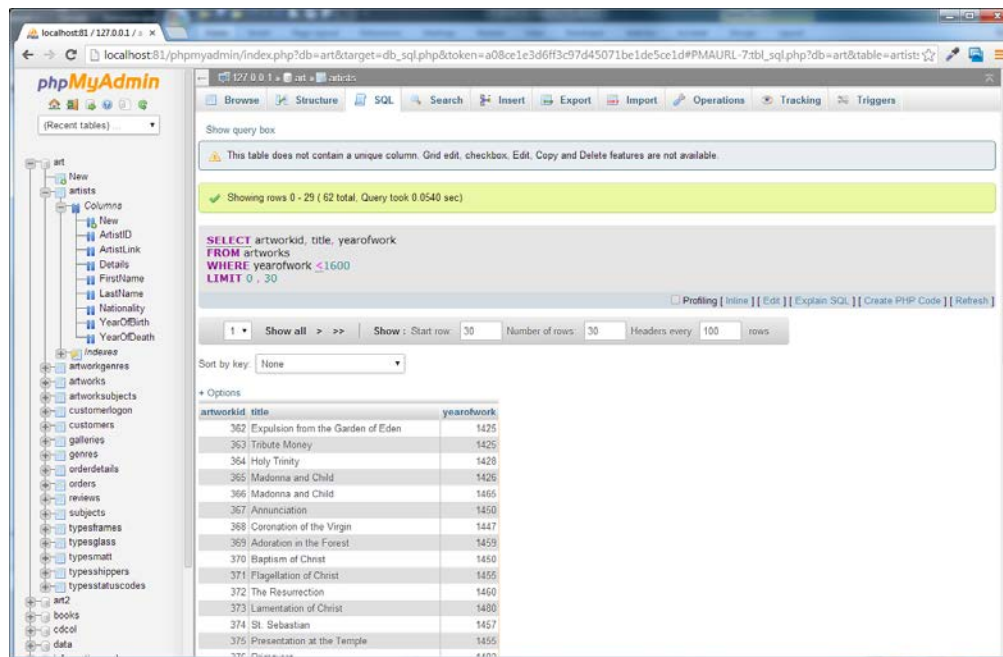


Figure 14.2 – EXERCISE 14.4

EXERCISE 14.5 — MODIFYING RECORDS

- 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:”
- In the blank SQL window, enter the following.

```
insert into Artists (firstname, lastname, nationality, yearofbirth,
yearofdeath) values ('Palma','Vecchio','Italy',1480,1528)
```
- Press the Go button.
You should see message about one row being inserted.
- 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.
- Run the following new query:

```
update Artists
set details='Palmo Vecchio was an Italian painter of the Venetian school'
```
- 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 14.6 — 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 14.3. Click Go.

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

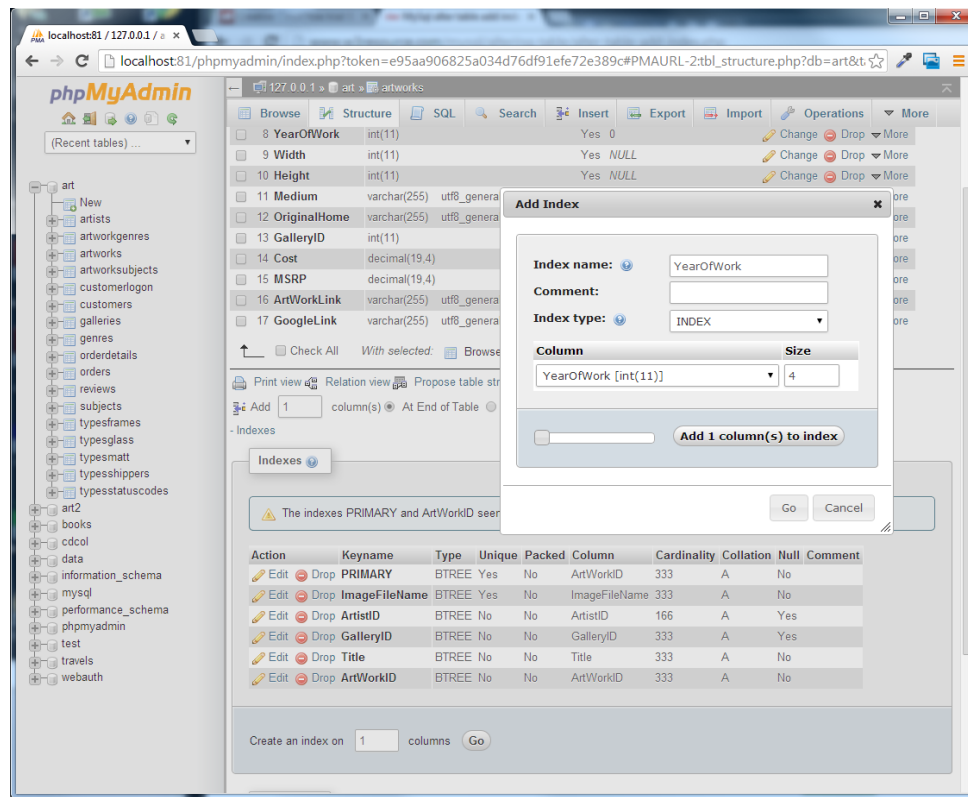


Figure 14.3 – Creating an index in phpMyAdmin

EXERCISE 14.7 — 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. 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 14.5).

- 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** 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.

The screenshot shows the phpMyAdmin interface for adding a new user. The 'Login Information' section has 'User name' set to 'testuser', 'Host' set to 'localhost', and 'Password' and 'Re-type' fields with masked input. The 'Database for user' section has the checkbox 'Grant all privileges on database "art"' checked. The 'Global privileges' section has the 'Data' tab selected, with checkboxes for 'SELECT', 'INSERT', 'UPDATE', 'DELETE', and 'FILE' all checked. The 'Structure', 'Administration', and 'Resource limits' sections are empty. A 'Go' button is at the bottom right.

Figure 14.4 – Creating a user

ACCESSING MySQL IN PHP

EXERCISE 14.8 — MySQL THROUGH PHP

- 1 Open [config.php](#) and modify the file as follows (if you used different user name and password in EXERCISE 14.7, 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-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.

- 4 Open [lab14-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 14.9 — INTEGRATING USER INPUTS

- 1 Open and examine [lab14-exercise09.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">
                  "
                    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.

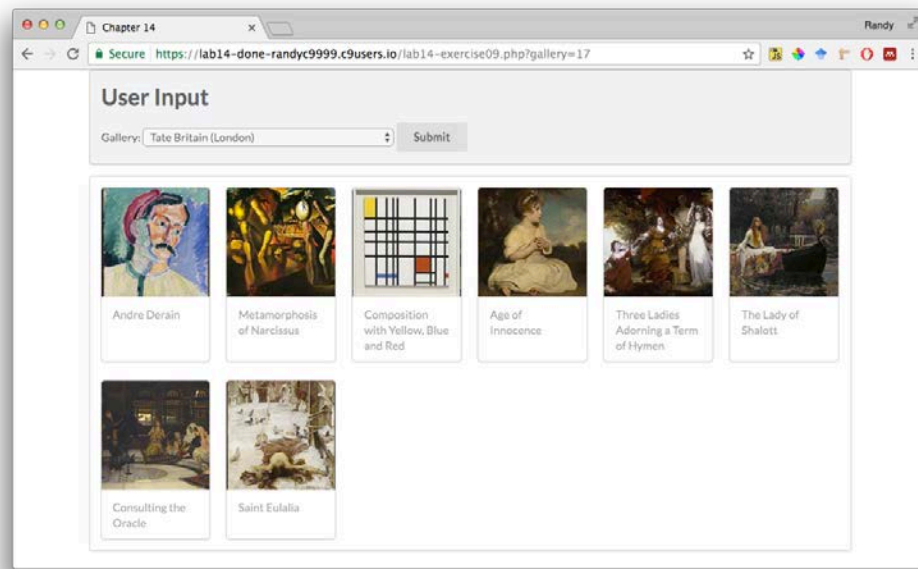


Figure 14.5 – EXERCISE 14.9 complete

EXERCISE 14.10 — USING FUNCTIONS

- 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>
      <div class="twelve wide column">
        <div class="ui items">
          <?php outputPaintings(); ?>
        </div>
      </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 '';
    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 14.7.

Note, not all paintings contain an excerpt.

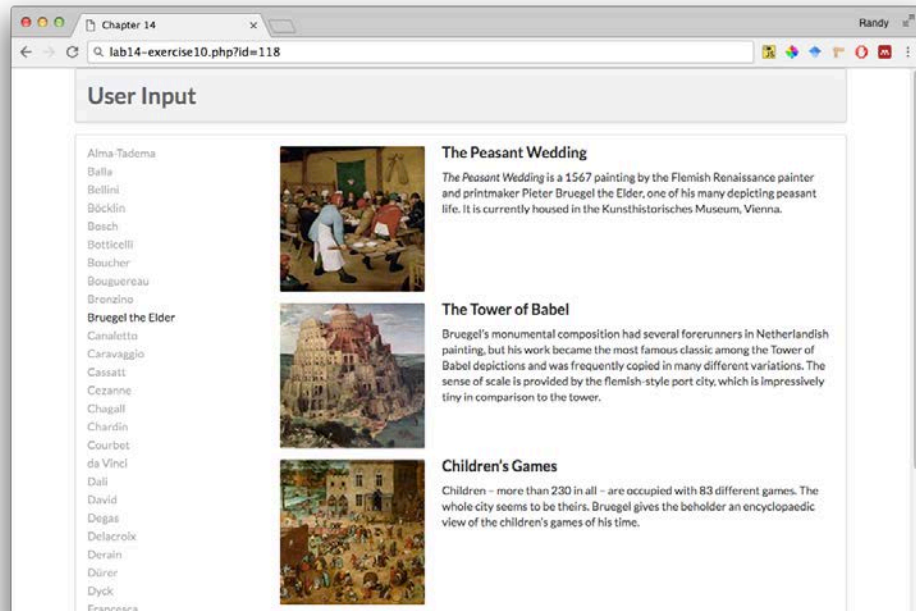


Figure 14.6 – EXERCISE 14.10 complete

EXERCISE 14.11 — PREPARED STATEMENTS

- 1 Open and examine [lab14-exercisen.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 = $pdo->prepare($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 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.

```
/*
  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 EraID';
        $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="ui fluid card">';
    echo '<div class="ui fluid image">';
    $img = '';
    echo constructGenreLink($row['GenreId'], $img);
    echo '</div>';
    echo '<div class="extra">';
    echo '<h4>';
    echo constructGenreLink($row['GenreId'], $row['GenreName']);
    echo '</h4>';
    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.

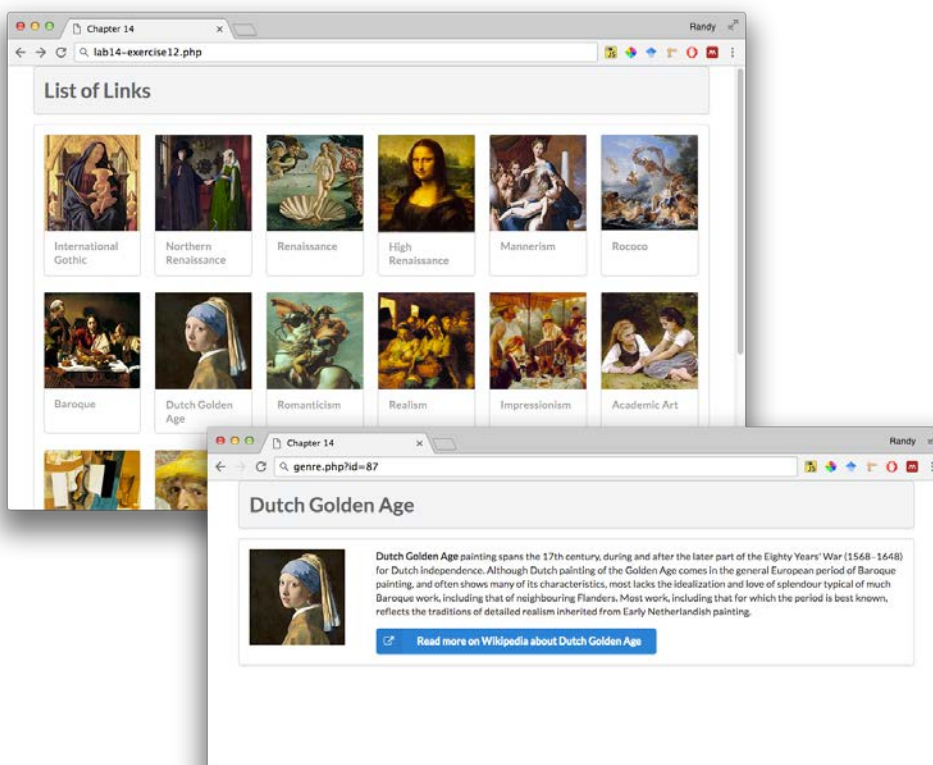


Figure 14.7 – Exercise 14.12 complete