LAB 9

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, 2nd Ed

Randy Connolly and Ricardo Hoar

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

Terminal/Shell Version, Date Last Revised: Mar 2, 2017

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 (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 `Lab09-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 9.1 — PHPMyAdmin

1 You may need to install PHPMyAdmin for you environment.

2 Start phpAdmin.

You should see the phpMyAdmin panel as shown in Figure 9.1.

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 9.1). If not, jump to Exercise 9.2 and then return to step 4.

4 Click on the **art** database.

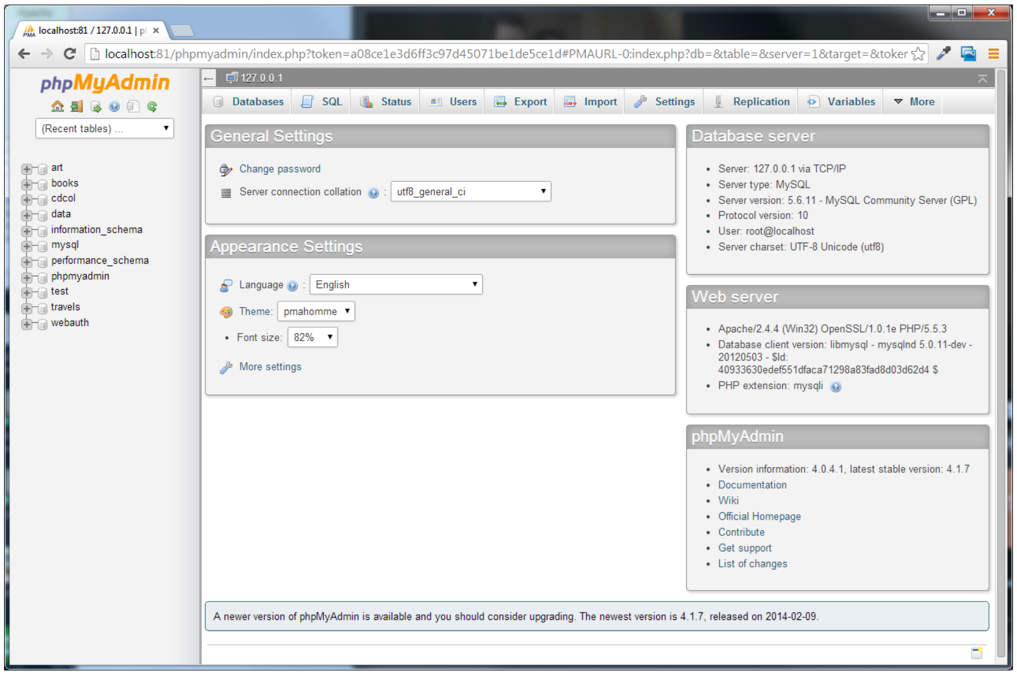
This will display the tables in the art database.

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.*



localhost:81/phpmyadmin/index.php?token=a08ce1e3d6ff3c97d45071be1de5ce1d#PMAURL-0:index.php?db=&table=&server=1&target=&toker

art
books
cdcol
data
information_schema
mysql
performance_schema
phpmyadmin
test
travels
webauth

Change password
Server connection collation : utf8_general_ci

Language : English
Theme: pmahomme
- Font size: 82%

More settings

- Server: 127.0.0.1 via TCP/IP
- Server type: MySQL
- Server version: 5.6.11 - MySQL Community Server (GPL)
- Protocol version: 10
- User: root@localhost
- Server charset: UTF-8 Unicode (utf8)

- Apache/2.4.4 (Win32) OpenSSL/1.0.1e PHP/5.5.3
- Database client version: libmysql - mysqlnd 5.0.11-dev - 20120503 - $Id: 40933630edef551dfaca71298a83fad8d03d62d4 $
- PHP extension: mysqli

- Version information: 4.0.4.1, latest stable version: 4.1.7
- Documentation
- Wiki
- Official Homepage
- Contribute
- Get support
- List of changes

A newer version of phpMyAdmin is available and you should consider upgrading. The newest version is 4.1.7, released on 2014-02-09.

Figure 9.1 – phpMyAdmin

EXERCISE 9.2 — 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

Copyright © 2017 Randy Connolly and Ricardo Hoar

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 9.3 — 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 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.

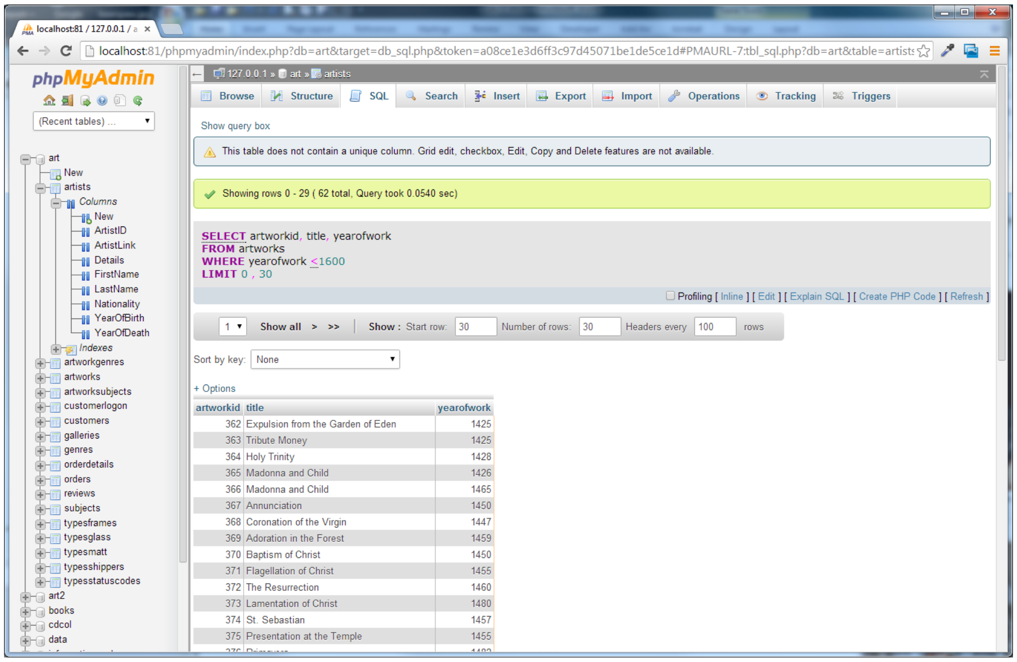
```
SELECT Artists.ArtistID, Title, YearOfWork, LastName FROM Artists
INNER JOIN Paintings ON Artists.ArtistID = Paintings.ArtistID
```

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.



This table does not contain a unique column. Grid edit, checkbox, Edit, Copy and Delete features are not available.

Showing rows 0 - 29 (62 total, Query took 0.0540 sec)

```
SELECT artworkid, title, yearofwork
FROM artworks
WHERE yearofwork <1600
LIMIT 0 , 30
```

Profiling [Inline] [Edit] [Explain SQL] [Create PHP Code] [Refresh]
1
Show all > >>
Show : Start row: 30 Number of rows: 30 Headers every 100 rows
Sort by key: None
+ Options

artworkid	title	yearofwork
362	Expulsion from the Garden of Eden	1425
363	Tribute Money	1425
364	Holy Trinity	1428
365	Madonna and Child	1426
366	Madonna and Child	1465
367	Annunciation	1450
368	Coronation of the Virgin	1447
369	Adoration in the Forest	1459
370	Baptism of Christ	1450
371	Flagellation of Christ	1455
372	The Resurrection	1460
373	Lamentation of Christ	1480
374	St. Sebastian	1457
375	Presentation at the Temple	1455

Figure 9.2 – EXERCISE 9.3

EXERCISE 9.4 — MODIFYING RECORDS

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

Copyright © 2017 Randy Connolly and Ricardo Hoar

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 9.5 — 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 9.3).

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

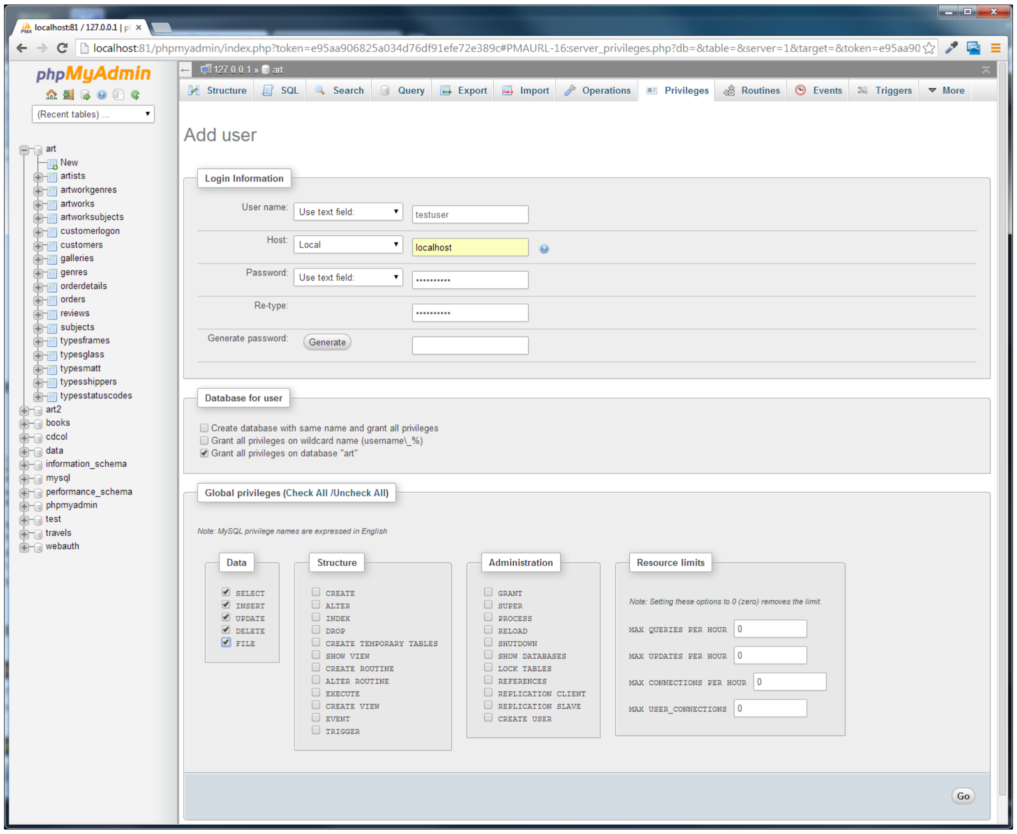
```
User name (use text filed): testuser
Host (Local):
Password (use text filed): mypassword
Re-Type: mypassword
```

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.



Add user

Login Information

User name:	Use text field:	testuser
Host:	Local	localhost
Password:	Use text field:	••••••••••
Re-type:		••••••••••
Generate password:	Generate	

Database for user

- [ ] Create database with same name and grant all privileges
- [ ] Grant all privileges on wildcard name (username_%)
- [x] Grant all privileges on database "art"

Global privileges (Check All /Uncheck All)

Note: MySQL privilege names are expressed in English

Data

- [x] SELECT
- [x] INSERT
- [x] UPDATE
- [x] DELETE
- [x] FILE

Structure

- [ ] CREATE
- [ ] ALTER
- [ ] INDEX
- [ ] DROP
- [ ] CREATE TEMPORARY TABLES
- [ ] SHOW VIEW
- [ ] CREATE ROUTINE
- [ ] ALTER ROUTINE
- [ ] EXECUTE
- [ ] CREATE VIEW
- [ ] EVENT
- [ ] TRIGGER

Administration

- [ ] GRANT
- [ ] SUPER
- [ ] PROCESS
- [ ] RELOAD
- [ ] SHUTDOWN
- [ ] SHOW DATABASES
- [ ] LOCK TABLES
- [ ] REFERENCES
- [ ] REPLICATION CLIENT
- [ ] REPLICATION SLAVE
- [ ] CREATE USER

Resource limits

Note: Setting these options to 0 (zero) removes the limit.

MAX QUERIES PER HOUR	0
MAX UPDATES PER HOUR	0
MAX CONNECTIONS PER HOUR	0
MAX USER_CONNECTIONS	0

Go

Figure 9.3 – Creating a user

Accessing MySQL in PHP

Copyright © 2017 Randy Connolly and Ricardo Hoar

EXERCISE 9.6 — MySQL Through PHP

1 Open config.php and modify the file as follows (if you used different user name and password in EXERCISE 9.5, 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 lab09-exercise06-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 lab09-exercise06-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 9.7 — INTEGRATING USER INPUTS (MYSQLI)

1 Open and examine lab09-exercise07.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.

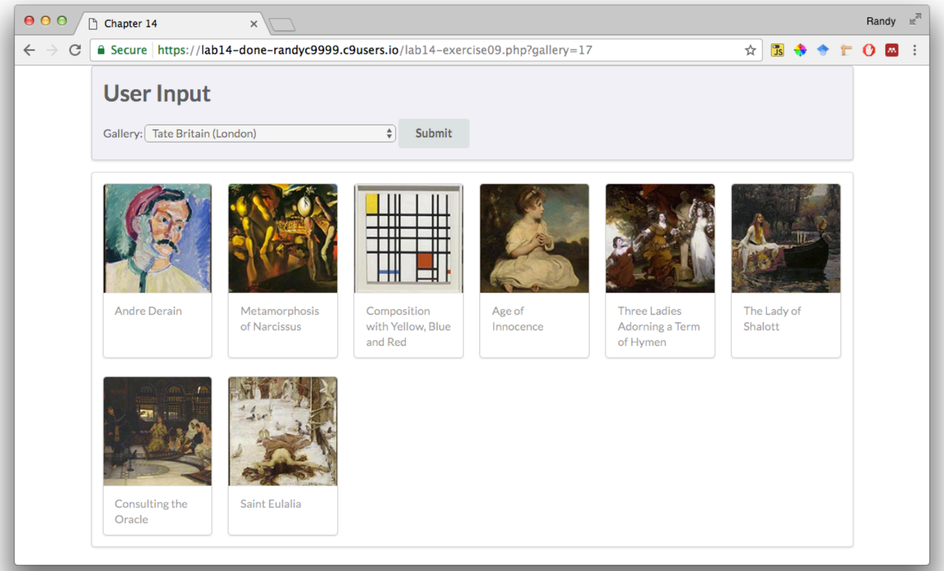
2 Add the following code and test.

```
<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 9.4. 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.



https://lab14-done-randyc9999.c9users.io/lab14-exercise09.php?gallery=17

User Input

Gallery: Tate Britain (London)

Andre Derain

Metamorphosis of Narcissus

Composition with Yellow, Blue and Red

Age of Innocence

Three Ladies Adorning a Term of Hymen

The Lady of Shalott

Consulting the Oracle

Saint Eulalia

Figure 9.4 – EXERCISE 9.7 complete

EXERCISE 9.8 — INTEGRATING USER INPUTS (PDO)

1 Open and examine lab09-exercise08.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()`.

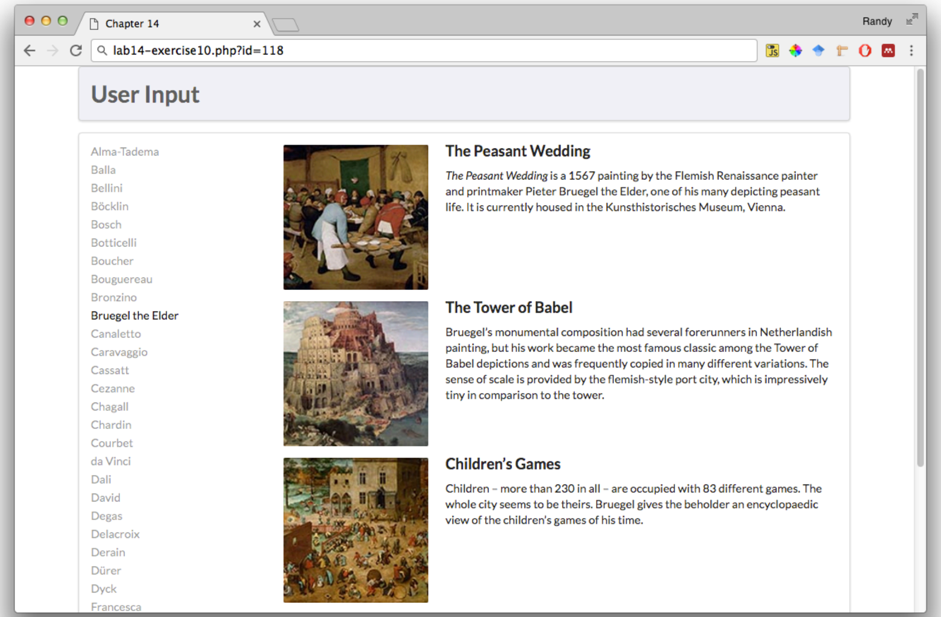
```
/*
 Displays the list of paintings for the artist id specified in the id
 query string
*/
function outputPaintings() {
   try {
      if (isset($_GET['id']) && $_GET['id'] > 0) {
         $pdo = new PDO(DBCONNSTRING,DBUSER,DBPASS);
         $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

         $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 9.5.

Note, not all paintings contain an excerpt.



The Peasant Wedding

The Peasant Wedding is a 1567 painting by the Flemish Renaissance painter and printmaker Pieter Bruegel the Elder, one of his many depicting peasant life. It is currently housed in the Kunsthistorisches Museum, Vienna.

The Tower of Babel

Bruegel's monumental composition had several forerunners in Netherlandish painting, but his work became the most famous classic among the Tower of Babel depictions and was frequently copied in many different variations. The sense of scale is provided by the flemish-style port city, which is impressively tiny in comparison to the tower.

Children's Games

Children – more than 230 in all – are occupied with 83 different games. The whole city seems to be theirs. Bruegel gives the beholder an encyclopaedic view of the children's games of his time.

Figure 9.5 – EXERCISE 9.8 complete

EXERCISE 9.9 — PREPARED STATEMENTS

1 Open and examine lab09-exercise09.php.

This file is the same as the finished version of exercise 8.

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 9.10 — HTML List and Results

1 Open and examine lab09-exercise10.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 GenreName';
      $result = $pdo->query($sql);
      while ($row = $result->fetch()) {
         outputSingleGenre($row);
```

Copyright © 2017 Randy Connolly and Ricardo Hoar

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

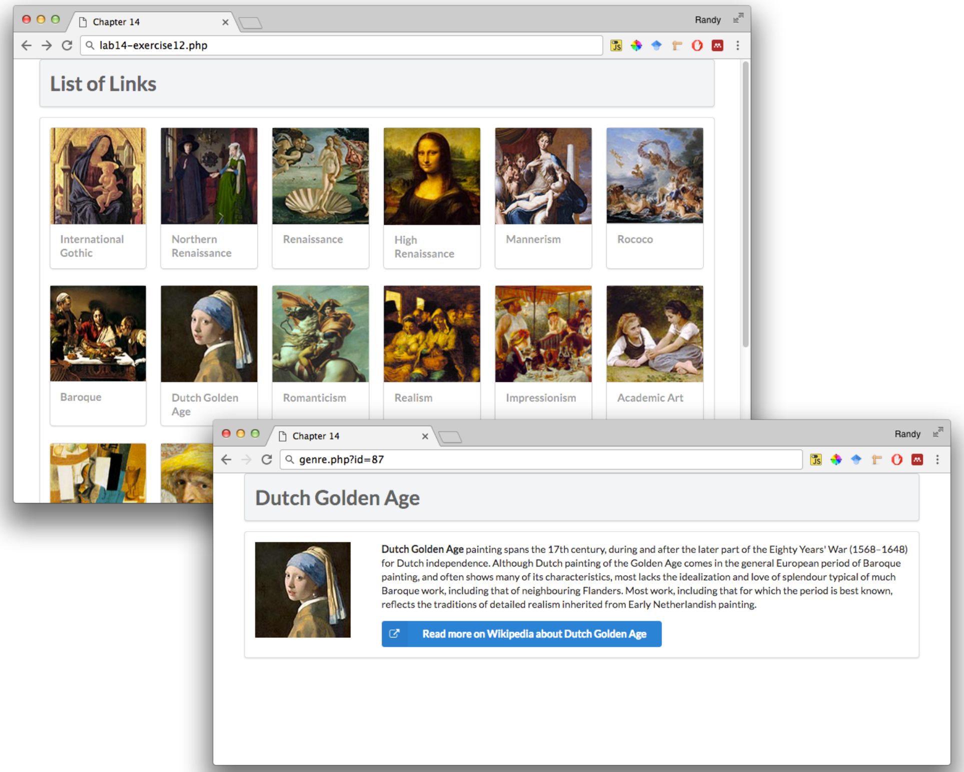
3 Define the following two functions.

```
/*
 Displays a single genre
*/
function outputSingleGenre($row) {
   echo '<div class="ui fluid card">';
   echo '<div class="ui fluid image">';
   $img = '<img src="images/art/genres/square-medium/' .
                   $row['GenreId'] .'.jpg">';
   echo constructGenreLink($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 9.6. Test the links. They should display the appropriate genre page.



Dutch Golden Age painting spans the 17th century, during and after the later part of the Eighty Years' War (1568–1648) for Dutch independence. Although Dutch painting of the Golden Age comes in the general European period of Baroque painting, and often shows many of its characteristics, most lacks the idealization and love of splendour typical of much Baroque work, including that of neighbouring Flanders. Most work, including that for which the period is best known, reflects the traditions of detailed realism inherited from Early Netherlandish painting.

Figure 9.6 – EXERCISE 9.10 complete

Copyright © 2017 Randy Connolly and Ricardo Hoar