LAB 17

WEB APPLICATION DESIGN

What You Will Learn

- How to construct a simple adapter class for database access
-

Approximate Time

The exercises in this lab should take approximately 50 minutes to complete.

Fundamentals of Web Development, 2nd Ed

Randy Connolly and Ricardo Hoar

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

Date Last Revised: Oct 27, 2017

Creating a Data Adapter

Preparing Directories

1 If you haven't done so already, create a folder in your personal drive for all the labs for this book.

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 `Lab17` to your course folder created in step one.

3 Be sure to create and populate your database.

This lab walks you through the creation of a simplified data access layer. To begin, you will create a simple adapter class for the PDO API.

Exercise 17.1 — Including Class Files

1 Examine book-config.inc.php in the includes folder.

2 Add the following code after the comment to this file then save:

```
// auto load all classes so we don't have to explicitly include them
spl_autoload_register(function ($class) {
    $file = 'lib/' . $class . '.class.php';
    if (file_exists($file))
      include $file;
});
```

As the code comment indicates, this code will automatically load all the PHP files contained with the lib folder that have the .class.php extension.

Exercise 17.2 — Create a Data Adapter Class

1 Examine DatabaseHelper.class.php in the lib folder.

2 Add the following methods to this class:

```
class DatabaseHelper {

   // Create the connection to the database
   public static function createConnectionInfo($values=array()) {
       // pass in the connection string, username, and password as array
       $connString = $values[0];
       $user = $values[1];
       $password = $values[2];

       $pdo = new PDO($connString,$user,$password);
       $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
       return $pdo;
   }

   // run an SQL query and return the cursor to the database
   public static function runQuery($connection, $sql, $parameters=array()) {
       // Ensure parameters are in an array
       if (!is_array($parameters)) {
           $parameters = array($parameters);
       }

       $statement = null;
       if (count($parameters) > 0) {
           // Use a prepared statement if parameters
           $statement = $connection->prepare($sql);
           $executedOk = $statement->execute($parameters);
           if (! $executedOk) {
              throw new PDOException;
           }
       } else {
           // Execute a normal query
           $statement = $connection->query($sql);
           if (!$statement) {
               throw new PDOException;
           }
       }
       return $statement;
   }
}
```

Our adapter simply allows a user to create a connection and to run queries.

3 To test this adapter, we will first need to create a connection to the database. Add the following code to the end of book-config.inc.php in the includes folder.

```
// connect to the database
$connection = DatabaseHelper::createConnectionInfo(array(DBCONNECTION,
                                     DBUSER, DBPASS));
```

4 To test the adapter, add the following code to the top of tester-adapter.php.

```
<?php
include 'includes/book-config.inc.php';

$sql = "select * from Imprints";
$statement = DatabaseHelper::runQuery($connection, $sql, null);
?>
```

5 Then add the following code to the body of the page then test.

```
<body>
<h1>Imprints (using DatabaseHelper)</h1>
<?php
   while ($row = $statement->fetch()) {
      echo $row['ImprintID'] . ' ' . $row['Imprint'] . '<br>';

   }
?>
</body>
```

This should display the information from the database table.

CREATING A GATEWAY CLASS

Exercise 17.3 - CREATE A SIMPLE GATEWAY CLASS

1 Add the following methods to ImprintDB.class.php in the lib folder.

```
class ImprintDB {

    private $connect = null;

    private static $baseSQL = "SELECT ImprintID,Imprint FROM Imprints";
    private static $constraint = " ORDER BY Imprint";

    // constructor will be passed the database connection
    public function __construct($connection) {
        $this->connect = $connection;
    }

    // return all the records
    public function getAll()
    {
        $sql = self::$baseSQL . self::$constraint;
        $statement = DatabaseHelper::runQuery($this->connect, $sql, null);
        return $statement->fetchAll();
    }

    // return just a single record whose key value = passed parameter
    public function findById($id)
    {
        $sql = self::$baseSQL .  ' WHERE ImprintID=:id ';
        $statement = DatabaseHelper::runQuery($this->connect, $sql,
                                              Array(':id' => $id));
        return $statement->fetch();
    }

}
```

Exercise 17.4 — Testing the Gateway

1 To test this adapter, add the following code to the body of tester-gateway.php.

```
try {
    $db = new ImprintDB($connection );
    $result = $db->findById(5);
    echo '<h3>Sample Imprint (id=5)</h3>';
    echo $result['ImprintID'] . ' ' . $result['Imprint'];

    $result = $db->getAll();
    echo '<h3>All Imprints</h3>';
    foreach ($result as $row) {
      echo $row['ImprintID'] . ' ' . $row['Imprint'] . ', ';
    }
}
catch (Exception $e) {
   die( $e->getMessage() );
}
```

This should display the information from the database table.

A Better Gateway

Exercise 17.5 — A Bit more Complicated GateWay

1 Edit the class in the lib folder named TableGateway.class.php.

2 Add the following abstract method definitions:

```
/*
 The name of the table in the database
*/
abstract protected function getSelectStatement();

/*
 A list of fields that define the sort order
*/
abstract protected function getOrderFields();

/*
 The name of the primary keys in the database ... this can be overridden by
subclasses
*/
abstract protected function getPrimaryKeyName();
```

3 Add the following methods:

```
/*
  Returns all the records in the table
*/
public function findAll($sortFields=null)
{
  $sql = $this->getSelectStatement();
  // add sort order if required
  if (! is_null($sortFields)) {
     $sql .= ' ORDER BY ' . $sortFields;
  }
  $statement = DatabaseHelper::runQuery($this->connect, $sql, null);
  return $statement->fetchAll();
}

/*
  Returns all the records in the table sorted by the specified sort order
*/
public function findAllSorted($ascending)
{
  $sql = $this->getSelectStatement() . ' ORDER BY ' .
                    $this->getOrderFields();
  if (! $ascending) {
     $sql .= " DESC";
  }
  $statement = DatabaseHelper::runQuery($this->connect, $sql, null);
  return $statement->fetchAll();
}

/*
  Returns a record for the specificed ID
*/
public function findById($id)
{
  $sql = $this->getSelectStatement() . ' WHERE ' .
                    $this->getPrimaryKeyName() . '=:id';

  $statement = DatabaseHelper::runQuery($this->connect, $sql,
                    Array(':id' => $id));
  return $statement->fetch();
}
```

3 Create a new Gateway class in the lib folder named EmployeesGateway.class.php with the following code:

```
class EmployeesGateway extends TableDataGateway {

   public function __construct($connect)    {
      parent::__construct($connect);
   }

   protected function getSelectStatement()
   {
      return "SELECT EmployeeID, FirstName, LastName, Address, City,
                     Region, Country, Postal, Email FROM Employees ";
   }

   protected function getOrderFields()    {
      return 'LastName, FirstName';
   }
   protected function getPrimaryKeyName() {
      return "EmployeeID";
   }
}
```

Notice that this gateway class, in comparison to the ImprintDB version created earlier, has much less code in it.

4 Test this gateway class by adding the following to your tester-gateway.php page.

```
echo '<hr>';

$db = new EmployeesGateway($connection );
$result = $db->findById(11);
echo '<h3>Sample Employee (id=11)</h3>';
echo $result['EmployeeID'] . ' ' . $result['FirstName'] . ' ' .
             $result['LastName'] . ' ' . $result['Address'];

$result = $db->findAll();
echo '<h3>All Employees</h3>';
foreach ($result as $row) {
  echo $row['EmployeeID'] . ' ' . $row['LastName'] . ', ';
}
```