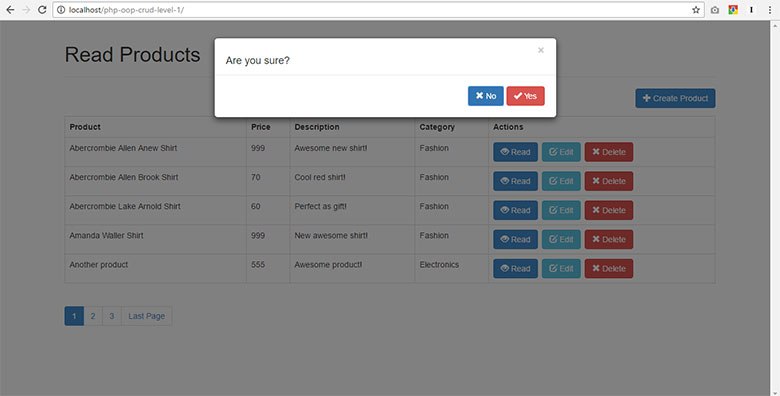
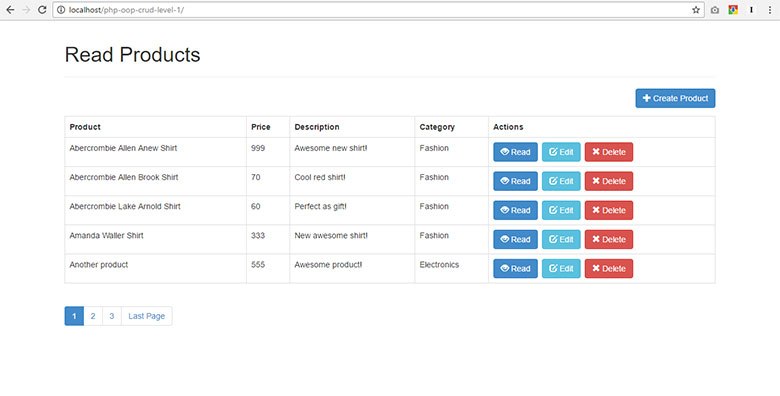
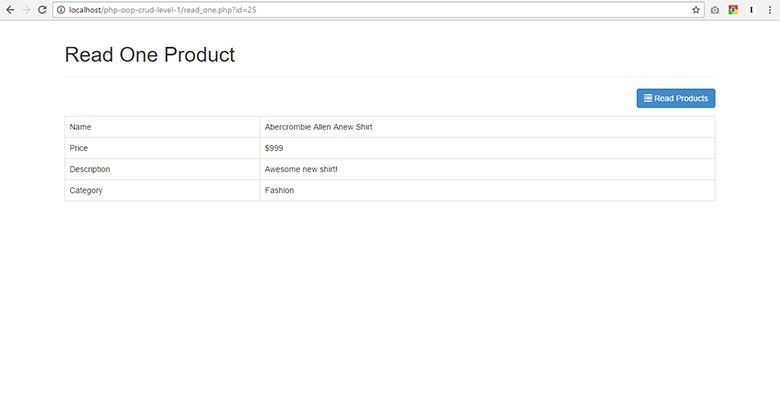
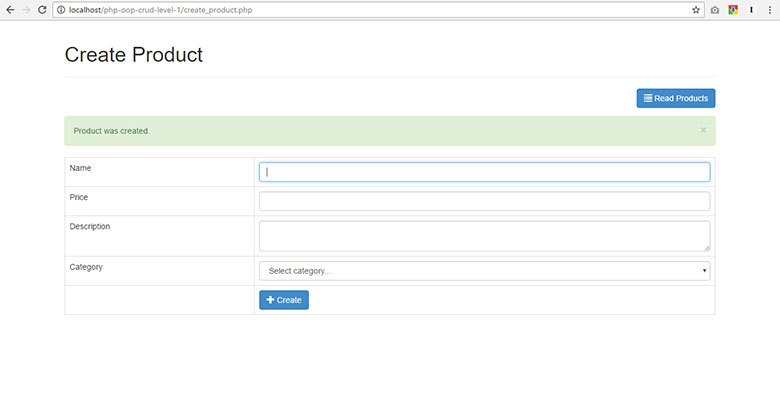
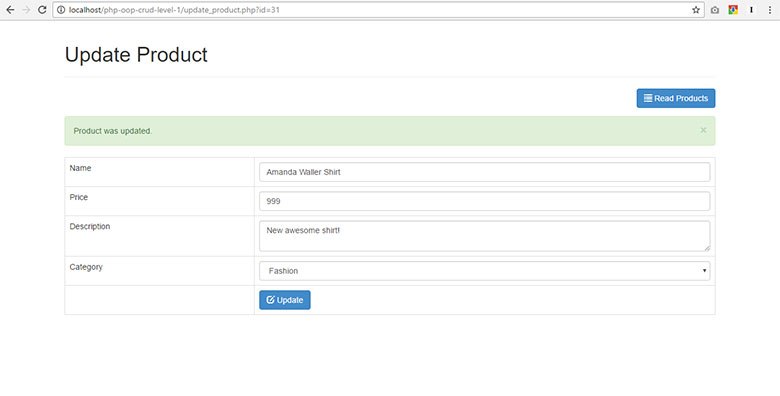
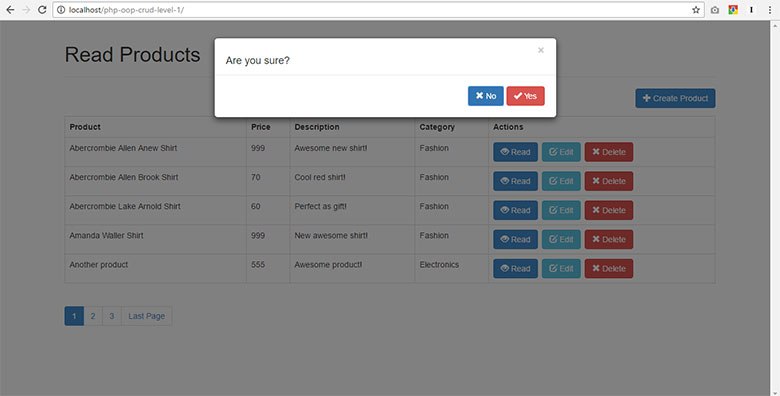
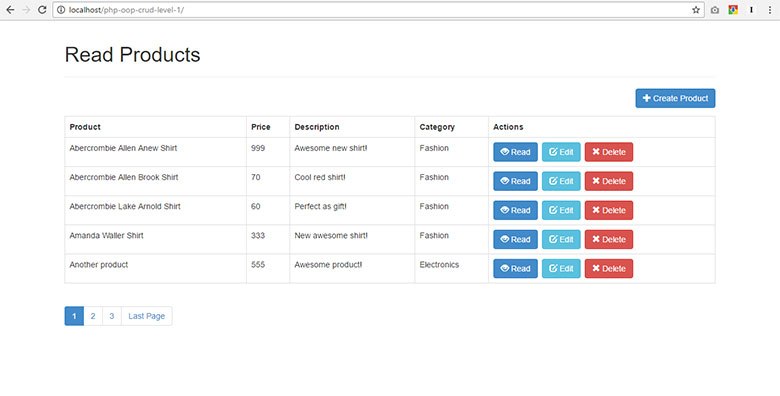
1. **Overview**

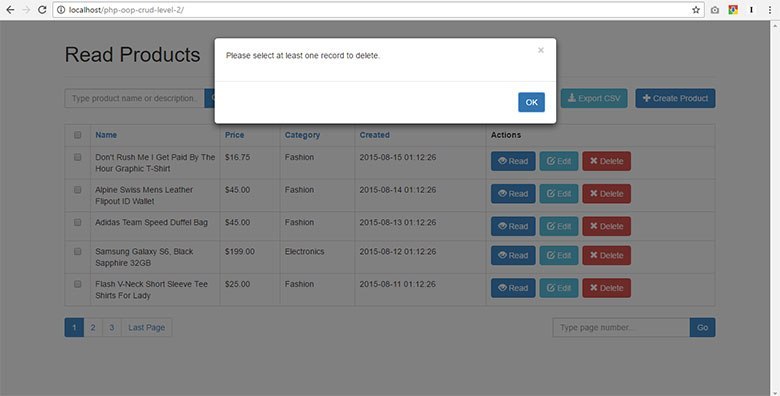
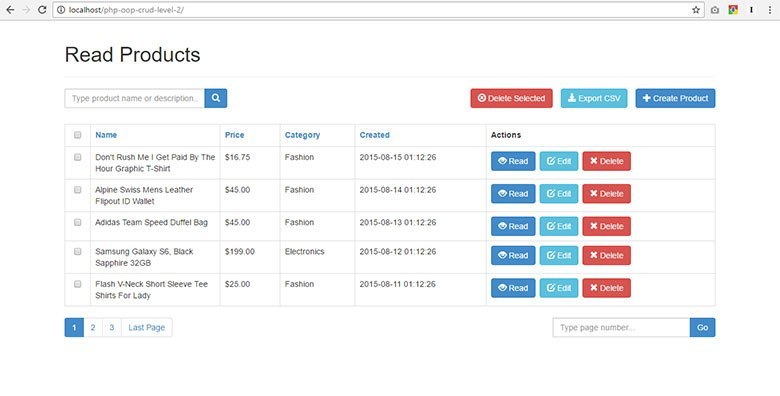
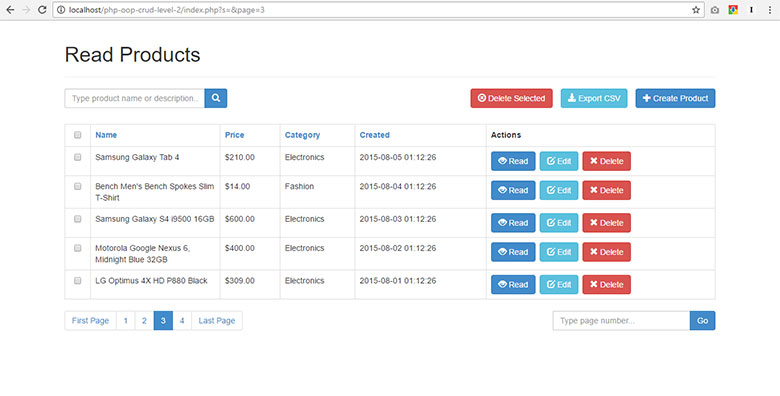
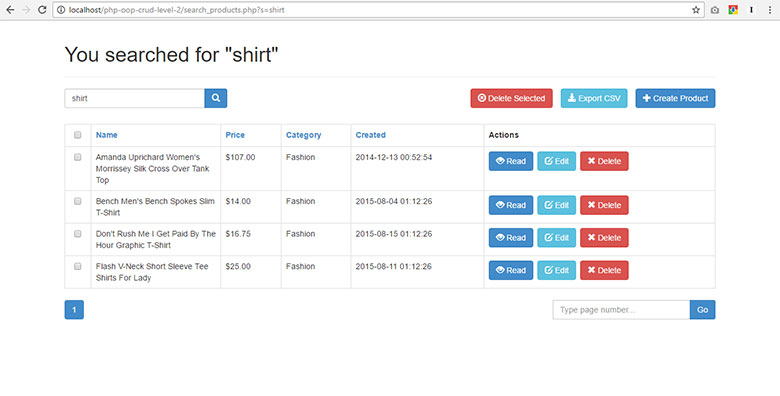
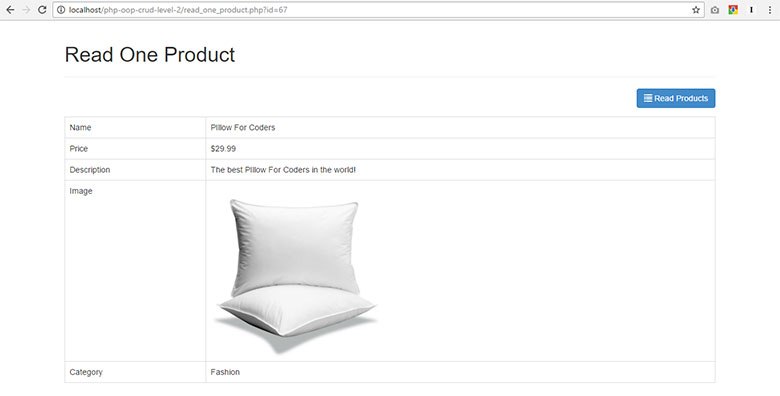
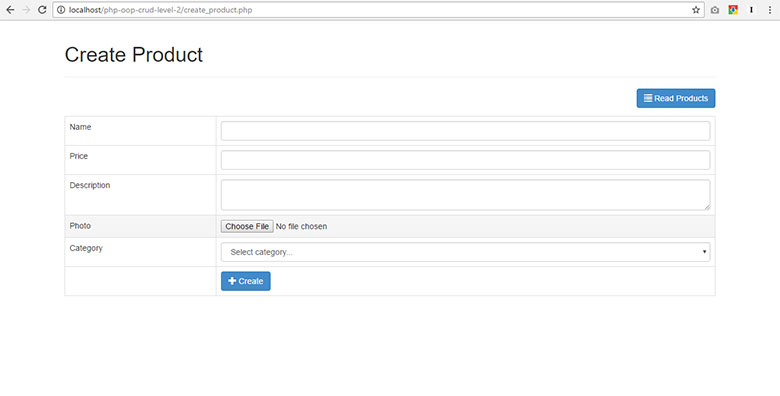
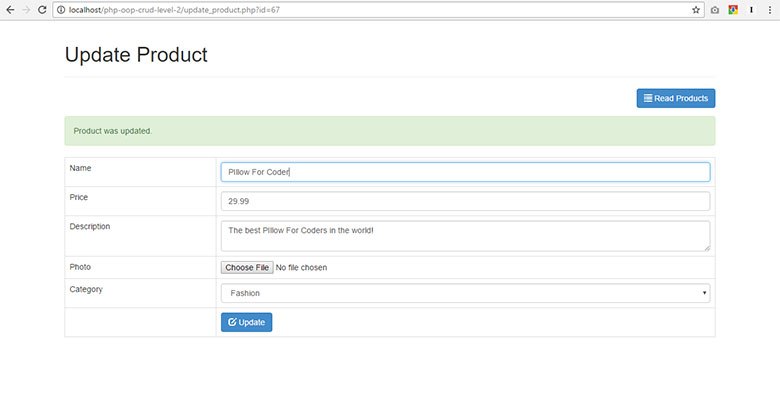
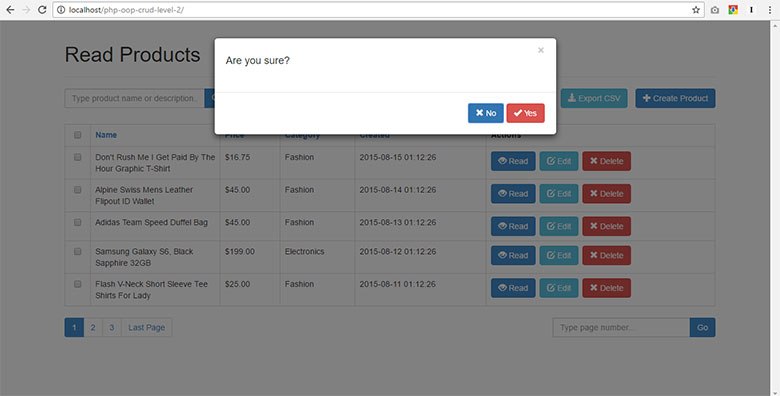
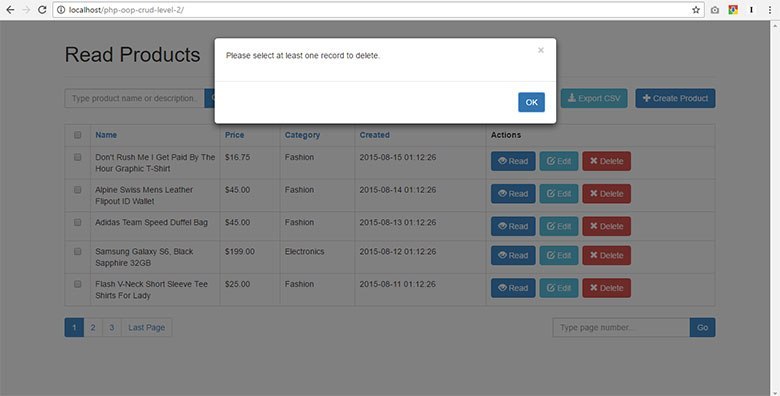
Create a simple database application. We will achieve this with a PHP OOP CRUD example. You can use this knowledge in your current or future projects.

1. **OUTPUT / UI**

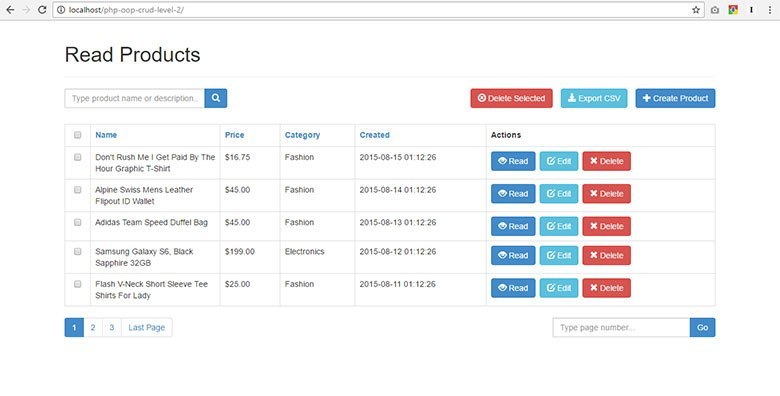
### 2.1 LEVEL 1 Source Code Output

* [](https://i0.wp.com/www.codeofaninja.com/wp-content/uploads/2016/12/5-delete-record.jpg?fit=780%2C396&ssl=1)
* [](https://i0.wp.com/www.codeofaninja.com/wp-content/uploads/2016/12/1-read-records-1.jpg?fit=780%2C396&ssl=1)
* [](https://i1.wp.com/www.codeofaninja.com/wp-content/uploads/2016/12/2-read-one-product-2.jpg?fit=780%2C396&ssl=1)
* [](https://i1.wp.com/www.codeofaninja.com/wp-content/uploads/2016/12/3-create-record.jpg?fit=780%2C396&ssl=1)
* [](https://i2.wp.com/www.codeofaninja.com/wp-content/uploads/2016/12/4-update-record.jpg?fit=780%2C396&ssl=1)
* [](https://i0.wp.com/www.codeofaninja.com/wp-content/uploads/2016/12/5-delete-record.jpg?fit=780%2C396&ssl=1)
* [](https://i0.wp.com/www.codeofaninja.com/wp-content/uploads/2016/12/1-read-records-1.jpg?fit=780%2C396&ssl=1)

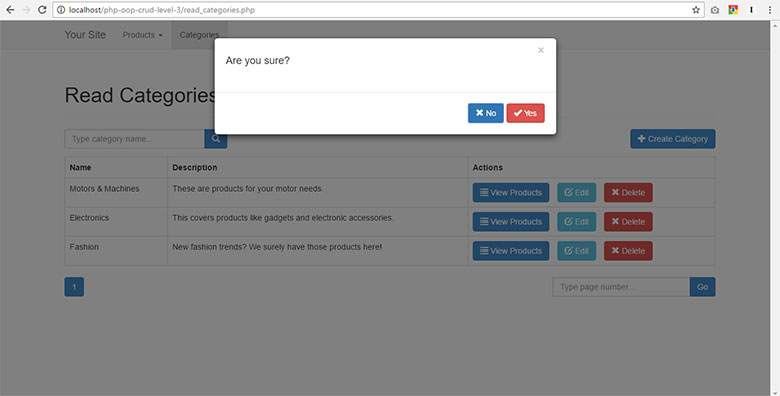
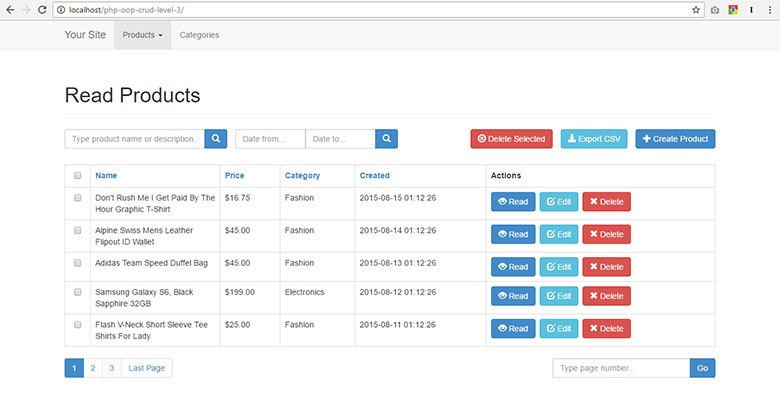
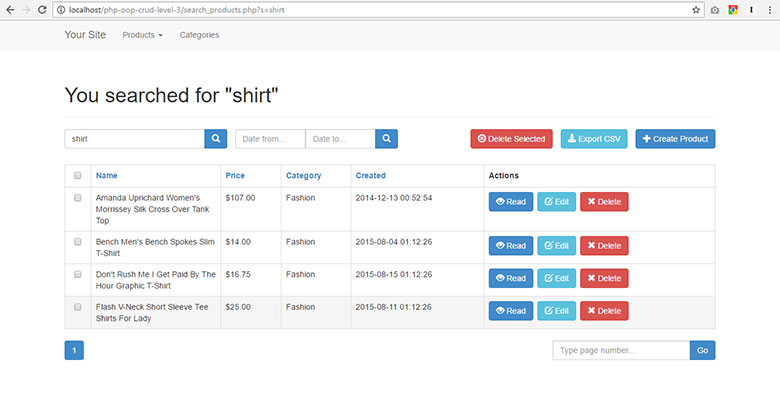
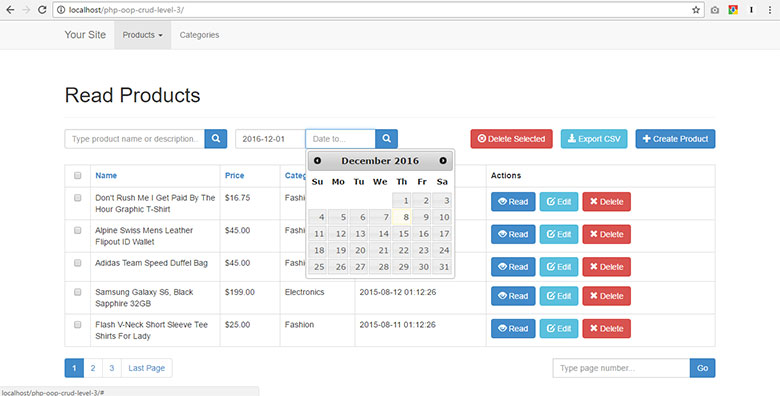
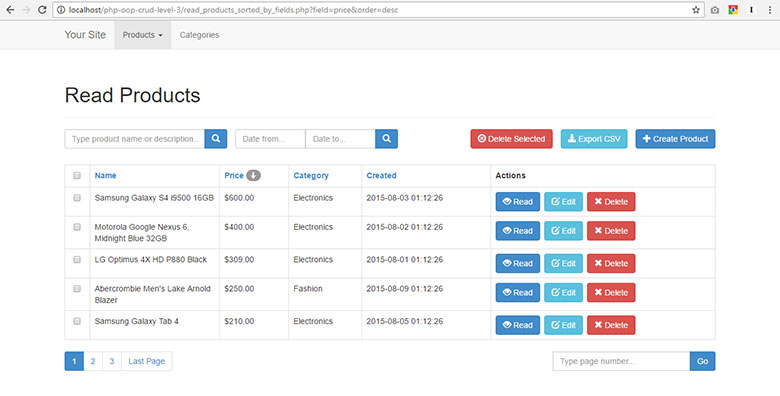
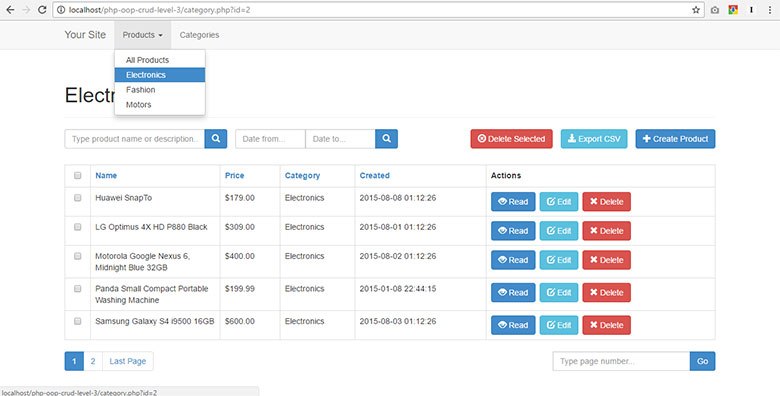
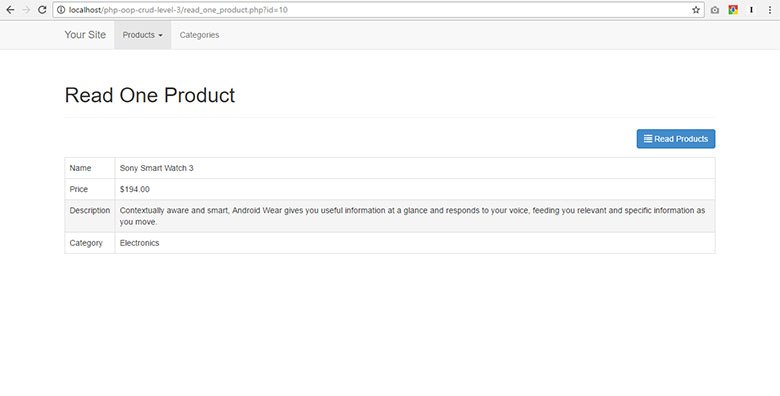
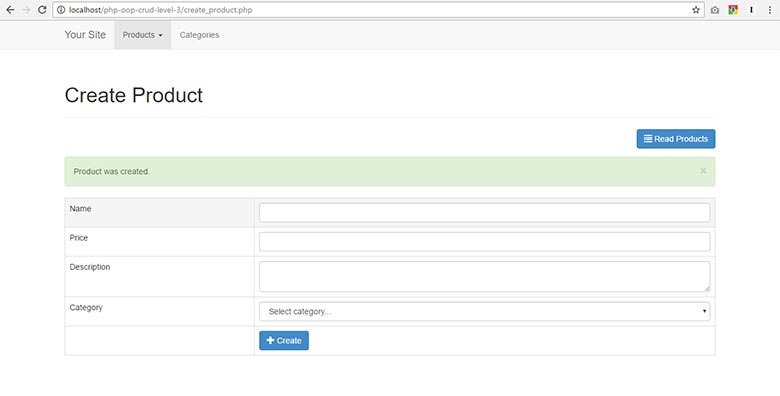
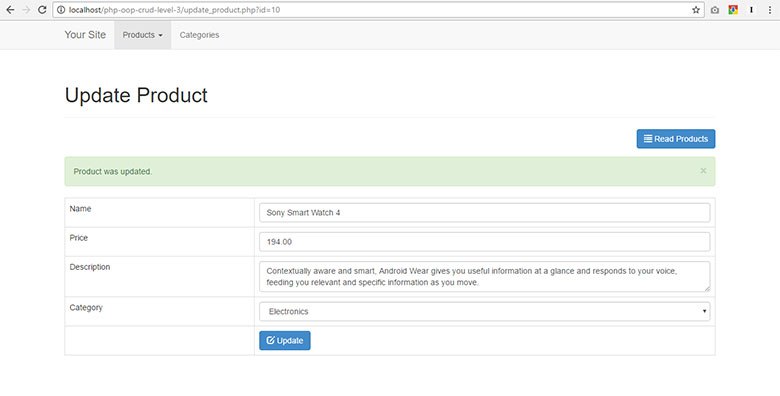
### 2.2 LEVEL 2 Source Code Output

* [](https://i1.wp.com/www.codeofaninja.com/wp-content/uploads/2016/12/8-delete-selected-records.jpg?fit=780%2C396&ssl=1)
* [](https://i0.wp.com/www.codeofaninja.com/wp-content/uploads/2016/12/1-read-records-2.jpg?fit=780%2C396&ssl=1)
* [](https://i0.wp.com/www.codeofaninja.com/wp-content/uploads/2016/12/2-records-pagination.jpg?fit=780%2C396&ssl=1)
* [](https://i0.wp.com/www.codeofaninja.com/wp-content/uploads/2016/12/3-search-records-1.jpg?fit=780%2C396&ssl=1)
* [](https://i1.wp.com/www.codeofaninja.com/wp-content/uploads/2017/01/read-one-product.jpg?fit=780%2C396&ssl=1)
* [](https://i2.wp.com/www.codeofaninja.com/wp-content/uploads/2017/01/create-product-record.jpg?fit=780%2C396&ssl=1)
* [](https://i2.wp.com/www.codeofaninja.com/wp-content/uploads/2017/01/update-product-record.jpg?fit=780%2C396&ssl=1)
* [](https://i1.wp.com/www.codeofaninja.com/wp-content/uploads/2016/12/7-delete-record.jpg?fit=780%2C396&ssl=1)
* [[](https://i1.wp.com/www.codeofaninja.com/wp-content/uploads/2016/12/8-delete-selected-records.jpg?fit=780%2C396&ssl=1)](https://i1.wp.com/www.codeofaninja.com/wp-content/uploads/2016/12/8-delete-selected-records.jpg?fit=780%2C396&ssl=1" \o "#8 Delete Selected Records (Pop Up When No Records Were Selected))

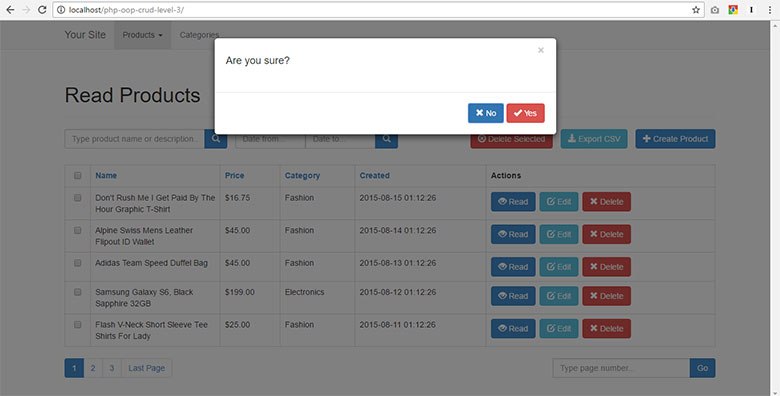
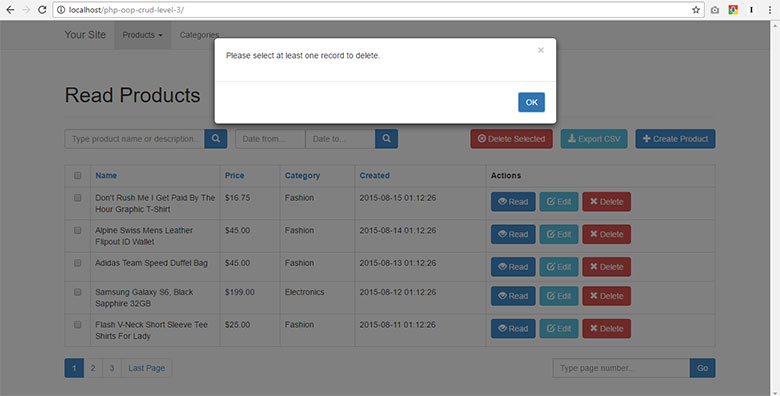
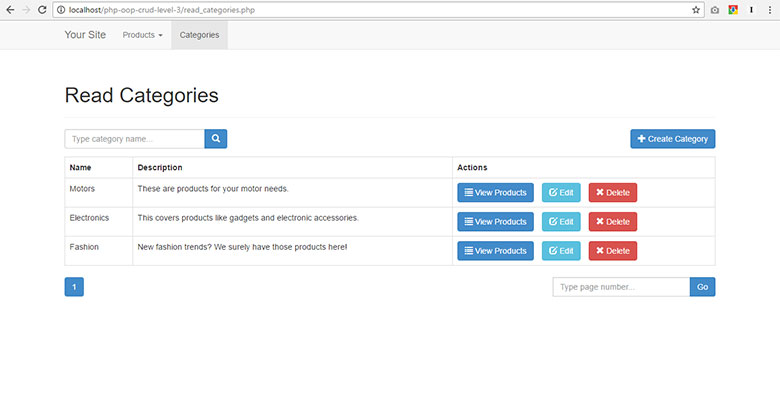
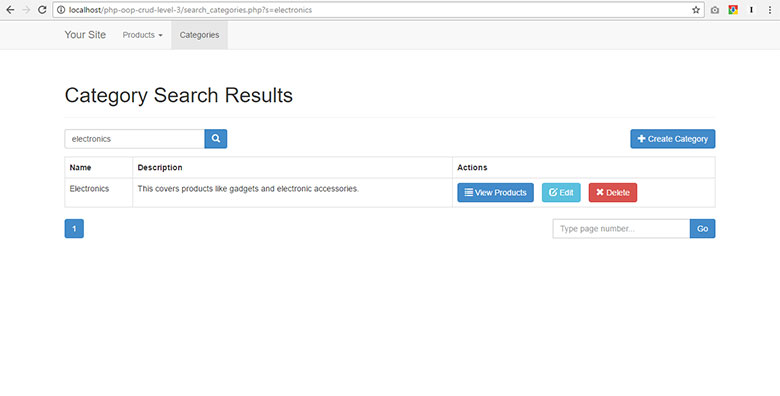
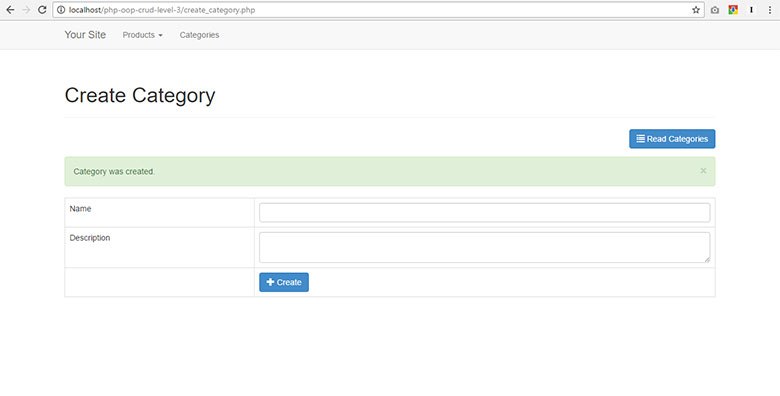
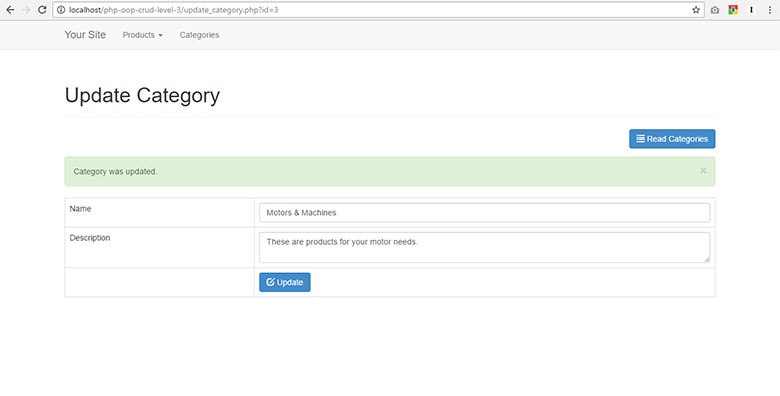
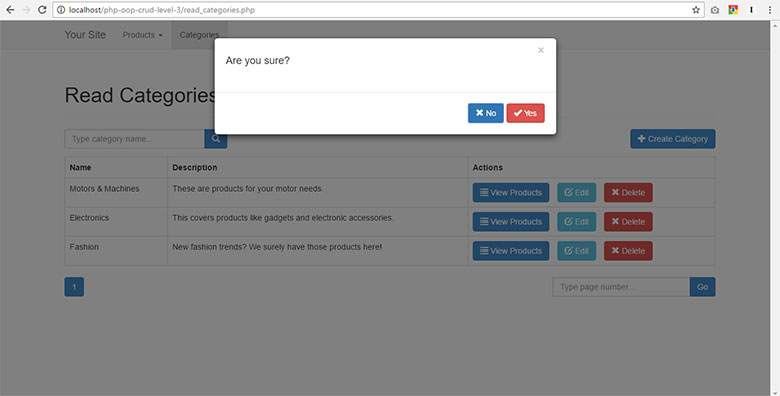
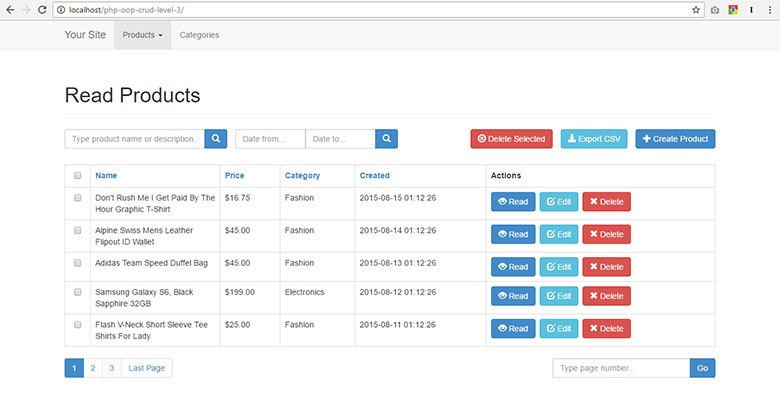
[#8 Delete Selected Records (Pop Up When No Records Were Selected)](https://i1.wp.com/www.codeofaninja.com/wp-content/uploads/2016/12/8-delete-selected-records.jpg?fit=780%2C396&ssl=1" \o "#8 Delete Selected Records (Pop Up When No Records Were Selected))

* [](https://i0.wp.com/www.codeofaninja.com/wp-content/uploads/2016/12/1-read-records-2.jpg?fit=780%2C396&ssl=1)

### 2.3 LEVEL 3 Source Code Output

* [](https://i2.wp.com/www.codeofaninja.com/wp-content/uploads/2016/12/15-delete-category.jpg?fit=780%2C396&ssl=1)
* [](https://i1.wp.com/www.codeofaninja.com/wp-content/uploads/2016/12/1-read-products-2.jpg?fit=780%2C396&ssl=1)
* [](https://i1.wp.com/www.codeofaninja.com/wp-content/uploads/2016/12/2-search-products.jpg?fit=780%2C396&ssl=1)
* [](https://i1.wp.com/www.codeofaninja.com/wp-content/uploads/2016/12/3-search-product-by-date-range.jpg?fit=780%2C396&ssl=1)
* [](https://i1.wp.com/www.codeofaninja.com/wp-content/uploads/2016/12/4-sort-products-by-price.jpg?fit=780%2C396&ssl=1)
* [](https://i2.wp.com/www.codeofaninja.com/wp-content/uploads/2016/12/5-read-products-by-category.jpg?fit=780%2C396&ssl=1)
* [](https://i1.wp.com/www.codeofaninja.com/wp-content/uploads/2016/12/6-read-one-product.jpg?fit=780%2C396&ssl=1)
* [](https://i0.wp.com/www.codeofaninja.com/wp-content/uploads/2016/12/7-create-product-1.jpg?fit=780%2C396&ssl=1)
* [[](https://i0.wp.com/www.codeofaninja.com/wp-content/uploads/2016/12/8-update-product-1.jpg?fit=780%2C396&ssl=1)](https://i0.wp.com/www.codeofaninja.com/wp-content/uploads/2016/12/8-update-product-1.jpg?fit=780%2C396&ssl=1" \o "#8 Update Product)

[#8 Update Product](https://i0.wp.com/www.codeofaninja.com/wp-content/uploads/2016/12/8-update-product-1.jpg?fit=780%2C396&ssl=1" \o "#8 Update Product)

* [](https://i0.wp.com/www.codeofaninja.com/wp-content/uploads/2016/12/9-delete-product-2.jpg?fit=780%2C396&ssl=1)
* [](https://i2.wp.com/www.codeofaninja.com/wp-content/uploads/2016/12/10-delete-selected-product.jpg?fit=780%2C396&ssl=1)
* [](https://i2.wp.com/www.codeofaninja.com/wp-content/uploads/2016/12/11-read-categories-1.jpg?fit=780%2C396&ssl=1)
* [](https://i0.wp.com/www.codeofaninja.com/wp-content/uploads/2016/12/12-search-categories.jpg?fit=780%2C396&ssl=1)
* [](https://i1.wp.com/www.codeofaninja.com/wp-content/uploads/2016/12/13-create-category.jpg?fit=780%2C396&ssl=1)
* [](https://i1.wp.com/www.codeofaninja.com/wp-content/uploads/2016/12/14-update-category.jpg?fit=780%2C396&ssl=1)
* [](https://i2.wp.com/www.codeofaninja.com/wp-content/uploads/2016/12/15-delete-category.jpg?fit=780%2C396&ssl=1)
* [](https://i1.wp.com/www.codeofaninja.com/wp-content/uploads/2016/12/1-read-products-2.jpg?fit=780%2C396&ssl=1)

**3.0 DATABASE TABLE STRUCTURE**

The files products.sql and categories.sql are also included in the code download, located at the **README** folder.

**3.1 Create a database**

* Open your PhpMyAdmin (http://localhost/phpmyadmin)
* Create a new database.
* Put php\_oop\_crud\_level\_1 as database name.
* Click "Create" button.

**3.2 Create products table**

In this section, we will create the "products" table (using PhpMyAdmin) on the database we just created.

Here's how to run an SQL statement using PhpMyAdmin.

* Click php\_oop\_crud\_level\_1 database.
* Click "SQL" tab.
* Copy the SQL statement below and paste it in the text area.
* Click the "Go" button.

|  |
| --- |
| -- Table structure for table `products`  CREATE TABLE IF NOT EXISTS `products` (    `id` int(11) NOT NULL AUTO\_INCREMENT,    `name` varchar(32) NOT NULL,    `description` text NOT NULL,    `price` int(11) NOT NULL,    `category\_id` int(11) NOT NULL,    `created` datetime NOT NULL,    `modified` timestamp NOT NULL DEFAULT CURRENT\_TIMESTAMP,    PRIMARY KEY (`id`)  ) ENGINE=MyISAM  DEFAULT CHARSET=latin1 AUTO\_INCREMENT=38 ; |

**3.3 Insert products sample data**

We have to put some database records.

Run the following SQL statement using your PhpMyAdmin.

|  |
| --- |
| -- Dumping data for table `products`  INSERT INTO `products` (`id`, `name`, `description`, `price`, `category\_id`, `created`, `modified`) VALUES  (1, 'LG P880 4X HD', 'My first awesome phone!', 336, 3, '2014-06-01 01:12:26', '2014-05-31 17:12:26'),  (2, 'Google Nexus 4', 'The most awesome phone of 2013!', 299, 2, '2014-06-01 01:12:26', '2014-05-31 17:12:26'),  (3, 'Samsung Galaxy S4', 'How about no?', 600, 3, '2014-06-01 01:12:26', '2014-05-31 17:12:26'),  (6, 'Bench Shirt', 'The best shirt!', 29, 1, '2014-06-01 01:12:26', '2014-05-31 02:12:21'),  (7, 'Lenovo Laptop', 'My business partner.', 399, 2, '2014-06-01 01:13:45', '2014-05-31 02:13:39'),  (8, 'Samsung Galaxy Tab 10.1', 'Good tablet.', 259, 2, '2014-06-01 01:14:13', '2014-05-31 02:14:08'),  (9, 'Spalding Watch', 'My sports watch.', 199, 1, '2014-06-01 01:18:36', '2014-05-31 02:18:31'),  (10, 'Sony Smart Watch', 'The coolest smart watch!', 300, 2, '2014-06-06 17:10:01', '2014-06-05 18:09:51'),  (11, 'Huawei Y300', 'For testing purposes.', 100, 2, '2014-06-06 17:11:04', '2014-06-05 18:10:54'),  (12, 'Abercrombie Lake Arnold Shirt', 'Perfect as gift!', 60, 1, '2014-06-06 17:12:21', '2014-06-05 18:12:11'),  (13, 'Abercrombie Allen Brook Shirt', 'Cool red shirt!', 70, 1, '2014-06-06 17:12:59', '2014-06-05 18:12:49'),  (25, 'Abercrombie Allen Anew Shirt', 'Awesome new shirt!', 999, 1, '2014-11-22 18:42:13', '2014-11-21 19:42:13'),  (26, 'Another product', 'Awesome product!', 555, 2, '2014-11-22 19:07:34', '2014-11-21 20:07:34'),  (27, 'Bag', 'Awesome bag for you!', 999, 1, '2014-12-04 21:11:36', '2014-12-03 22:11:36'),  (28, 'Wallet', 'You can absolutely use this one!', 799, 1, '2014-12-04 21:12:03', '2014-12-03 22:12:03'),  (30, 'Wal-mart Shirt', '', 555, 2, '2014-12-13 00:52:29', '2014-12-12 01:52:29'),  (31, 'Amanda Waller Shirt', 'New awesome shirt!', 333, 1, '2014-12-13 00:52:54', '2014-12-12 01:52:54'),  (32, 'Washing Machine Model PTRR', 'Some new product.', 999, 1, '2015-01-08 22:44:15', '2015-01-07 23:44:15'); |

**3.4 Create categories table**

Categories table are used to store product categories.

Run the following SQL statement using your PhpMyAdmin.

|  |
| --- |
| -- Table structure for table `categories`  CREATE TABLE IF NOT EXISTS `categories` (    `id` int(11) NOT NULL AUTO\_INCREMENT,    `name` varchar(256) NOT NULL,    `created` datetime NOT NULL,    `modified` timestamp NOT NULL DEFAULT CURRENT\_TIMESTAMP,    PRIMARY KEY (`id`)  ) ENGINE=MyISAM  DEFAULT CHARSET=utf8 AUTO\_INCREMENT=4 ; |

**3.5 Insert categories sample data**

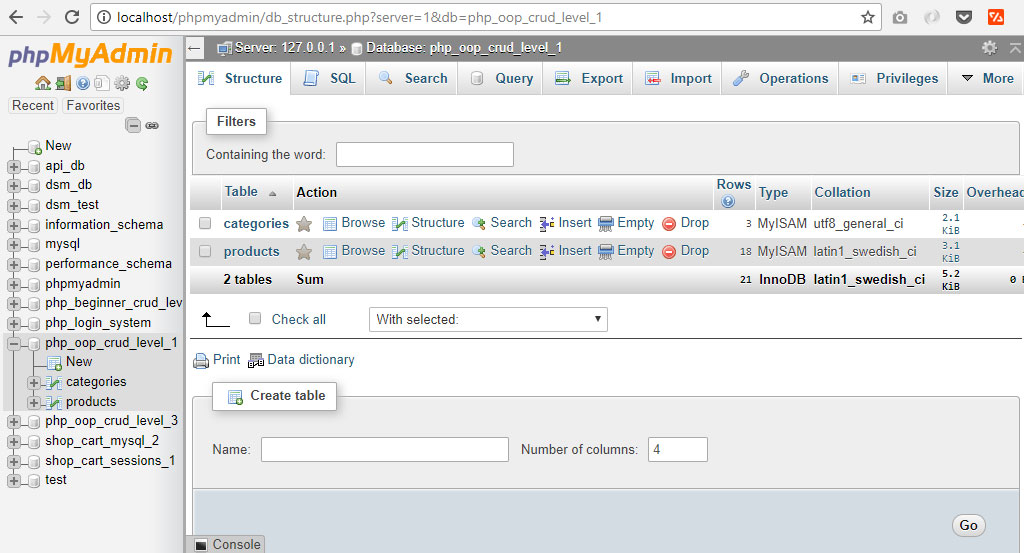
We are going to have "Fashion", "Electronics" and "Motors" as categories in our project. I got those three category ideas from eBay, haha!

Run the following SQL statement using your PhpMyAdmin.

|  |
| --- |
| -- Dumping data for table `categories`  INSERT INTO `categories` (`id`, `name`, `created`, `modified`) VALUES  (1, 'Fashion', '2014-06-01 00:35:07', '2014-05-30 17:34:33'),  (2, 'Electronics', '2014-06-01 00:35:07', '2014-05-30 17:34:33'),  (3, 'Motors', '2014-06-01 00:35:07', '2014-05-30 17:34:54'); |

**3.6 Output**

In this section, we were able to set up our database using PhpMyAdmin. It should look like the image below.



We don't have a PHP program output yet. Let's continue on the next section to achieve more output.

**4.0 CREATE THE LAYOUT FILES**

To reduce some code mess, we will create the layout files with the codes and assets it needs.

**4.1 Create header layout file**

This layout\_header.php file will be included at the beginning of the PHP files that will need it. This way, we won't have to write the same header codes every time.

We use the [Bootstrap framework](https://getbootstrap.com/) to make our project look good. If you're not yet familiar with you, please learn our [Bootstrap tutorial here](https://www.codeofaninja.com/2014/05/bootstrap-tutorial-beginners-step-step.html) first.

Bootstrap CSS asset will be included inside the head tags.

* Create php-oop-crud-level-1 folder and open it.
* Create layout\_header.php file.
* Place the following code.

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>        <meta charset="utf-8">      <meta http-equiv="X-UA-Compatible" content="IE=edge">      <meta name="viewport" content="width=device-width, initial-scale=1">        <title><?php echo $page\_title; ?></title>        <!-- Latest compiled and minified Bootstrap CSS -->      <link rel="stylesheet" href="<https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css>" />        <!-- our custom CSS -->      <link rel="stylesheet" href="libs/css/custom.css" />    </head>  <body>        <!-- container -->      <div class="container">            <?php          // show page header          echo "<div class='page-header'>                  <h1>{$page\_title}</h1>              </div>";          ?> |

**4.2 Create footer layout file**

This layout\_footer.php will be included at the end of each PHP files that needs it. This way, we won't have to write the same footer codes every time.

The assets used in this file are:

* [jQuery](https://jquery.com/) - needed by Bootstrap JavaScript.
* [Bootstrap JavaScript](https://getbootstrap.com/docs/3.3/javascript/) - to make cool UI components work.
* [BootboxJS](http://bootboxjs.com/getting-started.html) - to show good looking alert or confirm dialog boxes.

Let's go on and create the footer layout file.

* Open php-oop-crud-level-1 folder.
* Create layout\_footer.php file.
* Place the following code.

|  |
| --- |
| </div>      <!-- /container -->    <!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->  <script src="<https://code.jquery.com/jquery-3.2.1.min.js>"></script>    <!-- Latest compiled and minified Bootstrap JavaScript -->  <script src="<https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js>"></script>    <!-- bootbox library -->  <script src="<https://cdnjs.cloudflare.com/ajax/libs/bootbox.js/4.4.0/bootbox.min.js>"></script>    </body>  </html> |

**4.3 Create custom CSS file**

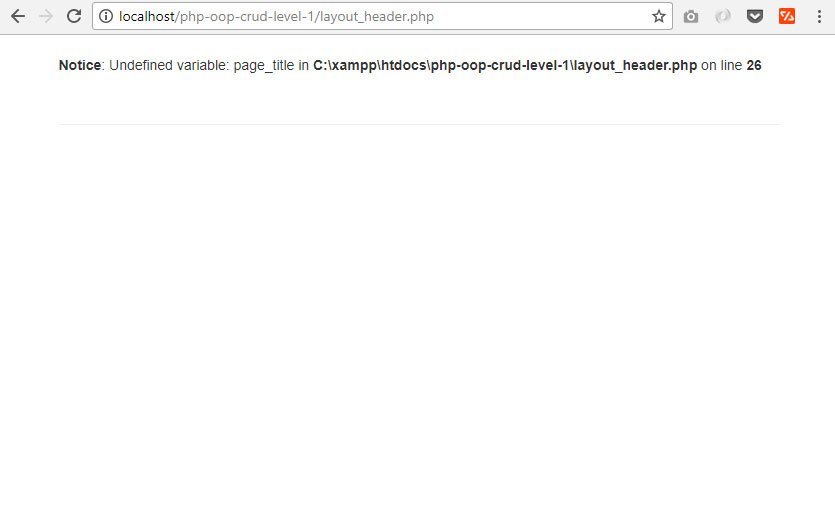
This file is used to change any style we want on our web page. It is also used to override the default style given by Bootstrap.

* Open php-oop-crud-level-1 folder.
* Create libs folder.
* Create css folder.
* Create custom.css file.
* Place the following code.

|  |
| --- |
| .left-margin{      margin:0 .5em 0 0;  }    .right-button-margin{      margin: 0 0 1em 0;      overflow: hidden;  }    /\* some changes in bootstrap modal \*/  .modal-body {      padding: 20px 20px 0px 20px !important;      text-align: center !important;  }    .modal-footer{      text-align: center !important;  } |

**4.4 Output**

The layout files we created in this section is meant to be used inside another PHP file. If we try to run the layout files alone, we won't get any meaningful output.

If you run layout\_header.php, it will look like this on the browser.  


The custom.css file will look like this.  


The layout\_footer.php is blank. Let's continue on the next section to see a more meaningful output.

**5.0 CREATING RECORD IN PHP THE OOP WAY**

**5.1 Create a file: create\_product.php**

Go back to php-oop-crud-level-1 folder, create a file with a name create\_product.php and put the following code inside it.

|  |
| --- |
| <?php  // set page headers  $page\_title = "Create Product";  include\_once "layout\_header.php";    // contents will be here    // footer  include\_once "layout\_footer.php";  ?> |

**5.2 Create a "Read Products" Button**

The following code will render a button. Replace the comments // contents will be here of the previous section with the following.

|  |
| --- |
| echo "<div class='right-button-margin'>";      echo "<a href='index.php' class='btn btn-default pull-right'>Read Products</a>";  echo "</div>";    ?>  <!-- 'create product' html form will be here -->  <?php |

**5.3 Get a Database Connection**

We can use it for retrieving categories or saving new product record later. Put the following code before // set page headers comment of create\_product.php file.

|  |
| --- |
| // include database and object files  include\_once 'config/database.php';  include\_once 'objects/product.php';  include\_once 'objects/category.php';    // get database connection  $database = new Database();  $db = $database->getConnection();    // pass connection to objects  $product = new Product($db);  $category = new Category($db); |

**5.4 Create the Database Configuration Class**

Getting a database connection will not work without this class. This class file will be included in most PHP files of our PHP OOP CRUD Tutorial.

Create a config folder and inside that folder, create a database.php file. Open that file and put the following code.

|  |
| --- |
| <?php  class Database{        // specify your own database credentials      private $host = "localhost";      private $db\_name = "php\_oop\_crud\_level\_1";      private $username = "root";      private $password = "";      public $conn;        // get the database connection      public function getConnection(){            $this->conn = null;            try{              $this->conn = new PDO("mysql:host=" . $this->host . ";dbname=" . $this->db\_name, $this->username, $this->password);          }catch(PDOException $exception){              echo "Connection error: " . $exception->getMessage();          }            return $this->conn;      }  }  ?> |

**5.5 Create a Form in create\_product.php**

The following code will render an HTML form. Open create\_product.php file.

Replace <!-- 'create product' html form will be here --> comment with the following code.

|  |
| --- |
| <!-- PHP post code will be here -->    <!-- HTML form for creating a product -->  <form action="<?php echo htmlspecialchars($\_SERVER["PHP\_SELF"]);?>" method="post">        <table class='table table-hover table-responsive table-bordered'>            <tr>              <td>Name</td>              <td><input type='text' name='name' class='form-control' /></td>          </tr>            <tr>              <td>Price</td>              <td><input type='text' name='price' class='form-control' /></td>          </tr>            <tr>              <td>Description</td>              <td><textarea name='description' class='form-control'></textarea></td>          </tr>            <tr>              <td>Category</td>              <td>              <!-- categories from database will be here -->              </td>          </tr>            <tr>              <td></td>              <td>                  <button type="submit" class="btn btn-primary">Create</button>              </td>          </tr>        </table>  </form> |

**5.6 Loop Through the Categories Records to show as Drop-down**

The following code will retrieve categories and put it in a "select" drop-down.

Replace <!-- categories from database will be here --> comment of the previous section with the following code.

|  |
| --- |
| <?php  // read the product categories from the database  $stmt = $category->read();    // put them in a select drop-down  echo "<select class='form-control' name='category\_id'>";      echo "<option>Select category...</option>";        while ($row\_category = $stmt->fetch(PDO::FETCH\_ASSOC)){          extract($row\_category);          echo "<option value='{$id}'>{$name}</option>";      }    echo "</select>";  ?> |

**5.7 Create the Object Class for Categories**

Of course, the previous section won't work without the category object class. Create objects folder. Create category.php file. Place the following code.

|  |
| --- |
| <?php  class Category{        // database connection and table name      private $conn;      private $table\_name = "categories";        // object properties      public $id;      public $name;        public function \_\_construct($db){          $this->conn = $db;      }        // used by select drop-down list      function read(){          //select all data          $query = "SELECT                      id, name                  FROM                      " . $this->table\_name . "                  ORDER BY                      name";            $stmt = $this->conn->prepare( $query );          $stmt->execute();            return $stmt;      }    }  ?> |

**5.8 Prepare readName() method**

It will get the category name instead of showing just an ID. Add the following code inside our category.php, you will see this method used in the next few sections.

|  |
| --- |
| // used to read category name by its ID  function readName(){        $query = "SELECT name FROM " . $this->table\_name . " WHERE id = ? limit 0,1";        $stmt = $this->conn->prepare( $query );      $stmt->bindParam(1, $this->id);      $stmt->execute();        $row = $stmt->fetch(PDO::FETCH\_ASSOC);        $this->name = $row['name'];  } |

**5.9 Code when the Form was Submitted**

The user will enter the values in the HTML form and when the create (submit) button was clicked, values will be sent via POST request, the code below will save it in the database.

Open create\_product.php file. Replace <!-- PHP post code will be here --> comment with the following code.

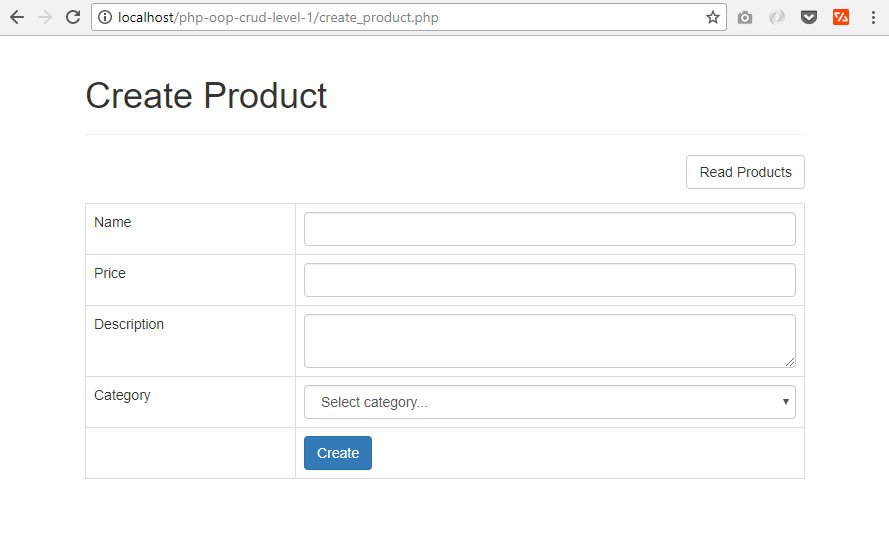
|  |
| --- |
| <?php  // if the form was submitted - PHP OOP CRUD Tutorial  if($\_POST){        // set product property values      $product->name = $\_POST['name'];      $product->price = $\_POST['price'];      $product->description = $\_POST['description'];      $product->category\_id = $\_POST['category\_id'];        // create the product      if($product->create()){          echo "<div class='alert alert-success'>Product was created.</div>";      }        // if unable to create the product, tell the user      else{          echo "<div class='alert alert-danger'>Unable to create product.</div>";      }  }  ?> |

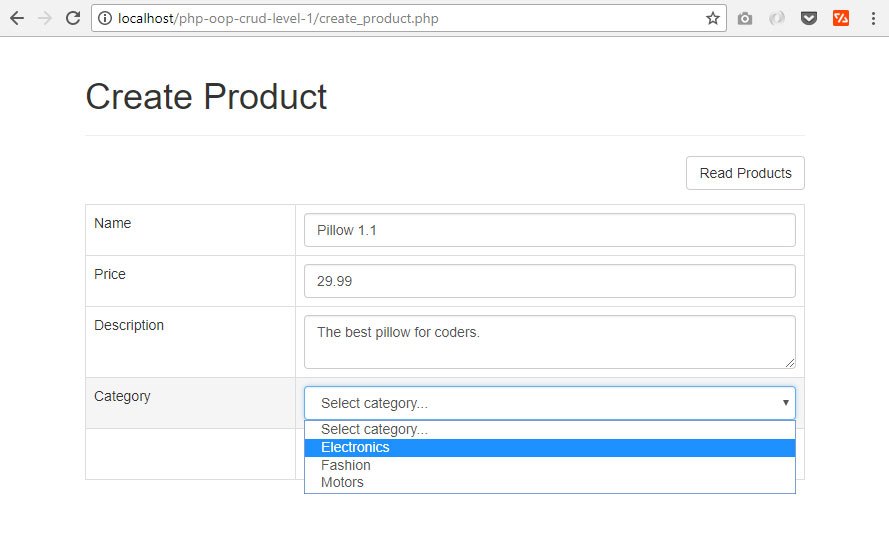
**5.10 Create the Object Class for Products**

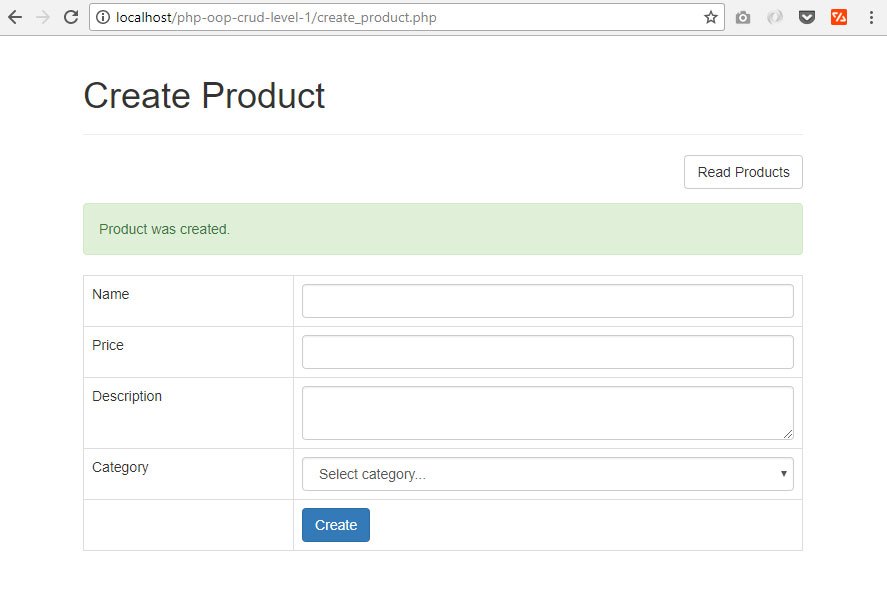
The previous section will not work without the product object. Open objects folder. Create product.php file. Open that file and put the following code.

|  |
| --- |
| <?php  class Product{        // database connection and table name      private $conn;      private $table\_name = "products";        // object properties      public $id;      public $name;      public $price;      public $description;      public $category\_id;      public $timestamp;        public function \_\_construct($db){          $this->conn = $db;      }        // create product      function create(){            //write query          $query = "INSERT INTO                      " . $this->table\_name . "                  SET                      name=:name, price=:price, description=:description, category\_id=:category\_id, created=:created";            $stmt = $this->conn->prepare($query);            // posted values          $this->name=htmlspecialchars(strip\_tags($this->name));          $this->price=htmlspecialchars(strip\_tags($this->price));          $this->description=htmlspecialchars(strip\_tags($this->description));          $this->category\_id=htmlspecialchars(strip\_tags($this->category\_id));            // to get time-stamp for 'created' field          $this->timestamp = date('Y-m-d H:i:s');            // bind values          $stmt->bindParam(":name", $this->name);          $stmt->bindParam(":price", $this->price);          $stmt->bindParam(":description", $this->description);          $stmt->bindParam(":category\_id", $this->category\_id);          $stmt->bindParam(":created", $this->timestamp);            if($stmt->execute()){              return true;          }else{              return false;          }        }  }  ?> |

**5.11 Output**

Form to create product.  


Categories drop down in the form.  


When you fill out the form and clicked the "Create" button.  


Changes in the database.  


**6.0 READING AND PAGING RECORD IN PHP THE OOP WAY**

In this part of our PHP OOP CRUD tutorial, we will list the records from the database.

**6.1 Create File: index.php**

Create a new file and name it index.php. This file will show the main page of our web app. Put the following code inside it.

|  |
| --- |
| <?php  // set page header  $page\_title = "Read Products";  include\_once "layout\_header.php";    // contents will be here    // set page footer  include\_once "layout\_footer.php";  ?> |

**6.2 Add a "Create Product" button**

The following code will render a button. When this button was clicked, it will show us a page where we can create a record. Replace the // contents will be here comments in the previous section with the following code.

|  |
| --- |
| echo "<div class='right-button-margin'>      <a href='create\_product.php' class='btn btn-default pull-right'>Create Product</a>  </div>"; |

**6.3 Configure Pagination Variables**

Pagination is very important if you have thousands of data from the database. Put the following code before the set page header comment of index.php file.

|  |
| --- |
| // page given in URL parameter, default page is one  $page = isset($\_GET['page']) ? $\_GET['page'] : 1;    // set number of records per page  $records\_per\_page = 5;    // calculate for the query LIMIT clause  $from\_record\_num = ($records\_per\_page \* $page) - $records\_per\_page;    // retrieve records here |

**6.4 Retrieve Records from the Database**

Now we will retrieve data from the database. Replace // retrieve records here comment of index.php with the following code.

|  |
| --- |
| // include database and object files  include\_once 'config/database.php';  include\_once 'objects/product.php';  include\_once 'objects/category.php';    // instantiate database and objects  $database = new Database();  $db = $database->getConnection();    $product = new Product($db);  $category = new Category($db);    // query products  $stmt = $product->readAll($from\_record\_num, $records\_per\_page);  $num = $stmt->rowCount(); |

**6.5 Add readAll() Method in product.php**

Retrieving records in the previous section won't work without this method. Put the following code inside our "product.php" file which is inside the "objects" folder.

|  |
| --- |
| function readAll($from\_record\_num, $records\_per\_page){        $query = "SELECT                  id, name, description, price, category\_id              FROM                  " . $this->table\_name . "              ORDER BY                  name ASC              LIMIT                  {$from\_record\_num}, {$records\_per\_page}";        $stmt = $this->conn->prepare( $query );      $stmt->execute();        return $stmt;  } |

**6.6 Display data from the database**

This time, we will show the list of records to the user. An HTML table will hold our data. Put the following code after the section 6.2 code.

|  |
| --- |
| // display the products if there are any  if($num>0){        echo "<table class='table table-hover table-responsive table-bordered'>";          echo "<tr>";              echo "<th>Product</th>";              echo "<th>Price</th>";              echo "<th>Description</th>";              echo "<th>Category</th>";              echo "<th>Actions</th>";          echo "</tr>";            while ($row = $stmt->fetch(PDO::FETCH\_ASSOC)){                extract($row);                echo "<tr>";                  echo "<td>{$name}</td>";                  echo "<td>{$price}</td>";                  echo "<td>{$description}</td>";                  echo "<td>";                      $category->id = $category\_id;                      $category->readName();                      echo $category->name;                  echo "</td>";                    echo "<td>";                      // read one, edit and delete button will be here                  echo "</td>";                echo "</tr>";            }        echo "</table>";        // paging buttons will be here  }    // tell the user there are no products  else{      echo "<div class='alert alert-info'>No products found.</div>";  } |

**6.7 Put the Read, Edit and Delete Action Buttons**

The following code will render three buttons: Read, Edit and Delete button.

Inside the "while" loop of the previous section, there is a comment "read one, edit and delete button will be here", replace that with the following code.

|  |
| --- |
| // read, edit and delete buttons  echo "<a href='read\_one.php?id={$id}' class='btn btn-primary left-margin'>      <span class='glyphicon glyphicon-list'></span> Read  </a>    <a href='update\_product.php?id={$id}' class='btn btn-info left-margin'>      <span class='glyphicon glyphicon-edit'></span> Edit  </a>    <a delete-id='{$id}' class='btn btn-danger delete-object'>      <span class='glyphicon glyphicon-remove'></span> Delete  </a>"; |

**6.8 Create paging.php for Paging Buttons**

The following code will show our pagination buttons. Create a new file and name it "paging.php". Open that file and put the following code.

|  |
| --- |
| <?php  echo "<ul class='pagination'>";    // button for first page  if($page>1){      echo "<li><a href='{$page\_url}' title='Go to the first page.'>";          echo "First";      echo "</a></li>";  }    // calculate total pages  $total\_pages = ceil($total\_rows / $records\_per\_page);    // range of links to show  $range = 2;    // display links to 'range of pages' around 'current page'  $initial\_num = $page - $range;  $condition\_limit\_num = ($page + $range)  + 1;    for ($x=$initial\_num; $x<$condition\_limit\_num; $x++) {        // be sure '$x is greater than 0' AND 'less than or equal to the $total\_pages'      if (($x > 0) && ($x <= $total\_pages)) {            // current page          if ($x == $page) {              echo "<li class='active'><a href=\"#\">$x <span class=\"sr-only\">(current)</span></a></li>";          }            // not current page          else {              echo "<li><a href='{$page\_url}page=$x'>$x</a></li>";          }      }  }    // button for last page  if($page<$total\_pages){      echo "<li><a href='" .$page\_url. "page={$total\_pages}' title='Last page is {$total\_pages}.'>";          echo "Last";      echo "</a></li>";  }    echo "</ul>";  ?> |

**6.9 Add the countAll() method in objects/product.php**

The following code will be used to count the total number of records in the database. This will be used for pagination.

Open your product.php file which is inside the "objects" folder. Add the following method in the class.

|  |
| --- |
| // used for paging products  public function countAll(){        $query = "SELECT id FROM " . $this->table\_name . "";        $stmt = $this->conn->prepare( $query );      $stmt->execute();        $num = $stmt->rowCount();        return $num;  } |

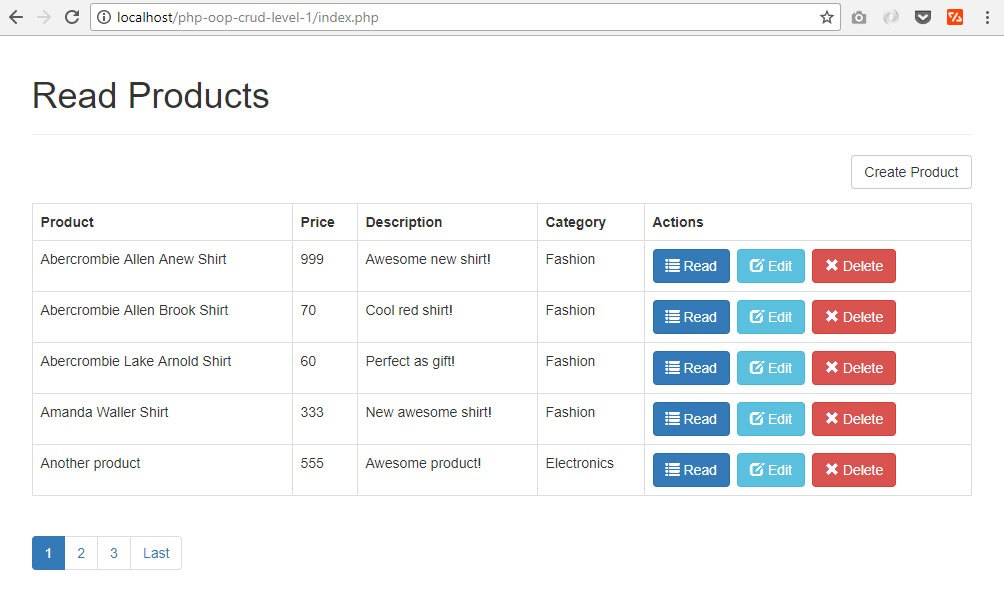
**6.10 Include paging.php in index.php**

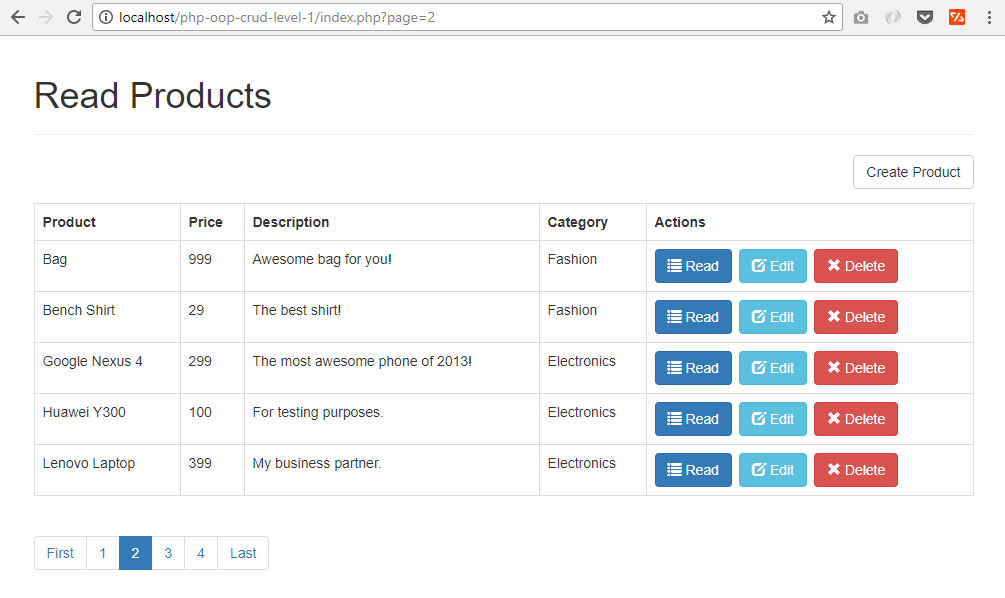
The following code will show our pagination buttons under our records list. Put the following code after the closing "table" tag of section 6.6 above.

|  |
| --- |
| // the page where this paging is used  $page\_url = "index.php?";    // count all products in the database to calculate total pages  $total\_rows = $product->countAll();    // paging buttons here  include\_once 'paging.php'; |

**6.11 Output**

Run <http://localhost/php-oop-crud-level-1/index.php> on your browser, you should see something like the image below.

List of records, page 1.  


List of records, page 2.  


**7.0 UPDATING RECORD IN PHP THE OOP WAY**

I know our PHP OOP CRUD tutorial is kinda long. Please take a break or drink some coffee first!

**7.1 Create File: update\_product.php**

Create update\_product.php file, open that file and put the following code.

|  |
| --- |
| <?php  // retrieve one product will be here    // set page header  $page\_title = "Update Product";  include\_once "layout\_header.php";    // contents will be here    // set page footer  include\_once "layout\_footer.php";  ?> |

**7.2 Create a "Read Products" Button**

The following code will render a button. This button, when clicked, will let us go back to the records list. Replace the previous section's "contents will be here" comments with the following code.

|  |
| --- |
| echo "<div class='right-button-margin'>";      echo "<a href='index.php' class='btn btn-default pull-right'>Read Products</a>";  echo "</div>"; |

**7.3 Retrieve One Product Information Based on the Given ID.**

The following code will retrieve data that will populate our HTML form. This is important because this will let the user know what exactly the record he is updating.

Open update\_product.php file. Replace "// retrieve one product will be here" comment with the following code.

|  |
| --- |
| // get ID of the product to be edited  $id = isset($\_GET['id']) ? $\_GET['id'] : die('ERROR: missing ID.');    // include database and object files  include\_once 'config/database.php';  include\_once 'objects/product.php';  include\_once 'objects/category.php';    // get database connection  $database = new Database();  $db = $database->getConnection();    // prepare objects  $product = new Product($db);  $category = new Category($db);    // set ID property of product to be edited  $product->id = $id;    // read the details of product to be edited  $product->readOne();    ?>  <!-- 'update product' form will be here -->  <?php |

**7.4 Add readOne() method in the Product Object Class.**

The readOne() method used in the previous section will not work without the following code inside /objects/product.php file.

|  |
| --- |
| function readOne(){        $query = "SELECT                  name, price, description, category\_id              FROM                  " . $this->table\_name . "              WHERE                  id = ?              LIMIT                  0,1";        $stmt = $this->conn->prepare( $query );      $stmt->bindParam(1, $this->id);      $stmt->execute();        $row = $stmt->fetch(PDO::FETCH\_ASSOC);        $this->name = $row['name'];      $this->price = $row['price'];      $this->description = $row['description'];      $this->category\_id = $row['category\_id'];  } |

**7.5 Put the Values in the Form.**

Now we can put the latest values to each form elements. Replace "<!-- 'update product' form will be here -->" comment of update\_product.php with the following code.

|  |
| --- |
| <!-- post code will be here -->    <form action="<?php echo htmlspecialchars($\_SERVER["PHP\_SELF"] . "?id={$id}");?>" method="post">      <table class='table table-hover table-responsive table-bordered'>            <tr>              <td>Name</td>              <td><input type='text' name='name' value='<?php echo $product->name; ?>' class='form-control' /></td>          </tr>            <tr>              <td>Price</td>              <td><input type='text' name='price' value='<?php echo $product->price; ?>' class='form-control' /></td>          </tr>            <tr>              <td>Description</td>              <td><textarea name='description' class='form-control'><?php echo $product->description; ?></textarea></td>          </tr>            <tr>              <td>Category</td>              <td>                  <!-- categories select drop-down will be here -->              </td>          </tr>            <tr>              <td></td>              <td>                  <button type="submit" class="btn btn-primary">Update</button>              </td>          </tr>        </table>  </form> |

**7.6 Loop Through the Categories Records to show as Drop-down**

The following code will list the categories in a drop-down.

Notice that we put "if($product->category\_id==$category\_id){..." inside the while loop. This is to pre-select the option of the current record.

Replace the previouse section's comments "categories select drop-down will be here" with the following code.

|  |
| --- |
| <?php  $stmt = $category->read();    // put them in a select drop-down  echo "<select class='form-control' name='category\_id'>";        echo "<option>Please select...</option>";      while ($row\_category = $stmt->fetch(PDO::FETCH\_ASSOC)){          $category\_id=$row\_category['id'];          $category\_name = $row\_category['name'];            // current category of the product must be selected          if($product->category\_id==$category\_id){              echo "<option value='$category\_id' selected>";          }else{              echo "<option value='$category\_id'>";          }            echo "$category\_name</option>";      }  echo "</select>";  ?> |

**7.7 Code When Form was Submitted**

The following code will assign the "posted" values to the object properties. Once assigned, it will update the database with those values using the update() method.

Open update\_product.php file. Replace <!-- post code will be here -->" comment with the following code.

|  |
| --- |
| <?php  // if the form was submitted  if($\_POST){        // set product property values      $product->name = $\_POST['name'];      $product->price = $\_POST['price'];      $product->description = $\_POST['description'];      $product->category\_id = $\_POST['category\_id'];        // update the product      if($product->update()){          echo "<div class='alert alert-success alert-dismissable'>";              echo "Product was updated.";          echo "</div>";      }        // if unable to update the product, tell the user      else{          echo "<div class='alert alert-danger alert-dismissable'>";              echo "Unable to update product.";          echo "</div>";      }  }  ?> |

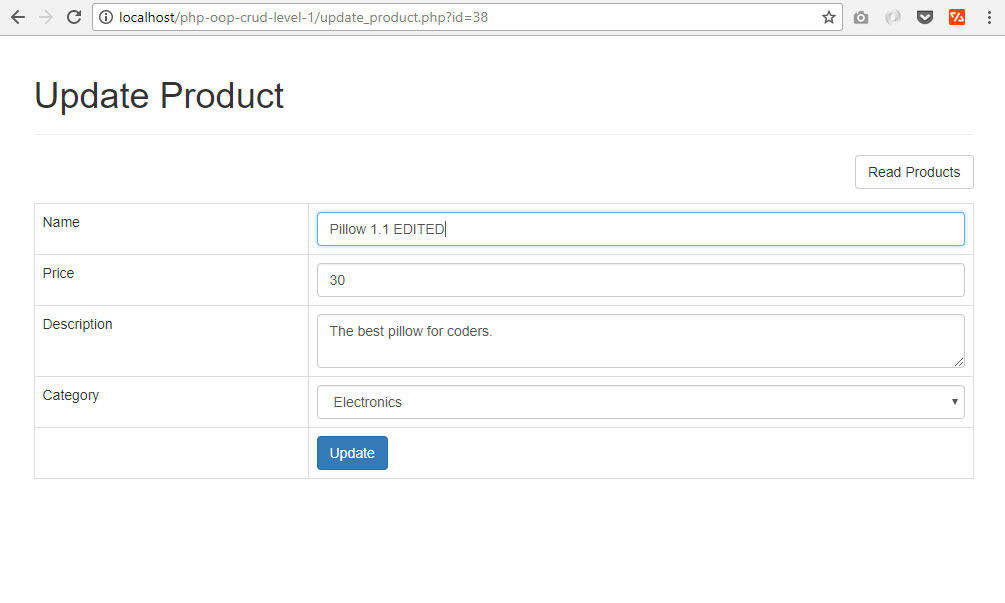
**7.8 Update Code in the Product Class**

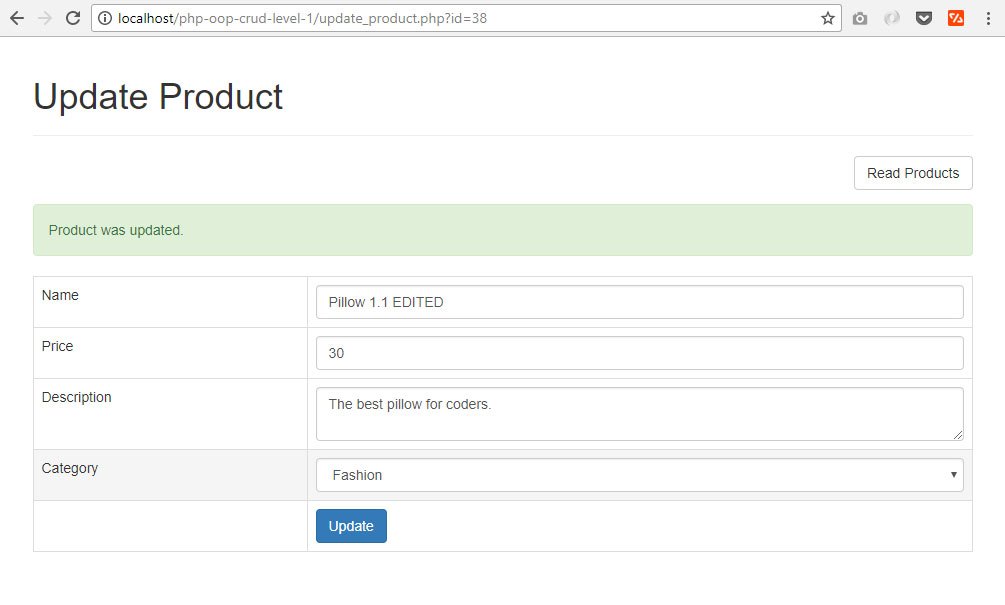
The following code will make the previous section's "$product->update()" method work. Open our "product.php" which is inside the "objects" folder and add the following code.

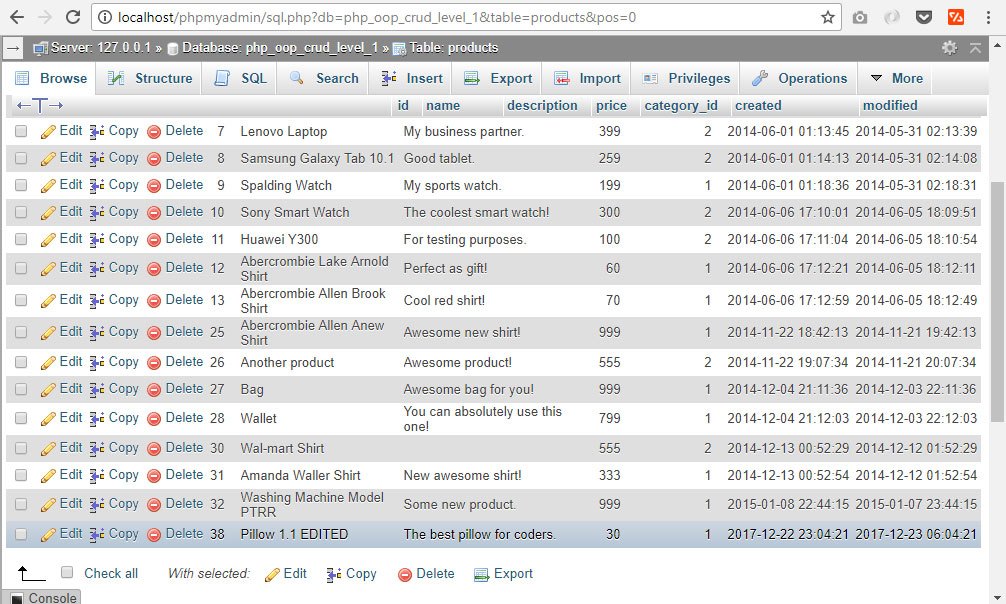
|  |
| --- |
| function update(){        $query = "UPDATE                  " . $this->table\_name . "              SET                  name = :name,                  price = :price,                  description = :description,                  category\_id  = :category\_id              WHERE                  id = :id";        $stmt = $this->conn->prepare($query);        // posted values      $this->name=htmlspecialchars(strip\_tags($this->name));      $this->price=htmlspecialchars(strip\_tags($this->price));      $this->description=htmlspecialchars(strip\_tags($this->description));      $this->category\_id=htmlspecialchars(strip\_tags($this->category\_id));      $this->id=htmlspecialchars(strip\_tags($this->id));        // bind parameters      $stmt->bindParam(':name', $this->name);      $stmt->bindParam(':price', $this->price);      $stmt->bindParam(':description', $this->description);      $stmt->bindParam(':category\_id', $this->category\_id);      $stmt->bindParam(':id', $this->id);        // execute the query      if($stmt->execute()){          return true;      }        return false;    } |

**7.9 Output**

Click any "Edit" button in the index page. The update record form should look like the following.



When you submit the form, a message will be shown.  


A record was changed in the database.  


**8.0 READ ONE RECORD IN PHP THE OOP WAY**

We previously made the code for "update record", this section for reading one record from a database will be easier to do.

**8.1 Create read\_one.php file**

This is the page where the data of a single record will be displayed. Create a new file and name it "read\_one.php", open that file and put the following code.

|  |
| --- |
| <?php  // set page headers  $page\_title = "Read One Product";  include\_once "layout\_header.php";    // read products button  echo "<div class='right-button-margin'>";      echo "<a href='index.php' class='btn btn-primary pull-right'>";          echo "<span class='glyphicon glyphicon-list'></span> Read Products";      echo "</a>";  echo "</div>";    // set footer  include\_once "layout\_footer.php";  ?> |

**8.2 Read one record based on given record ID**

The following code will read a single record from the database. Put the following code before the "set page headers" comments of the previous section.

|  |
| --- |
| // get ID of the product to be read  $id = isset($\_GET['id']) ? $\_GET['id'] : die('ERROR: missing ID.');    // include database and object files  include\_once 'config/database.php';  include\_once 'objects/product.php';  include\_once 'objects/category.php';    // get database connection  $database = new Database();  $db = $database->getConnection();    // prepare objects  $product = new Product($db);  $category = new Category($db);    // set ID property of product to be read  $product->id = $id;    // read the details of product to be read  $product->readOne(); |

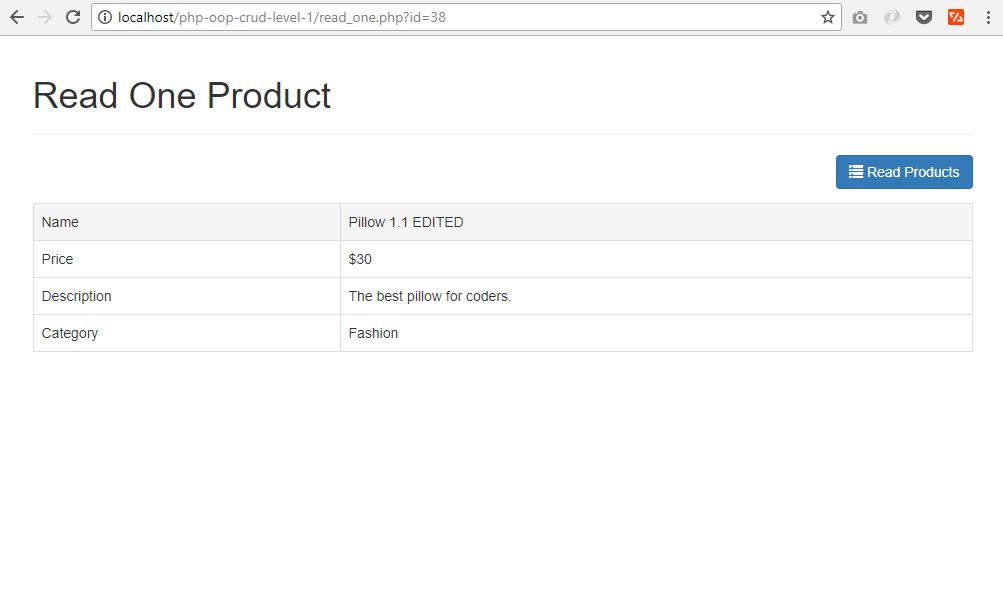
**8.3 Display record on HTML table**

This time, we will display the record details on an HTML table. Put the following code under the closing "div" tag of "Read Products" button.

|  |
| --- |
| // HTML table for displaying a product details  echo "<table class='table table-hover table-responsive table-bordered'>";        echo "<tr>";          echo "<td>Name</td>";          echo "<td>{$product->name}</td>";      echo "</tr>";        echo "<tr>";          echo "<td>Price</td>";          echo "<td>&#36;{$product->price}</td>";      echo "</tr>";        echo "<tr>";          echo "<td>Description</td>";          echo "<td>{$product->description}</td>";      echo "</tr>";        echo "<tr>";          echo "<td>Category</td>";          echo "<td>";              // display category name              $category->id=$product->category\_id;              $category->readName();              echo $category->name;          echo "</td>";      echo "</tr>";    echo "</table>"; |

**8.4 Output**

Click any "Read" button in the index page, you should see something like the image below.



**9.0 DELETING RECORD IN PHP THE OOP WAY**

This is the last coding part of our PHP OOP CRUD Tutorial. Enjoy every code!

**9.1 Put this JavaScript code in layout\_footer.php**

Put the following JavaScript code before the closing "body" tag in layout\_footer.php file. We used [Bootbox.js](http://bootboxjs.com/) to make a Bootstrap-style confirm dialog box.

|  |
| --- |
| <script>  // JavaScript for deleting product  $(document).on('click', '.delete-object', function(){        var id = $(this).attr('delete-id');        bootbox.confirm({          message: "<h4>Are you sure?</h4>",          buttons: {              confirm: {                  label: '<span class="glyphicon glyphicon-ok"></span> Yes',                  className: 'btn-danger'              },              cancel: {                  label: '<span class="glyphicon glyphicon-remove"></span> No',                  className: 'btn-primary'              }          },          callback: function (result) {                if(result==true){                  $.post('delete\_product.php', {                      object\_id: id                  }, function(data){                      location.reload();                  }).fail(function() {                      alert('Unable to delete.');                  });              }          }      });        return false;  });  </script> |

**9.2 Create delete\_product.php**

Create a new file and name it "delete\_product.php". This file accepts the ID posted by the JavaScript code in the previous section. A record will be deleted from the database based on posted ID.

Open delete\_product.php and put the following code.

|  |
| --- |
| <?php  // check if value was posted  if($\_POST){        // include database and object file      include\_once 'config/database.php';      include\_once 'objects/product.php';        // get database connection      $database = new Database();      $db = $database->getConnection();        // prepare product object      $product = new Product($db);        // set product id to be deleted      $product->id = $\_POST['object\_id'];        // delete the product      if($product->delete()){          echo "Object was deleted.";      }        // if unable to delete the product      else{          echo "Unable to delete object.";      }  }  ?> |

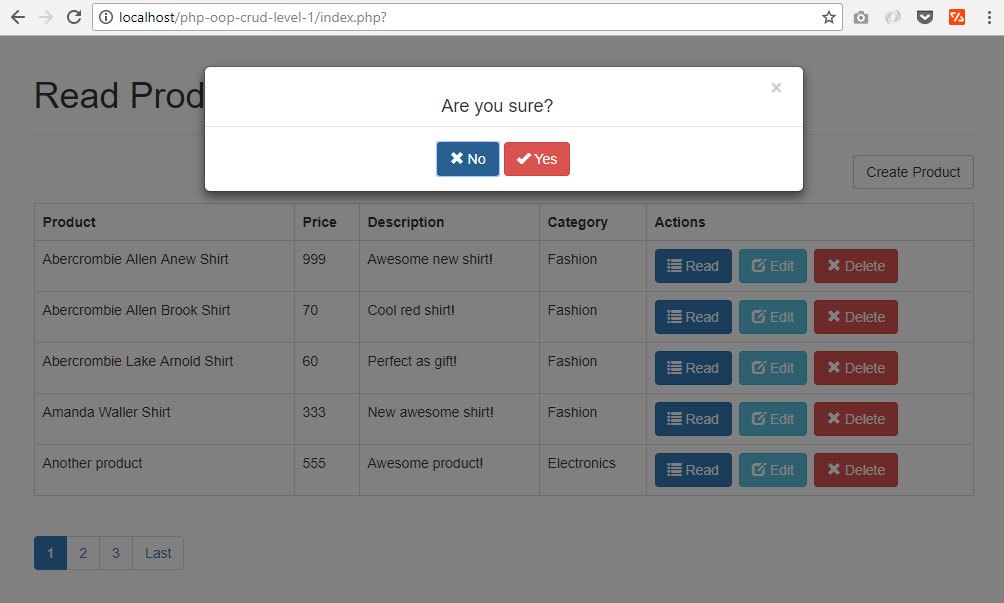
**9.3 Delete Code in Product Class**

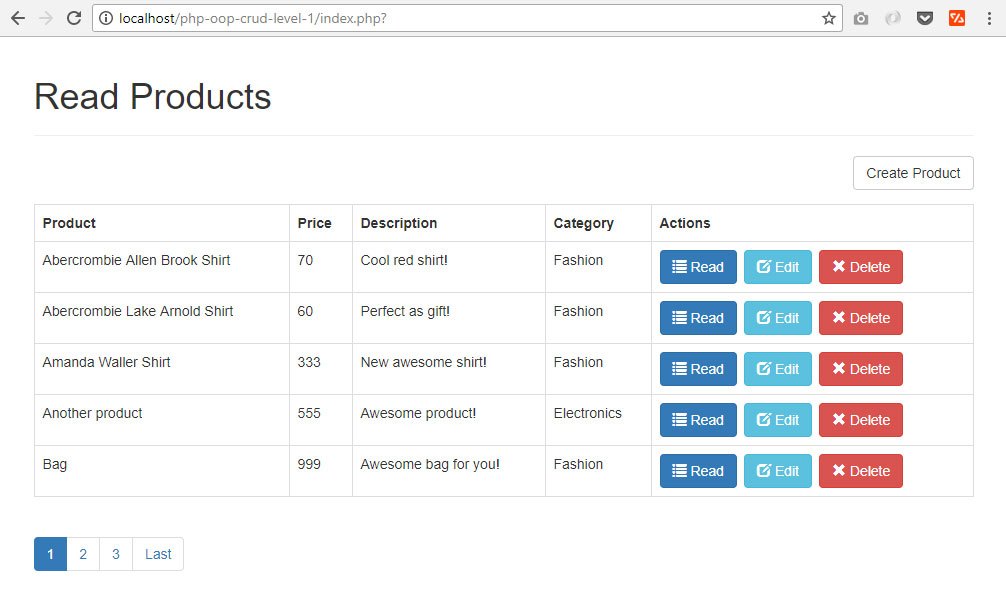
The previous section will not work with the "delete()" method in the product object. Open "product.php" which is inside the "objects" folder and put the following code.

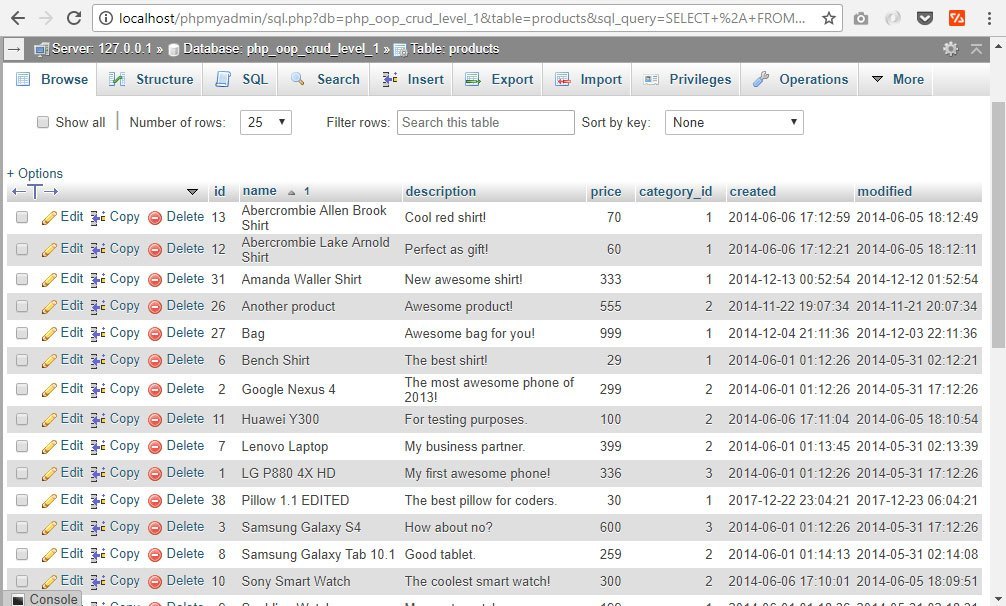
|  |
| --- |
| // delete the product  function delete(){        $query = "DELETE FROM " . $this->table\_name . " WHERE id = ?";        $stmt = $this->conn->prepare($query);      $stmt->bindParam(1, $this->id);        if($result = $stmt->execute()){          return true;      }else{          return false;      }  } |

**9.4 Output**

Click any "Delete" button in the index page. A pop up confirmation will be shown.



If the user clicks "OK" the record will be deleted and gone in the table.  


A record was deleted in the database.  


**10.0 SEARCH RECORDS IN PHP THE OOP WAY**

We'll continue by adding the search feature. This will answer the question: How to search data from database in php? This is a very useful feature because you enable your users to easily search a certain data from our MySQL database.

Please note that this is a bonus section. The code in this section is not included in our LEVEL 1 source code download.

**10.1 Change index.php**

We have to change index.php because we are adding a “search” feature and we want our code to be short. Our index.php will now look like the following code.

|  |
| --- |
| <?php  // core.php holds pagination variables  include\_once 'config/core.php';    // include database and object files  include\_once 'config/database.php';  include\_once 'objects/product.php';  include\_once 'objects/category.php';    // instantiate database and product object  $database = new Database();  $db = $database->getConnection();    $product = new Product($db);  $category = new Category($db);    $page\_title = "Read Products";  include\_once "layout\_header.php";    // query products  $stmt = $product->readAll($from\_record\_num, $records\_per\_page);    // specify the page where paging is used  $page\_url = "index.php?";    // count total rows - used for pagination  $total\_rows=$product->countAll();    // read\_template.php controls how the product list will be rendered  include\_once "read\_template.php";    // layout\_footer.php holds our javascript and closing html tags  include\_once "layout\_footer.php";  ?> |

**10.2 Create read\_template.php**

Why do we need this template? We need it because exactly the same code can be used by index.php and search.php for displaying a list of records. Using a template means lesser code.

This template holds our search form as well.

|  |
| --- |
| <?php  // search form  echo "<form role='search' action='search.php'>";      echo "<div class='input-group col-md-3 pull-left margin-right-1em'>";          $search\_value=isset($search\_term) ? "value='{$search\_term}'" : "";          echo "<input type='text' class='form-control' placeholder='Type product name or description...' name='s' id='srch-term' required {$search\_value} />";          echo "<div class='input-group-btn'>";              echo "<button class='btn btn-primary' type='submit'><i class='glyphicon glyphicon-search'></i></button>";          echo "</div>";      echo "</div>";  echo "</form>";    // create product button  echo "<div class='right-button-margin'>";      echo "<a href='create\_product.php' class='btn btn-primary pull-right'>";          echo "<span class='glyphicon glyphicon-plus'></span> Create Product";      echo "</a>";  echo "</div>";    // display the products if there are any  if($total\_rows>0){        echo "<table class='table table-hover table-responsive table-bordered'>";          echo "<tr>";              echo "<th>Product</th>";              echo "<th>Price</th>";              echo "<th>Description</th>";              echo "<th>Category</th>";              echo "<th>Actions</th>";          echo "</tr>";            while ($row = $stmt->fetch(PDO::FETCH\_ASSOC)){                extract($row);                echo "<tr>";                  echo "<td>{$name}</td>";                  echo "<td>{$price}</td>";                  echo "<td>{$description}</td>";                  echo "<td>";                      $category->id = $category\_id;                      $category->readName();                      echo $category->name;                  echo "</td>";                    echo "<td>";                        // read product button                      echo "<a href='read\_one.php?id={$id}' class='btn btn-primary left-margin'>";                          echo "<span class='glyphicon glyphicon-list'></span> Read";                      echo "</a>";                        // edit product button                      echo "<a href='update\_product.php?id={$id}' class='btn btn-info left-margin'>";                          echo "<span class='glyphicon glyphicon-edit'></span> Edit";                      echo "</a>";                        // delete product button                      echo "<a delete-id='{$id}' class='btn btn-danger delete-object'>";                          echo "<span class='glyphicon glyphicon-remove'></span> Delete";                      echo "</a>";                    echo "</td>";                echo "</tr>";            }        echo "</table>";        // paging buttons      include\_once 'paging.php';  }    // tell the user there are no products  else{      echo "<div class='alert alert-danger'>No products found.</div>";  }  ?> |

**10.3 Create core.php in "config" folder**

Create a new folder and name it "config". Inside that folder, create a new file and name it "core.php".

This file will hold our pagination variables. Using a core.php file is a good practice, it can be used to hold other configuration values that you might need in the future.

Open core.php and put the following code.

|  |
| --- |
| <?php  // page given in URL parameter, default page is one  $page = isset($\_GET['page']) ? $\_GET['page'] : 1;    // set number of records per page  $records\_per\_page = 5;    // calculate for the query LIMIT clause  $from\_record\_num = ($records\_per\_page \* $page) - $records\_per\_page;  ?> |

**10.4 Change paging.php code**

The new paging.php code will look like the following.

|  |
| --- |
| <?php  echo "<ul class=\"pagination\">";    // button for first page  if($page>1){      echo "<li><a href='{$page\_url}' title='Go to the first page.'>";          echo "First Page";      echo "</a></li>";  }    // count all products in the database to calculate total pages  $total\_pages = ceil($total\_rows / $records\_per\_page);    // range of links to show  $range = 2;    // display links to 'range of pages' around 'current page'  $initial\_num = $page - $range;  $condition\_limit\_num = ($page + $range)  + 1;    for ($x=$initial\_num; $x<$condition\_limit\_num; $x++) {        // be sure '$x is greater than 0' AND 'less than or equal to the $total\_pages'      if (($x > 0) && ($x <= $total\_pages)) {            // current page          if ($x == $page) {              echo "<li class='active'><a href=\"#\">$x <span class=\"sr-only\">(current)</span></a></li>";          }            // not current page          else {              echo "<li><a href='{$page\_url}page=$x'>$x</a></li>";          }      }  }    // button for last page  if($page<$total\_pages){      echo "<li><a href='" .$page\_url . "page={$total\_pages}' title='Last page is {$total\_pages}.'>";          echo "Last Page";      echo "</a></li>";  }    echo "</ul>";  ?> |

**10.5 Include core.php and read\_template.php**

The core.php file will be included at the beginning of index.php file. The read\_template.php will be included before the layout\_footer.php inclusion. The new index.php will look like the following code

|  |
| --- |
| <?php  // core.php holds pagination variables  include\_once 'config/core.php';    // include database and object files  include\_once 'config/database.php';  include\_once 'objects/product.php';  include\_once 'objects/category.php';    // instantiate database and product object  $database = new Database();  $db = $database->getConnection();    $product = new Product($db);  $category = new Category($db);    $page\_title = "Read Products";  include\_once "layout\_header.php";    // query products  $stmt = $product->readAll($from\_record\_num, $records\_per\_page);    // specify the page where paging is used  $page\_url = "index.php?";    // count total rows - used for pagination  $total\_rows=$product->countAll();    // read\_template.php controls how the product list will be rendered  include\_once "read\_template.php";    // layout\_footer.php holds our javascript and closing html tags  include\_once "layout\_footer.php";  ?> |

**10.6 Create search.php**

This is the most important file of this section. This file will display the records based on a user's search term.

Create a new file and name it "search.php". Open that file and put the following code.

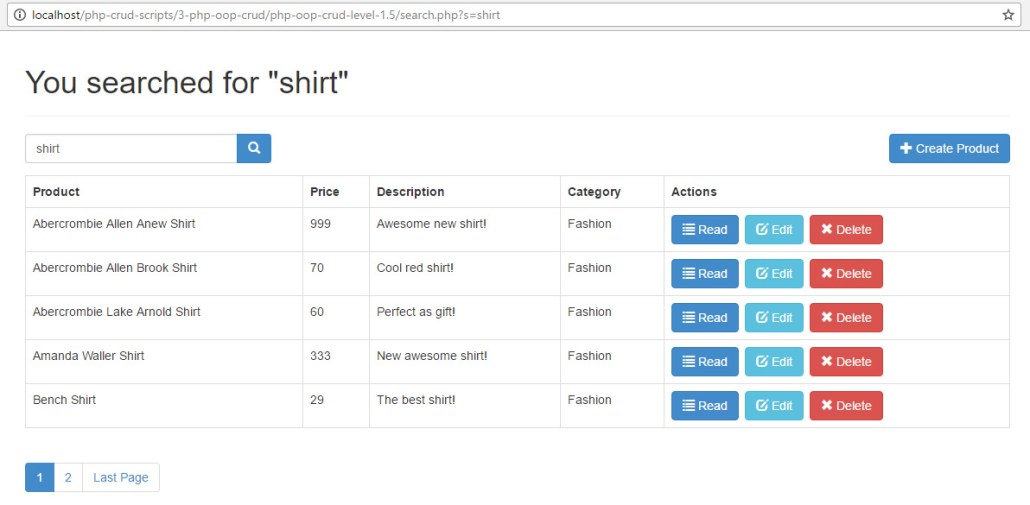
|  |
| --- |
| <?php  // core.php holds pagination variables  include\_once 'config/core.php';    // include database and object files  include\_once 'config/database.php';  include\_once 'objects/product.php';  include\_once 'objects/category.php';    // instantiate database and product object  $database = new Database();  $db = $database->getConnection();    $product = new Product($db);  $category = new Category($db);    // get search term  $search\_term=isset($\_GET['s']) ? $\_GET['s'] : '';    $page\_title = "You searched for \"{$search\_term}\"";  include\_once "layout\_header.php";    // query products  $stmt = $product->search($search\_term, $from\_record\_num, $records\_per\_page);    // specify the page where paging is used  $page\_url="search.php?s={$search\_term}&";    // count total rows - used for pagination  $total\_rows=$product->countAll\_BySearch($search\_term);    // read\_template.php controls how the product list will be rendered  include\_once "read\_template.php";    // layout\_footer.php holds our javascript and closing html tags  include\_once "layout\_footer.php";  ?> |

**10.7 Add search() and countAll\_BySearch() methods**

Open "product.php" file which is inside the "objects" folder. Add the following methods in the class.

|  |
| --- |
| // read products by search term  public function search($search\_term, $from\_record\_num, $records\_per\_page){        // select query      $query = "SELECT                  c.name as category\_name, p.id, p.name, p.description, p.price, p.category\_id, p.created              FROM                  " . $this->table\_name . " p                  LEFT JOIN                      categories c                          ON p.category\_id = c.id              WHERE                  p.name LIKE ? OR p.description LIKE ?              ORDER BY                  p.name ASC              LIMIT                  ?, ?";        // prepare query statement      $stmt = $this->conn->prepare( $query );        // bind variable values      $search\_term = "%{$search\_term}%";      $stmt->bindParam(1, $search\_term);      $stmt->bindParam(2, $search\_term);      $stmt->bindParam(3, $from\_record\_num, PDO::PARAM\_INT);      $stmt->bindParam(4, $records\_per\_page, PDO::PARAM\_INT);        // execute query      $stmt->execute();        // return values from database      return $stmt;  }    public function countAll\_BySearch($search\_term){        // select query      $query = "SELECT                  COUNT(\*) as total\_rows              FROM                  " . $this->table\_name . " p              WHERE                  p.name LIKE ? OR p.description LIKE ?";        // prepare query statement      $stmt = $this->conn->prepare( $query );        // bind variable values      $search\_term = "%{$search\_term}%";      $stmt->bindParam(1, $search\_term);      $stmt->bindParam(2, $search\_term);        $stmt->execute();      $row = $stmt->fetch(PDO::FETCH\_ASSOC);        return $row['total\_rows'];  } |

**10.8 Output**

[](https://i1.wp.com/www.codeofaninja.com/wp-content/uploads/2014/06/php-mysql-oop-crud-tutorial.jpg?ssl=1)

**11.0 FILE UPLOAD IN PHP THE OOP WAY**

In this section, we will add a "file upload" feature. This feature is included in the LEVEL 2 source code download.

**11.1 Change HTML form**

Open create\_product.php and find the "form" tag. Change that line to the following code. The "enctype" enables the form to submit a file to the server.

|  |
| --- |
| <form action="<?php echo htmlspecialchars($\_SERVER["PHP\_SELF"]);?>" method="post" enctype="multipart/form-data"> |

On the same HTML table, find the closing "tr" tag of the "Category" field. Add the following code. This adds an input field where the user can browse the file he wants to upload.

|  |
| --- |
| <tr>      <td>Photo</td>      <td><input type="file" name="image" /></td>  </tr> |

**11.2 Set value of "image" field**

Open create\_product.php and add the new "image" field. The value will be the file name of the submitted file. We used the built-in sha1\_file() function the make the file name unique.

Open create\_product.php file. Place the following code under $product->category\_id = $\_POST['category\_id']; code.

|  |
| --- |
| $image=!empty($\_FILES["image"]["name"])          ? sha1\_file($\_FILES['image']['tmp\_name']) . "-" . basename($\_FILES["image"]["name"]) : "";  $product->image = $image; |

**11.3 Change create() method**

Open "objects" folder and open the "product.php" file inside it. Find the "create()" method.

Add the "image" field by changing the query to:

|  |
| --- |
| // insert query  $query = "INSERT INTO " . $this->table\_name . "              SET name=:name, price=:price, description=:description,                  category\_id=:category\_id, image=:image, created=:created"; |

On the sanitize section, it will be:

|  |
| --- |
| $this->image=htmlspecialchars(strip\_tags($this->image)); |

Then bind the value.

|  |
| --- |
| $stmt->bindParam(":image", $this->image); |

Add the "image" property at the top of the class, maybe after public $category\_id;

|  |
| --- |
| public $image; |

Using the PhpMyAdmin, add an "image" field in the products table. Set the type to VARCHAR with 512 in length.

**11.4 Call uploadPhoto() method**

Open create\_product.php and find this line.

|  |
| --- |
| // product was created in database  echo "<div class='alert alert-success'>Product was created.</div>"; |

Put the following code under the code above. This will call the uploadPhoto() method that will try to upload the file to server.

|  |
| --- |
| // try to upload the submitted file  // uploadPhoto() method will return an error message, if any.  echo $product->uploadPhoto(); |

**11.5 Add uploadPhoto() method**

The previous section will not work without the complete code of uploadPhoto() method.

Open "objects" folder and open the "product.php" file inside it. Add the following method inside the class.

|  |
| --- |
| // will upload image file to server  function uploadPhoto(){        $result\_message="";        // now, if image is not empty, try to upload the image      if($this->image){            // sha1\_file() function is used to make a unique file name          $target\_directory = "uploads/";          $target\_file = $target\_directory . $this->image;          $file\_type = pathinfo($target\_file, PATHINFO\_EXTENSION);            // error message is empty          $file\_upload\_error\_messages="";        }        return $result\_message;  } |

**11.6 Validate submitted file**

Now we will validate the submitted file by:

* Identifying if it's a real or fake image.
* Limit the allowed file types.
* Prevent multiple file on the server.
* Deny uploading files with large file size.
* Making sure the "uploads" directory exists.

Add the following code after $file\_upload\_error\_messages=""; of the previous section.

|  |
| --- |
| // make sure that file is a real image  $check = getimagesize($\_FILES["image"]["tmp\_name"]);  if($check!==false){      // submitted file is an image  }else{      $file\_upload\_error\_messages.="<div>Submitted file is not an image.</div>";  }    // make sure certain file types are allowed  $allowed\_file\_types=array("jpg", "jpeg", "png", "gif");  if(!in\_array($file\_type, $allowed\_file\_types)){      $file\_upload\_error\_messages.="<div>Only JPG, JPEG, PNG, GIF files are allowed.</div>";  }    // make sure file does not exist  if(file\_exists($target\_file)){      $file\_upload\_error\_messages.="<div>Image already exists. Try to change file name.</div>";  }    // make sure submitted file is not too large, can't be larger than 1 MB  if($\_FILES['image']['size'] > (1024000)){      $file\_upload\_error\_messages.="<div>Image must be less than 1 MB in size.</div>";  }    // make sure the 'uploads' folder exists  // if not, create it  if(!is\_dir($target\_directory)){      mkdir($target\_directory, 0777, true);  } |

**11.7 Return error messages**

If the file is valid, we will upload the file to server. Specifically, in the "uploads" folder. If there's any error, we will return it to be shown to the user.

Place the following code after the previous section's code.

|  |
| --- |
| // if $file\_upload\_error\_messages is still empty  if(empty($file\_upload\_error\_messages)){      // it means there are no errors, so try to upload the file      if(move\_uploaded\_file($\_FILES["image"]["tmp\_name"], $target\_file)){          // it means photo was uploaded      }else{          $result\_message.="<div class='alert alert-danger'>";              $result\_message.="<div>Unable to upload photo.</div>";              $result\_message.="<div>Update the record to upload photo.</div>";          $result\_message.="</div>";      }  }    // if $file\_upload\_error\_messages is NOT empty  else{      // it means there are some errors, so show them to user      $result\_message.="<div class='alert alert-danger'>";          $result\_message.="{$file\_upload\_error\_messages}";          $result\_message.="<div>Update the record to upload photo.</div>";      $result\_message.="</div>";  } |

**11.8 Show uploaded image file**

Open "objects" folder and open "product.php" file. Find readOne() method. Add the "image" field in the method. The new method should look like the following.

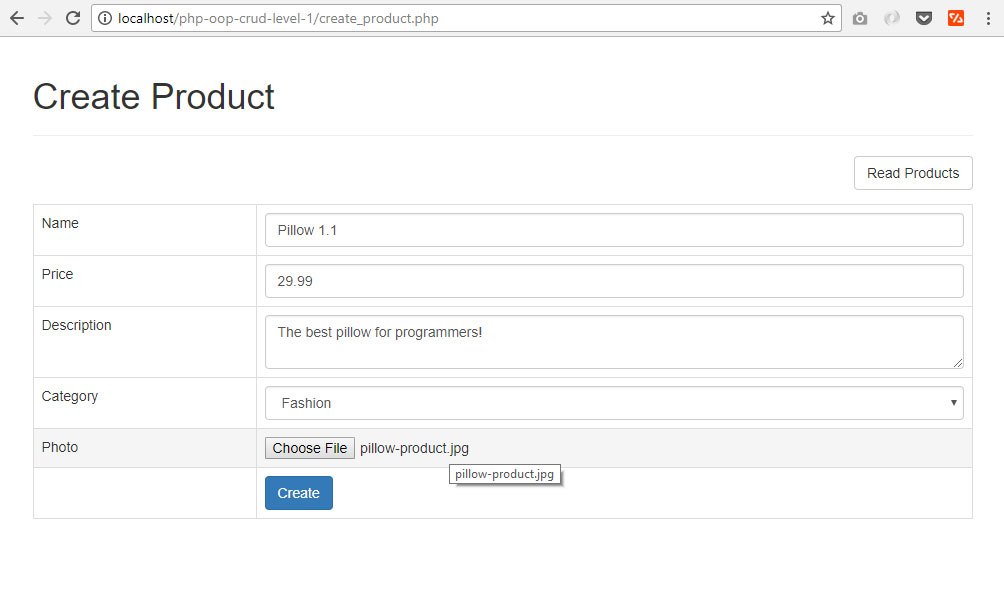
|  |
| --- |
| function readOne(){        $query = "SELECT name, price, description, category\_id, image          FROM " . $this->table\_name . "          WHERE id = ?          LIMIT 0,1";        $stmt = $this->conn->prepare( $query );      $stmt->bindParam(1, $this->id);      $stmt->execute();        $row = $stmt->fetch(PDO::FETCH\_ASSOC);        $this->name = $row['name'];      $this->price = $row['price'];      $this->description = $row['description'];      $this->category\_id = $row['category\_id'];      $this->image = $row['image'];  } |

Open read\_one.php file and find the closing "tr" tag of the "Category" field in the HTML table. Add the following code. This will show the uploaded image.

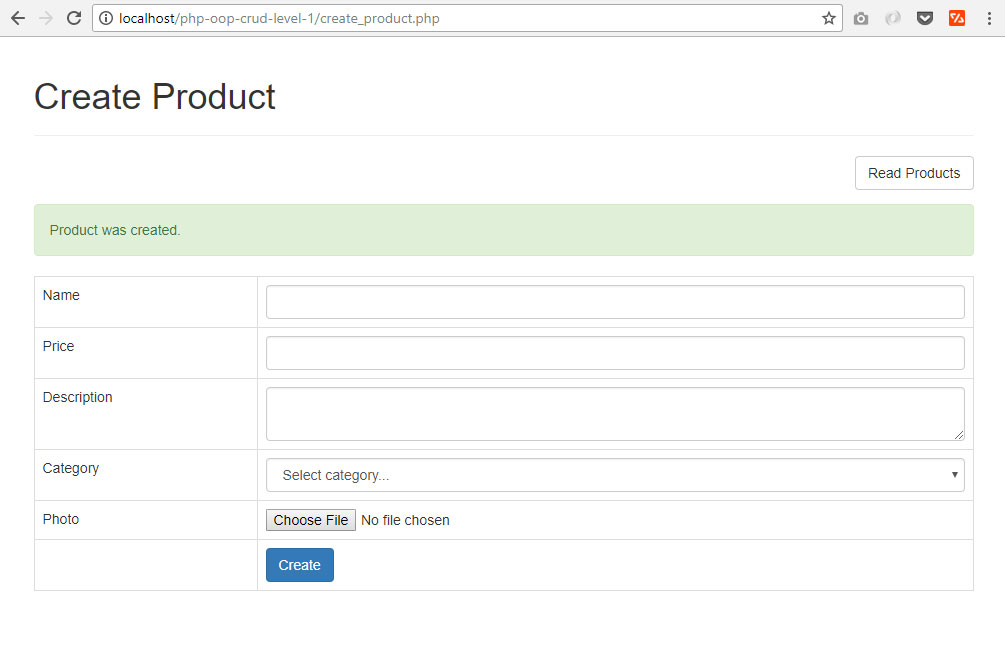
|  |
| --- |
| echo "<tr>";      echo "<td>Image</td>";      echo "<td>";          echo $product->image ? "<img src='uploads/{$product->image}' style='width:300px;' />" : "No image found.";      echo "</td>";  echo "</tr>"; |

**11.9 Output**

Click the "Create" button, you will see something like the image below.



When you submitted the form, it will show a message prompt.



If you