# Creating CRUD Application with PHP and MySQL

This document defines the homework assignments for ["PHP Web Development Basic" Course @ Software University](https://softuni.bg/trainings/2163/php-web-development-basics-september2018).

## Content:

### 1.0 Overview

### 2.0 Program output

### 3.0 Database table structure 3.1 Create a database

### 3.2 Create categories table 3.3 Insert categories sample data 3.4 Create products table 3.5 Insert products sample data 3.6 Output

### 4.0 Create the layout files 4.1 Create header layout file 4.2 Create footer layout file 4.3 Create custom CSS file

### 5.0 Creating record in PHP 5.1 Create create\_product.php file 5.2 Create "Read Products" button 5.3 Get a database connection 5.4 Create the database class 5.5 Create an HTML form 5.6 Show categories drop down 5.7 Create "categories" object 5.8 Add readName() method 5.9 Code when the form was submitted 5.10 Create "products" object 5.11 Output

### 6.0 Reading and paging record in PHP 6.1 Create index.php file 6.2 Add "Create Product" button 6.3 Configure pagination variables 6.4 Retrieve records from the database 6.5 Add readAll() method 6.6 Display data from the database 6.7 Add action buttons 6.8 Create paging.php file 6.9 Add countAll() method 6.10 Include paging.php file 6.11 Output

### 7.0 Updating record in PHP 7.1 Create update\_product.php file 7.2 Create "Read Products" button 7.3 Read one record 7.4 Add readOne() method 7.5 Put form values 7.6 Show categories dropdown 7.7 Code when form was submitted 7.8 Add update() method 7.9 Output

### 8.0 Read One Record in PHP 8.1 Create read\_one.php file 8.2 Read one record 8.3 Display record on HTML table 8.4 Output

### 9.0 Deleting record in PHP 9.1 Put this JavaScript code in layout\_footer.php 9.2 Create delete\_product.php 9.3 Add delete() method 9.4 Output

## The Project

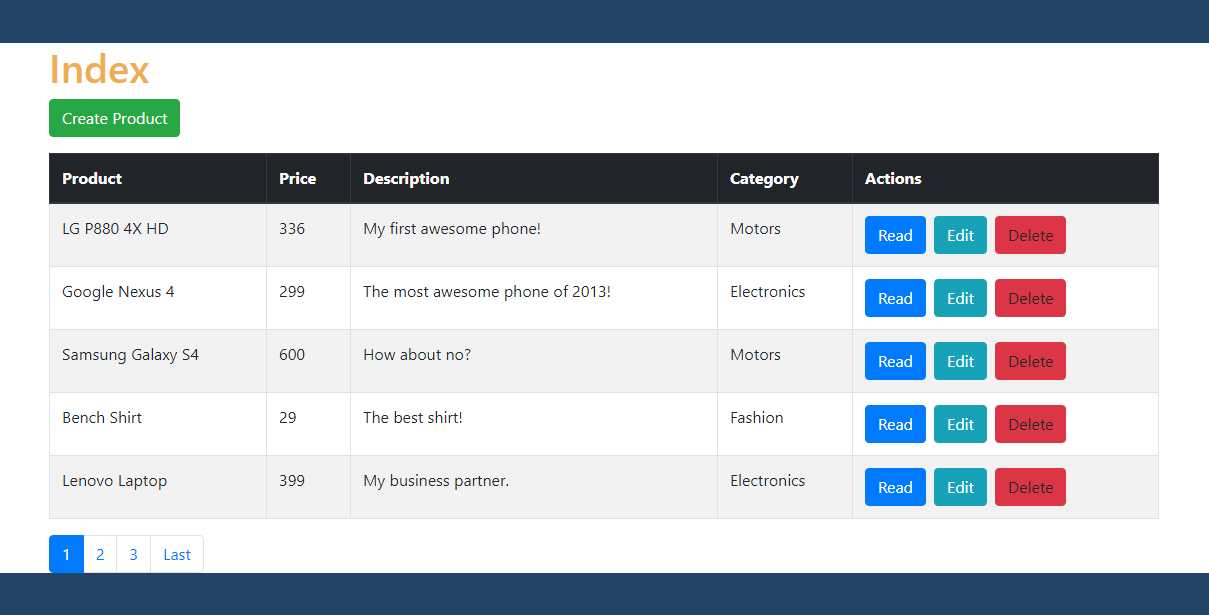
### 1.0 Overview

We use Bootstrap so that our application will have a decent UI. If you're not yet familiar what Bootstrap is, and you want to learn how to use it click here <https://getbootstrap.com/docs/4.1/getting-tarted/introduction/>.

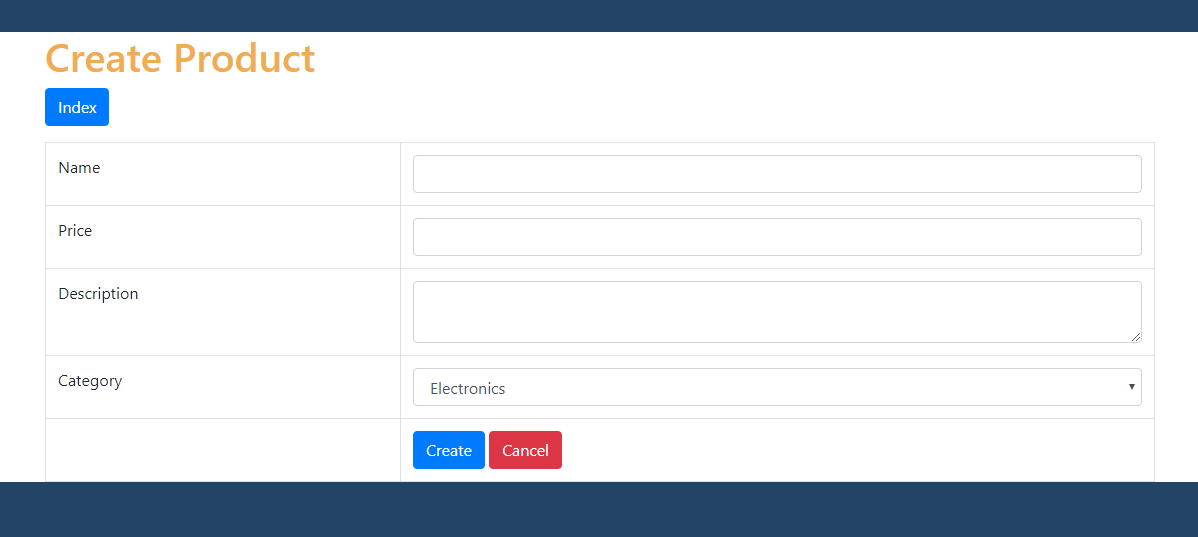
### 2.0 Program output

Below are some screenshots of our application.

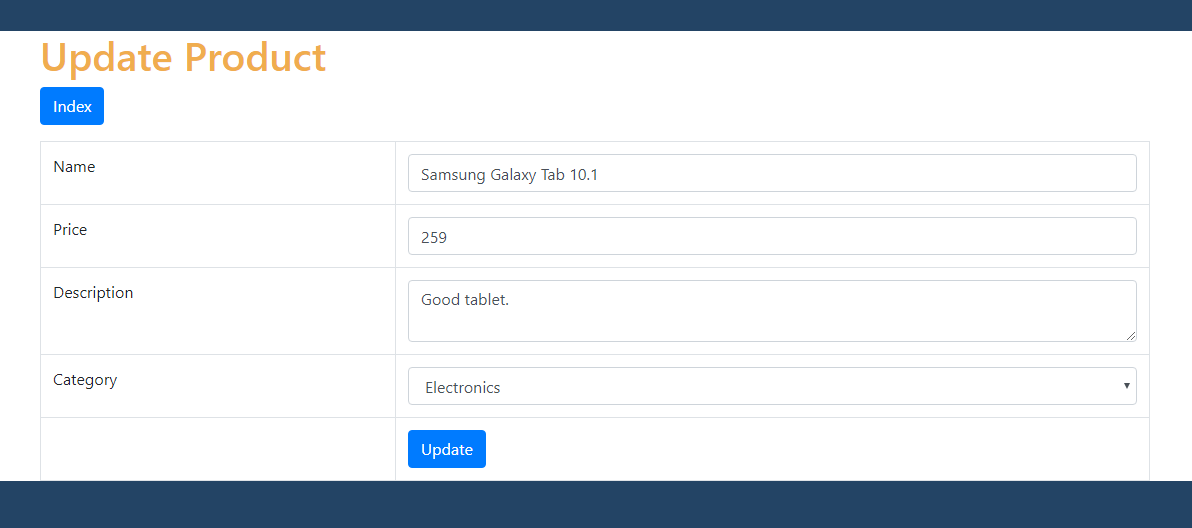
**Home Page** – index.php



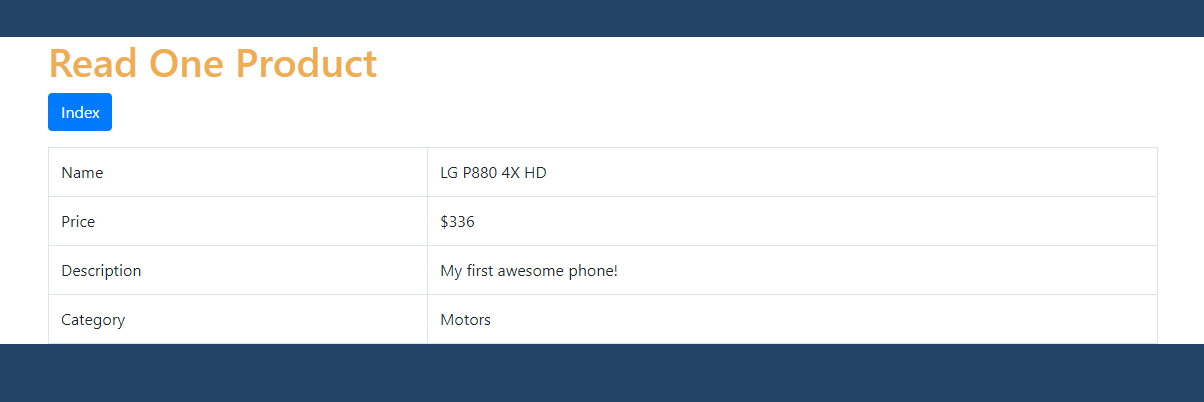
**Create Product** – create\_product.php



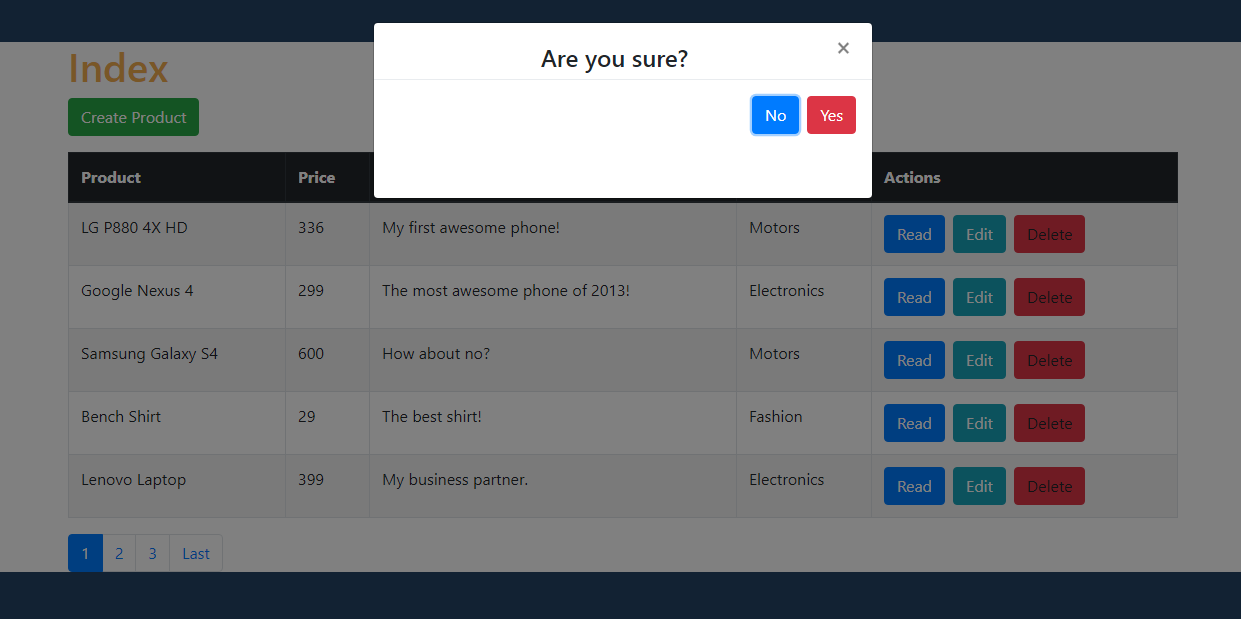
**Update Product** – update\_product.php



**Read Product** – read\_one.php



**Delete Product** – delete\_product.php



### 3.0 Database Table Structure

#### 3.1 Create a database

* Open your Xampp and start MySQL. After that open HeidiSql.
* Create a new database.
* Put php\_application as database name.
* CREATE DATABASE `php\_application` **CHARACTER** **SET** **utf8** **COLLATE** **utf8\_general\_ci**;
* Click "F9" button to execute query.

#### 3.2 Create categories table

Categories table are used to store product categories.

Run the following SQL statement using your HeidiSQL.

* Click php\_applicationdatabase.
* USE php\_application;
* Click enter.
* Copy the SQL statement below and paste it in the text area.
* Click the "F9" button.

*-- 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**=**InnoDB** **DEFAULT** **CHARSET**=**utf8** **AUTO\_INCREMENT**=4;

#### 3.3 Insert categories sample data

We are going to have "Fashion", "Electronics" and "Motors" as categories in our project.

Run the following SQL statement using your HeidiSQL.

*-- 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');

#### 3.4 Create products table

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

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

|  |  |
| --- | --- |
| *-- 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`),  **CONSTRAINT** FK\_products\_categories  **FOREIGN** **KEY** (`category\_id`) **REFERENCES** `categories`(`id`) **ON** **DELETE** **CASCADE**  ) **ENGINE**=**InnoDB** **DEFAULT** **CHARSET**=**utf8** **AUTO\_INCREMENT**=38 ; 3.5 Insert products sample data We have to put some database records.  Run the following SQL statement using your HeidiSQL.   |  | | --- | | *-- 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'); | |

#### 3.6 Output

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

Database Visualization

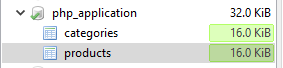


Table Categories:

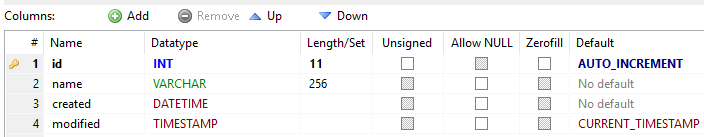
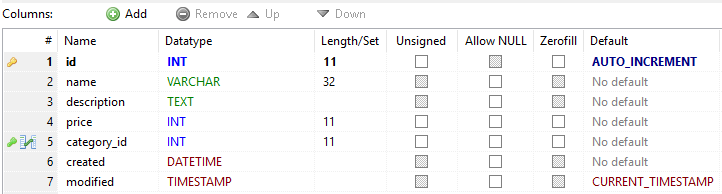


Table Products:



### Creating a root folder for application – php\_crud

### 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

We use the [Bootstrap framework](http://getbootstrap.com/) to make our project look good.

Bootstrap CSS asset will be included inside the head tags.

* Create views folder and open it.
* Create 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  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 views folder.
* Create footer.php file.
* Place the following code.

|  |
| --- |
| </**div**> *<!-- /container -->  <!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->* <**script src="https://code.jquery.com/jquery-3.3.1.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 views folder.
* Create libs folder.
* Create css folder.
* Create custom.css file.
* Place the following code.

html {

background: #234465;

}

.left-margin {

margin: 0 .5em 0 0;

}

.right-button-margin {

margin: 0 0 1em 0;

overflow: hidden;

}

.page-header{

margin-top: 3em;

color: #f0ad4e;

}

/\* some changes in bootstrap modal \*/

.modal-body {

padding: 20px 20px 0px 20px !important;

text-align: center !important;

}

.modal-footer {

margin-bottom: 3em;

text-align: center !important;

}

### 5.0 CREATING RECORD IN PHP

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

Go back to views folder, create a file with a name create\_product.phpand put the following code inside it.

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

#### 5.2 Create a "Read Products" Button

The following code will render a button. Replace the comments // contents will be hereof 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 -->* |

#### 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 objects 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.

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\_application"**;  **private $username** = **"root"**;  **private $password** = **""**;  **private $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.

|  |
| --- |
| *<!-- HTML form for creating a product -->* <**form method="post"**>   <**table class='table table-hover table-responsive table-bordered'**>   <**tr**>  <**td**>Name</**td**>  <**td**><**input type='text' name='name' class='form-control' required="required"** /></**td**>  </**tr**>   <**tr**>  <**td**>Price</**td**>  <**td**><**input type='text' name='price' class='form-control' required="required"** /></**td**>  </**tr**>   <**tr**>  <**td**>Description</**td**>  <**td**><**textarea name='description' class='form-control' required="required"**></**textarea**></**td**>  </**tr**>   <**tr**>  <**td**>Category</**td**>  <**td**>  *<!-- categories from database will be here -->* **<?php** *// read the product categories from the database* $stmt = $category->read();   *// put them in a select drop-down* **echo "<select class='form-control' name='category\_id'>"**;   **while** ($row\_category = $stmt->fetch(PDO::***FETCH\_ASSOC***)){  *extract*($row\_category);  **echo "<option value='**{$id}**'>**{$name}**</option>"**;  }   **echo "</select>"**;  **?>** </**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 objectsfolder. 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>"**;  } } **?>** |

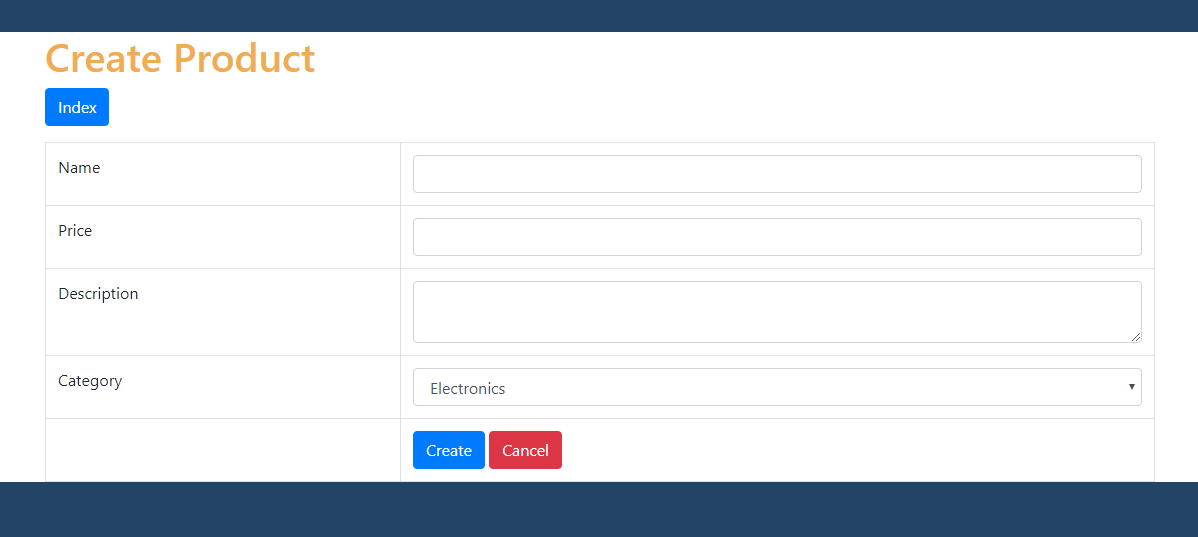
#### 5.10 Create the Object Class for Products

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.

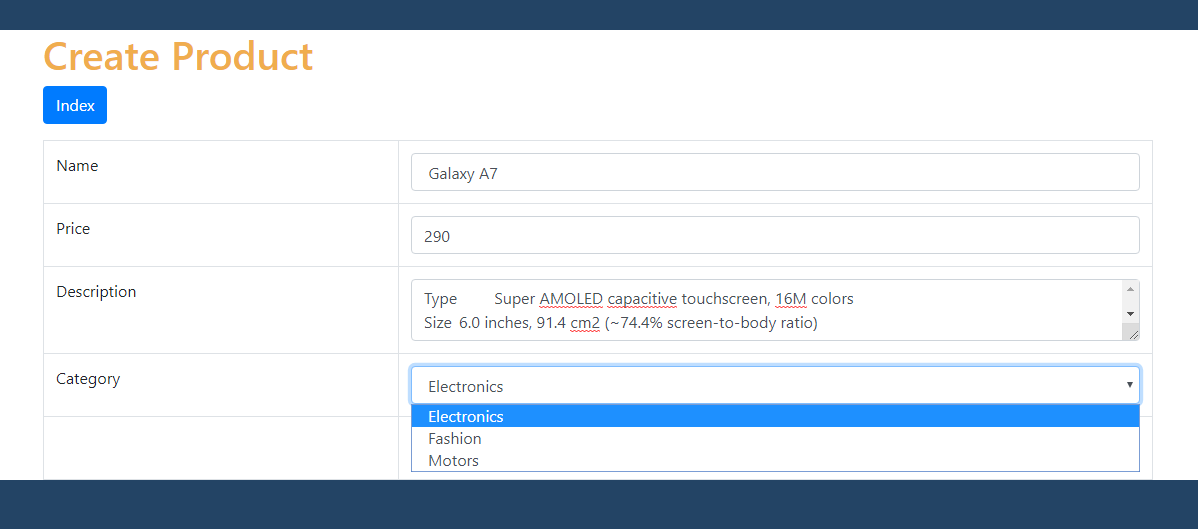
|  |
| --- |
| **<?php class** Product {   *//* ***TODO*** *// database connection and table name* **private $conn**;  **private $table\_name** = **"products"**;   *// object properties* **public $id**;  **public $name**;  **public $price**;  **public $description**;  **public $category\_id**;  **public $timestamp**;   **public function** \_\_construct($db)  {  $this->**conn** = $db;  }   *// create product* **function** create()  {   $stmt = $this->**conn**->prepare(**"INSERT INTO  "** . $this->**table\_name** . **"  SET  name = ?, price = ?, description = ?,  category\_id = ?, created = ?"**);   *// to get time-stamp for 'created' field* $this->**timestamp** = *date*(**'Y-m-d H:i:s'**);   *// bind values* $stmt->bindParam(1, $this->**name**);  $stmt->bindParam(2, $this->**price**);  $stmt->bindParam(3, $this->**description**);  $stmt->bindParam(4, $this->**category\_id**);  $stmt->bindParam(5, $this->**timestamp**);   **if** ($stmt->execute()) {  **return true**;  } **else** {  **return false**;  }  } }  **?>** |

#### 5.11 Output

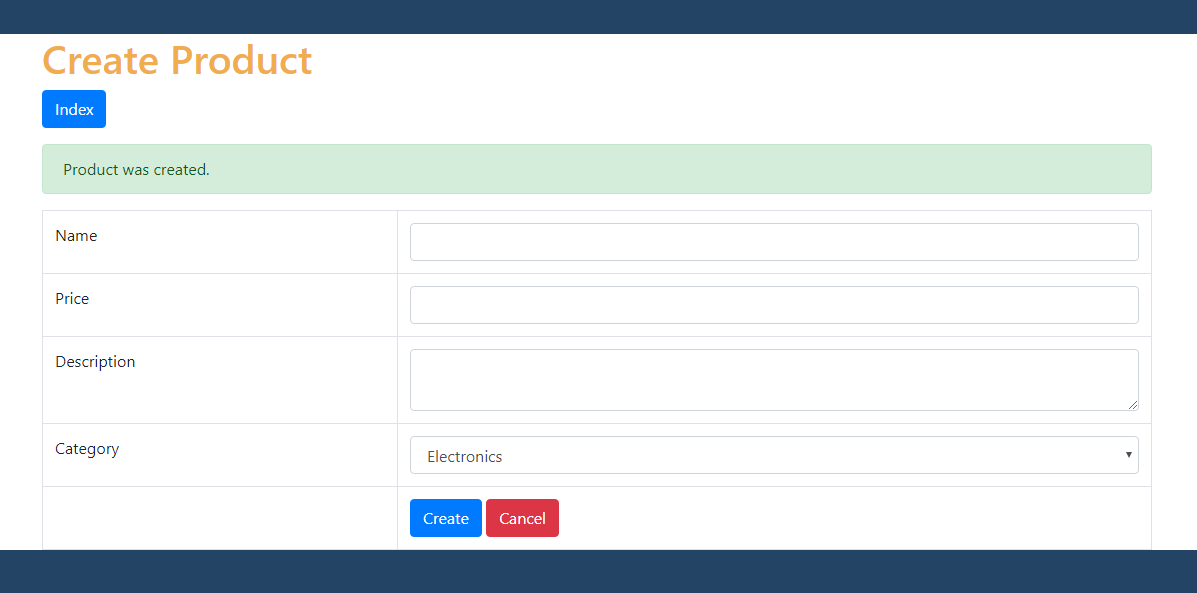
Form to create product.



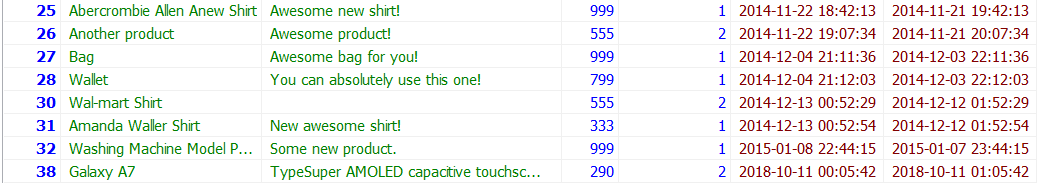
Categories drop down in the form.



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



Changes in the database.



### 6.0 READING AND PAGING RECORD IN PHP

Now we will list the records from the database.

#### 6.1 Create File: index.php

Create a new file and name it "index.php" in root directory – **php\_crud**. This file will show the main page. Put the following code inside it.

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

#### 6.2 Add a "Create Product" button

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

|  |
| --- |
| **echo "<div class='right-button-margin'>"**; **echo "<a href='views/create\_product.php'   class='btn btn-default pull-right'>Create Product</a>"**; **echo "</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 section 6.1 above.

|  |
| --- |
| *// 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; |

#### 6.4 Retrieve Records from the Database

Now we will retrieve data from the database. Put the following code after the previous section's code, befoure //contents will be here // in index.php file.

|  |
| --- |
| *// 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 product button* **echo "<a href='views/read\_one.php?id=**{$id}**' class='btn btn-primary left-margin'>"**;  **echo "<span class='glyphicon glyphicon-list'></span> Read"**;  **echo "</a>"**;  *// edit product button* **echo "<a href='views/update\_product.php?id=**{$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>"**; |

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

The following code will show our pagination buttons. Open folder **views** and 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() : int {  **return** $this->**conn**->query(**"select** *count***(***\****) from** $this->**table\_name"**)->fetchColumn(); } |

#### 6.10 Include paging.php in index.php

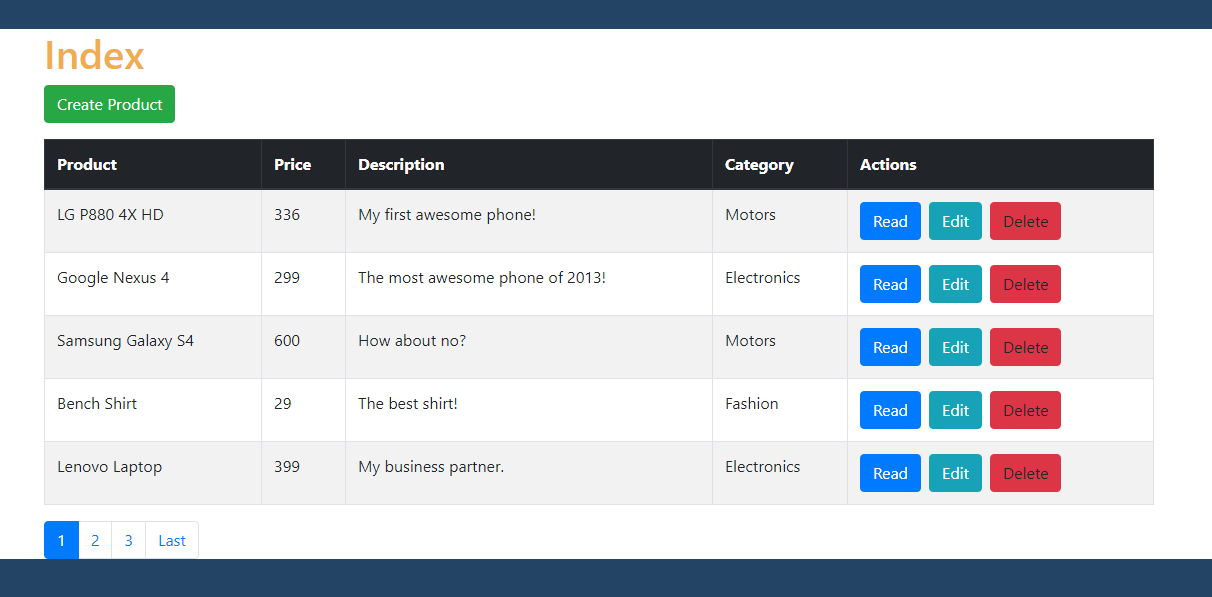
The following code will show our pagination buttons under our records list. Put the following code after the closing "table" tag of section 6.6 above.

|  |
| --- |
| *// 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 'views/paging.php'**; |

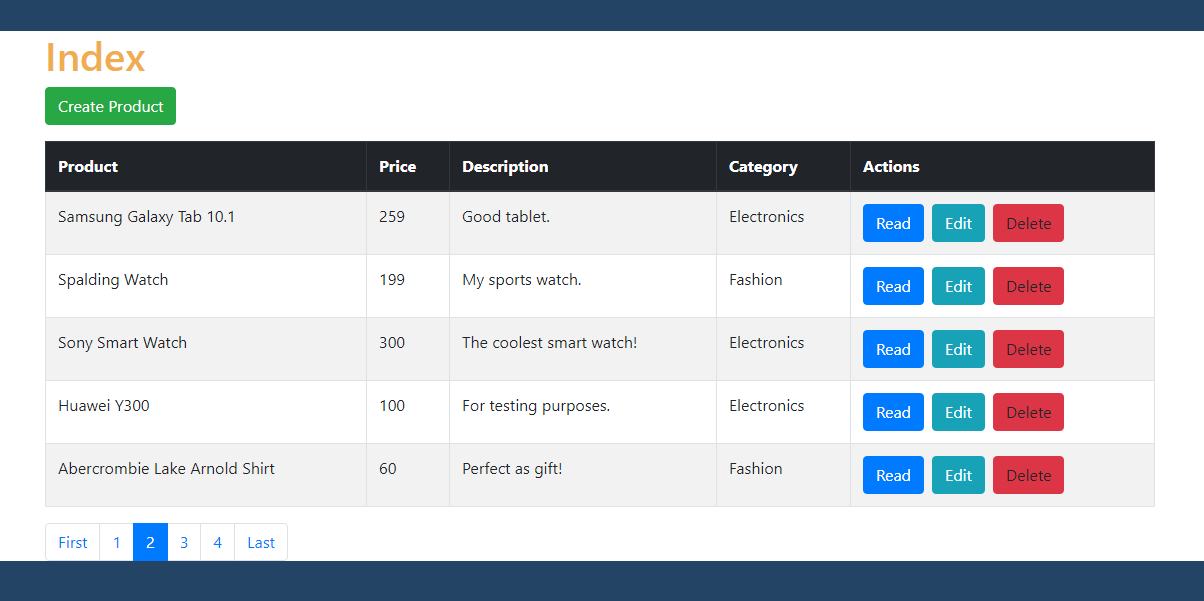
#### 6.11 Output

Run <http://localhost/php-crud/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

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

#### 7.1 Create File: update\_product.php in // views // folder

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 "../views/header.php"**;  *// contents will be here  // set page footer* **include\_once "../views/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' required="required" name='name'  value='<?php echo** $product->**name**; **?>' class='form-control'** /></**td**>  </**tr**>   <**tr**>  <**td**>Price</**td**>  <**td**><**input type='text' required="required" name='price'  value='<?php echo** $product->**price**; **?>' class='form-control'** /></**td**>  </**tr**>   <**tr**>  <**td**>Description</**td**>  <**td**><**textarea name='description' required="required" 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>"**;  } } **?>** |

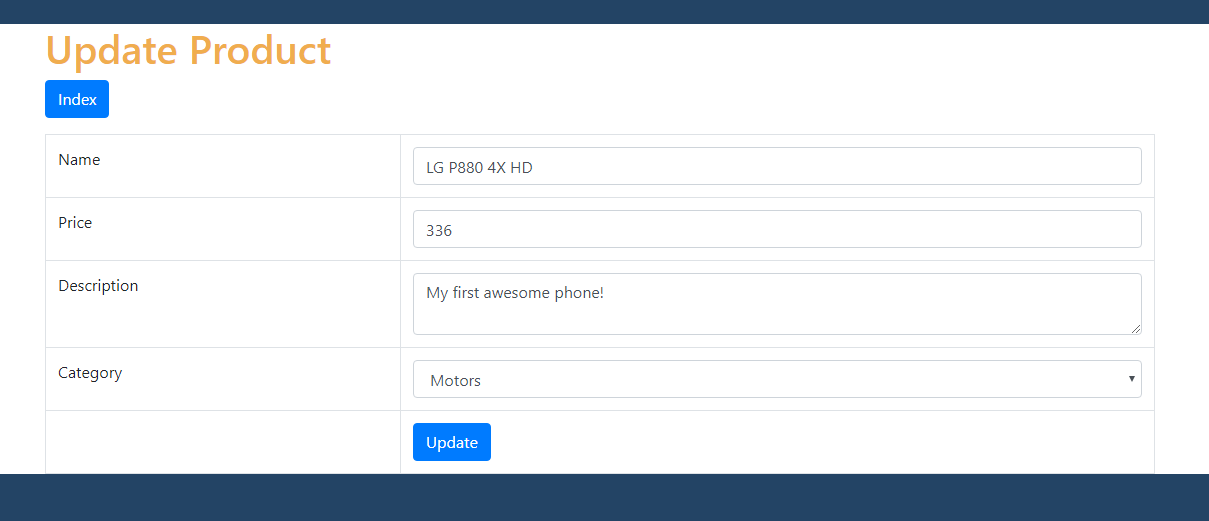
#### 7.8 Update Code in the Product Class

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.

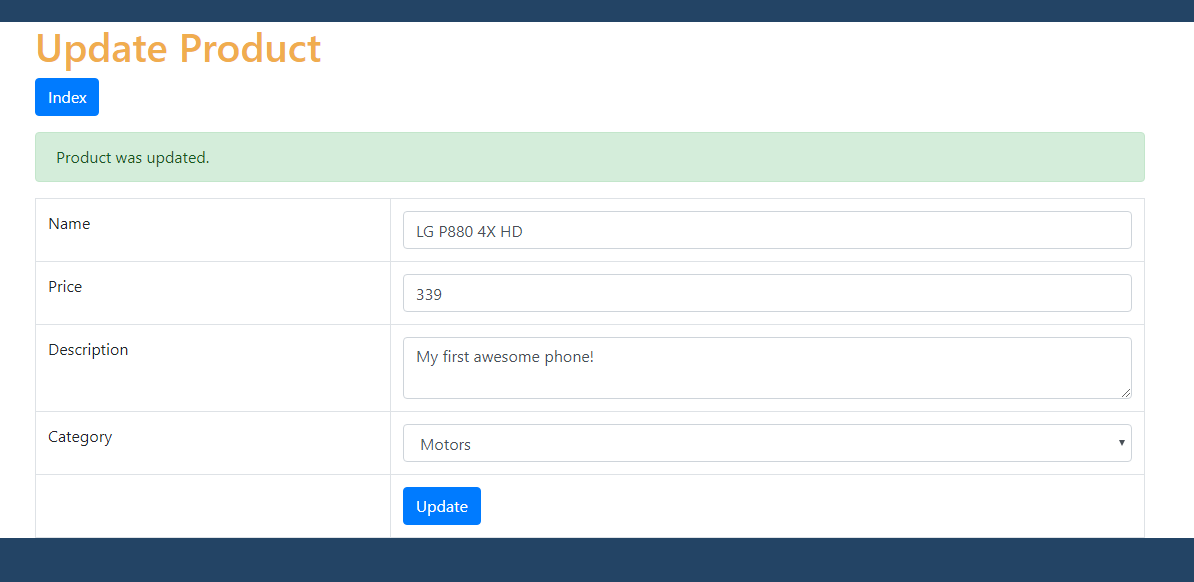
|  |
| --- |
| **function** update(){   $query = **"UPDATE  "** . $this->**table\_name** . **"  SET  name = ?,  price = ?,  description = ?,  category\_id = ?  WHERE  id = ?"**;   $stmt = $this->**conn**->prepare($query);   *// bind parameters* $stmt->bindParam(1, $this->**name**);  $stmt->bindParam(2, $this->**price**);  $stmt->bindParam(3, $this->**description**);  $stmt->bindParam(4, $this->**category\_id**);  $stmt->bindParam(5, $this->**id**);   *// execute the query* **if**($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

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. Open **views** folder and 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 "../views/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 "../views/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(); |

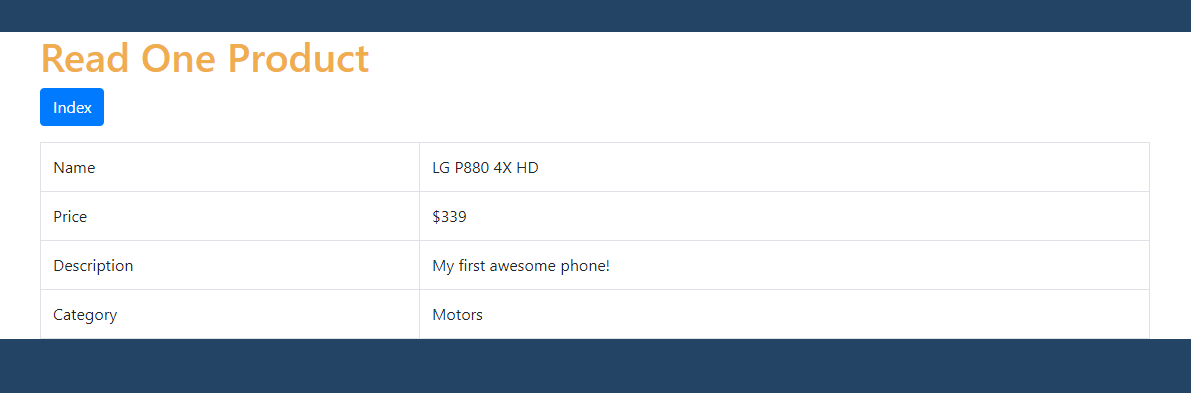
#### 8.3 Display record on HTML table

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

This is the last coding part of CRUD Application.

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

Put the following JavaScript code before the closing "body" tag in 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(**'views/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

Open folder **views** and 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."**;  } } **?>** |

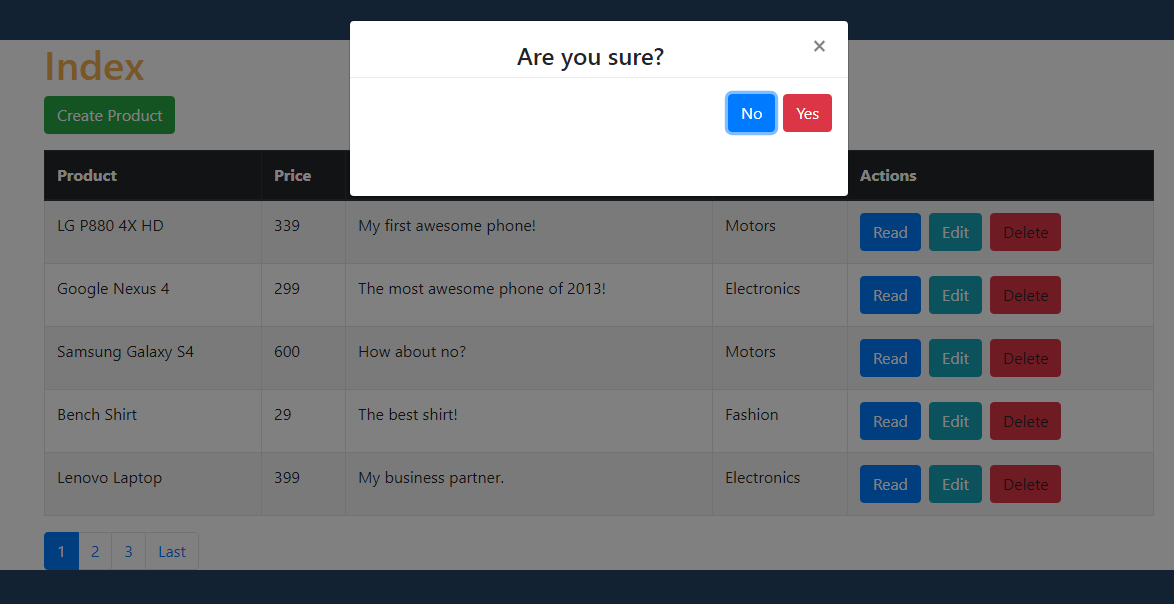
#### 9.3 Delete Code in Product Class

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.

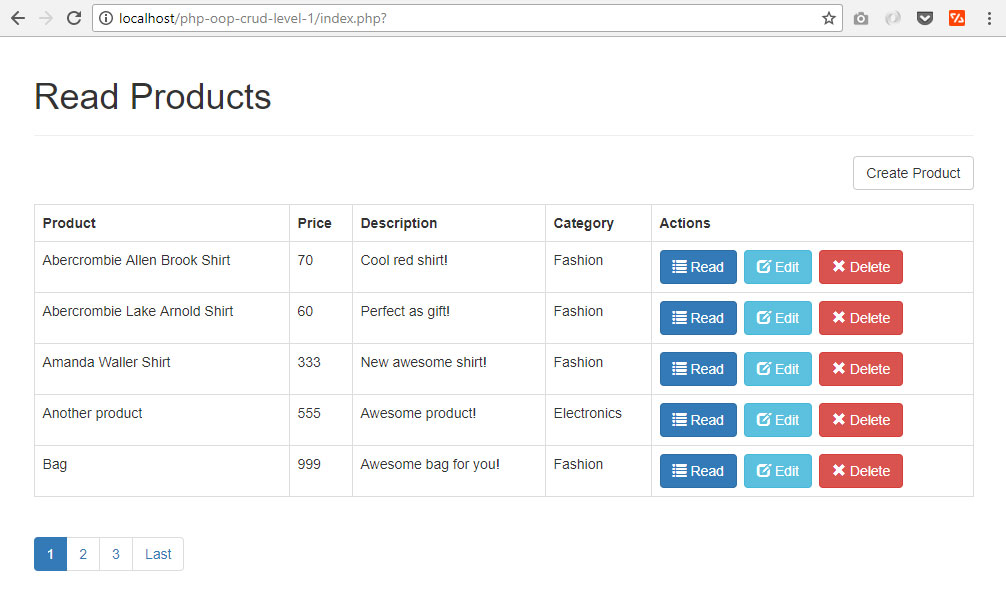
|  |
| --- |
| *// delete the product* **function** delete(){   $query = **"DELETE FROM "** . $this->**table\_name** . **" WHERE id = ?"**;   $stmt = $this->**conn**->prepare($query);  $stmt->bindParam(1, $this->**id**);   **if**($result = $stmt->execute()){  **return true**;  }**else**{  **return false**;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

#### 9.4 Output

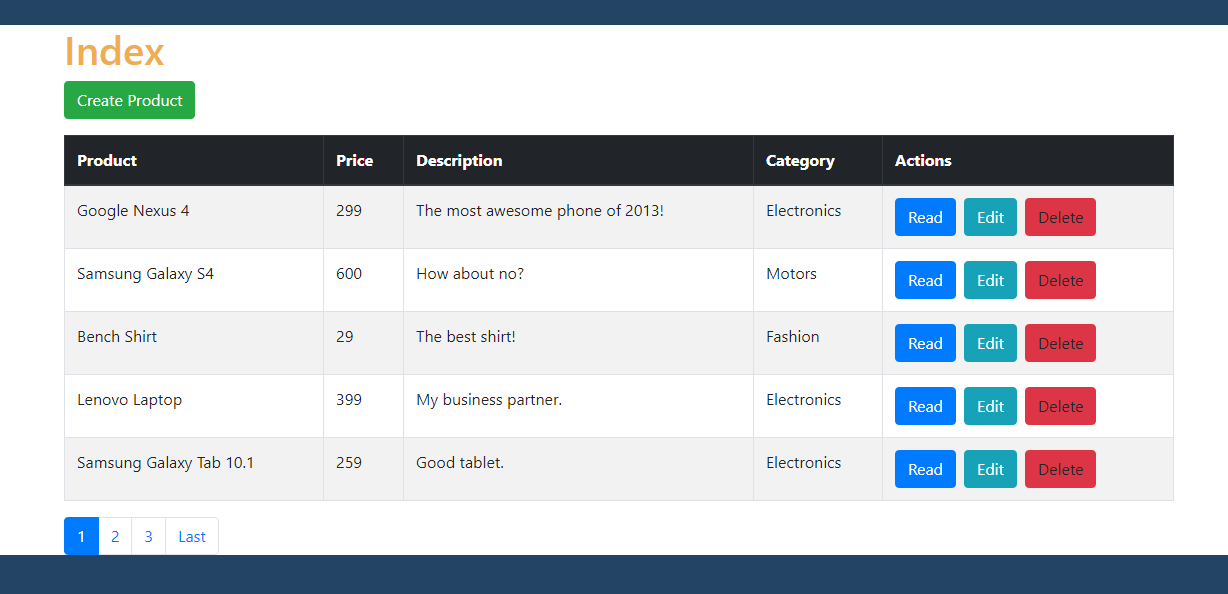
Click any "Delete" button in the index page. A pop up confirmation will be shown.



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



A record was deleted in the database.



The End ☺