CODE OF A NINJA < https://codeofaninja.com/>

PHP Shopping Cart Tutorial using MySQL database

This PHP shopping cart tutorial using MySQL database will help you create a shopping cart system using PHP and MySQL. Previously, we learned how to create, read, update and delete database records on our **PHP OOP CRUD tutorial < https://codeofaninja.com/php-oop-crud-tutorial/>** . We will apply that knowledge by building a simple PHP shopping cart application.

Contents [**hide**]

Overview

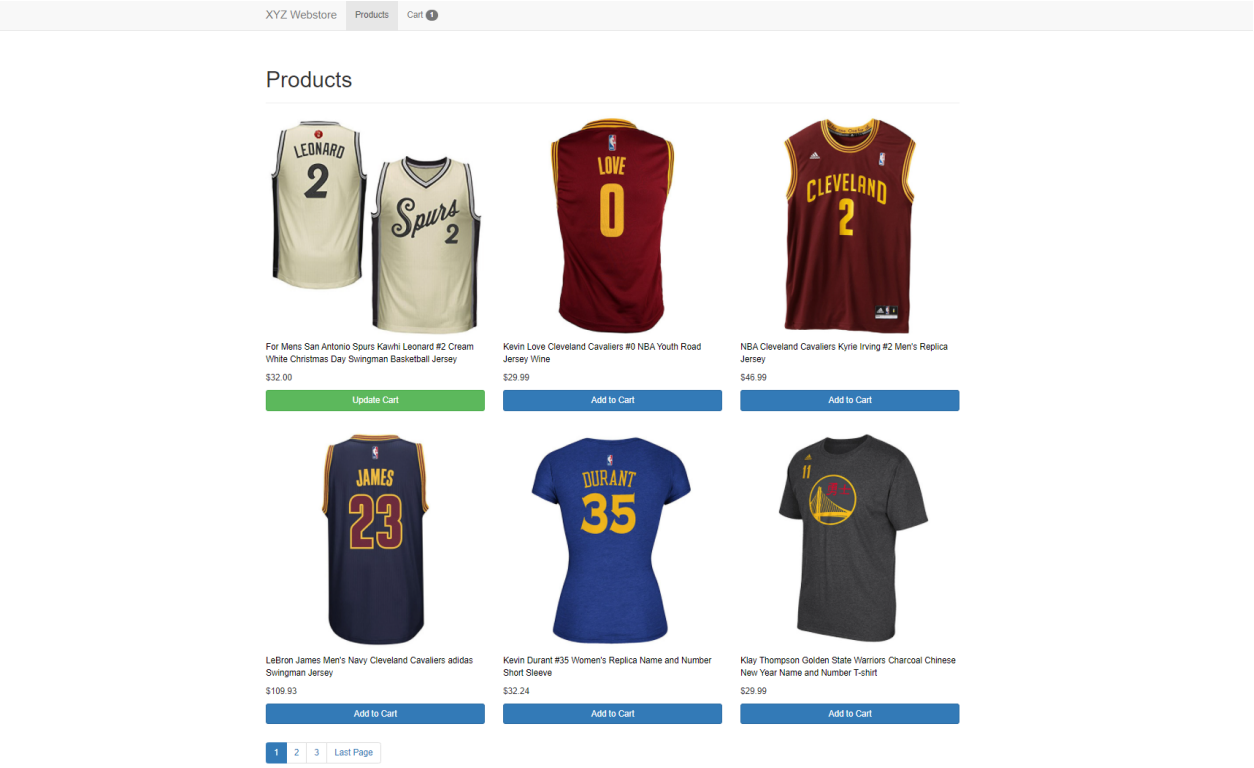
In some cases, you might need to use **cookies < https://codeofaninja.com/php-shopping-cart-tutorial-using-cookies/>** or **sessions < https://codeofaninja.com/php-shopping-cart-tutorial-using-sessions/>** for storing products added to the cart. But this tutorial focuses on using the MySQL database to store products or items in the cart.

Pros & Cons

What are the pros of this approach? The user can view the products added to his cart on any computer or device as long as this user is logged in. The customer will be reminded of products in the cart so there is a higher chance of purchases in your online store.

What are the cons of this approach? If you have thousands of users, frequent access to the MySQL database might make your site slow. To prevent this, you will need to use a **type of caching < https://developer.mozilla.org/en-US/docs/Web/HTTP/Caching>** that is not covered in this tutorial.

Screenshots



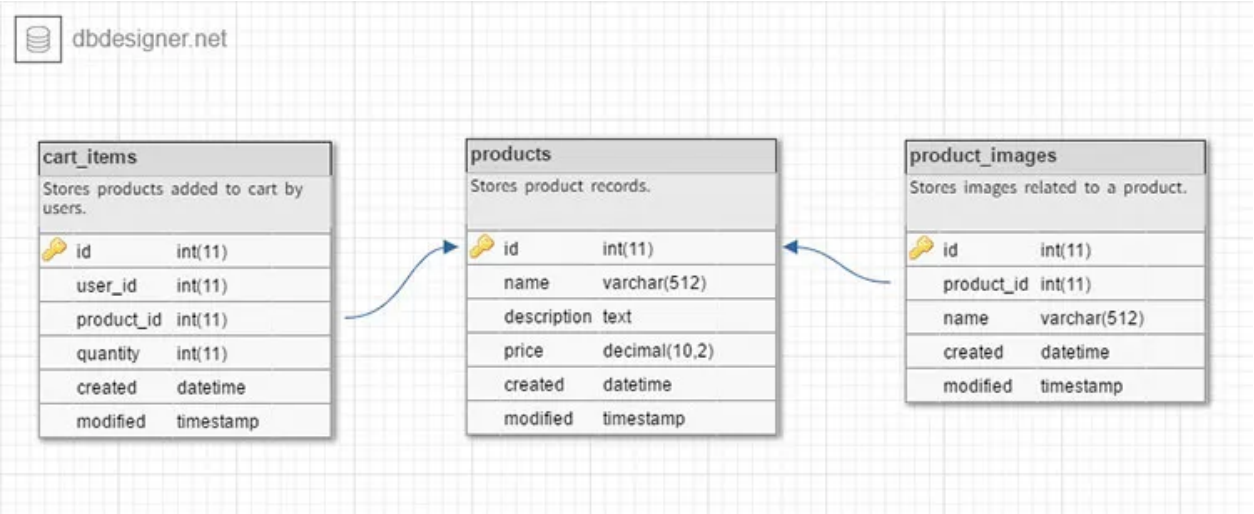
< https://i0.wp.com/codeofaninja.com/wp-content/uploads/2022/03/image-14.png?ssl=1>

PHP Shopping Cart Tutorial using MySQL database

Prepare the database

Database Design

Our database name is “php_shopping_cart”, and we will have three (3) tables.



< https://i0.wp.com/www.codeofaninja.com/wp-content/uploads/2015/08/database-design.jpg?ssl=1>

Create MySQL Tables

I’m using PhpMyAdmin to run the following queries on our “php_shopping_cart” database.

Create the **products** table. This table will store the products records.

```
CREATE TABLE IF NOT EXISTS `products` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(512) NOT NULL,
  `description` text NOT NULL,
  `price` decimal(10,2) NOT NULL,
  `created` datetime NOT NULL,
  `modified` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
) ENGINE=MyISAM  DEFAULT CHARSET=latin1 COMMENT='products
```

Create **product_images** table. This table will hold images related to the product.

```
CREATE TABLE IF NOT EXISTS `product_images` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `product_id` int(11) NOT NULL,
  `name` varchar(512) NOT NULL,
  `created` datetime NOT NULL,
  `modified` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP
  PRIMARY KEY (`id`)
) ENGINE=MyISAM  DEFAULT CHARSET=utf8 COMMENT='image files
```

Create **cart_items** table. This table will hold cart items of our user.

```
CREATE TABLE IF NOT EXISTS `cart_items` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `product_id` int(11) NOT NULL,
  `quantity` double NOT NULL,
  `user_id` int(11) NOT NULL,
  `created` datetime NOT NULL,
  `modified` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP
  PRIMARY KEY (`id`)
) ENGINE=MyISAM  DEFAULT CHARSET=utf8 AUTO_INCREMENT=40 ;
```

Download sample data

The **products** and **product_images** table will not fully work without the sample products data and images in the "/uploads/images/" directory.

You'll have to download the SQL and image files. Use the download link below. What will you get? A **ZIP file** with **products_data.sql** and **28 image files** inside (1.30 MB).

> Download SQL and image files <
> https://www.dropbox.com/sh/tspkfx1ovo72xyr/AADMz0Mm9JbupeJ0DFsOP9S6a?dl=1>

Once downloaded, import the SQL file using PhpMyAdmin. Put the image files in the “/uploads/images/” directory.

Database connection file

Create the “config” folder and inside it, create the “database.php” file with the following code. This code will enable us to connect to our database and do database operations such as adding products to the cart or removing products from our shopping cart.

```
<?php
// used to get mysql database connection
class Database{

    // specify your own database credentials
    private $host = "localhost";
    private $db_name = "php_shopping_cart";
    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->h
        }catch(PDOException $exception){
            echo "Connection error: " . $exception->getMes
        }

        return $this->conn;
    }

}
?>
```

Prepare the user interface

We are about to create the layout files. Layout files contain reusable code that we can use on each of our web pages. This will help us have a consistent user interface.

Create header layout file

Create layout_head.php file and place the following code. We have some PHP code inside the title tags because we want the page title to be dynamic.

We're using Boostrap to make our user interface look better. We also have some custom CSS.

```
<!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, ini

    <title><?php echo isset($page_title) ? $page_title : "

    <!-- Latest compiled and minified Bootstrap CSS -->
    <link rel="stylesheet" href="https://maxcdn.bootstrapc

    <!-- custom css for users -->
    <link href="libs/css/custom.css" rel="stylesheet" medi

</head>
<body>

    <?php include 'navigation.php'; ?>

    <!-- container -->
    <div class="container">
        <div class="row">

        <div class="col-md-12">
            <div class="page-header">
                <h1><?php echo isset($page_title) ? $page_
            </div>
        </div>
```

Create footer layout file

Create `layout_foot.php` file and place the following code. The script tags are needed by Bootstrap to function.

```
        </div>
        <!-- /row -->

    </div>
    <!-- /container -->

<!-- jQuery (necessary for Bootstrap's JavaScript plugins)
<script src="https://code.jquery.com/jquery-3.2.1.min.js <

<!-- Latest compiled and minified Bootstrap JavaScript -->
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3

<!-- custom script will be here -->

</body>
</html>
```

Create navigation.php

Create `navigation.php` file and put the following code inside. The navigation bar will contain links to the list of products and cart items.

```
<!-- navbar -->
<div class="navbar navbar-default navbar-static-top" role=
    <div class="container">

        <div class="navbar-header">
            <button type="button" class="navbar-toggle" da
            <span class="sr-only">Toggle navigation</span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
            </button>
            <a class="navbar-brand" href="products.php">XY
        </div>

        <div class="navbar-collapse collapse">
            <ul class="nav navbar-nav">

                <!-- highlight if $page_title has 'Product
                <li <?php echo strpos($page_title, "Produc
                    <a href="products.php">Products</a>
                </li>

                <li <?php echo $page_title=="Cart" ? "clas
                    <a href="cart.php">
                        <!--later, we'll put a PHP code he
                        Cart <span class="badge" id="compa
                    </a>
                </li>
            </ul>

        </div><!--/.nav-collapse -->

    </div>
</div>
<!-- /navbar -->
```

Add Custom CSS

Create the “libs” folder. And inside it, create a “css” folder. Create a custom.css file with the following code inside it. Custom CSS are needed to make our web page look better and more user-friendly.

```
.text-align-center{ text-align:center; }
.f-w-b{ font-weight:bold; }
.display-none{ display:none; }

.w-5-pct{ width:5%; }
.w-10-pct{ width:10%; }
.w-15-pct{ width:15%; }
.w-20-pct{ width:20%; }
.w-25-pct{ width:25%; }
.w-30-pct{ width:30%; }
.w-35-pct{ width:35%; }
.w-40-pct{ width:40%; }
.w-45-pct{ width:45%; }
.w-50-pct{ width:50%; }
.w-55-pct{ width:55%; }
.w-60-pct{ width:60%; }
.w-65-pct{ width:65%; }
.w-70-pct{ width:70%; }
.w-75-pct{ width:75%; }
.w-80-pct{ width:80%; }
.w-85-pct{ width:85%; }
.w-90-pct{ width:90%; }
.w-95-pct{ width:95%; }
.w-100-pct{ width:100%; }

.m-t-0px{ margin-top:0px; }
.m-b-10px{ margin-bottom:10px; }
.m-b-20px{ margin-bottom:20px; }
.m-b-30px{ margin-bottom:30px; }
.m-b-40px{ margin-bottom:40px; }

.stock-text {
    font-weight: bold;
    color: #008a00;
}

.stock-text-red{
    font-weight:bold;
    color:#b12704;
}

.product-detail {
    font-weight: bold;
    margin: 0 0 5px 0;
}
```

```
.blueimp-gallery>.prev, .blueimp-gallery>.next{ border:non

.update-quantity-form {
    width: 150px;
    float: left;
    margin: 0 10px 0 0;
}

.cart-row {
    border-bottom: thin solid #f1f1f1;
    overflow: hidden;
    width: 100%;
    padding: 20px 0 20px 0;
}

.product-link{
    color:#000000;
}

.product-link:hover{
    color:#000000;
    text-decoration:none;
}

.product-img-thumb {
    margin: 0 0 10px 0;
    width: 100%;
    cursor: pointer;
}
```

Display products with pagination

Create products.php

Now we are going to start displaying products from the database. Create `products.php` with the following basic code. This is one instance where we use our layout files. At the top we specify our page title.

```php
<?php
// set page title
$page_title="Products";

// page header html
include 'layout_head.php';

// contents will be here

// layout footer code
include 'layout_foot.php';
?>
```

Include PHP Classes

Put this code after the opening “php” tag of products.php file. This code will enable us to connect to the database and then do database operations on specific tables.

```php
// for database connection
include 'config/database.php';

// include objects
include_once "objects/product.php";
include_once "objects/product_image.php";
include_once "objects/cart_item.php";

// database connection request will be here
```

Create “product” object file

We’re going to create the “object” files. We have three objects: products, product images, and cart items.

Create “`objects`” folder. Inside it, create `product.php` file. Use the following code. This file will handle our queries on the products table.

```
```

```php
<?php
// 'product' object
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 $category_name;
    public $timestamp;

    // constructor
    public function __construct($db){
        $this->conn = $db;
    }
}
```

Create “product image” object file

Create `product_image.php` file inside “objects” folder. This file will handle our queries on the product_images table.

```php
<?php
// 'product image' object
class ProductImage{

    // database connection and table name
    private $conn;
    private $table_name = "product_images";

    // object properties
    public $id;
    public $product_id;
    public $name;
    public $timestamp;

    // constructor
    public function __construct($db){
        $this->conn = $db;
    }
}
```

Create “cart item” object file

Create `cart_item.php` file inside “objects” folder. This file will handle our queries on the cart_items table.

```php
<?php
// 'cart item' object
class CartItem{

    // database connection and table name
    private $conn;
    private $table_name = "cart_items";

    // object properties
    public $id;
    public $product_id;
    public $quantity;
    public $user_id;
    public $created;
    public $modified;

    // constructor
    public function __construct($db){
        $this->conn = $db;
    }
}
```

Connect to the database

Open products.php file. Replace "`// database connection request will be here`" comment in `products.php` file with the following code.

This code will request a database connection and initialize the objects we created earlier.

```php
// get database connection
$database = new Database();
$db = $database->getConnection();

// initialize objects
$product = new Product($db);
$product_image = new ProductImage($db);
$cart_item = new CartItem($db);

// action and pagination will be here
```

Initialize action and pagination

We're going to initialize some variables using the following code.

The $action variable will be used to display custom prompt messages such as "Added to cart!" or "Removed from cart."

The pagination variables will be used to control the pagination of products list.

Replace "// action and pagination will be here" comment with the following code.

```
// action for custom messages
$action = isset($_GET['action']) ? $_GET['action'] : "";

// for pagination purposes
$page = isset($_GET['page']) ? $_GET['page'] : 1; // page
$records_per_page = 6; // set records or rows of data per
$from_record_num = ($records_per_page * $page) - $records_
```

Request data from the database

This code will display the list of products from the database. If there are no products, it will display a "No products found." message.

Replace "// contents will be here" comment in products.php with the following code.

```php
// read all products in the database
$stmt=$product->read($from_record_num, $records_per_page);

// count number of retrieved products
$num = $stmt->rowCount();

// if products retrieved were more than zero
if($num>0){
    // needed for paging
    $page_url="products.php?";
    $total_rows=$product->count();

    // show products
    include_once "read_products_template.php";
}

// tell the user if there's no products in the database
else{
    echo "<div class='col-md-12'>
        <div class='alert alert-danger'>No products found.
    </div>";
}
```

Add “read” method

We’re using the “read” and “count” methods of the product object but they do not exist yet. Open “`objects/product.php`” file.

Put the following “read” method. This code contains the actual query to retrieve the list of products from the products database.

```php
// read all products
function read($from_record_num, $records_per_page)
{

    // select all products query
    $query = "SELECT id, name, description, price
            FROM " . $this->table_name . "
            ORDER BY created DESC
            LIMIT ?, ?";

    // prepare query statement
    $stmt = $this->conn->prepare($query);

    // bind limit clause variables
    $stmt->bindParam(1, $from_record_num, PDO::PARAM_INT);
    $stmt->bindParam(2, $records_per_page, PDO::PARAM_INT)

    // execute query
    $stmt->execute();

    // return values
    return $stmt;
}
```

Add “count” method

Next, put the following count method. This code contains a query to count the total number of products in the database. It will be used for pagination.

```php
// used for paging products
public function count(){

    // query to count all product records
    $query = "SELECT count(*) FROM " . $this->table_name;

    // prepare query statement
    $stmt = $this->conn->prepare( $query );

    // execute query
    $stmt->execute();

    // get row value
    $rows = $stmt->fetch(PDO::FETCH_NUM);

    // return count
    return $rows[0];
}
```

Template to display products

Let's create a template to display the products. Create "`read_products_template.php`" file. We will use the following code.

In this code, we loop through the list of products retrieved from the database so we can display them. On each loop of the product, we have a query to read and display a product image.

We default the user ID to "1" for now because this code is not yet integrated with a "user login" code like what we have in our **PHP Login Script with Session tutorial < https://codeofaninja.com/php-login-script/>** .

We will also display the product's name, price, category name, and the "Add to cart" button. If the item exists in our cart, we'll display an "Update cart" button instead.

```php
<?php
while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
    extract($row);

    // creating box
    echo "<div class='col-md-4 m-b-20px'>";

    echo "<a href='product.php?id={$id}' class='product-li

    // select and show first product image
    $product_image->product_id = $id;
    $stmt_product_image = $product_image->readFirst();
    while ($row_product_image = $stmt_product_image->fetch
        echo "<div class='m-b-10px'>
                    <img src='uploads/images/{$row_product
                </div>";
    }

    // product name
    echo "<div class='product-name m-b-10px'>{$name}</div>
        </a>";

    // product price and category name
    echo "<div class='m-b-10px'>$" . number_format($price,

    // add to cart button
    echo "<div class='m-b-10px'>";

    // cart item settings
    $cart_item->user_id = 1; // we default to a user with
    $cart_item->product_id = $id;

    // if product was already added in the cart
    if ($cart_item->exists()) {
        echo "<a href='cart.php' class='btn btn-success w-
    } else {
        echo "<a href='add_to_cart.php?id={$id}&page={$pag
    }

    echo "</div>";

    echo "</div>";
}

include_once "paging.php";
```

Add “readFirst()” method

We used the readFirst() method of the product_image object but it does not exist yet.

Open “objects/product_image.php” file. Put the following code. This code will read the first product image of a specific product.

```
// read the first product image related to a product
function readFirst(){

    // select query
    $query = "SELECT id, product_id, name
            FROM " . $this->table_name . "
            WHERE product_id = ?
            ORDER BY name DESC
            LIMIT 0, 1";

    // prepare query statement
    $stmt = $this->conn->prepare( $query );

    // sanitize
    $this->id=htmlspecialchars(strip_tags($this->id));

    // bind prodcut id variable
    $stmt->bindParam(1, $this->product_id);

    // execute query
    $stmt->execute();

    // return values
    return $stmt;
}
```

Add “exists()” method

We used the exists() method of the cart_items object but it does not exist yet.

Open /objects/cart_item.php file. Add the following code. This code will check if an item is already added to the cart or not.

```php
// check if a cart item exists
public function exists(){

    // query to count existing cart item
    $query = "SELECT count(*) FROM " . $this->table_name .

    // prepare query statement
    $stmt = $this->conn->prepare( $query );

    // sanitize
    $this->product_id=htmlspecialchars(strip_tags($this->p
    $this->user_id=htmlspecialchars(strip_tags($this->user

    // bind category id variable
    $stmt->bindParam(":product_id", $this->product_id);
    $stmt->bindParam(":user_id", $this->user_id);

    // execute query
    $stmt->execute();

    // get row value
    $rows = $stmt->fetch(PDO::FETCH_NUM);

    // return
    if($rows[0]>0){
        return true;
    }

    return false;
}
```

Create pagination file

If we have a lot of products, it is ideal to paginate the product list so our web page will not be very long. The web page will also load faster which is great for the user experience.

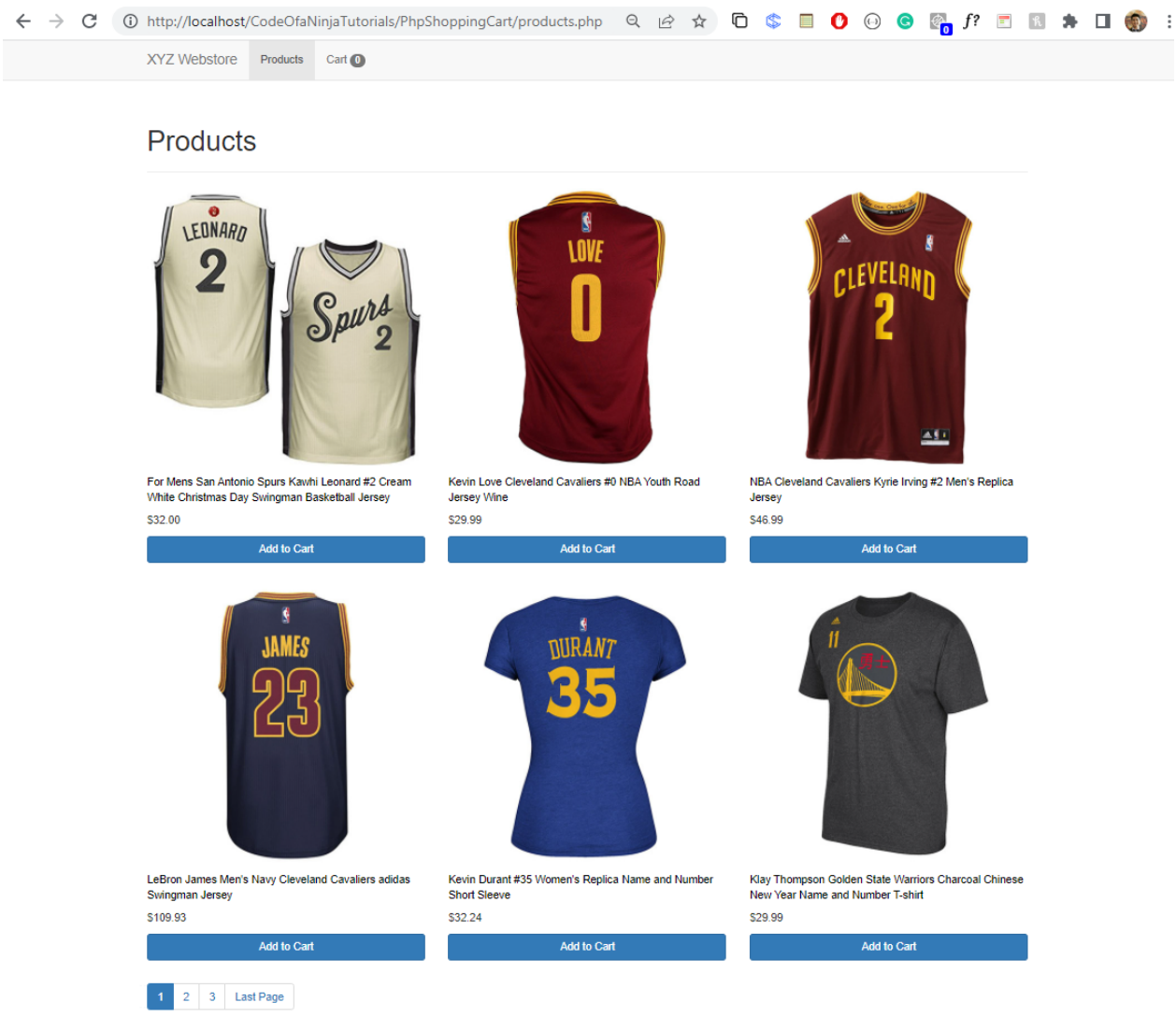
Create `paging.php` file. Put the following code. This code will display the pagination buttons. It will have the first-page button, the page numbers, and then the last page button.

```php
<?php
echo "<div class='col-md-12'>
<ul class='pagination m-b-20px m-t-0px'>";

// button for first page
if ($page > 1) {
    echo "<li>
            <a href='{$page_url}' title='Go to the first p
        </li>";
}

$total_pages = ceil($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 '$x is greater than 0' AND 'less than or eq
    if (($x > 0) && ($x <= $total_pages)) {

        // current page
        if ($x == $page) {
            echo "<li class='active'><a href=\"#\">$x <spa
        }

        // not current page
        else {
            echo "<li><a href='{$page_url}page=$x'>$x</a><
        }
    }
}

// button for last page
if ($page < $total_pages) {
    echo "<li>
            <a href='" . $page_url . "page={$total_pages}'
                Last Page
            </a>
        </li>";
}
```

```
echo "</ul>
</div>";
```

Output

Let’s run our product.php file on the browser. We should see the following output.



< https://i0.wp.com/codeofaninja.com/wp-content/uploads/2022/03/image-1.png?ssl=1>

PHP shopping cart tutorial using MySQL database – output

How to count items in the cart?

Display count in navigation bar

We will display the number of products the user has added to his cart. Open **navigation.php** file. Find the following lines of code.

```
<!--later, we'll put a PHP code here that will count items
Cart <span class="badge" id="comparison-count">0</span>
```

Replace that code with the following code. This code uses the count() method of the cart_item object to count the user's products in his cart.

```
<a href="cart.php">
<?php
// count products in cart
$cart_item->user_id=1; // default to user with ID "1" for
$cart_count=$cart_item->count();
?>
Cart <span class="badge" id="comparison-count"><?php echo
</a>
```

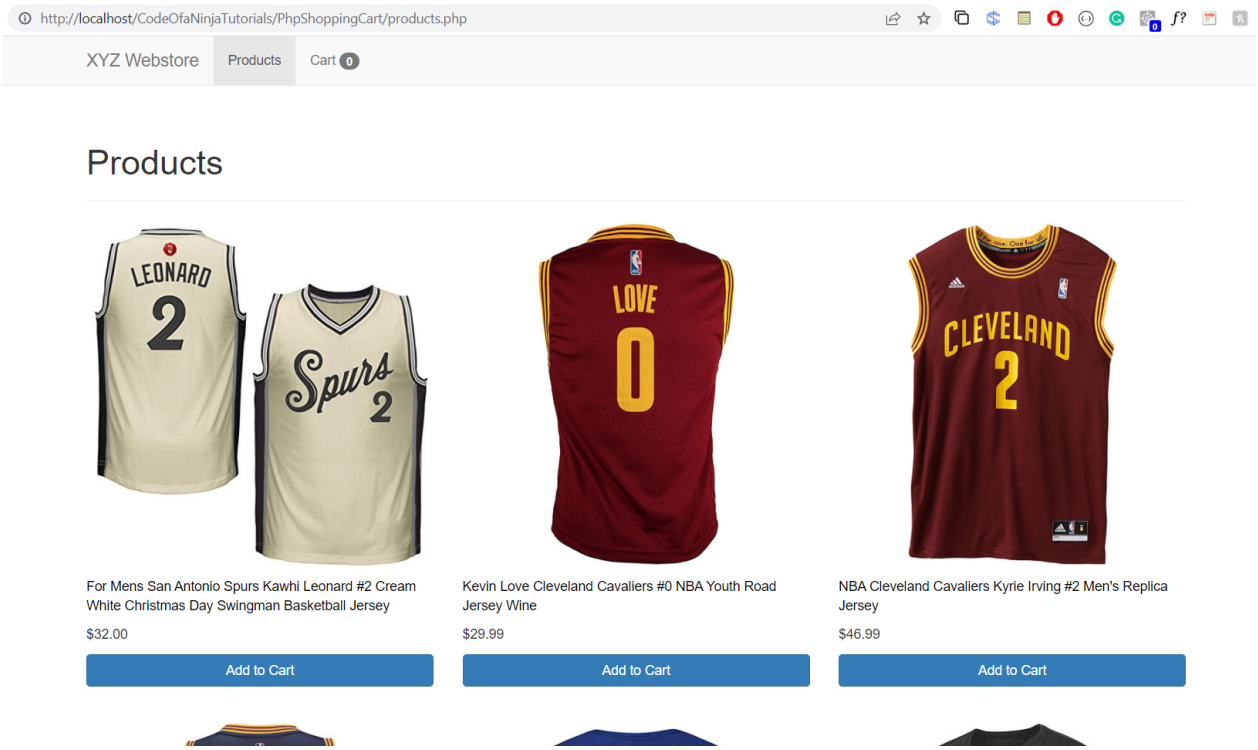
Cart item count() method

The count() method of the cart_item object does not exist yet. Open **/objects/cart_item.php** file. Put the following code. This code queries the database for the number of products a specific user has added to his cart.

```php
// count user's items in the cart
public function count()
{

    // query to count existing cart item
    $query = "SELECT count(*) FROM " . $this->table_name .

    // prepare query statement
    $stmt = $this->conn->prepare($query);

    // sanitize
    $this->user_id = htmlspecialchars(strip_tags($this->us

    // bind category id variable
    $stmt->bindParam(":user_id", $this->user_id);

    // execute query
    $stmt->execute();

    // get row value
    $rows = $stmt->fetch(PDO::FETCH_NUM);

    // return
    return $rows[0];
}
```

Output

There are not many changes yet. But we know that the number “0” will increase once we added items to the cart.



< https://i0.wp.com/codeofaninja.com/wp-content/uploads/2022/03/image-2.png?ssl=1>

How to add to cart?

Make “add to cart” button work

Let us make the “Add to Cart” button work. Create `add_to_cart.php` file. Put the following code. Our “Add to cart” buttons are linked to this file. There’s one parameter in the link which is the “product ID”. The default quantity is 1.

If the product exists in the cart, the user will be redirected to a page with a message saying the product already exists in the cart. Else, the system will create a cart_item record and redirect to a page where it says the product was added to the cart.

```php
<?php
// parameters
$product_id = isset($_GET['id']) ? $_GET['id'] : "";
$quantity = 1;

// connect to database
include 'config/database.php';

// include object
include_once "objects/cart_item.php";

// get database connection
$database = new Database();
$db = $database->getConnection();

// initialize objects
$cart_item = new CartItem($db);

// set cart item values
$cart_item->user_id = 1; // we default to '1' because we d
$cart_item->product_id = $product_id;
$cart_item->quantity = $quantity;

// check if the item is in the cart, if it is, do not add
if ($cart_item->exists()) {
    // redirect to product list and tell the user it was a
    header("Location: cart.php?action=exists");
}

// else, add the item to cart
else {
    // add to cart
    if ($cart_item->create()) {
        // redirect to product list and tell the user it w
        header("Location: cart.php?id={$product_id}&action
    } else {
        header("Location: cart.php?id={$product_id}&action
    }
}
```

Add a create() method

We used the create() method of the cart_item object but it does not exist yet. Open `/objects/cart_item.php` file. Add the following code. This code contains a database query that will add the product as a cart item.

```
// create cart item record
function create()
{

    // to get times-tamp for 'created' field
    $this->created = date('Y-m-d H:i:s');

    // query to insert cart item record
    $query = "INSERT INTO
                " . $this->table_name . "
            SET
                product_id = :product_id,
                quantity = :quantity,
                user_id = :user_id,
                created = :created";

    // prepare query statement
    $stmt = $this->conn->prepare($query);

    // sanitize
    $this->product_id = htmlspecialchars(strip_tags($this-
    $this->quantity = htmlspecialchars(strip_tags($this->q
    $this->user_id = htmlspecialchars(strip_tags($this->us

    // bind values
    $stmt->bindParam(":product_id", $this->product_id);
    $stmt->bindParam(":quantity", $this->quantity);
    $stmt->bindParam(":user_id", $this->user_id);
    $stmt->bindParam(":created", $this->created);

    // execute query
    if ($stmt->execute()) {
        return true;
    }

    return false;
}
```

Create the cart page

We will now create the “cart” page. Create `cart.php` file. Put the following basic code. This page will list the products the user has added to the cart.

```
<?php
// connect to database
include 'config/database.php';

// include objects
include_once "objects/product.php";
include_once "objects/product_image.php";
include_once "objects/cart_item.php";

// get database connection
$database = new Database();
$db = $database->getConnection();

// initialize objects
$product = new Product($db);
$product_image = new ProductImage($db);
$cart_item = new CartItem($db);

// set page title
$page_title="Cart";

// include page header html
include 'layout_head.php';

// contents will be here

// layout footer
include 'layout_foot.php';
?>
```

Display message based on action

We’ll display messages on the cart page based on a given action. Put the following code after the “**include ‘layout_head.php’;**” code of cart.php file.

We have messages if the product was removed from the cart, the quantity was updated, the product already exists, and so on.

```php
<?php
$action = isset($_GET['action']) ? $_GET['action'] : "";

echo "<div class='col-md-12'>";
if ($action == 'removed') {
    echo "<div class='alert alert-info'>
            Product was removed from your cart!
        </div>";
} else if ($action == 'added') {
    echo "<div class='alert alert-info'>
            Product was added to cart!
        </div>";
} else if ($action == 'quantity_updated') {
    echo "<div class='alert alert-info'>
            Product quantity was updated!
        </div>";
} else if ($action == 'exists') {
    echo "<div class='alert alert-info'>
            Product already exists in your cart!
        </div>";
} else if ($action == 'cart_emptied') {
    echo "<div class='alert alert-info'>
            Cart was emptied.
        </div>";
} else if ($action == 'updated') {
    echo "<div class='alert alert-info'>
            Quantity was updated.
        </div>";
} else if ($action == 'unable_to_update') {
    echo "<div class='alert alert-danger'>
            Unable to update quantity.
        </div>";
}
echo "</div>";
```

Display products added to cart

We will now display the contents of the cart. Replace “// contents will be here” of the cart.php file with the following code.

This code below will show the product name, the update quantity input box and button, the delete from cart button, the price, and the total amount of products

added to the cart.

If there are no products added to the cart yet, it will display a “No products found in your cart!” message

```php
// $cart_count variable is initialized in navigation.php
if ($cart_count > 0) {

    $cart_item->user_id = "1";
    $stmt = $cart_item->read();

    $total = 0;
    $item_count = 0;

    while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
        extract($row);

        $sub_total = $price * $quantity;

        echo "<div class='cart-row'>
            <div class='col-md-8'>";

        // product name
        echo "<div class='product-name m-b-10px'>
                    <h4>{$name}</h4>
                </div>";

        // update quantity
        echo "<form class='update-quantity-form'>
                    <div class='product-id' style='display
                    <div class='input-group'>
                        <input type='number' name='quantit
                            <span class='input-group-btn'>
                            <button class='btn btn-default
                            </span>
                    </div>
                </form>";

        // delete from cart
        echo "<a href='remove_from_cart.php?id={$id}' clas
                    Delete
                </a>
            </div>

            <div class='col-md-4'>
                <h4>$" . number_format($price, 2, '.', ','
            </div>
        </div>";

        $item_count += $quantity;
        $total += $sub_total;
```

```
    }

    echo "<div class='col-md-8'></div>
    <div class='col-md-4'>
        <div class='cart-row'>
            <h4 class='m-b-10px'>Total ({$item_count} item
            <h4>$" . number_format($total, 2, '.', ',') .
            <a href='checkout.php' class='btn btn-success
                <span class='glyphicon glyphicon-shopping-
            </a>
        </div>
    </div>";
} else {
    echo "<div class='col-md-12'>
        <div class='alert alert-danger'>
            No products found in your cart!
        </div>
    </div>";
}
```

Put read() method in cart item object

We are using the read() method of the cart_item object in cart.php but it does not exist yet. Open `/objects/cart_item.php` file. Add the following code.

This code will request the list of items added to the cart by a specific user. It is using a MySQL LEFT JOIN keyword so we can get the product information such as the name and price.

```php
// read items in the cart
public function read(){

    $query="SELECT p.id, p.name, p.price, ci.quantity, ci.
            FROM " . $this->table_name . " ci
                LEFT JOIN products p
                    ON ci.product_id = p.id
            WHERE ci.user_id=:user_id";

    // prepare query statement
    $stmt = $this->conn->prepare($query);

    // sanitize
    $this->user_id=htmlspecialchars(strip_tags($this->user

    // bind value
    $stmt->bindParam(":user_id", $this->user_id, PDO::PARA

    // execute query
    $stmt->execute();

    // return values
    return $stmt;
}
```

Output

Our cart page is not working! This is what it should look like.

< https://i0.wp.com/codeofaninja.com/wp-content/uploads/2022/03/image-3.png?ssl=1>

How to update cart?

Update product quantity with JavaScript

We will update the product quantity with the help of JavaScript. We have the ‘update’ button on the cart.php file. When that button was clicked, it will trigger the following JavaScript code.

Open layout_foot.php file. Put the following code. This code will get the product ID and quantity entered by the user. It will pass those parameters using a redirect to the update_quantity.php file.

```
<script>
    $(document).ready(function() {
        // update quantity button listener
        $('.update-quantity-form').on('submit', function()

            // get basic information for updating the cart
            var id = $(this).find('.product-id').text();
            var quantity = $(this).find('.cart-quantity').

            // redirect to update_quantity.php, with param
            window.location.href = "update_quantity.php?id
            return false;
        });

        // image hover js will be here
    });
</script>
```

PHP script to update cart

Create the update_quantity.php file. Put the following code. This code will update the product quantity in the database. It will use the values of product_id, quantity, and user_id. It will redirect to the cart.php file to show a success or failure message.

```php
<?php
// get the product id
$product_id = isset($_GET['id']) ? $_GET['id'] : 1;
$quantity = isset($_GET['quantity']) ? $_GET['quantity'] :

// make quantity a minimum of 1
$quantity=$quantity<=0 ? 1 : $quantity;

// connect to database
include 'config/database.php';

// include object
include_once "objects/cart_item.php";

// get database connection
$database = new Database();
$db = $database->getConnection();

// initialize objects
$cart_item = new CartItem($db);

// set cart item values
$cart_item->user_id=1; // we default to '1' because we do
$cart_item->product_id=$product_id;
$cart_item->quantity=$quantity;

// add to cart
if($cart_item->update()){
    // redirect to product list and tell the user it was a
    header("Location: cart.php?action=updated");
}else{
    header("Location: cart.php?action=unable_to_update");
}
```

Update cart item in database

The update() method of the cart_item object does not exist yet. Open /objects/cart_item.php file. Add the following code. This code contains the MySQL update query that will update the product quantity in the cart_items table.

```php
// create cart item record
function update(){

    // query to insert cart item record
    $query = "UPDATE " . $this->table_name . "
            SET quantity=:quantity
            WHERE product_id=:product_id AND user_id=:user

    // prepare query statement
    $stmt = $this->conn->prepare($query);

    // sanitize
    $this->quantity=htmlspecialchars(strip_tags($this->qua
    $this->product_id=htmlspecialchars(strip_tags($this->p
    $this->user_id=htmlspecialchars(strip_tags($this->user

    // bind values
    $stmt->bindParam(":quantity", $this->quantity);
    $stmt->bindParam(":product_id", $this->product_id);
    $stmt->bindParam(":user_id", $this->user_id);

    // execute query
    if($stmt->execute()){
        return true;
    }

    return false;
}
```

How to remove product from cart?

In the cart.php file, we have the “Delete” button. When this button was clicked it will trigger the remove_from_cart.php file. This will remove a specific product from the cart.

Create remove_from_cart.php file. Put the following code. This code uses the values of product_id and user_id to delete the product from the cart. Once deleted, the user will be redirected to the cart page with a confirmation message.

```php
<?php
// get the product id
$product_id = isset($_GET['id']) ? $_GET['id'] : "";

// connect to database
include 'config/database.php';

// include object
include_once "objects/cart_item.php";

// get database connection
$database = new Database();
$db = $database->getConnection();

// initialize objects
$cart_item = new CartItem($db);

// remove cart item from database
$cart_item->user_id=1; // we default to '1' because we do
$cart_item->product_id=$product_id;
$cart_item->delete();

// redirect to product list and tell the user it was added
header('Location: cart.php?action=removed&id=' . $id);
```

Put delete() method

The delete() method does not exist yet in the cart_item object. Open /objects/cart_item.php file. Add the following code. This code contains a DELETE MySQL query that will delete the product from the cart.

```php
// remove cart item by user and product
public function delete(){

    // delete query
    $query = "DELETE FROM " . $this->table_name . " WHERE
    $stmt = $this->conn->prepare($query);

    // sanitize
    $this->product_id=htmlspecialchars(strip_tags($this->p
    $this->user_id=htmlspecialchars(strip_tags($this->user

    // bind ids
    $stmt->bindParam(":product_id", $this->product_id);
    $stmt->bindParam(":user_id", $this->user_id);

    if($stmt->execute()){
        return true;
    }

    return false;
}
```

Create the checkout page

The checkout page looks like the cart page but the items cannot be updated or removed. It is just like the summary of orders with the total amount presented to the customer. It will also act as a confirmation page before the customer place the order.

Create checkout.php file. Put the following code. This code will request the cart data from the database. It will loop through the records to display the products added to the cart. It will also display the total cost of the order.

```php
<?php
// connect to database
include 'config/database.php';

// include objects
include_once "objects/product.php";
include_once "objects/product_image.php";
include_once "objects/cart_item.php";

// get database connection
$database = new Database();
$db = $database->getConnection();

// initialize objects
$product = new Product($db);
$product_image = new ProductImage($db);
$cart_item = new CartItem($db);

// set page title
$page_title = "Checkout";

// include page header html
include 'layout_head.php';

// $cart_count variable is initialized in navigation.php
if ($cart_count > 0) {

    $cart_item->user_id = "1";
    $stmt = $cart_item->read();

    $total = 0;
    $item_count = 0;

    while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
        extract($row);

        $sub_total = $price * $quantity;

        echo "<div class='cart-row'>
                <div class='col-md-8'>
                    <div class='product-name m-b-10px'><h4
        echo $quantity > 1 ? "<div>{$quantity} items</div>
        echo "</div>";

        echo "<div class='col-md-4'>
                <h4>$" . number_format($price, 2, '.', ','
```

```
            </div>
        </div>";

        $item_count += $quantity;
        $total += $sub_total;
    }

    echo "<div class='col-md-12 text-align-center'>
        <div class='cart-row'>";

    if ($item_count > 1) {
        echo "<h4 class='m-b-10px'>Total ({$item_count} it
    } else {
        echo "<h4 class='m-b-10px'>Total ({$item_count} it
    }

    echo "<h4>$" . number_format($total, 2, '.', ',') . "<

            <a href='place_order.php' class='btn btn-lg bt
                <span class='glyphicon glyphicon-shopping-
            </a>
        </div>
    </div>";
} else {
    echo "<div class='col-md-12'>
        <div class='alert alert-danger'>
            No products found in your cart!
        </div>
    </div>";
}

include 'layout_foot.php';
```

Create place_order.php

Create a place_order.php file. Put the following code. This code will show a “thank you” message. It will also remove all items in the cart so the customer can start with a new cart.

You can do a lot more with this file, for example, saving the order in your orders database. Unfortunately, we cannot cover that part in this tutorial for now.

```php
<?php
// include classes
include_once "config/database.php";
include_once "objects/cart_item.php";

// get database connection
$database = new Database();
$db = $database->getConnection();

// initialize objects
$cart_item = new CartItem($db);

// remove all cart item by user, from database
$cart_item->user_id = 1; // we default to '1' because we d
$cart_item->deleteByUser();

// set page title
$page_title = "Thank You!";

// include page header HTML
include_once 'layout_head.php';

// tell the user order has been placed
echo "<div class='col-md-12'>
    <div class='alert alert-success'>
        <strong>Your order has been placed!</strong> Thank
    </div>
</div>";

// include page footer HTML
include_once 'layout_foot.php';
```

Delete all items in the cart

The deleteByUser() method does not exist yet in the cart_item object. Open /objects/cart_item.php file. Add the following code. This code contains a DELETE query that will delete all the products added to the cart of a specific user.

```php
// remove cart items by user
public function deleteByUser(){
    // delete query
    $query = "DELETE FROM " . $this->table_name . " WHERE
    $stmt = $this->conn->prepare($query);

    // sanitize
    $this->user_id=htmlspecialchars(strip_tags($this->user

    // bind id
    $stmt->bindParam(":user_id", $this->user_id);

    if($stmt->execute()){
        return true;
    }

    return false;
}
```

Output

< https://i0.wp.com/codeofaninja.com/wp-content/uploads/2022/03/image-4.png?ssl=1>

< https://i0.wp.com/codeofaninja.com/wp-content/uploads/2022/03/image-5.png?ssl=1>

< https://i0.wp.com/codeofaninja.com/wp-content/uploads/2022/03/image-6.png?ssl=1>

How to make the product page?

Product details page

Create `product.php` file. Put the following code. Once we completed the code on this product.php file, it will display the product details of a specific product.

```php
<?php
// include classes
include_once "config/database.php";
include_once "objects/product.php";
include_once "objects/product_image.php";
include_once "objects/cart_item.php";

// get database connection
$database = new Database();
$db = $database->getConnection();

// initialize objects
$product = new Product($db);
$product_image = new ProductImage($db);
$cart_item = new CartItem($db);

// get ID of the product to be edited
$id = isset($_GET['id']) ? $_GET['id'] : die('ERROR: missi
$action = isset($_GET['action']) ? $_GET['action'] : "";

// set the id as product id property
$product->id = $id;

// to read single record product
$product->readOne();

// set page title
$page_title = $product->name;

// include page header HTML
include_once 'layout_head.php';

// content will be here

// include page footer HTML
include_once 'layout_foot.php';
?>
```

Read one product method

We are using the “readOne()” method of product object but it does not exist yet. Open the “objects” folder. Open product.php file. Put the following code. This

code contains a SELECT query that will read product details for a specific product.

```
// used when reading product details
function readOne()
{

    // query to select single record
    $query = "SELECT name, description, price
                FROM " . $this->table_name . "
                WHERE id = ?
                LIMIT 0,1";

    // prepare query statement
    $stmt = $this->conn->prepare($query);

    // sanitize
    $this->id = htmlspecialchars(strip_tags($this->id));

    // bind product id value
    $stmt->bindParam(1, $this->id);

    // execute query
    $stmt->execute();

    // get row values
    $row = $stmt->fetch(PDO::FETCH_ASSOC);

    // assign retrieved row value to object properties
    $this->name = $row['name'];
    $this->description = $row['description'];
    $this->price = $row['price'];
}
```

Display product thumbnails

Let's go back to the "product.php" file. Replace the "// content will be here" comment with the following code. This code will display the product images as thumbnails. When a thumbnail was hovered, it will display a larger version of that image.

```php
// set product id
$product_image->product_id = $id;

// read all related product image
$stmt_product_image = $product_image->readByProductId();

// count all relatd product image
$num_product_image = $stmt_product_image->rowCount();

echo "<div class='col-md-1'>";
// if count is more than zero
if ($num_product_image > 0) {

    // loop through all product images
    while ($row = $stmt_product_image->fetch(PDO::FETCH_AS

        // image name and source url
        $product_image_name = $row['name'];
        $source = "uploads/images/{$product_image_name}";
        echo "<img src='{$source}' class='product-img-thum
    }
} else {
    echo "No images.";
}
echo "</div>";

// large image will be here
```

Read images related to product

The “readByProductId()” method does not exist yet in the product_image object. Open the “objects” folder. Open the “product_image.php” file. Put the following code. This code contains a SELECT query that will return the list of images related to the product.

```php
// read all product image related to a product
function readByProductId()
{

    // select query
    $query = "SELECT id, product_id, name
            FROM " . $this->table_name . "
            WHERE product_id = ?
            ORDER BY name ASC";

    // prepare query statement
    $stmt = $this->conn->prepare($query);

    // sanitize
    $this->product_id = htmlspecialchars(strip_tags($this-

    // bind prodcut id variable
    $stmt->bindParam(1, $this->product_id);

    // execute query
    $stmt->execute();

    // return values
    return $stmt;
}
```

Display large image

We’re going to display a larger version of the product image using the following code. Replace the “// large image will be here” comment of the product.php file with the following code.

```
echo "<div class='col-md-4' id='product-img'>";

// read all related product image
$stmt_product_image = $product_image->readByProductId();
$num_product_image = $stmt_product_image->rowCount();

// if count is more than zero
if ($num_product_image > 0) {
    // loop through all product images
    $x = 0;
    while ($row = $stmt_product_image->fetch(PDO::FETCH_AS
        // image name and source url
        $product_image_name = $row['name'];
        $source = "uploads/images/{$product_image_name}";
        $show_product_img = $x == 0 ? "display-block" : "d
        echo "<a href='{$source}' target='_blank' id='prod
                <img src='{$source}' style='width:100%;' /
            </a>";
        $x++;
    }
} else {
    echo "No images.";
}
echo "</div>";

// product details will be here
```

Make image hover work

When the user hovers a thumbnail image, the larger version of the image must show up. We're going to use JavaScript code to achieve this. Open the "layout_foot.php" file. Replace "// image hover js will be here" comment with the following code.

```
// change product image on hover
$(document).on('mouseenter', '.product-img-thumb', functio
    var data_img_id = $(this).attr('data-img-id');
    $('.product-img').hide();
    $('#product-img-' + data_img_id).show();
});
```

Display product details

We will display the product price and description using the following code. Open product.php file. Replace the “// product details will be here” comment with the following code.

```
$page_description = htmlspecialchars_decode(htmlspecialcha

echo "<div class='col-md-5'>
        <h4 class='m-b-10px price-description'>$" . number
        <div class='m-b-10px'>
            {$page_description}
        </div>
    </div>";

// cart buttons will be here
```

Display a cart button

We will display the “Add to cart” button if the product is not yet added to the cart. Else, we will display the “Update cart” button. Open product.php file. Replace “// cart buttons will be here” comment with the following code.

```
echo "<div class='col-md-2'>";

// cart item settings
$cart_item->user_id = 1; // we default to a user with ID "
$cart_item->product_id = $id;

// if product was already added in the cart
if ($cart_item->exists()) {
    echo "<div class='m-b-10px'>This product is already in
    <a href='cart.php' class='btn btn-success w-100-pct'>
        Update Cart
    </a>";
}

// if product was not added to the cart yet
else {
    echo "<a href='add_to_cart.php?id={$id}' class='btn bt
}

echo "</div>";
```

Ouput

< https://i0.wp.com/codeofaninja.com/wp-content/uploads/2022/03/image-15.png?ssl=1>

< https://i0.wp.com/codeofaninja.com/wp-content/uploads/2022/03/image-16.png?ssl=1>

What People Say?

I'm so glad that this code delights other people. The following are some of them from the comments section!

★★★★★ "Hey Mike, my name is Leonardo from Argentina. I've been reading your blog since like 4 months from now, and I really must say: your tutorials are very good, they has helped me in many of my works… Well, thank you very much man. I really admire your work." ~ **Leonardo < https://www.codeofaninja.com/wp-content/uploads/2015/03/leonardo-codeofaninja-review.jpg>**

★★★★★ "Man, your tut's are awesome. Im so glad ive found your blog. Big respect!" ~ **Milos < https://www.codeofaninja.com/wp-content/uploads/2015/03/milos-codeofaninja-review.jpg>**

★★★★★ "I bought your level-2 source code and it was so good, very big help for me. It was worth it. Thank you very much!" ~ **Ashley Deanna Plata <**

https://www.codeofaninja.com/wp-content/uploads/2015/03/ashley-deanna-plata-codeofaninja-review.jpg>

★★★★★ “Hello, This is a great script and I have paid 6.99 for your work (it Worth it).” ~ **Louis Blais < https://www.codeofaninja.com/wp-content/uploads/2015/03/louis-blais-codeofaninja-review.jpg>**

★★★★★ “Words can’t express how grateful I am for the work and the articles you post, had some troubles with doing somethings but your articles as per usual hit the hammer right on the head. They are a great way for expanding upon later too!” ~ **Jeremy Smith < https://www.codeofaninja.com/wp-content/uploads/2015/03/jeremy-smith-codeofaninja-review.jpg>**

Download source code

You can get the LEVEL 1 source code by following the whole, well-detailed, and free tutorial above. But isn’t it more convenient if you can just download the complete source code we used, and play around with it?

There’s a small fee in getting the complete source code, it is small compared to the:

- Value or skill upgrade it can bring you.
- The income you can get from your website project or business.
- Precious time you will save.
- Expert advice and support (if you have any questions related to the source code)

Source code features	Level 1	Level 2
List all products from the MySQL database	YES	YES
Pagination of products list	YES	YES
Add to cart button	YES	YES
Remove from cart button	YES	YES
Show message if a product was added to the cart	YES	YES
Show message if a product was removed to cart	YES	YES
Navigation bar highlights which page is selected	YES	YES
Cart menu shows the number of products added to the cart	YES	YES
Show message if no products found	YES	YES
Show message if no product found in the cart	YES	YES
Bootstrap enabled UI	YES	YES
Cart page that lists all products added to cart	YES	YES
Compute the total cost of all products added to the cart	YES	YES
Update cart with the desired quantity	YES	YES
Single product view with add to cart button	YES	YES
Check out button with cart icon	YES	YES
The navigation bar has a drop-down of product categories	NO	YES
The selected category is highlighted in the drop-down	NO	YES
Categories are retrieved from the database	NO	YES