Web Design and Programming

Week 10

Assessing MySQL using PHP, MVC pattern

Peeraya Sripian

27 June 2024

Course schedule

Week	Date	Topic
1	4/18	Intro to WWW, Intro to HTML
2	4/25	CSS Fundamental
	5/2	Holiday (GW)
3	5/9	CSS and Bootstrap
4	5/16	Work on midterm project
5	**COIL 22 MAY 18:00-19:30 (counted as 1 class, replacing 23 May)**	
6	5/30	Midterm project presentation week
7	6/6	PHP fundamentals + Installation XAMPP
8	6/13	PHP fundamentals 2 + Intro of Final project
9	6/20	mySQL fundamentals
10	6/27	Assessing MySQL using PHP, MVC pattern
11	7/4	Cookies, sessions, and authentication **+ Proposal of final project**
12	7/11	Javascript and PHP validation
13	7/18	Final project development
14	7/25	Final project presentation

MySQL

- MySQL is a very popular, open source database for web servers.
- Officially pronounced “my Ess Que Ell” (not my sequel).
- Handles very large databases; very fast performance.
- Why are we using MySQL?
  - Free (much cheaper than Oracle!)
  - Each student can install MySQL locally.
  - Easy to use Shell for creating tables, querying tables, etc.
  - Easy to use with Java JDBC

Relational database

One-to-One

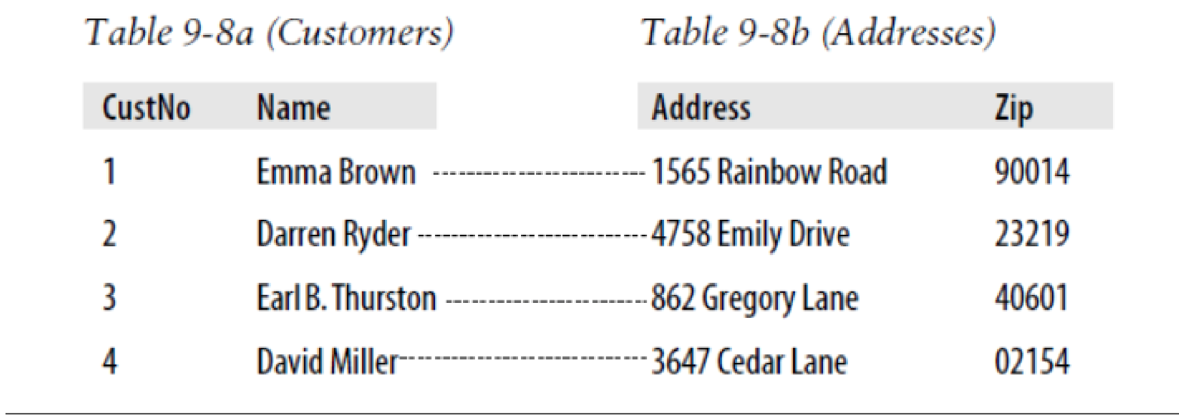


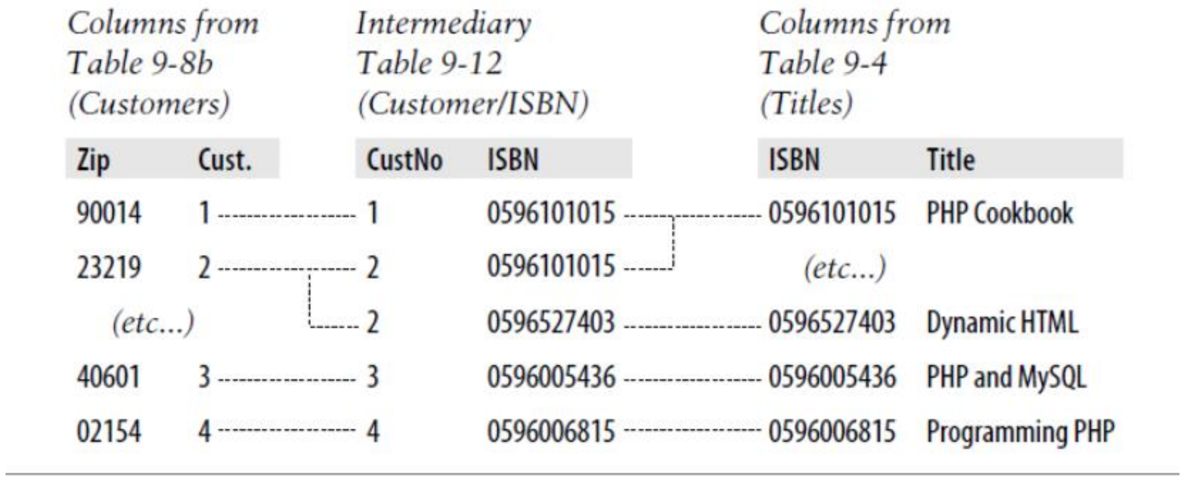
Table 9-8a (Customers)

CustNo	Name
1	Emma Brown
2	Darren Ryder
3	Earl B. Thurston
4	David Miller

Table 9-8b (Addresses)

Address	Zip
1565 Rainbow Road	90014
4758 Emily Drive	23219
862 Gregory Lane	40601
3647 Cedar Lane	02154

Many-to-Many



Columns from Table 9-8b (Customers)

Zip	Cust.
90014	1
23219	2
(etc...)	
40601	3
02154	4

Intermediary Table 9-12 (Customer/ISBN)

CustNo	ISBN
1	0596101015
2	0596101015
2	0596527403
3	0596005436
4	0596006815

Columns from Table 9-4 (Titles)

ISBN	Title
0596101015	PHP Cookbook
(etc...)	
0596527403	Dynamic HTML
0596005436	PHP and MySQL
0596006815	Programming PHP

One-to-Many

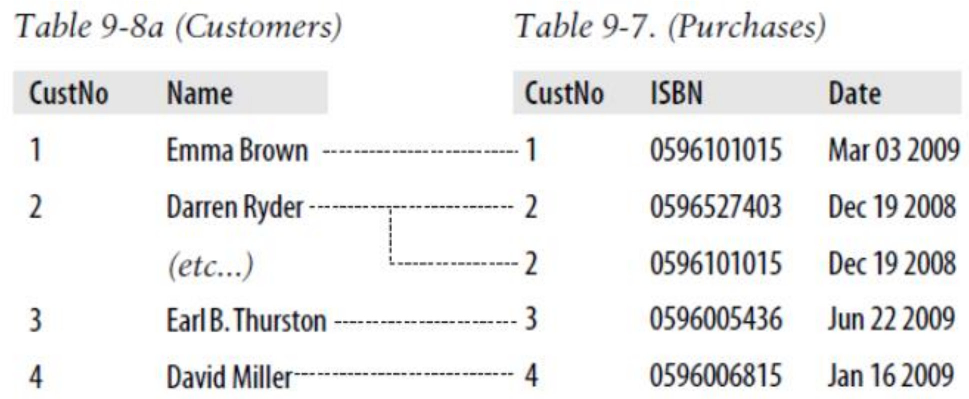


Table 9-8a (Customers)

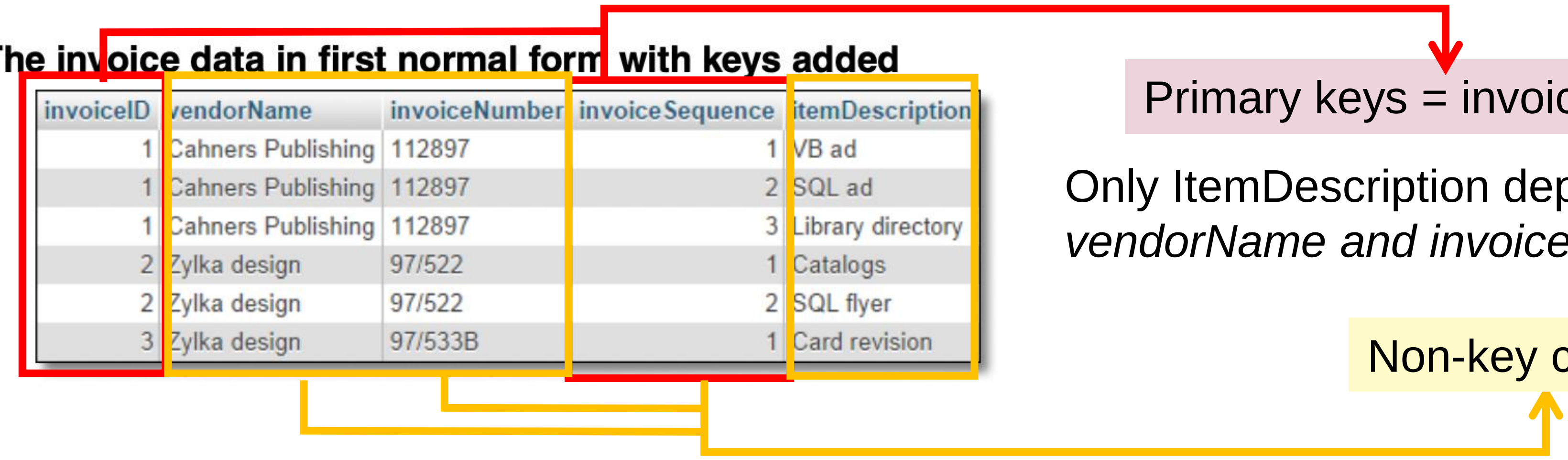
CustNo	Name
1	Emma Brown
2	Darren Ryder
	(etc...)
3	Earl B. Thurston
4	David Miller

Table 9-7. (Purchases)

CustNo	ISBN	Date
1	0596101015	Mar 03 2009
2	0596527403	Dec 19 2008
2	0596101015	Dec 19 2008
3	0596005436	Jun 22 2009
4	0596006815	Jan 16 2009

How to apply the second normal form

The invoice data in first normal form with keys added



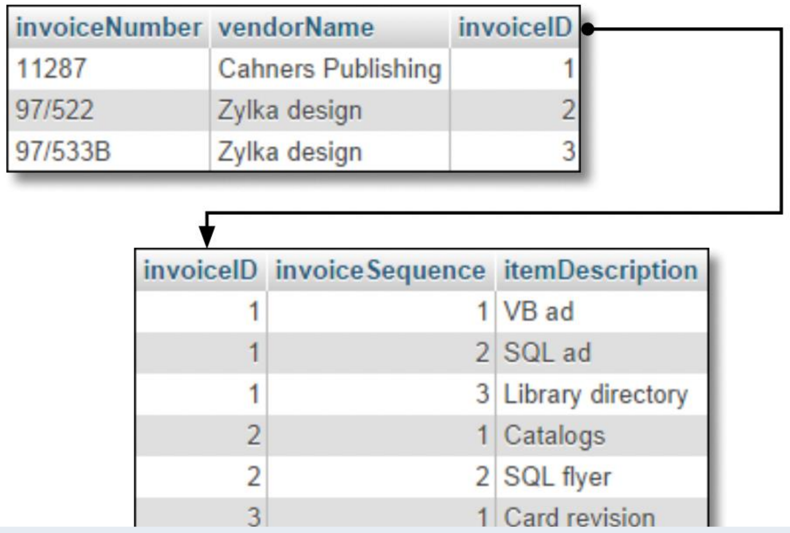
invoiceID	vendorName	invoiceNumber	invoiceSequence	itemDescription
1	Cahners Publishing	112897	1	VB ad
1	Cahners Publishing	112897	2	SQL ad
1	Cahners Publishing	112897	3	Library directory
2	Zylka design	97/522	1	Catalogs
2	Zylka design	97/522	2	SQL flyer
3	Zylka design	97/533B	1	Card revision

Primary keys = invoiceID & invoiceSequence

Only ItemDescription depends on **Primary keys**
*vendorName and invoiceNumber depends on **invoiceID***

Non-key columns

The invoice data in second normal form



invoiceNumber	vendorName	invoiceID
11287	Cahners Publishing	1
97/522	Zylka design	2
97/533B	Zylka design	3

invoiceID	invoiceSequence	itemDescription
1	1	VB ad
1	2	SQL ad
1	3	Library directory
2	1	Catalogs
2	2	SQL flyer
3	1	Card revision

- Move the columns that does not depend on Primary keys to another table
- 2 Tables
  1. Info related to invoice
  2. Info related to individual line items
- The relationship between tables is based on InvoiceID
  - InvoiceID is primary key of Table#1
  - InvoiceID is foreign key of Table #2

Course schedule

Week	Date	Teaching plan
1	13 Apr	Introduction to WWW
2	20 Apr	HTML and CSS Fundamentals 1
3	27 Apr	HTML and CSS Fundamentals 2
4	11 May	HTML and CSS: Framework
5	18 May	Midterm project presentation
6	25 May	PHP fundamentals 1
7	1 June	PHP fundamentals 2
8	8 June	mySQL fundamentals
9	15 June	Assessing MySQL using PHP, MVC pattern
10	22 June	Cookies, sessions, and authentication
11	29 June	Proposal of final project
12	6 July	Javascript and PHP validation
13	13 July	Final project development
14	20 July	Final project presentation

COIL: **C**ollaborative **O**nline **I**nternational **L**earning.

Pedagogy to connect with overseas universities online and provide an interactive and collaborative learning environment in and outside class.

COIL Implementation

26 May 10:00-12:00: COIL zoom discussion

Today’s topic

- How to connect to a database and handle exceptions
- Let’s put them in practice

- Break

- The MVC Pattern, code explanation
- **Homework 7**

Connecting to the database

3 Ways to use PHP to work with MySQL

- API (Application Programming Interface) provides a way for an application to work with other applications

- **PDO (PHP Data Objects)**
  - PDO (PHP Data Objects) extension to PHP defines a consistent interface for accessing databases.
  - PDO supports most popular databases, this lets you write PHP code that can be used for more than one type of database.
  - PDO is included with PHP 5.1 and later and is available as a PECL extension for PHP 5.0.

- **mysqli extension**
  - OO interface and procedural interface

- **MySQL extension**
  - The oldest PHP interface for working with MySQL.
  - Deprecated as of PHP5.5

PDO (PHP Data Objects)

Pros

- Is included with PHP 5.1 and later and available for 5.0.
- Provides an object-oriented interface.
- Provides a consistent interface that’s portable between other database servers such as Oracle, DB2, Microsoft SQL Server, and PostgreSQL.
- Takes advantage of most new features found in MySQL 4.1.3 and later.

Cons

- Doesn’t work with versions of PHP 4.x, 3.x, or earlier.
- Doesn’t take advantage of some advanced features found in MySQL 4.1.3 and later, such as multiple statements.

mysqli (MySQL improved extension)

Pros

- Is included with PHP 5 and later.
- Provides both an object-oriented interface and a procedural interface.
- Takes advantage of all new features found in MySQL 4.1.3 and later.

Cons

- Can’t be used with other database servers.

MySQL (MySQL extension)

Pros

- Works with older versions of PHP such as 3.x and 4.x

Cons

- Doesn’t take advantage of the advanced features found in MySQL 4.1.3 and later.
- Was deprecated with PHP 5.5 and is not included with PHP 7.

Connecting to a database

- The syntax for creating an object from any class

```
new ClassName(arguments);
```

- The syntax for creating a database object from the PDO class

```
new PDO($dsn, $username, $password);
```

- The syntax for a DSN (Data Source Name) for a MySQL database

```
mysql:host=host_address;
dbname=database_name;
```

- How to connect to a MySQL database named **my_guitar_shop1**

```
$dsn =         'mysql:host=localhost;
dbname=        'my_guitar_shop1';
$username =    'mgs_user';
$password =    'pa55word';

$db = new PDO($dsn, $username, $password);
      // creates PDO object
```

Your database name

The username you set

The password you set

Handling exceptions

- Sometimes PDO object cannot be created using the PDO class
- The class must throws an **exception**
  - Object that contains information about the error that occurred.
  - If the exception isn’t handled, the applications **ENDS**.
- `try/catch` statement to handle an exception

The syntax for a try/catch statement

```
try {
      // statements that might throw an exception
}
catch (ExceptionClass $exception_name) {
      // statements that handle the exception
}
```

try/catch

How to handle a PDO exception

```
try {
  $db = new PDO($dsn, $username, $password);
  echo '<p>You are connected to the database!</p>';
}
catch (PDOException $e) {
  $error_message = $e->getMessage();
  echo "<p>An error occurred while connecting to the database:
     $error_message </p>";
}
```

The statement that may throw an exception

The statement that will be executed if an exception is thrown

To call a method from any object, use
Nameofobject → nameofmethod

How to handle any type of exception

```
try {
// statements that might throw an exception
}
catch (Exception $e) {
  $error_message = $e->getMessage(); echo "<p>Error message: $error_message
  </p>";
}
```

How to select data

Methods of the PDO class for selecting data

Method	Description
`query($select_statement)`	Executes the specified SQL SELECT statement and returns a PDOStatement object that contains the result set. If no result set is returned, this method returns a FALSE value.
`quote($input)`	Places quotes around the input and escapes special characters.

How to select data (cont.)

A query() method with the SELECT statement coded in a variable

```
$query = 'SELECT * FROM products WHERE categoryID = 1 ORDER BY productID';
$products = $db->query($query); // $products contains the result set
```

A query() method with the SELECT statement coded as the argument

```
$products = $db->query('SELECT * FROM products');
```

An unquoted parameter (not secure!)

```
$query = "SELECT productCode, productName, listPrice FROM products
   WHERE productID = $product_id";
$products = $db->query($query);
```

These are **dynamic SQL statements**
Data that's input by users can be malicious. To protect against this, you can use the `quote()` method or **prepared statements**.

How to select data (cont.)

A quoted parameter (more secure)

```
$product_id_q = $db->quote($product_id);
$query = "SELECT productCode, productName, listPrice FROM products
   WHERE productID = $product_id_q";
$products = $db->query($query);
```

- If some parameter (such as `$product_id`) is included in the SQL statement, it is prone to a risk of *XSS or SQL injection attack*.
- To prevent this, use `quote()` around the input.
- Not all databases implements `quote()` method
- Therefore, ***prepared statement*** is better for protection against malicious input

How to insert, update, and delete data

Method	Description
`exec($sql_statement)`	Executes the specified SQL statement and returns the number of affected rows. If no rows were affected, the method returns zero.

- Use `exec()` method of the PDO object to execute dynamic statements
- Can also affect more than one row

How to insert, update, and delete data

How to execute an INSERT statement

```
$category_id_q = $db->quote($category_id);
$code_q        = $db->quote($code);
$name_q        = $db->quote($name);
$price_q       = $db->quote($price);
$query = "INSERT INTO products VALUES
  (categoryID, productCode, productName, listPrice)
  ($category_id_q, $code_q, $name_q, $price_q)";
$insert_count  = $db->exec($query);
```

How to execute an UPDATE statement

```
$product_id_q  = $db->quote($product_id);
$price_q       = $db->quote($price);
$query = "UPDATE products
  SET listPrice = $price_q
  WHERE productID = $product_id_q";
$update_count  = $db->exec($query);
```

How to execute a DELETE statement

```
$product_id_q  = $db->quote($product_id);
$query  = "DELETE FROM products
  WHERE productID = $product_id_q";
$delete_count  = $db->exec($query);
```

How to display the row counts

```
<p>Insert count: <?php echo $insert_count; ?></p>
<p>Update count: <?php echo $update_count; ?></p>
<p>Delete count: <?php echo $delete_count; ?></p>
```

Prepared statements

- To execute SQL statement, there are two methods
  - Prepared statements
  - Dynamic stetements

Method	Description
`prepare($sql_statement)`	Prepares the specified SQL statement for execution and returns a PDOStatement object. The specified statement can contain zero or more named (`:name`) or question mark (`?`) parameters.
`lastInsertId()`	After an INSERT statement has been executed, this method gets the ID that was automatically generated by MySQL for the row.

Execute SQL statements

- Methods of the PDOStatement class

Method	Description
`bindValue($param, $value)`	Binds the specified value to the specified parameter in the prepared statement. Returns TRUE for success and FALSE for failure.
`execute()`	Executes the prepared statement. Returns TRUE for success and FALSE for failure.
`fetchAll()`	Returns an array for all of the rows in the result set.
`fetch()`	Returns an array for the next row in the result set.
`rowCount()`	Returns the number of rows affected by the last statement.
`closeCursor()`	Closes the cursor and frees the connection to the server so other SQL statements may be issued.

Execute SQL statements

How to use the `fetchAll()` method to return a result set

```
$query = 'SELECT * FROM products';
$statement = $db->prepare($query);
$statement->execute();
$products = $statement->fetchAll();
$statement->closeCursor();
foreach ($products as $product)
{
  echo $product['productName'] . '<br>';
}
```

No parameters in this SELECT statement

All rows in the result are stored in `$product`

`fetchAll()` use more memory than `fetch()`

How to use the `fetch()` method to loop through a result set

```
$query = 'SELECT * FROM products';
$statement = $db->prepare($query);
$statement->execute();
$product = $statement->fetch();
while ($product != null) {
       echo $product['productName'] . '<br>';
       $product = $statement->fetch();
} $statement->closeCursor();
```

Returns the first row in the result set or a NULL value if the result has no rows

While loop to process each row

Get the next result row

Named parameter

- Prepared statements may include more than one parameters
- A named parameter begins with **:** followed by the name of the parameter

How to use named parameters

```
$query = ‘SELECT * FROM products
         WHERE categoryID = :category_id
         AND listPrice > :price’;
$statement = $db->prepare($query);
$statement->bindValue(':category_id', $category_id);
$statement->bindValue(':price', $price);
$statement->execute();
$products = $statement->fetchAll();
$statement->closeCursor();
```

Parameters in this SELECT statement

Bind the values in $category_id variable to the parameter :category_id

Question mark parameters

- A *question mark parameter* use ? To indicate the <u>location</u> of the parameter in the SQL statement

How to use question mark parameters

```
$query = 'SELECT * FROM products WHERE categoryID = ? AND listPrice > ?';
$statement = $db->prepare($query);
$statement->bindValue(1, $category_id);
$statement->bindValue(2, $price);
$statement->execute();
$products = $statement->fetchAll();
$statement->closeCursor();
```

Bind `$category_id` to the first ? parameter in SQL statement

Bind `$price` to the second ? In SQL statement

How to modify data

Prepares and executes an INSERT statement to insert a row into the database.

```
// Sample data
$category_id = 2;
$code = 'hofner';
$name = 'Hofner Icon';
$price = '499.99';
// Prepare and execute the statement
$query = 'INSERT INTO products VALUES (categoryID, productCode, productName, listPrice)
(:category_id, :code, :name, :price)';
$statement = $db->prepare($query);
$statement->bindValue(':category_id', $category_id);
$statement->bindValue(':code', $code);
$statement->bindValue(':name', $name);
$statement->bindValue(':price', $price);
$success = $statement->execute();
$row_count = $statement->rowCount();
$statement->closeCursor();
// Get the last product ID that was automatically generated
$product_id = $db->lastInsertId();
// Display a message to the user
if ($success) {
  echo "<p>$row_count row(s) was inserted with this ID: $product_id</p>";
}
else {
  echo "<p>No rows were inserted.</p>";
}
```

Setting error mode for PDO

- Error mode determines what happens when there is an error executing a SQL statement
- It does not affect what happens when PDO connect to DB
  - PDO always use the ”exception mode”
  - PDO emits a standard PHP warning message, throw exception that you can catch and handle
- PDO use silent mode when executing SQL statements
  - If there is an error, PDO doesn’t throw exception and doesn’t issue PHP warning
  - It instead sets the error in the database object (dynamic), or in the statement object (prepared)
- To view the error, you can use `errorCode()` and `errorInfo()` on the object

Error modes for PDO

Name	Description
ERRMODE_SILENT	This is the default error mode. PDO sets the error in the database or statement object, but it doesn't emit a PHP warning message or throw an exception. To access the error, you can use the errorCode() and errorInfo() methods on the database or statement object. However, this requires you to check the error code after each database call.
ERRMODE_WARNING	PDO sets the error and doesn't throw an exception as in "silent" mode, but does emit a PHP warning message. This setting is useful during testing and debugging.
ERRMODE_EXCEPTION	PDO sets the error as in "silent" mode and throws a PDOException object that reflects the error code and error message. This setting is also useful during testing and debugging, and it makes it easier for you to structure your error-handling code.

Most recommended

Setting error mode in PDO

How to use the constructor of the PDO class to set the error mode

```
$dsn = 'mysql:host=localhost;dbname=my_guitar_shop2';
$username = 'mgs_user';
$password = 'pa55word';
$options = array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION);
try {
  $db = new PDO($dsn, $username, $password, $options);
}
catch (PDOException $e) {
  $error_message = $e->getMessage();
  echo "<p>Error connecting to database: $error_message </p>";
  exit();
}
```

How to use the `setAttribute()` method to set the error mode

```
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

Catching `PDOException` objects

- In this example, SELECT incorrectly refer to `product`, instead of `products`
- When code execute the prepared SELECT statement, it will throws a `PDOException`
- Display the message and exits the script

How to use a try/catch statement to catch PDOException objects

```
try {
  $query = ‘SELECT * FROM product’;
  $statement = $db->prepare($query);
  $statement->execute();
  $products = $statement->fetchAll();
  $statement->closeCursor();
}
catch (PDOException $e) {
  $error_message = $e->getMessage();
  echo "<p>Database error: $error_message </p>";
  exit();
}
```

Let's do this together in class

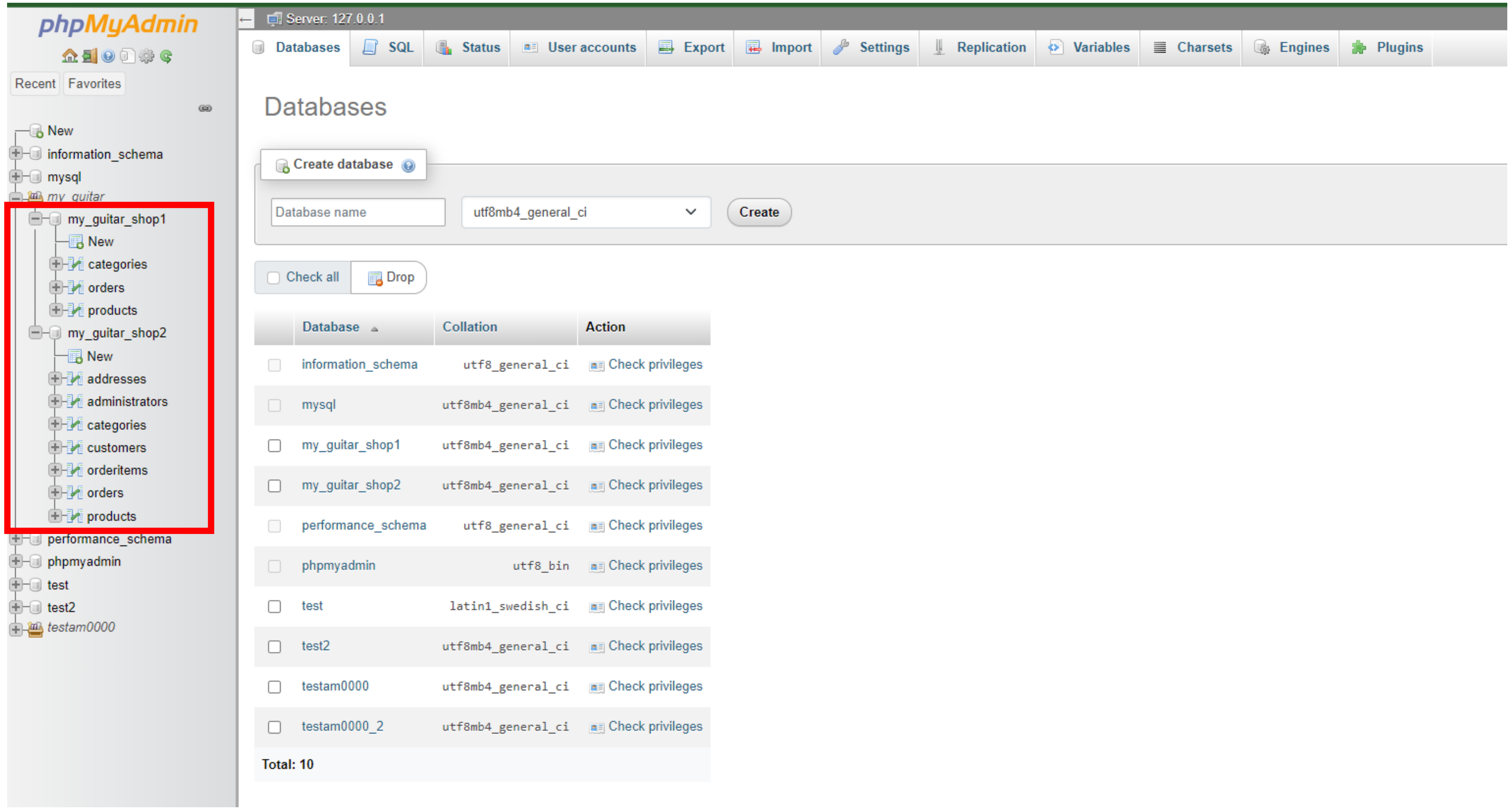
1. Download `material.zip` file from Github
2. Extract the files
3. Copy folders `class10_demo` and `class10_mvcdemo` to xampp/htdocs folder in your own PC
4. Import `create_db_class10.sql` in your phpMyadmin

Let's put in practice

- Code explanation
- We will do this in localhost first, so please turn on PHP and Mysql in XAMPP in your PC
- Make sure that you run **`create_db_class10.sql`** in your phpMyAdmin (in your localhost) **first**

Confirm that you have databases first

In your `localhost/phpmyadmin`



Databases

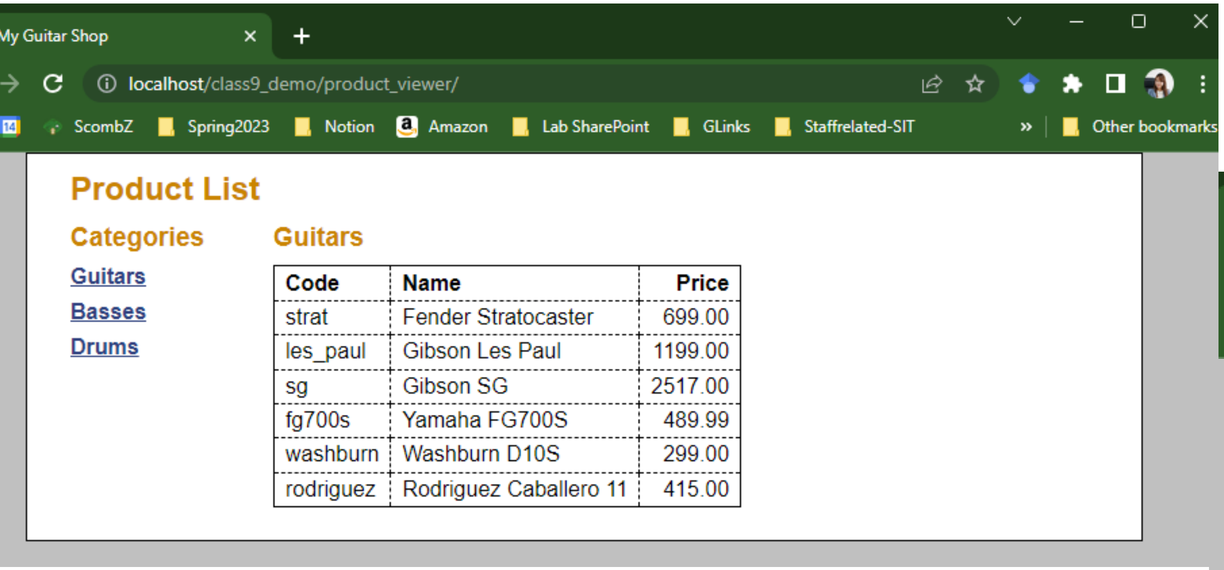
utf8mb4_general_ci

Database	Collation	Action
information_schema	utf8_general_ci	Check privileges
mysql	utf8mb4_general_ci	Check privileges
my_guitar_shop1	utf8mb4_general_ci	Check privileges
my_guitar_shop2	utf8mb4_general_ci	Check privileges
performance_schema	utf8_general_ci	Check privileges
phpmyadmin	utf8_bin	Check privileges
test	latin1_swedish_ci	Check privileges
test2	utf8mb4_general_ci	Check privileges
testam0000	utf8mb4_general_ci	Check privileges
testam0000_2	utf8mb4_general_ci	Check privileges

Total: 10

The Product Viewer application

- The user interface



Product List

Categories

Guitars

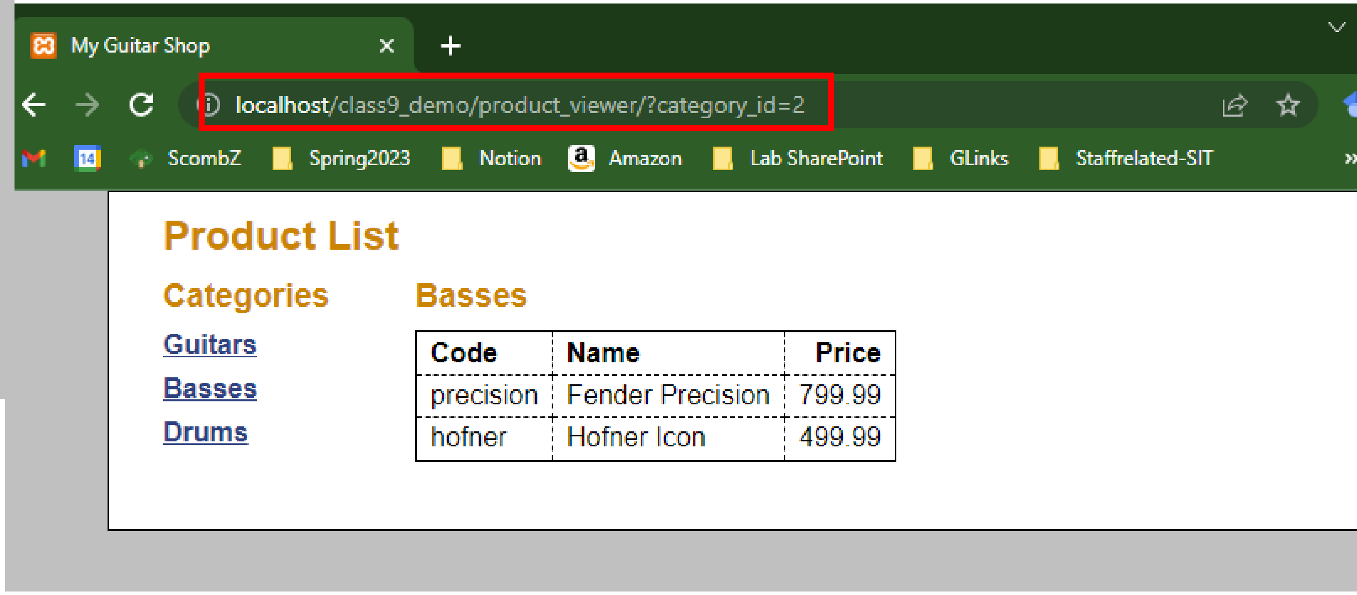
Basses

Drums

Guitars

Code	Name	Price
strat	Fender Stratocaster	699.00
les_paul	Gibson Les Paul	1199.00
sg	Gibson SG	2517.00
fg700s	Yamaha FG700S	489.99
washburn	Washburn D10S	299.00
rodriguez	Rodriguez Caballero 11	415.00

After user select Basses on the left menu bar



Product List

Categories

Guitars

Basses

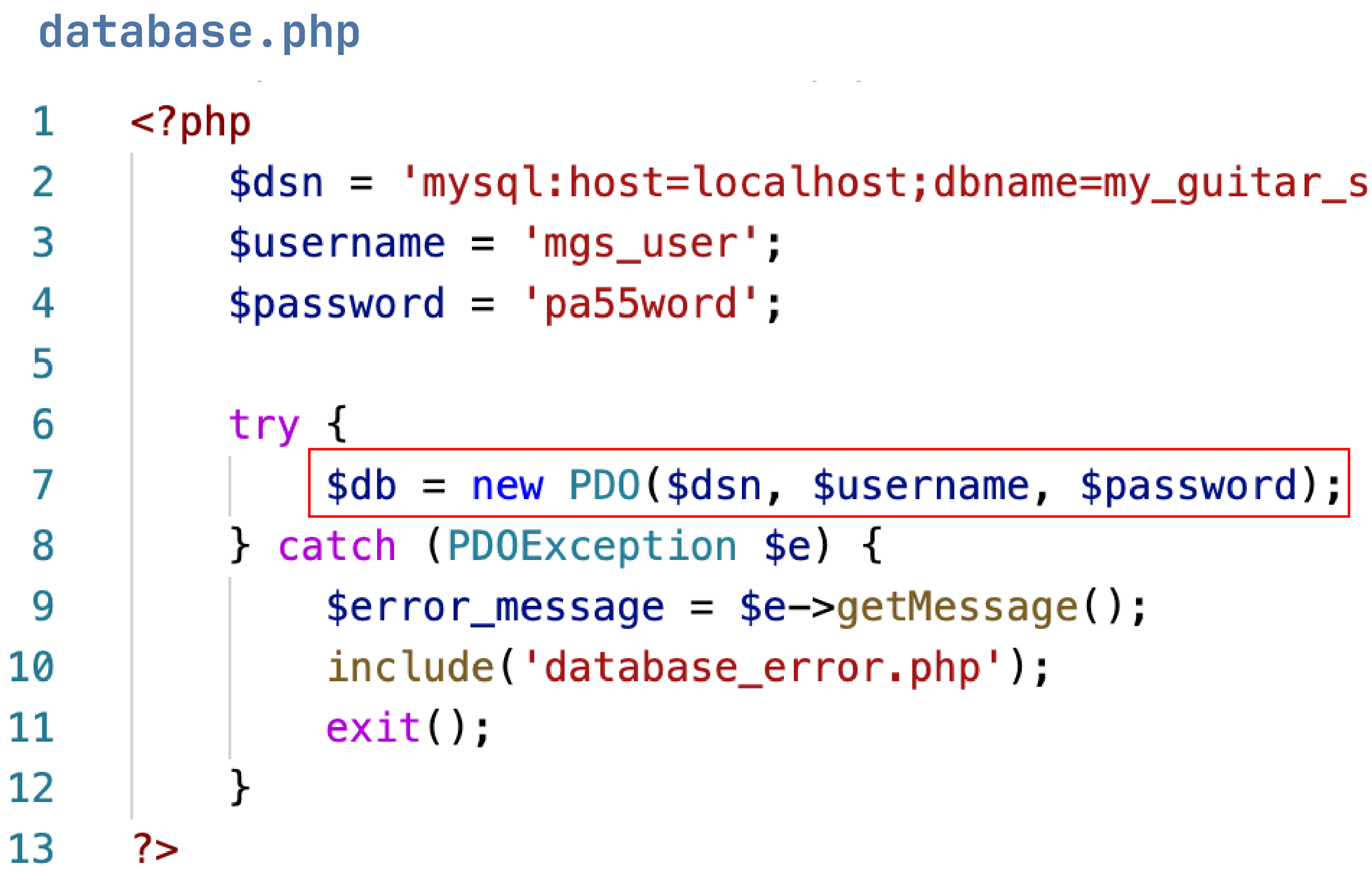
Drums

Basses

Code	Name	Price
precision	Fender Precision	799.99
hofner	Hofner Icon	499.99

Please refer to the explanation document along with the code files

The code



database.php

```php
<?php
    $dsn = 'mysql:host=localhost;dbname=my_guitar_shop1';
    $username = 'mgs_user';
    $password = 'pa55word';

    try {
        $db = new PDO($dsn, $username, $password);
    } catch (PDOException $e) {
        $error_message = $e->getMessage();
        include('database_error.php');
        exit();
    }
?>
```

Create a new PDO object

When an error occurs, display the error message

index.php (1/

```php
<?php
require_once('database.php');

// Get category ID
$category_id = filter_input(INPUT_GET, 'category_id', FILTER_VALIDATE_INT);
if ($category_id == NULL || $category_id == FALSE) {
    $category_id = 1;
}

// Get name for selected category
$queryCategory = 'SELECT * FROM categories
                  WHERE categoryID = :category_id';
$statement1 = $db->prepare($queryCategory);
$statement1->bindValue(':category_id', $category_id);
$statement1->execute();
$category = $statement1->fetch();
$category_name = $category['categoryName'];
$statement1->closeCursor();

// Get all categories
$queryAllCategories = 'SELECT * FROM categories
                       ORDER BY categoryID';
$statement2 = $db->prepare($queryAllCategories);
$statement2->execute();
$categories = $statement2->fetchAll();
$statement2->closeCursor();
```

Execute code in database.php

Get the category_id from $_GET

In case user hasn't yet clicked on category ID

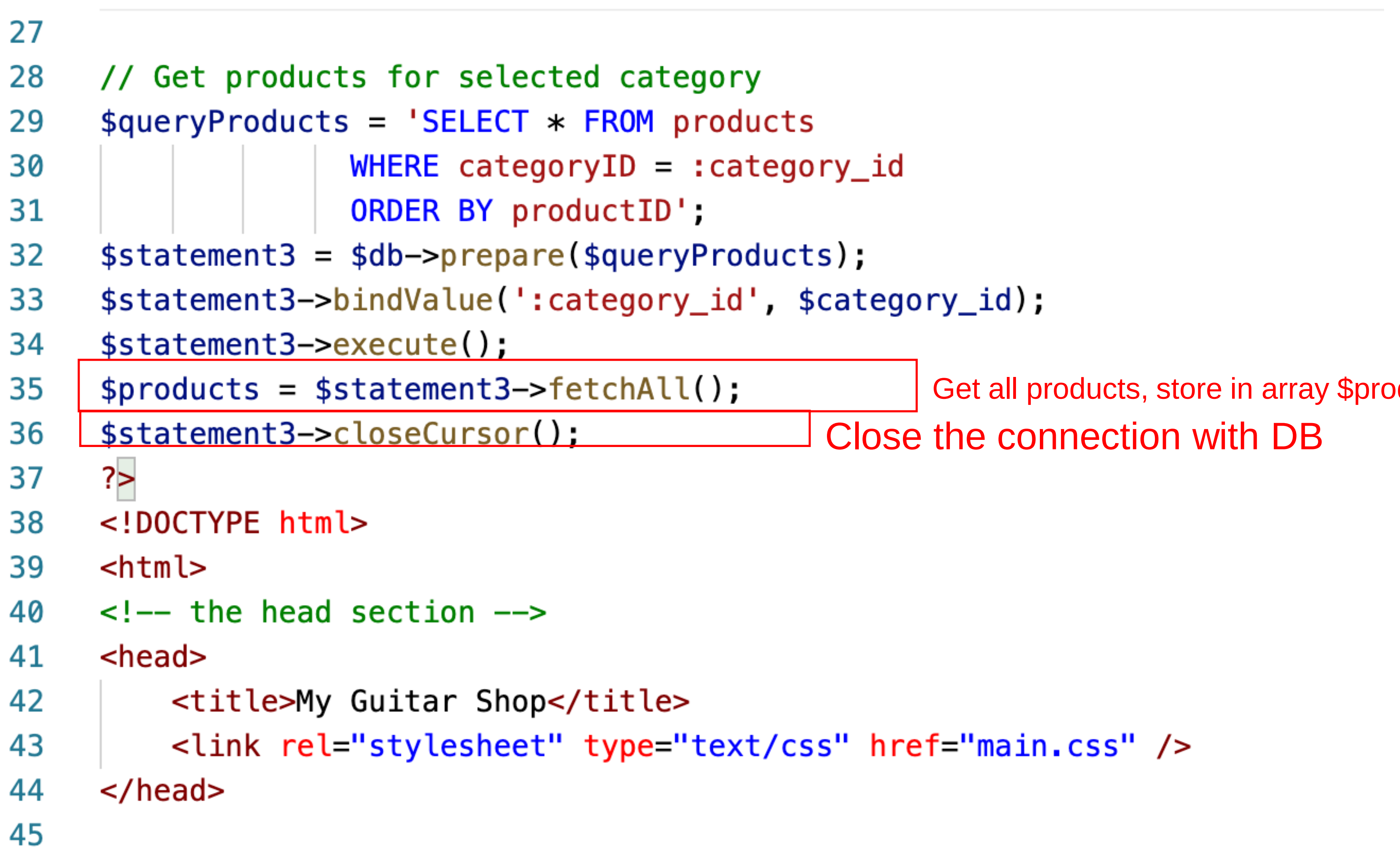
Get only the category name, store in $category_name

Close the connection with DB

Get all categories, store in array $categories

Close the connection with DB

`index.php` (2/3)



```
// Get products for selected category
$queryProducts = 'SELECT * FROM products
              WHERE categoryID = :category_id
              ORDER BY productID';
$statement3 = $db->prepare($queryProducts);
$statement3->bindValue(':category_id', $category_id);
$statement3->execute();
$products = $statement3->fetchAll();
$statement3->closeCursor();
?>
<!DOCTYPE html>
<html>
<!-- the head section -->
<head>
    <title>My Guitar Shop</title>
    <link rel="stylesheet" type="text/css" href="main.css" />
</head>
```

Get all products, store in array $products

Close the connection with DB

index.php (3/3)

```
    <!-- display a list of categories -->
    <h2>Categories</h2>
    <nav>
    <ul>
```

Show the category list that is stored in array $categories

```
        <?php foreach ($categories as $category) : ?>
        <li>
            <a href="?category_id=<?php echo $category['categoryID']; ?>">
                <?php echo $category['categoryName']; ?>
            </a>
        </li>
        <?php endforeach; ?>
    </ul>
    </nav>
</aside>

<section>
    <!-- display a table of products -->
```

Show the category name that is stored in $category_name

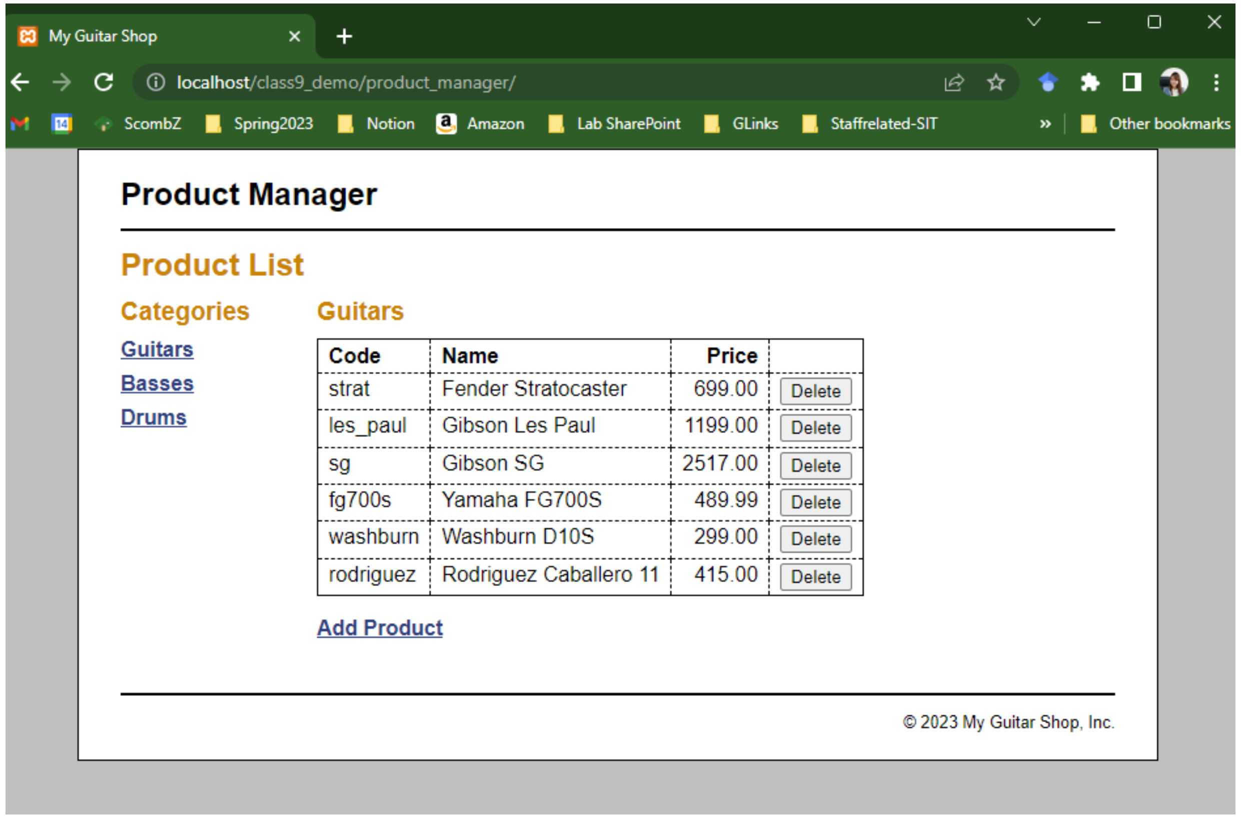
```
    <h2><?php echo $category_name; ?></h2>
    <table>
        <tr>
            <th>Code</th>
            <th>Name</th>
            <th class="right">Price</th>
        </tr>

        <?php foreach ($products as $product) : ?>
        <tr>
            <td><?php echo $product['productCode']; ?></td>
            <td><?php echo $product['productName']; ?></td>
            <td class="right"><?php echo $product['listPrice']; ?></td>
        </tr>
        <?php endforeach; ?>
    </table>
</section>
```

Show all products stored in array $product using foreach

The Product Manager application

- The user interface



My Guitar Shop

localhost/class9_demo/product_manager/

Product Manager

Product List

Categories

Guitars

Basses

Drums

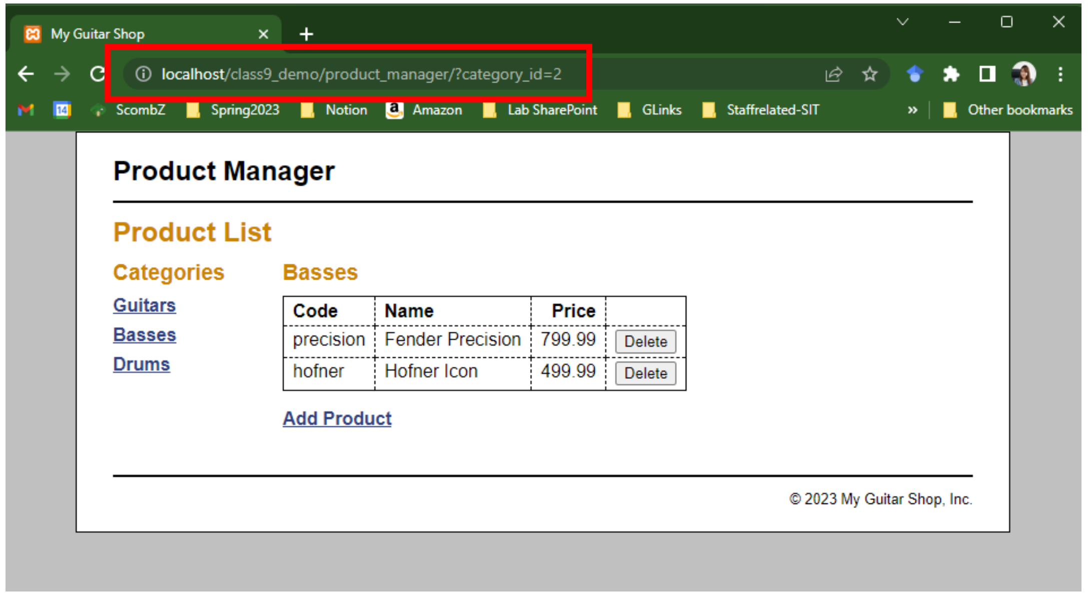
Guitars

Code	Name	Price	
strat	Fender Stratocaster	699.00	Delete
les_paul	Gibson Les Paul	1199.00	Delete
sg	Gibson SG	2517.00	Delete
fg700s	Yamaha FG700S	489.99	Delete
washburn	Washburn D10S	299.00	Delete
rodriguez	Rodriguez Caballero 11	415.00	Delete

Add Product

© 2023 My Guitar Shop, Inc.

After user select Basses on the left menu bar



My Guitar Shop

localhost/class9_demo/product_manager/?category_id=2

Product Manager

Product List

Categories

Guitars

Basses

Drums

Basses

Code	Name	Price	
precision	Fender Precision	799.99	Delete
hofner	Hofner Icon	499.99	Delete

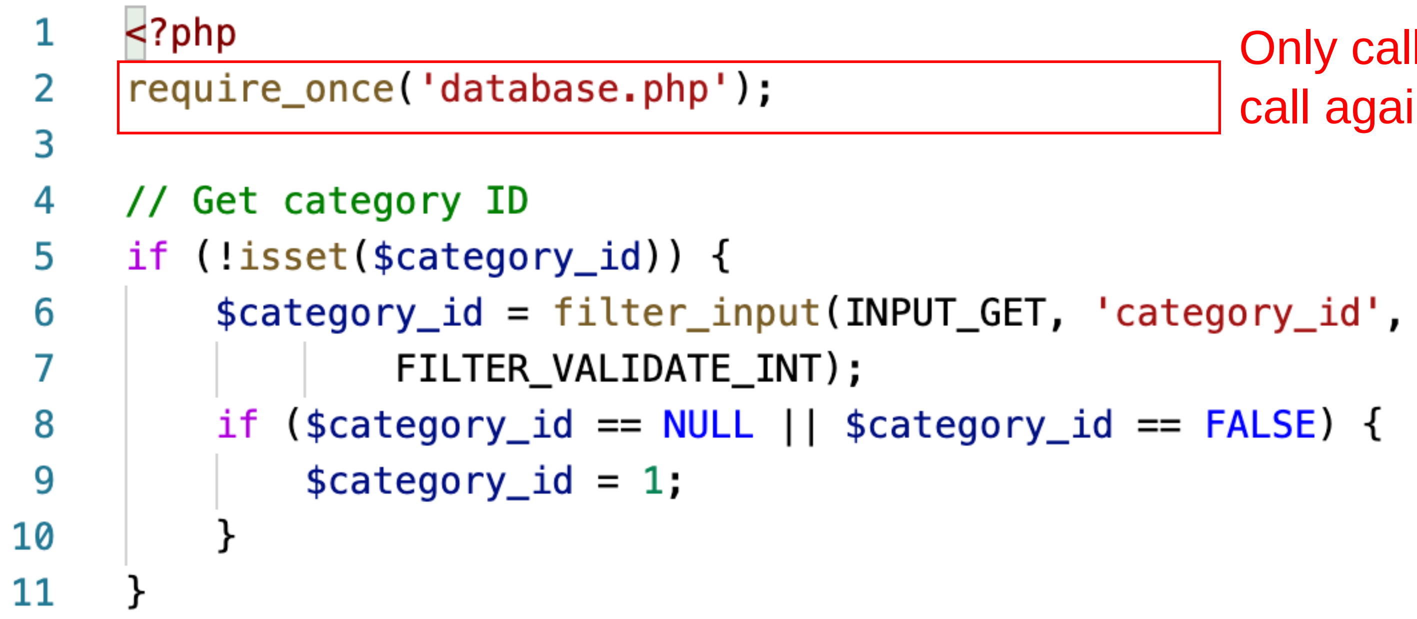
Add Product

© 2023 My Guitar Shop, Inc.

Please refer to the explanation document along with the code files

The code

`index.php` (showing only the part that is different)



```php
<?php
require_once('database.php');

// Get category ID
if (!isset($category_id)) {
    $category_id = filter_input(INPUT_GET, 'category_id',
            FILTER_VALIDATE_INT);
    if ($category_id == NULL || $category_id == FALSE) {
        $category_id = 1;
    }
}
```

Only call **`database.php`** once, so it will not call again if it is loaded in other pages `index.php` (showing only the part that is different)

```
<aside>
    <!-- display a list of categories -->
    <h2>Categories</h2>
    <nav>
    <ul>
        <?php foreach ($categories as $category) : ?>
        <li><a href=".?category_id=<?php echo $category['categoryID']; ?>">
                <?php echo $category['categoryName']; ?>
            </a>
        </li>
```

Notice the single dot .?category_id
= the URL starts with the current directory

```
<?php foreach ($products as $product) : ?>
<tr>
    <td><?php echo $product['productCode']; ?></td>
    <td><?php echo $product['productName']; ?></td>
    <td class="right"><?php echo $product['listPrice']; ?></td>
    <td><form action="delete_product.php" method="post">
        <input type="hidden" name="product_id"
               value="<?php echo $product['productID']; ?>">
        <input type="hidden" name="category_id"
               value="<?php echo $product['categoryID']; ?>">
        <input type="submit" value="Delete">
    </form></td>
</tr>
<?php endforeach; ?>
```

Use hidden field to pass predictID and categoryID to delete_product.php file

delete_product.php

```php
<?php
require_once('database.php');

// Get IDs
$product_id = filter_input(INPUT_POST, 'product_id', FILTER_VALIDATE_INT);
$category_id = filter_input(INPUT_POST, 'category_id', FILTER_VALIDATE_INT);

// Delete the product from the database
if ($product_id != false && $category_id != false) {
    $query = 'DELETE FROM products
              WHERE productID = :product_id';
    $statement = $db->prepare($query);
    $statement->bindValue(':product_id', $product_id);
    $success = $statement->execute();
    $statement->closeCursor();
}

// Display the Product List page
include('index.php');
```

Only call database.php once, so it will not call again if it is loaded in other pages

Get product ID and category ID with the $_POST array

Delete the row

add_product_form.php (showing only part that need explanation)

```
<?php
require('database.php');
$query = 'SELECT *
          FROM categories
          ORDER BY categoryID';
$statement = $db->prepare($query);
$statement->execute();
$categories = $statement->fetchAll();
$statement->closeCursor();
?>
<!DOCTYPE html>
<html>

<!-- the head section -->
<head>
    <title>My Guitar Shop</title>
    <link rel="stylesheet" type="text/css" href="main.css">
</head>

<!-- the body section -->
<body>
    <header><h1>Product Manager</h1></header>

    <main>
        <h1>Add Product</h1>
        <form action="add_product.php" method="post"
              id="add_product_form">
```

Retrieve categories into array $categories to be used later in drop down list in the form

```
            <label>Category:</label>
            <select name="category_id">
            <?php foreach ($categories as $category) : ?>
                <option value="<?php echo $category['categoryID
                    <?php echo $category['categoryName']; ?>
                </option>
            <?php endforeach; ?>
            </select><br>
```

Use POST method to pass 4 product variables to add_product.php

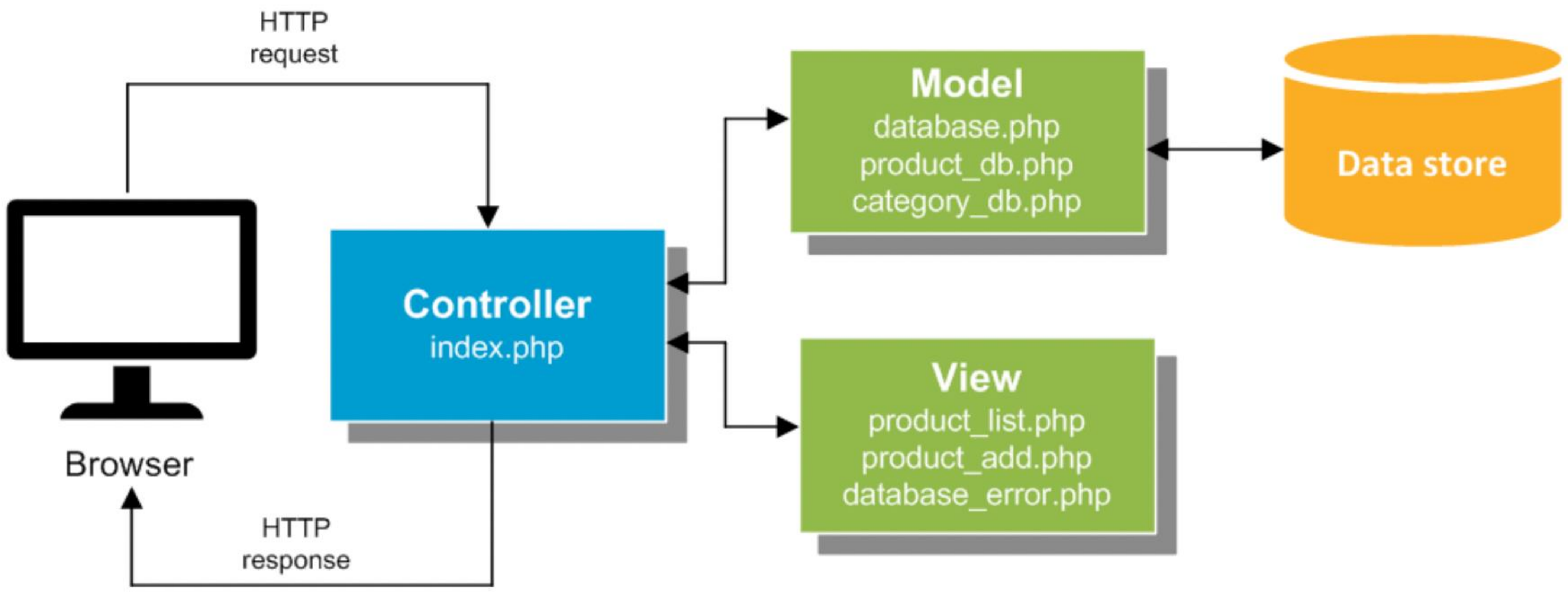
The MVC pattern

The MVC pattern

- The previous application demo mixes many codes
  - PHP code that accesses database
  - HTML codes that define the webpage
  - PHP code that controls the flow of application from one page to another
- This approach makes it difficult to code, test, debug, and maintain larger applications
- Professional web developers commonly use a programming pattern “MVC” Model-View-Controller pattern

Introduction to the MVC pattern

The MVC pattern



MVC

- The MVC (Model-View-Controller) pattern is commonly used to structure web applications that have significant processing requirements. That makes them easier to code and maintain.
  - The model consists of the PHP files that **represent the data of the application**. Normally, no HTML in model files.
  - The view consists of the **HTML and PHP** files that represent the **user interface** of the application.
  - The controller consists of the PHP files that **receive requests from users**, **get the appropriate data** from the model, and **return the appropriate views** to the users.

- Construct each layer to become as independent as possible
  - Ex: Web designers can work on user interface without any help from PHP programmers

Redirect requests

- `header()` function is used to *redirect* a request to another URL
  - Redirect a request = return a response to the browser to make a new request for the specified URL (if filename is not included, it use the default file)

- `include()` - display URL of the original page in the browser (forward)

- `header()` – display URL of the page you are redirected to in the browser (redirect)

Name	Description
`header($header)`	Sends an HTTP header to the browser. For example, you can use this function to send an HTTP Location header to the browser to redirect the browser to another URL.

The header function

```
header('Location: .');          //the current directory
header('Location: ..');         //navigate up one directory
header('Location: ./admin');    //navigate down one directory
header('Location: error.php');
```

How to redirect a request

```
if ($action == 'delete_product') {
    $product_id = filter_input(INPUT_POST, 'product_id', FILTER_VALIDATE_INT);
    if ($product_id != NULL && $product_id != FALSE) {
        delete_product($product_id);
        header("Location: .");
    }
}
```

How to redirect a request that includes a parameter

```
if ($action == 'delete_product') {
    $product_id = filter_input(INPUT_POST, 'product_id', FILTER_VALIDATE_INT);
    $category_id = filter_input(INPUT_POST, 'category_id', FILTER_VALIDATE_INT);
    if ($category_id != NULL && $category_id != FALSE && $product_id != NULL && $product_id != FALSE) {
        delete_product($product_id);
        header("Location: .?category_id=$category_id");
    }
}
```

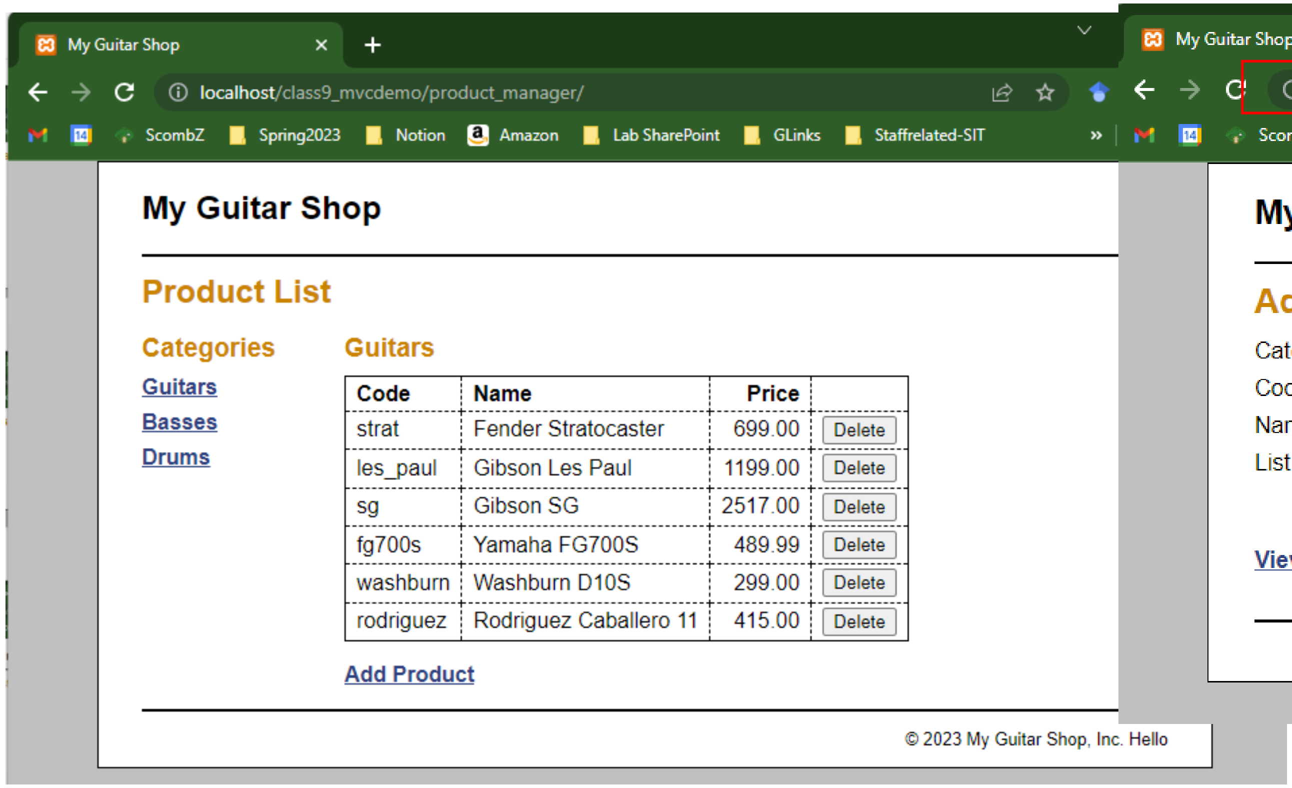
The modified Product Manager application

- Let’s look at the Product Manager application
- Modify to implement the MVC pattern
- Refactoring = modifying an application, where the code changes but the function stays the same

The codes are in `class10_mvcdemo` folder

The User Interface

Product List page



localhost/class9_mvcdemo/product_manager/

My Guitar Shop

Product List

Categories
- Guitars
- Basses
- Drums

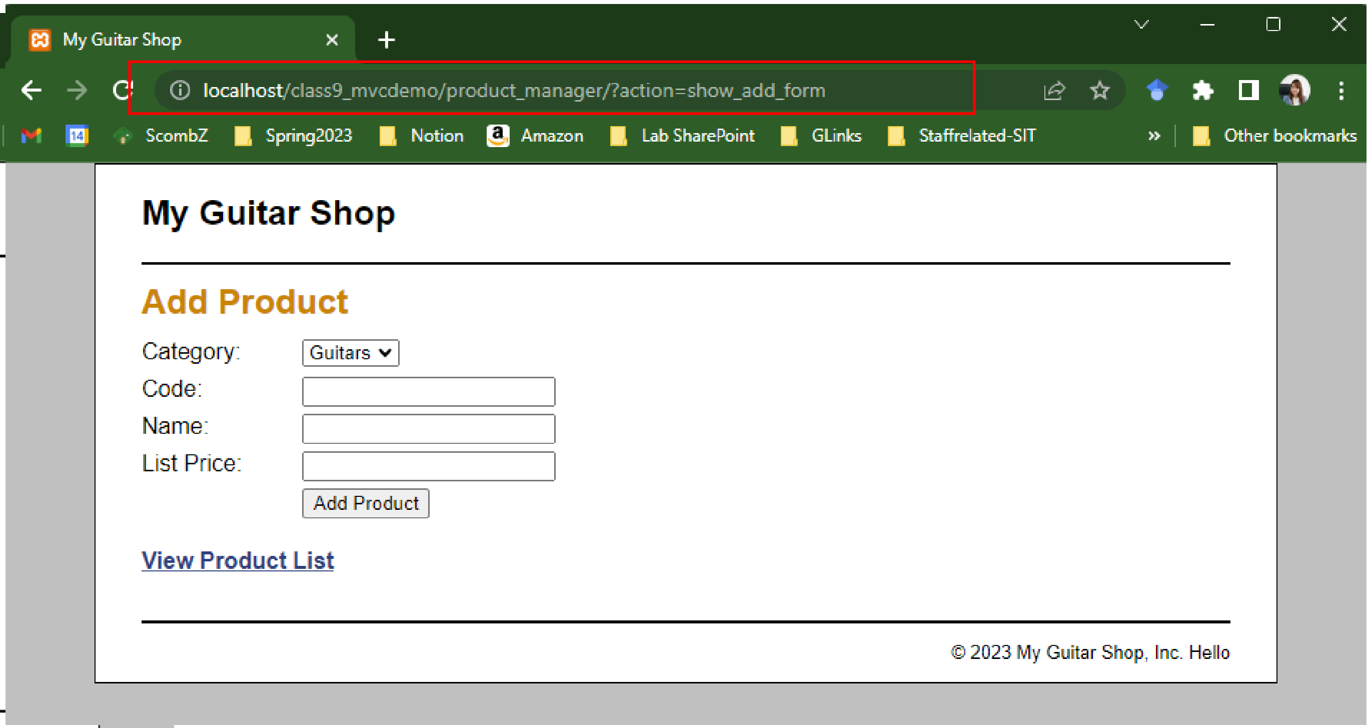
Guitars

Code	Name	Price	
strat	Fender Stratocaster	699.00	Delete
les_paul	Gibson Les Paul	1199.00	Delete
sg	Gibson SG	2517.00	Delete
fg700s	Yamaha FG700S	489.99	Delete
washburn	Washburn D10S	299.00	Delete
rodriguez	Rodriguez Caballero 11	415.00	Delete

Add Product

© 2023 My Guitar Shop, Inc. Hello

Add product page



localhost/class9_mvcdemo/product_manager/?action=show_add_form

My Guitar Shop

Add Product

Category:	Guitars
Code:	
Name:	
List Price:	
	Add Product

View Product List

© 2023 My Guitar Shop, Inc. Hello

The model

`model/category_db.php`

Return array that contain all rows and all columns in category table, sort by ID

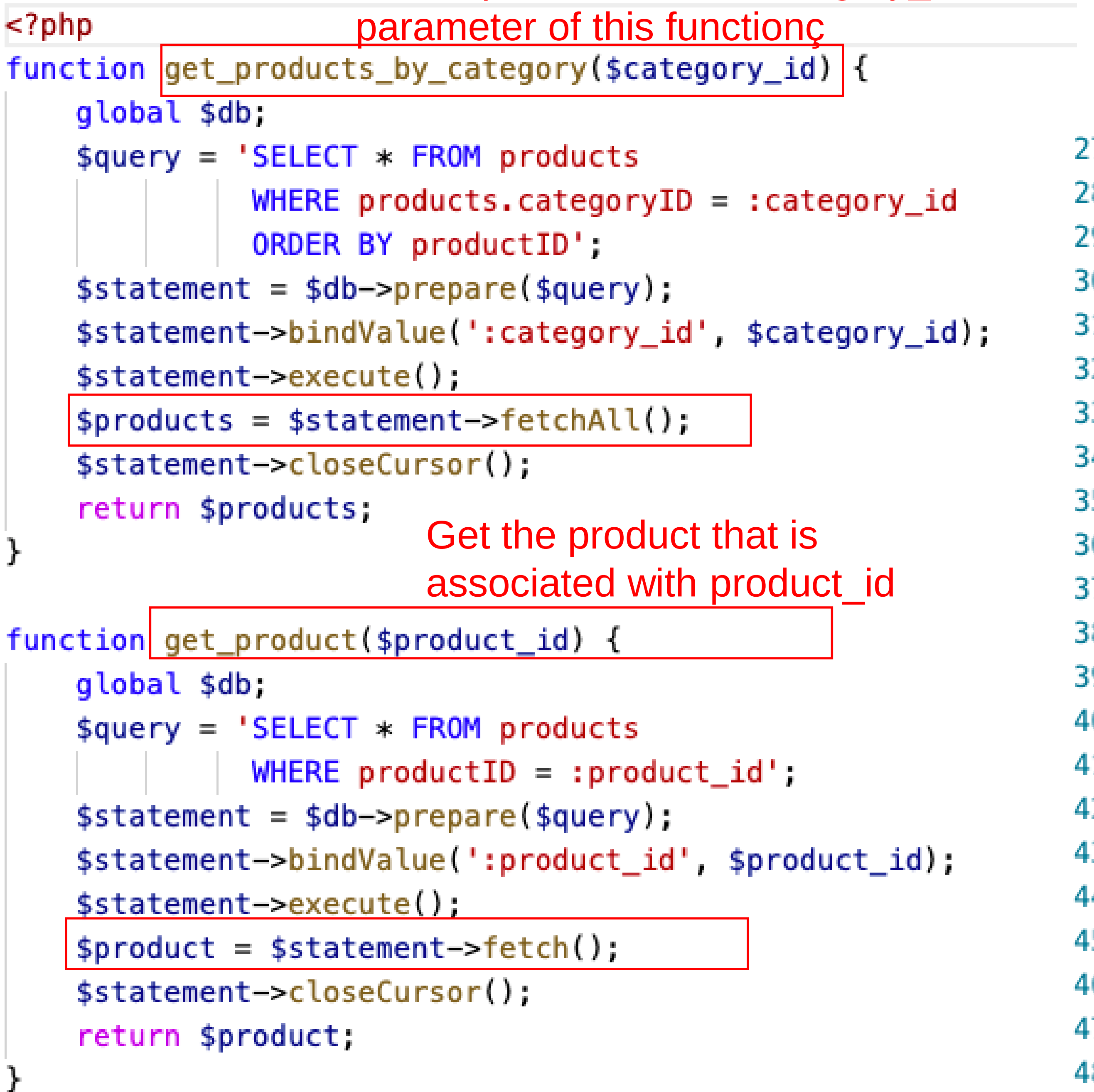
```php
<?php
function get_categories() {
    global $db;
    $query = 'SELECT * FROM categories
              ORDER BY categoryID';
    $statement = $db->prepare($query);
    $statement->execute();
    return $statement;
}

function get_category_name($category_id) {
    global $db;
    $query = 'SELECT * FROM categories
              WHERE categoryID = :category_id';
    $statement = $db->prepare($query);
    $statement->bindValue(':category_id', $category_id);
    $statement->execute();
    $category = $statement->fetch();
    $statement->closeCursor();
    $category_name = $category['categoryName'];
    return $category_name;
}
?>
```

Retrieve only one row from the category table, get the category name from that array and return only the category name model/product_db.php

Note the usage of prepare()

Get all products of the category_id parameter of this functionç

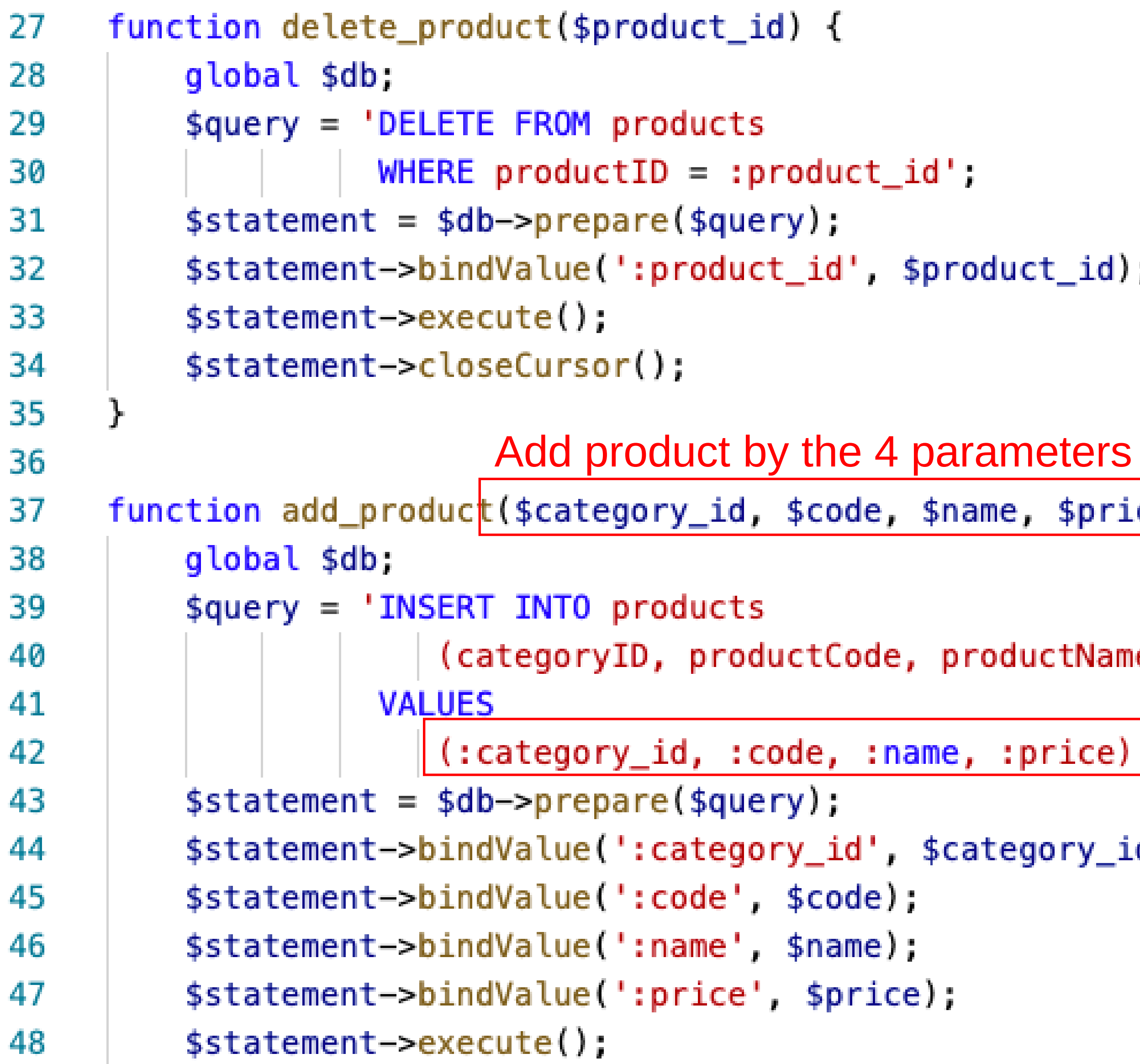


```php
<?php
function get_products_by_category($category_id) {
    global $db;
    $query = 'SELECT * FROM products
              WHERE products.categoryID = :category_id
              ORDER BY productID';
    $statement = $db->prepare($query);
    $statement->bindValue(':category_id', $category_id);
    $statement->execute();
    $products = $statement->fetchAll();
    $statement->closeCursor();
    return $products;
}

function get_product($product_id) {
    global $db;
    $query = 'SELECT * FROM products
              WHERE productID = :product_id';
    $statement = $db->prepare($query);
    $statement->bindValue(':product_id', $product_id);
    $statement->execute();
    $product = $statement->fetch();
    $statement->closeCursor();
    return $product;
}
```

Get the product that is associated with product_id

Delete a row by a specified product_id



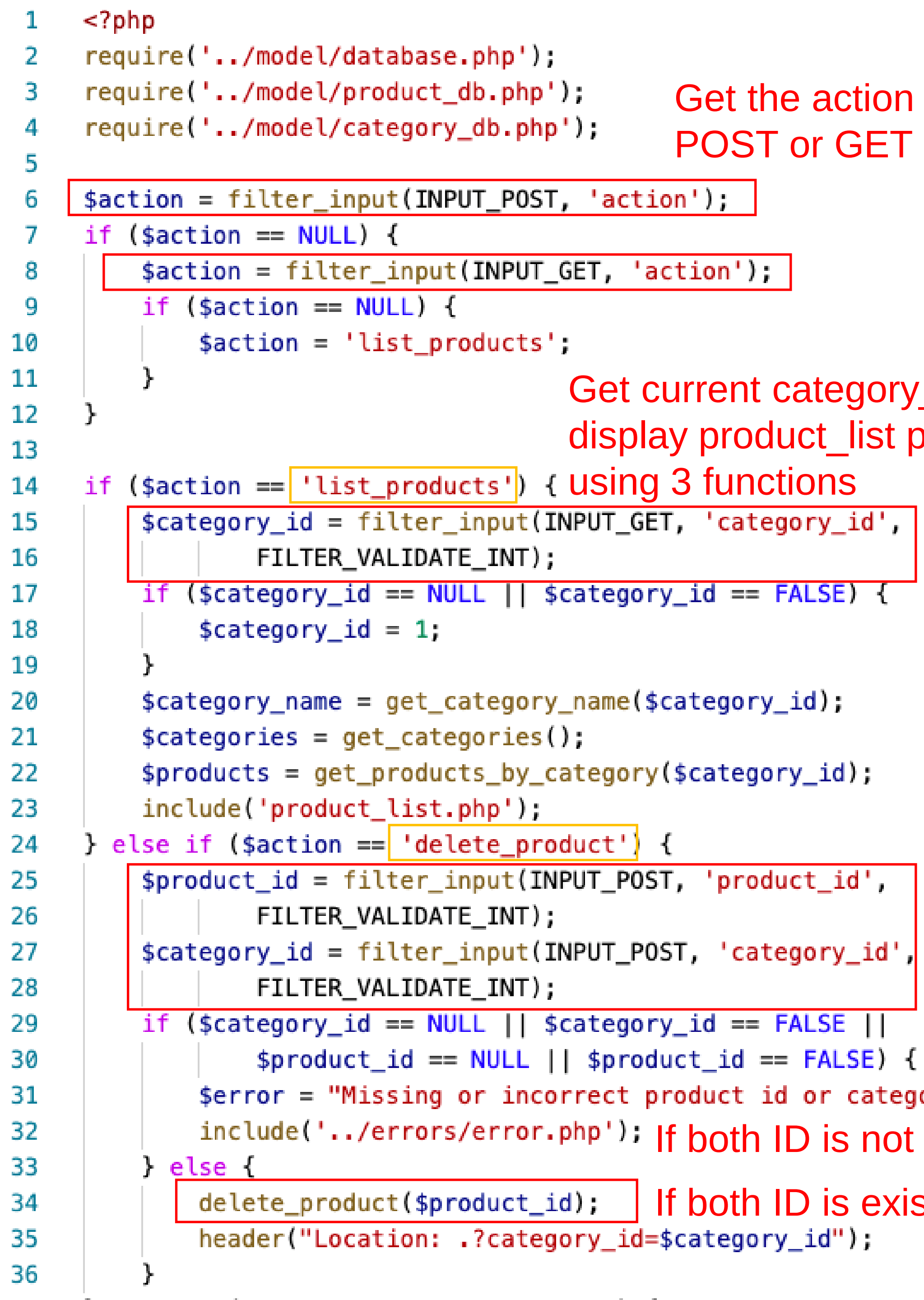
```php
function delete_product($product_id) {
    global $db;
    $query = 'DELETE FROM products
              WHERE productID = :product_id';
    $statement = $db->prepare($query);
    $statement->bindValue(':product_id', $product_id)
    $statement->execute();
    $statement->closeCursor();
}

function add_product($category_id, $code, $name, $pri
    global $db;
    $query = 'INSERT INTO products
                 (categoryID, productCode, productNam
              VALUES
                 (:category_id, :code, :name, :price)
    $statement = $db->prepare($query);
    $statement->bindValue(':category_id', $category_i
    $statement->bindValue(':code', $code);
    $statement->bindValue(':name', $name);
    $statement->bindValue(':price', $price);
    $statement->execute();
```

Add product by the 4 parameters

The controller

product_manager/index.php



```php
<?php
require('../model/database.php');
require('../model/product_db.php');
require('../model/category_db.php');

$action = filter_input(INPUT_POST, 'action');
if ($action == NULL) {
    $action = filter_input(INPUT_GET, 'action');
    if ($action == NULL) {
        $action = 'list_products';
    }
}

if ($action == 'list_products') {
    $category_id = filter_input(INPUT_GET, 'category_id',
            FILTER_VALIDATE_INT);
    if ($category_id == NULL || $category_id == FALSE) {
        $category_id = 1;
    }
    $category_name = get_category_name($category_id);
    $categories = get_categories();
    $products = get_products_by_category($category_id);
    include('product_list.php');
} else if ($action == 'delete_product') {
    $product_id = filter_input(INPUT_POST, 'product_id',
            FILTER_VALIDATE_INT);
    $category_id = filter_input(INPUT_POST, 'category_id',
            FILTER_VALIDATE_INT);
    if ($category_id == NULL || $category_id == FALSE ||
            $product_id == NULL || $product_id == FALSE) {
        $error = "Missing or incorrect product id or category id.";
        include('../errors/error.php');
    } else {
        delete_product($product_id);
        header("Location: .?category_id=$category_id");
    }
```

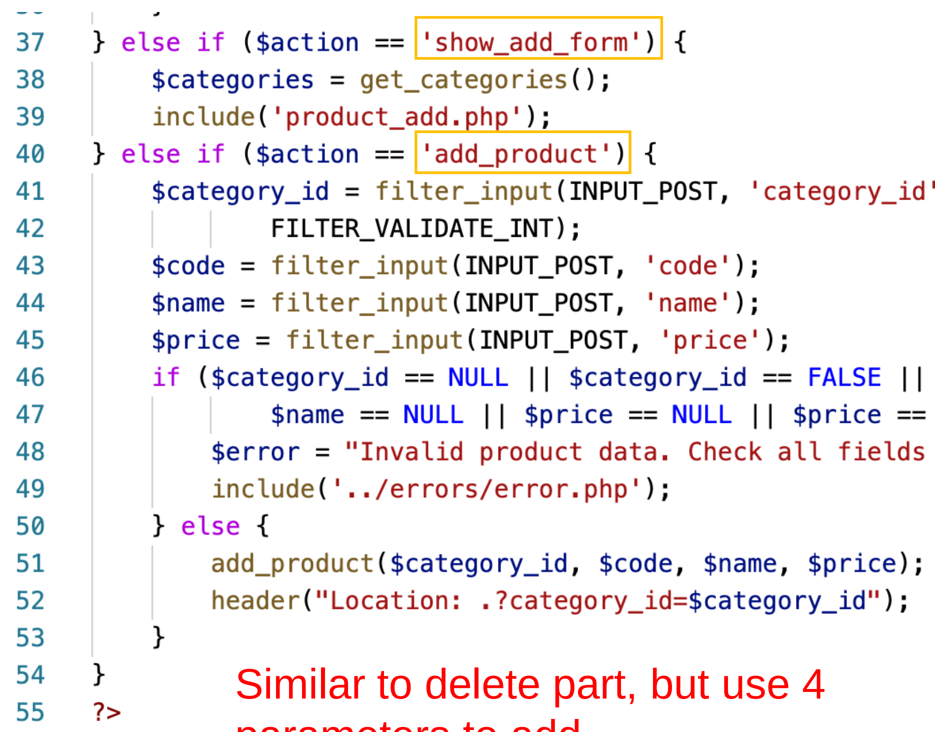
Get the action parameter from POST or GET

Get current category_id and display product_list page using 3 functions

Get product ID, category ID for the product to delete from $_POST array

If both ID is not exist, show error

If both ID is exist, delete the product



```php
} else if ($action == 'show_add_form') {
    $categories = get_categories();
    include('product_add.php');
} else if ($action == 'add_product') {
    $category_id = filter_input(INPUT_POST, 'category_id'
            FILTER_VALIDATE_INT);
    $code = filter_input(INPUT_POST, 'code');
    $name = filter_input(INPUT_POST, 'name');
    $price = filter_input(INPUT_POST, 'price');
    if ($category_id == NULL || $category_id == FALSE ||
            $name == NULL || $price == NULL || $price ==
        $error = "Invalid product data. Check all fields
        include('../errors/error.php');
    } else {
        add_product($category_id, $code, $name, $price);
        header("Location: .?category_id=$category_id");
    }
}
?>
```

Similar to delete part, but use 4 parameters to add

The view

view/header.php

```
<!DOCTYPE html>
<html>
<!-- the head section -->
<head>
    <title>My Guitar Shop</title>
    <link rel="stylesheet" type="text/css"
            href="../main.css">
</head>

<!-- the body section -->
<body>
<header><h1>My Guitar Shop</h1></header>
```

view/footer.php

```
<footer>
    <p class="copyright">
        &copy; <?php echo date("Y"); ?> My Guitar Shop, Inc.
    </p>
</footer>
</body>
</html>
```

- Header and footer is not strictly a part of the MVC pattern, but using these kind of files can adhere to DRY (don’t repete yourself) principle
- Ex: to change the title for every page, you can just change in header.php file, or similar for copyright notice.

a view

product_manager/product_list.php

```
<?php include '../view/header.php'; ?>
<main>
    <h1>Product List</h1>

    <aside>
        <!-- display a list of categories -->
        <h2>Categories</h2>
        <nav>
        <ul>
        <?php foreach ($categories as $category) : ?>
            <li>
            <a href="?category_id=<?php echo $category['categoryID']; ?>">
                <?php echo $category['categoryName']; ?>
            </a>
            </li>
        <?php endforeach; ?>
        </ul>
        </nav>
    </aside>
```

Show list of categories from array $categories

```

    <section>
        <!-- display a table of products -->
        <h2><?php echo $category_name; ?></h2>
        <table>
            <tr>
                <th>Code</th>
                <th>Name</th>
                <th class="right">Price</th>
                <th>&nbsp;</th>
            </tr>
            <?php foreach ($products as $product) : ?>
            <tr>
                <td><?php echo $product['productCode']; ?></td>
                <td><?php echo $product['productName']; ?></td>
                <td class="right"><?php echo $product['listPrice']; ?></td>
                <td><form action="." method="post">
                    <input type="hidden" name="action"
                           value="delete_product">
                    <input type="hidden" name="product_id"
                           value="<?php echo $product['productID']; ?>">
                    <input type="hidden" name="category_id"
                           value="<?php echo $product['categoryID']; ?>">
                    <input type="submit" value="Delete">
                </form></td>
            </tr>
            <?php endforeach; ?>
        </table>
        <p><a href="?action=show_add_form">Add Product</a></p>
        <p class="last_paragraph"><a href="?action=list_categories">List Categories</a></p>
    </section>
</main>
<?php include '../view/footer.php'; ?>


```

Display product

Delete a product by passing productid and category ID in hidden form

In add product form page, we still need PDOStatement object to display drop down list of category a view

product_manager/product_add.php

```
<?php include '../view/header.php'; ?>
<main>
    <h1>Add Product</h1>
    <form action="index.php" method="post" id="add_product_form">
        <input type="hidden" name="action" value="add_product">

        <label>Category:</label>
        <select name="category_id">
        <?php foreach ( $categories as $category ) : ?>
            <option value="<?php echo $category['categoryID']; ?>">
                <?php echo $category['categoryName']; ?>
            </option>
        <?php endforeach; ?>
        </select>
        <br>

        <label>Code:</label>
        <input type="text" name="code" />
        <br>

        <label>Name:</label>
        <input type="text" name="name" />
        <br>

        <label>List Price:</label>
        <input type="text" name="price" />
        <br>

        <label>&nbsp;</label>
        <input type="submit" value="Add Product" />
        <br>
    </form>
    <p class="last_paragraph">
        <a href="index.php?action=list_products">View Product List</a>
    </p>

</main>
<?php include '../view/footer.php'; ?>
```

Use post method to add

action=add_product

Show categories from $categories in drop down list

Category_id is submitted with the form
But display category name to user

Do not want to display the name of file in URL

The Product Catalog application

- The Product Manager application – for admin
- The Product Catalog application – for end users of the website

Explain the code & Demo

End of class 10

Next week: Cookies and sessions