**[IS113] Extra Exercises - Database Interaction (Set A)**

**Resource Download**

* To download the **resource files**, go to: <https://smu.sg/hob>
* You will see THREE (3) folders (**cat, cat2, cat3**).
  + Download & place the folders inside **<web root>/is113/extra10/** on your local computer.
  + You should be able to see the following directory/file structure on your local computer:

o **<web root>/is113/extra10/**:

* cat/ (folder)
  + Cat.php (file)
  + CatDAO.php (file)
  + data.php (file)
  + display.php (file)
* cat2/ (folder)
  + test.php (file)
* cat3/ (folder)
  + ConnectionManager.php (file)
  + create.sql (file)

**Set B** can be accessed at: <https://smu.sg/hov>

**NOTE:** If you spot any mistakes/errors in the questions, please contact Kyong & Layfoo at {kjshim, lfthiang}@smu.edu.sg.

**Question 1 - Cat**

Go to **<web root>/is113/extra10/cat/.** Complete the following **Parts A** and **B**.

In the last few weeks, we have been using **Associative Arrays** or **Indexed Arrays** to represent and store “things” such as persons, books, fruits, students, etc.

For example (see **data.php** file):

* **$cats** is an **Indexed Array** where each cat is represented and stored as an **Associative Array**.
  + The **key** is an **attribute** of a cat (e.g. name, age, gender, status).

|  |
| --- |
| **$cats** = [  // 1st cat  [  'name' => 'Dirty',  'age' => 12,  'gender' => 'M',  'status' => 'A'  ],  // 2nd cat  [  'name' => 'Filthy',  'age' => 7,  'gender' => 'F',  'status' => 'A'  ],  // 3rd cat  [  'name' => 'Boring',  'age' => 3,  'gender' => 'M',  'status' => 'A'  ]  ***// ... and so on***  ]; |

As shown above, we previously created new **cats** and stored them in an **Indexed Array**. Another way to **store** cat information is by using **Classes/Objects**.

**Part A (\*)**

Complete **Cat.php** such that:

* **Cat** class defines what a cat should be. Every **Cat object** has the following FOUR (4) attributes:
  + **name** (e.g. 'Dirty')
  + **age** (e.g. 12)
  + **gender** (e.g. 'M' indicating ***male***, 'F' indicating ***female***)
  + **status** (e.g. 'A' indicating ***available***, 'P' indicating ***pending adoption***)
* Implement its constructor so that it takes in values for the FOUR (4) attributes
* Implement **Getter** methods for all of the attributes.

Great! Now that we have **Cat.php** that defines what every **cat** should look like, we can use this **definition** to create **new cats**!

|  |
| --- |
|  |

We declare a **DAO** (Data Access Object) **class** inside **CatDAO.php** file.

|  |
| --- |
| **CatDAO.php** |
| <?php  require\_once 'Cat.php';  class CatDAO {  private **$cats**;  // constructor  public function \_\_construct() {  **// Pre-populate static data**  **$this->cats** = [  new Cat('Dirty', 12, 'M', 'A'),  new Cat('Filthy', 7, 'F', 'A'),  new Cat('Boring', 3, 'M', 'A'),  new Cat('Needy', 3, 'M', 'P'),  new Cat('Lazy', 1, 'F', 'P')  ];  }  // whoever needs $cats, call this method off CatDAO object  **public function getCats() {**  **return $this->cats;**  **}**  }  ?> |

* **CatDAO** class’s constructor pre-populates **$cats** **Array** with FIVE (5) **Cat** objects.
* Via its **getCats()** public method, **CatDAO** class allows other PHP pages to access the **$cats** **Array**. For instance, later on, **display.php** will need to display all the **cats**.

**Part B (\*\*)**

Complete **display.php** file. It must show all cats’ information as shown below:

|  |
| --- |
| **display.php** |
|  |

Where can **display.php** obtain the information about all the **cats**?

* From **CatDAO.php** file!!!
* It **requires** **CatDAO.php** file (e.g. **require\_once**).
* It needs to **create a new CatDAO object**.
* Using this new **CatDAO object**, it can call all **public methods** of **CatDAO** class.
  + For now, we only have TWO (2) public method **getCats()** and the constructor.
  + The **getCats()** method will return an **Indexed Array** containing **Cat objects**.
* Can you now see how… the **concern of data retrieval** (**CatDAO.php**) is completely separated from **displaying of data** (**display.php**)?

|  |
| --- |
| **display.php** |
| <?php  **require\_once 'CatDAO.php';**  **$dao = new CatDAO();**  **$cats** = $dao->getCats(); **// $cats is an Indexed Array of Cat objects**  ?>  <html>  <body>  <h1>Our Cats</h1>  <table border='1'>  <tr>  <th>Name</th>  <th>Age</th>  <th>Gender</th>  <th>Status</th>  </tr>  <?php  foreach(**$cats** as $cat\_object) {  echo "  <tr>  <td>  **{$cat\_object->getName()}**  ***... and more code below...*** |

**Question 2 - Cat2**

Go to **<web root>/is113/extra10/cat2/.** Complete the following **Parts A** and **B**.

Copy the following files from **<web root>/is113/extra10/cat/** folder into the current folder **<web root>/is113/extra10/cat2/**.

* Cat.php
* CatDAO.php
* display.php

**Part A (\*\*)**

Edit **CatDAO.php** file:

* Implement **getCatsByStatus($status)** public method.
* This method takes ONE (1) parameter, **$status**, where the valid values are:
  + 'A'
  + 'P'
* Given the parameter value of 'A', it is to:
  + Look for one or more **Cat** objects in **$cats** **Array** where each cat’s **status** is 'A'
  + Insert all matching **Cat** objects into an Indexed Array and return it.
* Likewise, for the parameter value of 'P', it is to perform the same but this time, the returned **Indexed Array** will contain all **Cat** objects where each cat’s **status** is 'P'.
* **Test Cases**
  + In your web browser, open **test.php**.
  + There are TWO (2) test cases inside.
  + Each test case must produce correct results. Verify that your new method returns the correct results.

**Part B (\*\*)**

Edit **display.php** file.

When the page loads in a web browser **for the first time**, it must display display all cats.

|  |
| --- |
| **display.php** |
|  |

**Next**, when the user selects **Available** as the filtering value, and clicks on **Filter SUBMIT button**, the page displays:

|  |
| --- |
| **display.php** |
|  |

**NOTE:** The page must remember and pre-select the user’s form input **“Filter by Status”**. For instance, in the above example, “**Available**” option is pre-selected in the drop-down list.

**Next**, when the user selects **Pending Adoption** as the filtering value, and clicks on **Filter SUBMIT button**, the page displays:

|  |
| --- |
| **display.php** |
|  |

**NOTE:** The page must remember and pre-select the user’s form input **“Filter by Status”**. For instance, in the above example, “**Pending Adoption**” option is pre-selected in the drop-down list.

**Question 3 - Cat3**

Go to **<web root>/is113/extra10/cat3/.** Complete the following **Parts A** and **B**.

Copy the following files from **<web root>/is113/extra10/cat2/** folder into the current folder **<web root>/is113/extra10/cat3/**.

* Cat.php
* display.php

I’m VERY SORRY to share with you... that our cat **Dirty** passed away last night... (a moment of silence please).



**Layfoo**, the adoption agency head, would like to request you (programmer) to **not display** Dirty’s information as he is no longer with us. So, what are you (programmer) now going to do?

* Go open up page **CatDAO.php**.
* Go into the **constructor** method.
* Manually remove the line of code creating a new **Cat object** corresponding to **Dirty**.

|  |
| --- |
| **CatDAO.php** |
| <?php  require\_once 'Cat.php';  class CatDAO {  private $cats;  // constructor  public function \_\_construct() {  // Pre-populate static data  $this->cats = [  **new Cat('Dirty', 12, 'M', 'A'), // remove this line... Dirty died**  new Cat('Filthy', 7, 'F', 'A'),  new Cat('Boring', 3, 'M', 'A'),  new Cat('Needy', 3, 'M', 'P'),  new Cat('Lazy', 1, 'F', 'P')  ];  } |

**WAIT A SECOND…**

1. What if there are **users** currently accessing my website? Will they get an error message WHILE I make this code fix?
2. What if **more cats die** in the coming weeks/months? Will I have to change my **code** AGAIN?
3. What if there are **more cats** coming into the agency? Will I have to change my **code** AGAIN?

|  |
| --- |
|  |

Wow! Our data are **static**! It’s **hard-coded**… INSIDE our code (**CatDAO.php**) OMG!!!

Honestly, I (programmer) don’t think I can afford taking my website down EVERY TIME … the data need to be updated. Especially given Layfoo’s testimonial:

* On average, 3-4 new cats are abandoned and dropped off at his agency office;
* On average, 1-2 cats die every week;
* On average, 2-3 new adoptions happen per month.

Now, we foresee… **frequent data updates**. Definitely, you won’t want to keep updating data **hard-coded** INSIDE your PHP code!

**So, how do we tackle this problem?**

|  |
| --- |
|  |

* Above, we completely separated out **data storage** away from PHP code.
* **Data Storage** is now handled exclusively by the **MySQL Database**.
* **CatDAO.php** now can focus on:
  1. **Interacting with the Database**
     + Create (C)
     + Read (R)
     + Update (U)
     + Delete (D)
  2. Store retrieved data (from MySQL Database) into **variables** (e.g. Class objects) so that **other PHP files** can access the data and display, etc.

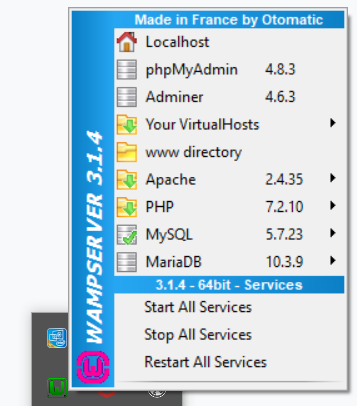
**Part A (Creating a Database)**

Agency Head Layfoo populates the MySQL database, and he will now maintain the database. you (programmer) do NOT have to worry about updating the data at all.

* During the software (web app) development, you NEVER have your code interact with the **production Database**. What if you make mistakes in your code… that **wipes out** the data in the **production Database**? → DIE ALREADY!!!
* Hence, typically, software developers will **create a replica of the production Database** (if it’s too large, then take a subset of it) in their **local development/testing environment**.

Since you’re still developing this website for the Cat Adoption Agency… you will have to create a replica of Layfoo’s MySQL database… locally on your computer.

1. In your web browser, go to: [**http://localhost/phpmyadmin/**](http://localhost/phpmyadmin/)
   1. Alternatively, you can go to **WAMP icon** and **LEFT-CLICK** on the icon.

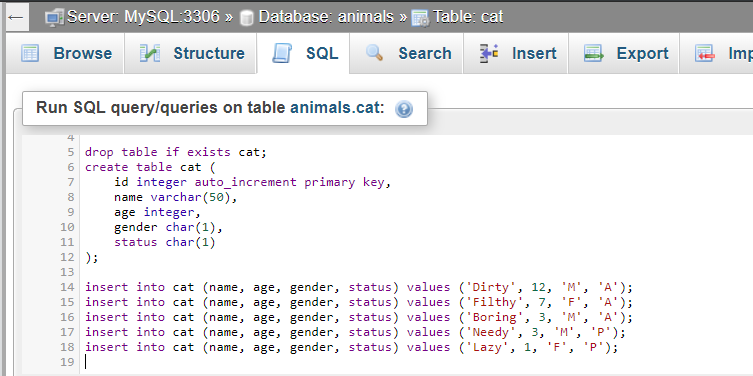


* 1. AND click on **phpMyAdmin**. It will pop up a web browser tab/window with the same URL **http://localhost/phpmyadmin**.
  2. If you are using MAMP or something else other than WAMP (or if you have changed WAMP’s configuration), then the URL will be different.

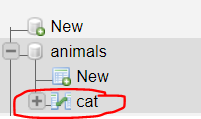
1. Sign in with:
   1. Username: **root**
   2. Password: <leave it empty> or something else depending on your setting
2. Click on **SQL** link at the top menu bar:



1. Open the file **create.sql**. Layfoo provided his MySQL database ‘animals’ **dump** in this SQL file. Copy the entire content and paste it into **PHPMyAdmin** SQL pane.



1. Click on **Go** button. The queries will be executed.
2. On the **LEFT SIDE** of the window, expand **animals** database and you should see a new table **cat**.



1. Click on **cat** table. You should see FIVE (5) rows of data inside:



Our **Database** is now ready!

NOW, we have to find a way to CONNECT to it.

**Part B (Connecting to Database)**

Open **CatDAO.php**. This new **CatDAO** is very similar to the previous **CatDAO**. Both make available these two **public methods** for OTHER PHP files to call:

* getCats()
* getCatsByStatus($status)

Both methods return exactly the same things (Indexed Array of Cat objects). Hence, the same **display.php** file should work without any coding changes.

But there ARE some changes. Let’s have a look.

**Connecting to Database**

**CatDAO.php** requires another file **ConnectionManager.php**.

|  |
| --- |
| **ConnectionManager.php** |
| <?php  class ConnectionManager {  **public function connect()** {  $servername = 'localhost';  $username = 'root';  $password = '';  $dbname = 'animals';    // Create connection  $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);  $conn->setAttribute(PDO::ATTR\_ERRMODE, PDO::ERRMODE\_EXCEPTION); // if fail, exception will be thrown  // Return connection object  return $conn;  }  } |

**ConnectionManager.php** handles **Database Connection** via PHP’s **PDO** (PHP Data Object) **extension**.

→ You do NOT need to know the details of **PDO** implementation.

→ You DO need to know how to **configure** the following in **ConnectionManager.php**:

* Server/Host name (e.g. **localhost**)
* Username: **Your DB username** (“root” by default)
* Password: **Your DB password** (empty by default)
* DB (instance) name (e.g. In our exercise, it is **animals**)

→ You do NOT need to memorize the code. But you are expected to know how to configure the above.

To use this from another PHP file, we must do the following:

* Import **ConnectionManager.php** file (e.g. **require\_once**).
* Create a new **ConnectionManager** object.
* Off that object, call the public method **connect()**.
* This **connect()** method will make a connection to the specified Database and **return a PDO object** **$conn**.
* We can then use this object **$conn** to perform SQL operations (SELECT, INSERT, UPDATE, etc.).

**Part C (Retrieving Data from Database)**

Let’s revisit **CatDAO.php**. It provides two **public methods** for OTHER PHP files to call:

* getCats()
* getCatsByStatus($status)

The same **display.php** file from **Question 2** should work without any modification.

Let’s have a look at getCats(). It consists of SIX (6) steps. In Step 6, it returns an Indexed Array of Cat objects (if any matching rows were found in the database table **cat**).

|  |
| --- |
| **CatDAO.php | Method getCats()** |
| public function getCats() {    **// STEP 1**  **// Connect to database 'animals'**  // See 'ConnectionManager.php'  $connMgr = new ConnectionManager();  $conn = $connMgr->connect();  **// STEP 2**  **// Prepare SQL statement**  $sql = "**SELECT name, age, gender, status FROM cat**";  $stmt = $conn->prepare($sql);  **// STEP 3**  **// Run SQL**  $stmt->execute();  $stmt->setFetchMode(PDO::FETCH\_ASSOC);  // Retrieve each row as an Associative Array  **// STEP 4**  **// Retrieve query results - ONE ROW AT A TIME**  **$cats = [];**  // Initialize an empty (indexed) Array  // so I can return it to whoever called this function  // Use WHILE loop to loop through  **while** ($row = **$stmt->fetch()** ) {  $cat = **new Cat(**  **$row['name'],**  **$row['age'],**  **$row['gender'],**  **$row['status']**  **)**;  **$cats[] = $cat;**  }    **// STEP 5**  **// Close DB Connection & SQL Statement**  $stmt = null;  $conn = null;  **// STEP 6**  **// YAY! We're ready to return the array!**  return $cats;  } |

Let’s have a look at getCatsByStatus($status). It consists of SIX (6) steps. In Step 6, it returns an Indexed Array of Cat objects **whose ‘status’** is equal to the **parameter** **$status**.

|  |
| --- |
| **CatDAO.php | Method getCatsByStatus($status)** |
| public function getCatsByStatus(**$status**) {  // $status == 'A' or 'P'  **// STEP 1**  // Connect to database 'animals'  // See 'ConnectionManager.php'  $connMgr = new ConnectionManager();  $conn = $connMgr->connect();  **// STEP 2**  // Prepare SQL statement  $sql = "**SELECT name, age, gender, status**  **FROM cat**  **WHERE status = :status** ";  $stmt = $conn->prepare($sql);  **// Parameter binding**  **$stmt->bindParam(':status', $status, PDO::PARAM\_STR);**  // It **binds** the value of **parameter $status** to **:status** in the **SQL statement**.  **// STEP 3**  // Run SQL  $stmt->execute();  $stmt->setFetchMode(PDO::FETCH\_ASSOC);  // Retrieve each row as an Associative Array  **// STEP 4**  // Retrieve query results - ONE ROW AT A TIME  $cats = [];  // Initialize an empty (indexed) Array  // so I can return it to whoever called this function  // Use WHILE loop to loop through  while ($row = $stmt->fetch() ) {  $cat = new Cat(  $row['name'],  $row['age'],  $row['gender'],  $row['status']  );  $cats[] = $cat;  }    **// STEP 5**  // Close DB Connection & SQL Statement  $stmt = null;  $conn = null;  **// STEP 6**  // YAY! We're ready to return the array!  return $cats;  } |

**Part D (Display Cat Information)**

In web browser, open **display.php** (you should have copied this file from **cat2** folder into **cat3** folder).

* You do NOT need to make any changes to **display.php**.
* The details of **CatDAO.php** is well-hidden from **display.php**.
* **display.php** does NOT need to know that **CatDAO** now retrieves data from **MySQL Database**.
* All **display.php** needs to know is … what are the public **methods** it can call - in order to retrieve needed data.
* **CatDAO** provides two **public methods** that **display.php** can call:
  + getCats()
  + getCatsByStatus($status)

When the page loads in a web browser **for the first time**, it must display display all cats.

|  |
| --- |
| **display.php** |
|  |

**Next**, when the user selects **Available** as the filtering value, and clicks on **Filter SUBMIT button**, the page displays:

|  |
| --- |
| **display.php** |
|  |

**NOTE:** The page must remember and pre-select the user’s form input **“Filter by Status”**. For instance, in the above example, “**Available**” option is pre-selected in the drop-down list.

**Next**, when the user selects **Pending Adoption** as the filtering value, and clicks on **Filter SUBMIT button**, the page displays:

|  |
| --- |
| **display.php** |
|  |

**NOTE:** The page must remember and pre-select the user’s form input **“Filter by Status”**. For instance, in the above example, “**Pending Adoption**” option is pre-selected in the drop-down list.