Database Retrieval

# *Objectives*

* Display records retrieved from a database via a web application.



1. We assume you have a subfolder contact\model in folder C:\wamp64\www\.
   1. We assume you have installed WAMP in C:\wamp64\.
   2. If you have installed WAMP in another folder, please change accordingly.
2. In folder model, create Contact.php which will define a class Contact. It has a constructor that takes in 4 parameters to initialize the respective private attributes of the Contact object.
   1. $email
   2. $name
   3. $tel with default value "-"
   4. $address with default value "-"
3. In folder contact, create testContact.php that
   1. Create a Contact object with email “dummy@gmail.com”, full name “Test Dummy”, telephone number “91114444”, and address “66 Crash Avenue”.
   2. Display the attributes of the Contact object.

If done correctly, output of testContact.php is as follows:

|  |
| --- |
| Email: dummy@gmail.com Name: Test Dummy Tel: 91114444 Address: 66 Crash Avenue |

Figure 1: Output of testContact.php.



1. Copy ContactDAO.php in the given resource to your folder contact\model\.
2. Class ContactDAO[[1]](#footnote-1) has
   1. Method getAll() that returns an array of Contact objects.
   2. Method get() that takes in a parameter $email and returns a Contact object with the given email.
3. In folder contact, create index.php that
   1. Get an array of Contact objects from a ContactDAO object.
   2. Display the details of each Contact object in a table with the following columns:
      1. S/N (which is the row number)
      2. Email
      3. Name

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | **S/N** | **Email** | **Name** | | 1 | ace@gmail.com | Ace Ang | | 2 | bay@live.com | Bay Ming Chun | | 3 | chan@smu.edu.sg | Chan Lay Keng | |

Figure 2: Display a list of contacts.



1. Copy form.html in the given resource to your folder contact\. This HTML file has a form that asks for an email and submits to view.php.
2. Create view.php that
   1. Obtain the email from the submitted form.
   2. Get the required Contact object via ContactDAO.
   3. Display the details of the Contact object.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **Email** | ace@gmail.com | | **Name** | Ace Ang | | **Tel** | - | | **Address** | - | |

Figure 3: view.php displays details of a contact with email “ace@gmail.com”.

1. The URL in your browser’s address bar should be <http://localhost/contact/view.php?email=ace%40gmail.com>.
   1. Certain characters are URL encoded by the browser upon form submission. Examples are space and the character @.
   2. Character @ is encoded to %40 by the browser when you submit the form.
   3. Reference <https://www.w3schools.com/tags/ref_urlencode.asp>
2. What is the URL that you can type into browser directly such that it loads view.php to display the contact details for email “bay@live.com”?
3. Edit form.html and insert the following hyperlink after the form.

<a href="view.php?email=chan@smu.edu.sg">Link </a>

Where does this hyperlink lead to? What will be displayed?

1. Update index.php to add a fourth column “Action”. For each row, this fourth column contains a hyperlink to view.php to view that contact details.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | S/N | Email | Name | Action | | 1 | ace@gmail.com | Ace Ang | view | | 2 | bay@live.com | Bay Ming Chun | view | | 3 | chan@smu.edu.sg | Chan Lay Keng | view | |

Figure 4: Fourth column with hyperlink to view.php to see the contact details.



1. Login to phpMyAdmin and click the “Import” at the top row of menu item of the right frame.

|  |
| --- |
|  |

Figure 5: Select menu item “Import” at the top row of left frame.

1. Click the button “Choose File” and select contact.sql from the given resource.
2. Click button “Go”.
3. When done successfully, a success message “Import has been successfully finished, …” (or equivalent) is displayed.

|  |
| --- |
|  |

Figure 6: Message “Import has been successfully finished, 14 queries executed. (contact.sql)”



1. Copy ConnectionManager.php and Logger.php from resource to folder contact\model\.
   1. Class ConnectionManager has two methods:
      1. Method connect() will return a PDO connection to MySQL database.
      2. Method handleError() provides a generic way to handle database connection and access errors by writing to a log file by calling class Logger’s log() method.
   2. Class Logger has only one method log(). This method will format the $message and $details nicely and write to C:\wamp64\logs\contact.log (assuming you have installed WAMP in C:\wamp64\).
2. Check the values of the variables $username and $password in class ConnectionManager’s method connect().
   1. We assume your MySQL login username is “root” and password is empty string “”.
   2. Update the two variables to the correct values if your MySQL database login is different.

**For MAMP users,**

1. the default password for MySQL is “root”.
2. MySQL is running on a different port 8889.

Hence, you have to make the changes highlighted in bold in the table below to ConnectionManager’s method connect(). If you have change the default MAMP’s MySQL port number, please update accordingly

|  |
| --- |
| function connect() {  $servername = "localhost";  $username = "root";  $password = "**root**"; **// Change password**  $dbname = "sis";  **$port = 8889; // Add port number**  // Create connection  **// Add port to connection string**  $conn = new PDO("mysql:host=$servername;**port=$port;**dbname=$dbname" , $username, $password);  $conn->setAttribute(PDO::ATTR\_ERRMODE, PDO::ERRMODE\_EXCEPTION);  return $conn;    // if fail, exception will be thrown  } |

1. Class ContactDAO will be using class ConnectionManager and class Logger.

|  |
| --- |
| <?php  **// insert the next two lines of code** require\_once 'model/ConnectionManager.php'; require\_once 'model/Logger.php';  **// end of code to be inserted**  class ContactDAO { |

Figure 7: Insert code in class ContactDAO to use classes ConnectionManager and Logger.

Side note: If a PHP file requires the use of a class definition in another PHP file, you will have to add the necessary “require\_once …” code. We will see how we can reduce this hassle later.

1. Replace class ContactDAO’s method getAll() with the following code such that it will retrieve from database instead of the attribute $contacts.

|  |
| --- |
| **// Start of new getAll()**  function getAll() {  $contactArr = array();    // connect to database  $connMgr = new ConnectionManager();  $conn = $connMgr->connect();    // prepare select  $sql = "SELECT `email`,`name`,`tel`,`address` FROM `contact`";  $stmt = $conn->prepare($sql);    if ( $stmt->execute() ) {    while ($result = $stmt->fetch(PDO::FETCH\_ASSOC) ) {  $contact = new Contact($result["email"], $result["name"], $result["tel"], $result["address"]);  $contactArr[] = $contact;  }  }  else {  $connMgr->handleError( $stmt, $sql );  }    // close connections  $stmt = null;  $conn = null;    return $contactArr;  }  **// End of new getAll()** |

Figure 8: New code for method getAll()to retrieve from database.

1. View the index.php page in your browser and you should see the list of contacts from database (which includes a new fourth contact dan@smu.edu.sg).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | **S/N** | **Email** | **Name** | **Action** | | 1 | ace@gmail.com | Ace Ang | view | | 2 | bay@live.com | Bay Ming Chun | view | | 3 | chan@smu.edu.sg | Chan Lay Keng | view | | 4 | dan@smu.edu.sg | Danny Fong | view | |

Figure 9: List of contacts from database displayed.



1. As class ContactDAO will be retrieving everything from database, the attribute $contacts and constructor are no longer required. Hence, remove the following lines of code in class ContactDAO.

|  |
| --- |
| private $contacts;  function \_\_construct() {  $this->contacts = array(  "ace@gmail.com" => new Contact("ace@gmail.com", "Ace Ang" ),  "bay@live.com" => new Contact("bay@live.com", "Bay Ming Chun", "91234567" ),  "chan@smu.edu.sg" => new Contact("chan@smu.edu.sg", "Chan Lay Keng", "98765432", "Catnip Park" )  );  } |

Figure 10: Remove attribute $contacts and constructor code that are not required.

1. Update class ContactDAO’s get() method to retrieve from database too.
   1. The SQL statement to retrieve a contact by email is  
      SELECT `email`,`name`,`tel`,`address` FROM `contact` WHERE `email` = :email
   2. You must use PDO PreparedStatement to bind the input parameter $email to “:email”.
2. When done correctly, you can view the details of each contact (including dan@smu.edu.sg) via the view.php page.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **Email** | dan@smu.edu.sg | | **Name** | Danny Fong | | **Tel** | 92225555 | | **Address** | DB Avenue | |

Figure 11: view.php displays details of [dan@smu.edu.sg](mailto:dan@smu.edu.sg).



We added “require\_once 'model/ConnectionManager.php';” and “require\_once 'model/Logger.php'” to ContactDAO.php because class Contact uses the other two classes. A real life application will have a large number of classes and there will be a lot of “require\_once …” code lines.

Hence, we can do the following to automatically find the class-based PHP files in folder contact\model\.

1. Copy the given common.php and error.php to folder contact\. Explanation of how common.php and error.php work will be given later.
2. Remove the various “require\_once …” code lines from ContactDAO.php too.
3. For index.php and view.php
   1. Remove the following two lines:

|  |
| --- |
| require\_once 'model/Contact.php'; require\_once 'model/ContactDAO.php'; |

* 1. Add the following one line:

|  |
| --- |
| require\_once 'common.php'; |

Note: For future PHP files that will be viewed in browser (i.e. non-class-based), you can just include the above line of code.

1. If done correctly, index.php and view.php should work as normal.



1. The content of common.php is as follows:

|  |
| --- |
| <?php  date\_default\_timezone\_set('Asia/Singapore');  /\*\*\*  to auto-load class definitions from PHP files  \*\*\*/  spl\_autoload\_register(function($class) {  $path = "./model/" . $class . ".php";  require\_once $path;    });  /\*\*\*  default handler for uncatch exceptions  \*\*\*/  set\_exception\_handler(function($ex){  $details = [  "exception" => [  "message" => $ex->getMessage(),  "file" => $ex->getFile(),  "line" => $ex->getLine(),  "trace" => $ex->getTrace()  ]  ];    Logger::log( "Unhandled Application Exception", $details );    header("Location: error.php");  }); |

1. PHP provides a predefined function spl\_autoload\_register() that allows you specify how to find your class-based PHP files.
2. Let’s look at the following line of code in function spl\_autoload\_register().

|  |
| --- |
| $path = "./model/" . $class . ".php"; |

* 1. The parameter $class that specifies the name of the class definition required
     1. E.g. Contact, ContactDAO, ConnectionManager, etc.
     2. This value will be passed in by the web server when it needs a particular class-based PHP file.
  2. Hence, if the class required is ContactDAO, the value of the $path is “./model/ContactDAO.php”.

1. And the last line of code in spl\_autoload\_register() is “require\_once $path” to include the class-based PHP file.
2. A second predefined PHP function is set\_exception\_handler() that allows you to specify how to handle errors that you did not expect, thus, did not specifically code for.
   1. Exception is an object oriented way of representing errors.
   2. In common.php, we just write the details of the error (aka exception) to log file using the class Logger and redirect to a generic page error.php.
3. You may wish to find out more in the links given in the references below.

# *References*

1. <https://www.w3schools.com/tags/ref_urlencode.asp>
2. <https://www.w3schools.com/html/html_filepaths.asp>
3. PDO
   1. <https://www.w3schools.com/php/php_mysql_connect.asp>
   2. <https://www.w3schools.com/php/php_mysql_prepared_statements.asp>
4. Predefined function spl\_autoload\_register()
   1. <http://php.net/manual/en/function.spl-autoload-register.php>
   2. <https://stackoverflow.com/questions/7651509/what-is-autoloading-how-do-you-use-spl-autoload-autoload-and-spl-autoload-re>
   3. Best practice <https://phpbestpractices.org/#auto-loading>
5. Predefined variable $\_SERVER
   1. <http://php.net/manual/en/reserved.variables.server.php>
6. Default exception (aka errors) handling
   1. <http://php.net/manual/en/function.set-exception-handler.php>
   2. <https://www.w3schools.com/php/php_exception.asp>

1. DAO stands for Data Access Object. For this exercise, class ContactDAO is very simple; it has only 2 methods.

   In later part of this lab and future labs, it will be updated with more methods to retrieve, create, update and delete Contact objects/records from a database. [↑](#footnote-ref-1)