|  |  |
| --- | --- |
|  | **Lima Bean is a division of Mindvate (Pty) Ltd**  Reg No: 2005/005418/07  P O Box 13094, Woodstock, 7915  Tel: (021) 486 1860 Fax: (021) 486 1886  Email: [supportHYPERLINK "mailto:support@limabean.co.za"@limabean.co.za](mailto:support@limabean.co.za) |

**PHP/MYSQL back-end developer training**Building a DVD store website/system (drafted by Nico Smit)

A typical client-developer scenario: You get contacted by a client who has a certain requirement. They want to be able to maintain a database for their business and they need your help to set up software to maintain this database. Usually, the system grows and changes over time, as the client realizes exactly what they need. This tutorial will simulate that process and you will be guided through it step by step to develop the end result for the client.

The goal of this tutorial is to get you trained and ready in the following areas:

* PHP coding basics
* MYSQL Database set up and query basics
* Software flow and logic principles

This is strictly a back-end tutorial and the focus on front-end (HTML,CSS,Javascript) will be minimal

***Synopsys:***

*The client is the owner of a DVD store. They currently have no electronic system in place, and they have to fill in all details on actual paper in books. The system is becoming too much for them to maintain. Desperate, the client comes to you and asks you to help them develop an online system to use that will be rolled out piece by piece.*

* **Task #1:** build a database and table to manage their customer base with
* The client wants to start very simple. They just want a database where they can keep all their customer info. They will still do all the dvd stock and renting on paper, but just having their clients accessible through an online database will help them a great deal.
* Set up a MYSQL database on your localhost, call it “dvd\_shop”
* Create a table in the database called “customer**s**” with the following fields
* id (this will be an auto-increment integer field, used as the primary key)
* name (customer’s first name)
* surname (customer’s surname)
* contact\_number (telephone or cellphone)
* email (customer’s email address)
* sa\_id\_number (The ID number of the customer, if they are a South African citizen)
* address (Physical address of the customer)
* Fill in a few dummy entries in the database so that there is test data to work with resources
* You can read up on MYSQL at w3schools (<http://www.w3schools.com/sql/>)
* As a tool to interact with the database, you can use Navicat (pre-installed on your PC) or HeidiSQL (<http://www.heidisql.com/download.php>)
* **Task #2:**Display all the data on a web page
* Start a new website project in your wamp/xamp www folder for this website
* Now build a php page that can interact with the MYSQL database. Call it customer.php.
* Run a query to the database that gets all the customers in the customer table
* Display the results on a page with HTML, let the display look similar to the following:  
  
* Resources
* Connecting PHP to a MYSQL database (<http://www.w3schools.com/php/php_mysql_connect.asp>)
* **Task #3:** Editing database entries
* You now have the ability to view all the customers in the database, but you still have to give the client the ability to edit the customers through your PHP website. You are going to add that functionality now.
* Add a new column at the end of the HTML table and give it the heading “Actions”
* In every row add a link in the actions column that only reads “edit”
* Add logic so that, when you click on “edit”, you can edit that specific record on-screen
* (Hint) Use the id field as index to pass to the edit page
* Once edit is clicked, open a page with a form where all the details of the customer are already filled in.
* Add a submit button at the bottom of the form. When the submit button is clicked, save all the data in the form in the database. Remember that you should update the existing record and not create a new one.
* Block the ability to submit blank fields. (functionaltiy working but prepopulated names on form must be shown)
* Don’t allow them to edit the id field. Users should never be able to edit your primary key
* If the form submits successfully, go back to the page that lists all the customers, and display a message “Customer successfully updated”
* Resources
* HTML forms (<http://www.w3schools.com/html/html_forms.asp>)
* MYSQL UPDATE query (<http://www.w3schools.com/php/php_mysql_update.asp>)
* **Task #4:** Adding new entries in the database
* Now that the client can update their customers, they also want the ability to add a new customer to their database.
* Add a button at the bottom of thecustomer.php page that reads “Add customer”
* When the Add customer button is clicked, go to an empty form that looks similar to the customer edit form. Give users the ability to fill in this form and submit it.
* On successful submission of the add form, write the data submitted to the database and go back to the customer.php page where you display a message “Customer successfully added to database”
* Your client now has the ability to add and edit customer in the database
* Resources
* MYSQL INSERT query (<http://www.w3schools.com/php/php_mysql_insert.asp>)
* **Task #5:** Deleting customers
* In the actions column on you customers table, add a new link that reads “delete”
* When clicking this link, delete the customer record concerned from the databse, and reload the customer.php page with a message “Customer successfully deleted from database”
* You have just completed your first CRUD (Create, Read, Update, Delete). This is the backbone of database systems.
* Resources:
* MYSQL DELETE query (<http://www.w3schools.com/php/php_mysql_delete.asp>)
* **Task #6:** Reusability
* Now that you have completed the first CRUD, you might have realized that there is the potential to keep on repeating code over and over again. For example, the code to connect to a database and then running queries on it. For one CRUD you run 4 different types of queries (insert, update, delete, select). A good programmer will always code software in such a way that you reuse code as much as possible. Let’s say, for example, that we want to add a new table to the database, and we want to access that through a new php file. Instead of writing all the database connections and calls again, we can implement reusability to reuse the same database calls on the other pages.
* Before we continue with our tutorial, we need to start making sure that what we already have is as optimized as possible. Take the part of your code that makes the connection to the database, and put this in a new file, called database.php. Now pull in this file in the top of your customer.php file using the php require function. You only need to make the connection with the database once in order to run queries from it. Everything in a file that you pull in using the require function, will be as if it is on the same page as where you pull it in. Variable names etc. will also still be available. Once you have this set up, you can easily just pull in the database functionality on every single page you work on.
* Create another file to import, called header.php, and pull in all your html head information into this file, so that you don’t have to include the html head code on every page.
* Resources:
* <http://www.w3schools.com/php/php_includes.asp>
* **Task #7:** Object Orientation (Classes)
* Now that you have the require function set up and your database connection set up in another file. You can also go the extra mile and put the whole database connection and querying in a class. So that you only have to instantiate the class once, and can then query the database through the class in the future. Before continuing, I strongly suggest you first understand object orientation and classes in PHP. Go through the resources below on this first.
* Set up a class in your database.php file. Call the class “mysql\_database”.
* In the class constructor function, accept the database access details as parameters (server host name, username, password, database name). And then make the connection with the database in your constructor. This way, whenever you instantiate the class [eg. $db = new mysql\_database(‘localhost’,’root’,’password’,’dvd\_shop’); ] – then your databse connection will already exist.
* Build a function to query the database, call it “fetch”. Let the fetch function accept a parameter called $query. You can just send the query to this function as a string, and it will query the database. Let it return the results in the form of an array (as PHP does)
* Build a function to update the database, call it update. Just as query, it will receive a $query parameter, but it returns only true or false: true if the query ran successfully, and false if not.
* Now rewrite your customer.php so that you do all your database manipulations through this class.
* Resources:
* <http://code.tutsplus.com/tutorials/object-oriented-php-for-beginners--net-12762>
* <http://php.net/manual/en/language.oop5.php>
* **Task #8:** Foreign keys and join queries
* With the database class in place, we will no longer have to rewrite the database queries for every call we make. So now we can proceed to implement the rest of the system without having to rewrite the code every time.
* Create a new table in your database, called “dvd”. This table will be used to store the dvds you have in stock. Create the following fields in the database, id (primary key, auto increment), name, description, release date.
* Create another table, called “category”, this table will store all the different categories that dvds can fall under (eg. Comedy, Action, Adventure, Horror, Family, Childrens etc). Create the fields: id, category\_name.
* Now add a new field to the “dvd” table, called “category\_id”. This field will serve as a foreign key value of the category, and should link the dvd to the category that it belongs to.
* Create a new php page, called dvd.php
* On this page, build a table (as before) that lists all the dvds in your collection. Please note that, instead of displaying the category\_id (which will make no sense to the user), display the name of the category under which that dvd falls. Use the appropriate MYSQL JOIN query to get all these results using only one query to lighten the server load. <?= ?>
* Also now add the CRUD (Add, edit, delete) for DVDs as you did with customers.
* Resources:
* <http://www.mysqltutorial.org/mysql-foreign-key/>
* <http://www.w3schools.com/sql/sql_foreignkey.asp>
* <http://www.w3schools.com/sql/sql_join.asp>
* **Task #9:** Link tables
* We now get to the actual ordering of dvds by customers. Create a table “order” in the database, you will store the orders by customers in this table. Add the fields: id, customer\_id (foreign key), rent\_date, due\_date, actual\_return\_date.
* As a customer can order multiple DVDs at once, it won’t help to just add a “dvd\_id” foreign key to this table, because that will limit the customer to order only one DVD. We are going to bridge this gap with the use of a link-table. Create a new table called “dvd\_order”, this table will be used to show what dvd’s were ordered by a customer on a specific order.
* Add the following fields to “dvd\_order”: id (primary key), dvd\_id (foreign key that links to dvd), order\_id (foreign key that links to order).
* With this structure, we can now list in “dvd\_order” all the dvds that were linked to a specific order.
* Create a new php page, order.php.
* Using this table structure as your base, and then getting the data using join queries, on the order.php, create a CRUD for orders. Remember that you should be able to add multiple dvds to one order.
* Resources
* <http://www.joinfu.com/2005/12/managing-many-to-many-relationships-in-mysql-part-1/>