**Using PHP with MySQL**

**Introduction**

In the activity *Introduction to PHP: Playing Catch With HTML,* we learned how to collect information from the user via a web form. Upon submission, that information was passed to a PHP file residing on a server. In this activity, we are going to do something constructive with that data: use it to select, update or delete records in a MySQL database.

Materials

* Computer
* Text editor, such as Notepad++, or an integrated development environment (IDE)
* Server environment, local (WAMP/MAMP) or remote (Cloud9)

In the activity *Introduction to MySQL*, you discovered that a record could be inserted into a data table using a SQL statement. For example, the SQL command

INSERT into movies (title, year, genre) VALUES (“Jaws”, 1975, “Drama”)

will add a record (row) to the *movies* table. Doing so requires, in addition to proper syntax, manual entry of that command on the mysql command line. It is possible to communicate with the MySQL server through a web page? Can we direct HTML form data to a php file, residing on a server, and have that file execute SQL commands for us? The answer is yes, and the goal of this activity is to show you how.

Procedure

**Part I: Obtaining permission to interact with the database**

In order to communicate with a database, a web page must obtain permission from the MySQL server hosting that database. The task of establishing a communications pipeline and obtaining permission to interact with a database is collectively called *establishing a connection*. If a connection attempt is successful, MySQL returns a token that must be used in any future interactions with the database. If a database refuses a connection, or if network resources are unavailable, then no token will be returned and interaction with the database will not be allowed. Consequently, establishing a connection is our first priority.

Our approach will be to modularize our work: we will create a file for connecting to a database and name it *db.php*. This file will reference another php file, called *config.php*, which defines all the variables needed to connect to the database: hostname, username, password, and database name. The reason for maintaining separate files is for ease of transition: changing to a different database in the future will be easy: all we have to do is swap out one variable definition file for another.

The main idea here is that information will be sent from an HTML form, residing on a client browser, to a php page residing on a server. PHP code will then extract and join this data to specific SQL verbiage, forming an appropriate SQL statement. (A SQL statement is deemed to be appropriate if it meets the needs of the client). For instance, if a user wants to select a movie having a certain title, we will create the SQL command

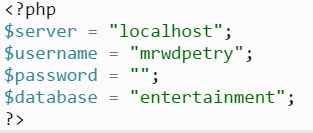
SELECT \* from movies where title = ‘**$title**’

The bolded variable **$title** contains information sent from an HTML form. (The single quotes around the variable are essential and their omission is often a source of error!).

Using variable data allows us to customize requests sent from the client side.

1. **Connect to a Database**. Create the files *process.php*, *db.php* and *config.php* as shown below. *db.php* and *config.php* are shown in their final form. We will add to process.php as we build our skill. As you type each, try to understand what each line does. Ask for clarification if the line’s meaning is unclear to you.

*config.php* (below) defines variables used in *db.php*:



*db.php* (below) uses the php command **include\_once** to link to *config.php* (above), which defines variables related to connecting. It then uses the php function **mysqli\_connect** to attempt a connection to the MySQL database *entertainment*. The code provide a contingency for failure and provides a token ($link) in case of success:



The previous two files (*config.php* and *db.php*) work together to connect to the MySQL database *entertainment*, obtaining a token (which we called $link) upon success. This variable is required to interact with the database in the future.

The file *process.php*, which is called by the HTML form, links to *db.php* which, of course, links to *config.php*:



*HTML form*

*process.php*

*db.php*

*config.php*

We can test the functionality of this chain of php files by building an abridged version of *process.php.* While *process.php* will eventually be called by an HTML form, we can set aside that functionality for now and create a simple version to test our ability to connect to the database:

*process.php*:



Make sure the project is running and point your browswer to

https://workspacename-username.c9users.io/process.php (verify correct path)

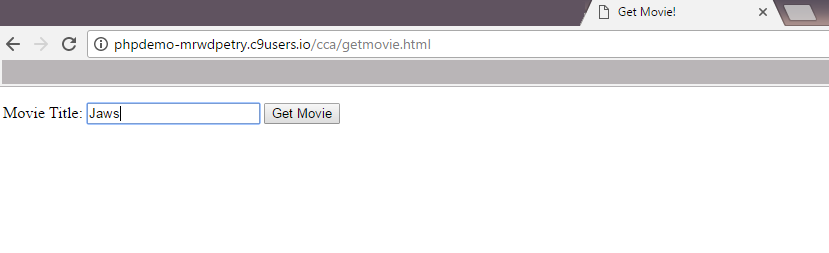
If we see the message “We are connected!” then we can conclude that we have connected to the *entertainment* database, obtaining a token ($link) that can be used for future interaction with that database. Now that we have verified connectivity with the database we can expand our project!

1. **Catch, Concatenate and Create a SQL Statement.**

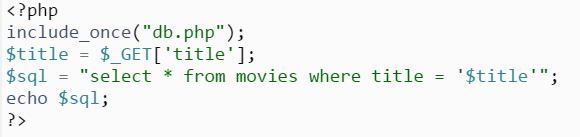
Let’s re-visit our HTML form and review what it does:



This form collects a movie title (assigned to the variable *title*) from the user and submits it to *process.php*. We have confirmed that *process.php*, using the chained files *db.php* and *config.php*,is able to connect to the *entertainment* database on MySQL using the supplied credentials. Next, we need to extract the submitted *title* form data and join it to specific SQL verbiage. This concatenation (i.e. joining) will be helpful in building an SQL statement . For instance, if we as for the movie jaws in the HTML form



and submit that form to *process.php*. We need to modify our process.php file to extract the sent data and join it to form a SQL statement:



Notice the line $title = $\_GET[‘title’];

This is an important line for two reasons: first, it illustrates that all php variables must begin with a dollar sign ($) and, second, it shows us how to extract information from form data, in this case information entered in a textbox named *title*. This line is an important part of a larger task: how to create an SQL statement that can be used to pull specific information from a database table.

Also, notice the line echo $sql;

This is a troubleshooting technique whereby the generated SQL is printed out in a browser. The resulting SQL is then copied, pasted, and executed in a mysql command line in order to determine if the code does what it is supposed to do.

1. **Hit the Database and Get Results.**

We will finish our example by adding code that uses the generated SQL statement ($sql) to pull information from a database. A command that requests information from, or alters information in a database is called a **query**. We will now add code to *process.php* that executes a query, runs through the result set, and sends data back to the client:



**Questions to Answer**

1. What roles do each of the following files play:

*config.php-*

*db.php-*

*getmovies.html-*

*process.php-*

1. How could you verify that each file is working as designed? Would your answer allow you to pinpoint which file caused a breakdown in the functioning of the whole?
2. In the space below, use pseudocode or a flowchart to describe how user entry in the HTML form allows for the retrieval of one or more records based on the user input.