Week 10 Lab – Databases, MySQL and PHP

Database Introduction: http://swinbrain.ict.swin.edu.au/wiki/Database_Introduction

Aims:
- The aim of this exercise is to practice the coding of SQL by creating and manipulating a database table on the MySQL database server **feenix-mariadb.swin.edu.au**
- Write PHP scripts to connect to the database, query tables, and display data in HTML

Getting Started:

Create a new folder '**lab10**' under the unit folder on the mercury server ~/*{your unit code}*/*www*/*htdocs* folder. Save today's work in this lab10 folder.

All Web pages should conform to HTML5 and must be validated.
You could also create and link an external stylesheet, to the pages, and this should be valid CSS3.

Task 1: How to login to your MySQL account:

(See also https://feenix.swin.edu.au/help/index.php?page=MySQL%20%28MariaDB%29)

1. Login to the **mercury.swin.edu.au** server using the 'PuTTY' client.

2. To access your mysql account type in:
   ```
   mysql    (press Enter key)
   ```

   The configuration file on **mercury.swin.edu.au** ensures that you are prompted for a password and to connect to your mysql account on the **feenix-mariadb.swin.edu.au** server.

   For most users, your **mysql username** will be the same as your **mercury username** and your **mysql password** will have been set initially to your date of birth in **ddmmyy** format.
   Press 'Enter' key after typing in your password.

   Your mysql password is not connected to your SIMS password.
   It is recommended that you change your mysql password and ***do not use*** the same password for your mysql account, as for other accounts, as later you will need to use your mysql password within PHP scripts. (See "How to change your mysql password" on the next page.).

3. You will arrive at the 'MySQL Monitor' or mysql command line client:
   ```
   mysql> …
   ```

4. On our mysql server, a database has already been created for you to use in labs and assignments, named as `username_db` where `username` is
   s<7 or 10 digit Swinburne id>, for example, s1234567 and `s1234567_db`.
   Access your database as follows:
   ```
   mysql> USE username_db;
   ```
   You will receive a confirmation message "`Database changed`".

5. The MariaDB SQL manual can be found here:
   https://mariadb.com/kb/en/mariadb/sql-structure-and-commands/   *… or see other simpler guides* ☺

> **How to change your MySQL password:**
>
>   mysql> SET password=PASSWORD('newpwd');
>
> Where `newpwd` represents the new password you would like to use.
>
> **Remember** that in Labs and in Assignments, you will have to **include (encode)** your database **password** in **all** PHP files that will access the database.
> So **do not use** your SIMS or your mercury password, as your mysql password.

Task 2: Using phpMyAdmin

Instead of using the 'MySQL Monitor' command line interface, you may find it easier to use the MySQL web interface called **phpMyAdmin** which is available at https://feenix-mariadb-web.swin.edu.au/
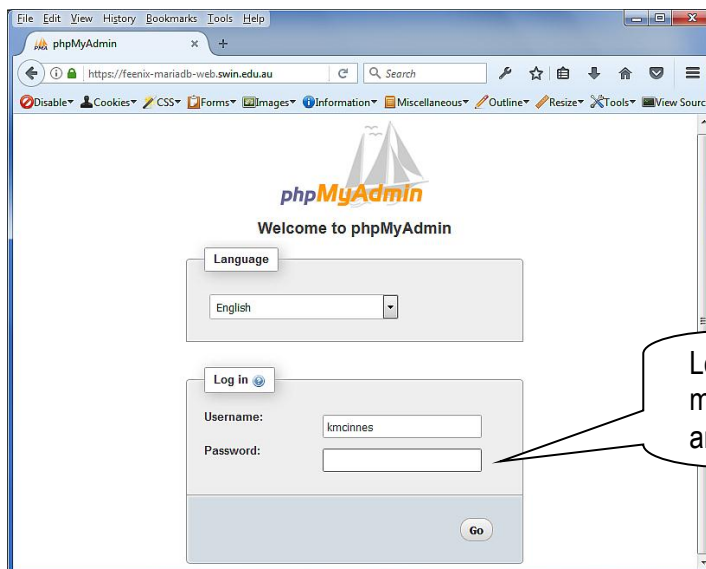
Log in to phpMyAdmin with your mariadb username (s1234567890) and mariadb password (ddmmyy).

From here you can manage the settings for phpMyAdmin and your database, and create, modify and delete tables and records.
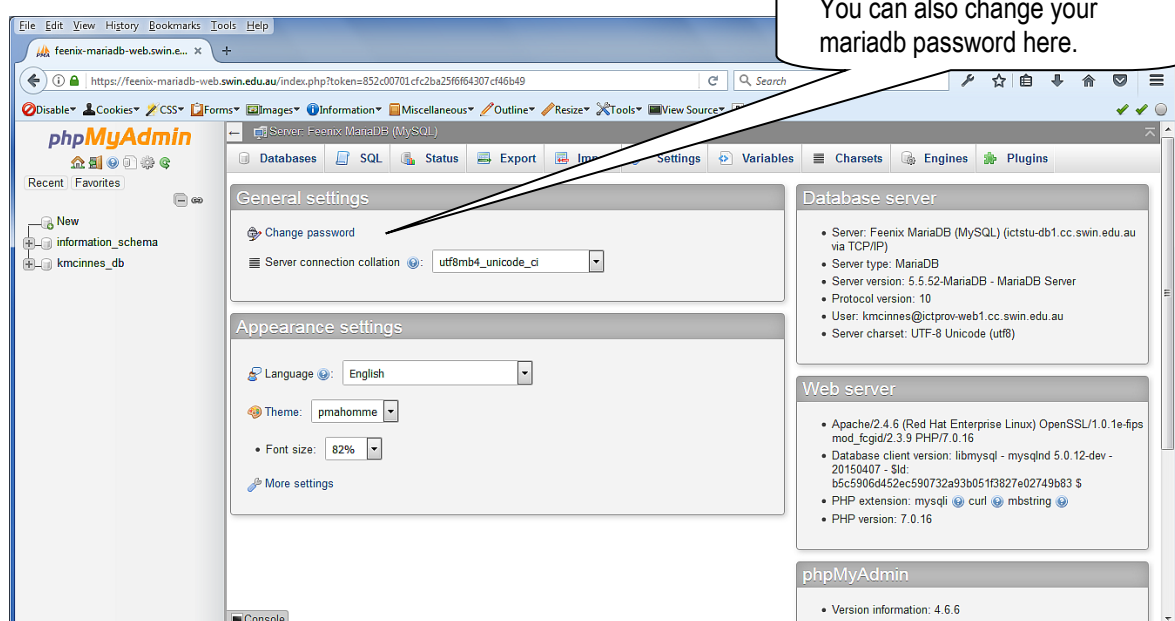
You can also change your mariadb password here.

Task 3: Creating a table

Note that if you managed your own website, you would need to first create a database before you would be able to create tables in your database. On our mysql server, a database has already been created for you.

Using your existing database '`s<Swinburne id>_db`', create a new table `cars` for a used car dealership.
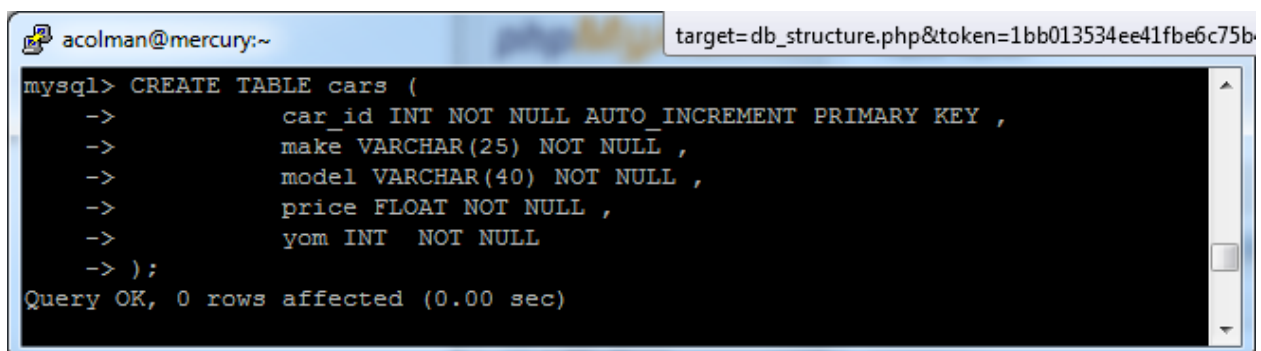
```
CREATE TABLE cars .....;
```

Include the following fields in the **cars** table with the following data types:

car_id	AUTO_INCREMENT PRIMARY KEY
make	string up to 25 characters – required
model	string up to 25 characters – required
price	string up to 25 characters – required
yom (year of manufacture).	integer - required

```
CREATE TABLE cars (
        car_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY ,
        make VARCHAR(25) NOT NULL ,
        model VARCHAR(40) NOT NULL ,
        price FLOAT NOT NULL ,
        yom INT  NOT NULL
);
```

> **Hint:** You can copy plain text from the clipboard into the MYSQL Monitor command prompt by right clicking. Make sure you have added a semicolon ";" at the end of the command.
> Pressing ↵ will execute the command.

In **'MySQL Monitor'**, if you key in the query, rather than copy and paste, you will need to press 'enter key' ↵ after each line. This will generate the "->" characters that indicate that the current MySQL line is a continuation to the above line **not** a new MsSQL line. These lines subsequently form a single MySQL query that is terminated by a semicolon ";". Pressing ↵ after the semicolon will then execute the query.

```
acolman@mercury:~                                    target=db_structure.php&token=1bb013534ee41fbe6c75b

mysql> CREATE TABLE cars (
    ->          car_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY ,
    ->          make VARCHAR(25) NOT NULL ,
    ->          model VARCHAR(40) NOT NULL ,
    ->          price FLOAT NOT NULL ,
    ->          yom INT  NOT NULL
    -> );
Query OK, 0 rows affected (0.00 sec)
```

Alternatively, in **phpMyAdmin**, you could create a table from the phpMyAdmin interface by selecting your database and then clicking the 'Create Table' tab, and then filling in the blanks OR enter the MySQL query by selecting the 'SQL' tab, typing or copying the SQL query, and clicking 'Go'.

Task 4: Adding data

Records can also be added (inserted) using either the 'MySQL Monitor' command line or the phpMyAdmin interface.

The SQL query will be:

```
INSERT INTO cars
     (car_id , make , model , price , yom)
          VALUES
     (NULL , 'BMW', 'X3', '35000', '2010');
```

Note: single quotes can also be used around numbers.

Alternatively you can use the phpMyAdmin interface, and type in the details for a new record.
As the car_id is auto-increment, we don't have to specify it.

The interface will then display the result along with the MySQL query it used to add the record:

Enter at least 10 records into the table using either the 'MySQL Monitor' command line or the phpMyAdmin interface. Sample data is shown in the table below.

Make	Model	Price	Year of Manufacture
Holden	Astra	$14,000.00	2009
BMW	X3	$35,000.00	2010
Ford	Falcon	$39,000.00	2013
Toyota	Corolla	$20,000.00	2012
Holden	Commodore	$13,500.00	2005
Holden	Astra	$8,000.00	2004
Holden	Commodore	$28,000.00	2009
Ford	Falcon	$14,000.00	2011
Ford	Falcon	$7,000.00	2003
Ford	Laser	$10,000.00	2010

Task 5: Querying the table

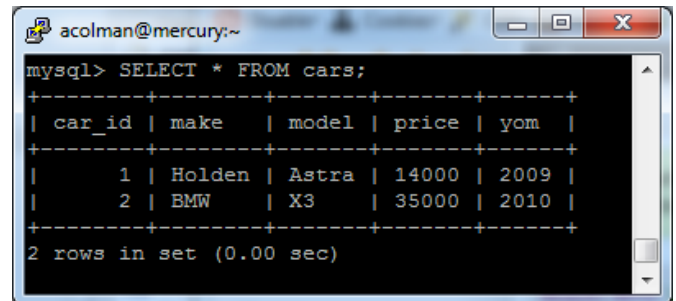In this task you will write queries that return records from the table.
In steps 1 to 4 below, the SQL is given. *Use SQL commands* to enter and send the following SQL queries to the database:

1. To select all records type:

    SELECT * FROM cars;

    **NOTE: The executed SQL query will display
    the dataset requested.**

2. To select make, model, and price,
    sorted by make and model

```
SELECT make, model, price FROM cars
          ORDER BY make, model;
```

3. To select make and model of the cars which cost $20,000.00 or more.

```
SELECT make, model FROM cars
          WHERE price >= 20000;
```
   Note that when entering numbers, i.e. price, you must not specify the unit i.e. $ or comma.

4. The average price of cars for similar make.

```
SELECT AVG(price) FROM cars
          GROUP BY make;
```
Now write SQL for the following and the test the queries:

5. The make and model of the cars which cost below $15,000.00.

    …………………………………………………………………………

6. Average price of FORD cars.

    ………………………………………………………………………….

7. Cars manufactured in 2010 or later and costing $15,000 or more.

    ………………………………………………………………………….

Task 6: Connecting to a database from PHP

In this task we will connect to a MySQL database from PHP and retrieve records from the table created in the previous tasks.

Step 1: Connection Info

First we will create a 'settings' file that stores the login information and can be reused whenever we want to connect to our database.

Create a file **settings.php** that contains variable declaration and initialisation of the database host, user name and password. It should not contain any HTML.

```php
<?php
    $host = "feenix-mariadb.swin.edu.au";
    $user = "s1234567890"; // your user name
    $pwd = "ddmmyy"; // your password (date of birth ddmmyy unless changed)
    $sql_db = "s1234567890_db"; // your database
?>
```

Change these to your mariadb username, password, database

Step 2: Connection and displaying records in a table

Use the **@mysqli_connect** command to create a connection to the database from PHP using the parameters that were set in the **settings.php** file.

Note that the @ symbol is not part of the command.
It is an *operator* used to suppress any error messages that may be generated by mysqli_connect.

```php
$conn = @mysqli_connect($host,
        $user,
        $pwd,
        $sql_db
    );
```

HUPD

As shown in the code on the following page, create a file **cars_display.php** that selects only make, model and price from the database table cars, created in task 3, and then displays them neatly in a well coded html table.

(The code for this is on the next page).

```php
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8" />
<meta name="description" content="Creating Web Applications Lab 10" />
<meta name="keywords" content="PHP, MySql" />
<title>Retrieving records to HTML</title>
</head>
<body>
<h1>Creating Web Applications - Lab10</h1>
<?php
    require_once ("settings.php");    //connection info

    $conn = @mysqli_connect($host,
        $user,
        $pwd,
        $sql_db
    );
    // Checks if connection is successful
    if (!$conn) {
        // Displays an error message
        echo "<p>Database connection failure</p>"; // not in production script
    } else {
        // Upon successful connection

        $sql_table="cars";

        // Set up the SQL command to query or add data into the table
        $query = "select make, model, price FROM cars ORDER BY make, model";

        // execute the query and store result into the result pointer
        $result = mysqli_query($conn, $query);

        // checks if the execution was successful
        if(!$result) {
            echo "<p>Something is wrong with ", $query, "</p>";
        } else {
            // Display the retrieved records
            echo "<table border=\"1\">\n";
            echo "<tr>\n "
                ."<th scope=\"col\">Make</th>\n "
                ."<th scope=\"col\">Model</th>\n "
                ."<th scope=\"col\">Price</th>\n "
                ."</tr>\n ";
            // retrieve current record pointed by the result pointer

            while ($row = mysqli_fetch_assoc($result)){
                echo "<tr>\n ";
                echo "<td>",$row["make"],"</td>\n ";
                echo "<td>",$row["model"],"</td>\n ";
                echo "<td>",$row["price"],"</td>\n ";
                echo "</tr>\n ";
            }
            echo "</table>\n ";
            // Frees up the memory, after using the result pointer
            mysqli_free_result($result);
        } // if successful query operation

        // close the database connection
        mysqli_close($conn);
    } // if successful database connection
?>
</body>
</html>
```

\n is optional.
Creates tidy served code.

Test in the browser. A table of cars should be displayed.

Check that the page is valid HTML5.

Task 7: Creating a form to post data to SQL queries

In this task we will create a form to add new records to the database table.

Step 1: Create a form to add records

Create an HTML page called `addcar.html` with a form that allows the user to enter data that will be sent to a server-side script that will then connect to the database and add records.
(A basic html and CSS are provided on Blackboard)

Lab10 Add Car Form

New Car Details

Car Make:	Holden
Last Name:	Commodore
Price:	13500
Year of Manufacture:	2005

[Add New Car]

Note: As discussed in the lecture we are *adding* information to the server so we will use HTTP **POST** method rather than **GET**

Have the form **post** name-value pairs to a file called `cars_add.php`

```
<form method="post" action="cars_add.php">
```

Step 2: Create a PHP file to talk to the database
Create the `cars_add.php` that creates a connection to the database. See Task 6 for a template. (Don't forget to close the connection when you have finished). This time if the connection is successful we will assign the values from the 'post' into the variables.

```php
$make = trim($_POST["carmake"]);
$model     = trim($_POST["carmodel"]);
$price     = trim($_POST["price"]);
$yom  = trim($_POST["yom"]);
```

Then in the query, instead of making a SELECT query as we did in Task 6, we will create an INSERT SQL query as done in Task 4.

```php
$query = "insert into $sql_table (make, model, price, yom) values ('$make', '$model', '$price', '$yom')";
// execute the query  -we should really check to see if the database exists first.
            $result = mysqli_query($conn, $query);
            // checks if the execution was successful
            if(!$result) {
                    echo "<p class=\"wrong\">Something is wrong with ",        $query, "</p>";
                //Would not show in a production script
            } else {
                    // display an operation successful message
                    echo "<p class=\"ok\">Successfully added New Car record</p>";
            } // if successful query operation

            // close the database connection
            mysqli_close($conn);
    } // if successful database connection
```

Step 3: Deploy and Test

Load to Mercury and test in the browser. Use MySQL Monitor command line or the phpMyAdmin interface to check that the new record was added to the cars table.

Run `cars_display.php` to check that all the records are still there. ☺

Check that the delivered page is valid HTML5.
Note that there are alternative outputs to check.

Step 4: Add some 'security'

Note: Now that we are using a form, we have created a user interface to our php code and to our mysql database. Never trust a user – NEVER. At present we have no 'checks' on the data that users can enter. For example, users could put html code in a form input, or even worse, users could put php or sql into a form input.

We do not want to be distracted *here* by writing php server-side data checking, but we can use some easy 'blockers' by simply converting any special characters entered
by using the php function   `htmlspecialchars($_POST["name"])`
or within mysqli using   `mysqli_escape_string($_POST["name"])`

Task 8: Create a search form.

In this task, using Task 7 as a model, create a form to search for a car, based on criteria entered by a user, and then display the results in a table, as done in Task 6.

Step 1: Create a form to search for records

Using Task 7 as a model, create an HTML page called `searchcar.html` with a form that allow users to enter search values, that will be sent to a new php script `cars_search.php` that will connect to the database, query the table for records, and display any records found, or a message if none are found.

Step 2: Create the PHP script that will action the search

Using Task 6 and 7 as a model, create the `cars_search.php`
Note for the query use 'like' and add the wildcard character '%'
eg. `select * from cars where make like 'hol%'`
Remember that you can test your SQL either using MySQL Monitor, or phpMyAdmin SQL interface.

Step 3: Deploy and Test

Load to Mercury and test in the browser.
Check that the delivered page is valid HTML5.
Check all alternative outputs.