RMIT UNIVERSITY

School of Computing Technologies

ISYS1101/ 1102 Database Applications

Week 2: Tute/Lab – Web Database Applications Primer

Semester 2 2022

1 Objective

The objective of this tute/lab session is to learn the basics of hosting a web database application, specifically:

- Make yourself familiarised with RMIT University's Core Teaching Linux Servers (Jupiter, Saturn, etc) setup;
- Familiarise with hosting a website on these servers;
- Learn how to use `oci` functions (PHP's Oracle API) to connect to your own database (we start with the Movies database you built in Week 1.);
- Learn how to embed SQL queries in PHP;
- Learn how use HTML forms to feed input into a PHP program, pass them as input into an SQL query and retrieve data from the backend database.

Once you successfully complete this Lab/ Tute session, you will be able to build a basic, but fully-functional Web Database Application. You will still require to learn further web programming skills to improve the usability and functionality of your web database application. Complete the LinkedIn Learning tutorials (refer to Pre-reading section) and other web resources to acquire these skills.

1.1 Preparation Tasks

1.1.1 Hosting your website

The three core teaching servers (Titan, Jupiter and Saturn) can host student websites that are built for learning and teaching process.

If you are not familiar with Linux/ Unix operating system and perform simple tasks (such as changing directories, editing text files, etc), you should invest some time to complete Unix Induction Module (https://sites.google.com/rmit.edu.au/unixinduction). Use your RMIT credentials to access this website.

This tute/lab session assumes that you have these skills.

If you are comfortable working directly on directories and files using unix commands (such as `cd`, `ls`, `chmod`) and unix file editors (such as vim) you don't need to follow the steps given in Section 1.1.2 below. In that case, create a directory called `public_html` and work in that directory.

1.1.2 Setting up Visual Studio Code to work with core teaching servers

1.1.2.1 Install Visual Studio Code

Visual Studio Code (VS Code, in short) is a free coding editor that helps you start coding quickly. Use it to code in any programming language, without switching editors. Visual Studio Code has support for many languages, including Python, Java, C++, JavaScript, and more.

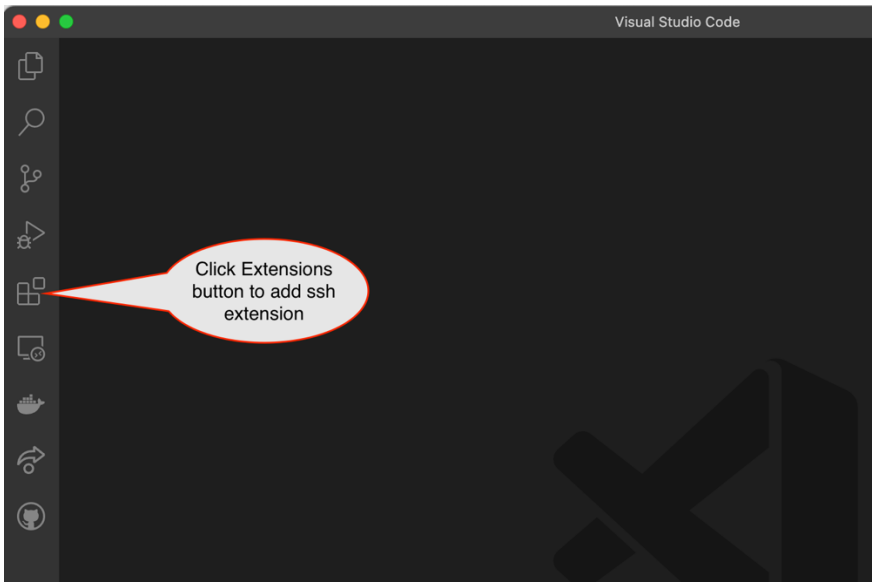
Download and install Visual Studio Code, downloadable from: https://code.visualstudio.com

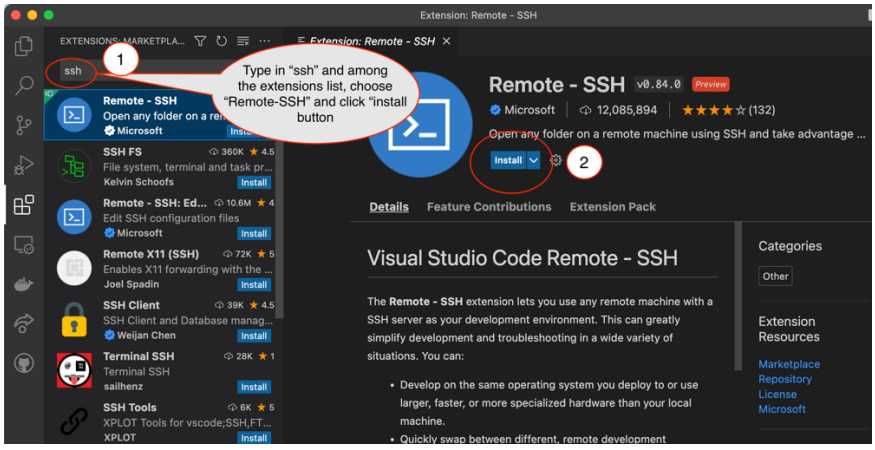
Useful resources:

- Configuring VS Code to work with core teaching servers – Week 2 Resources on Canvas
- Getting Started -- https://code.visualstudio.com/docs
- Remote Development using SSH -- https://code.visualstudio.com/docs/remote/ssh#_connect-to-a-remote-host

1.1.2.2 Add Remote-SSH extension

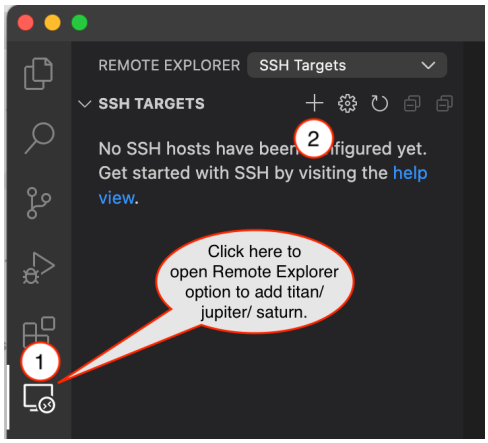
Adding extensions (adding more features to VS Code application) is super-easy. Click on Extensions button on the left-hand toolbar and enter 'ssh' to search the extension, choose 'Remote-SSH' and install.

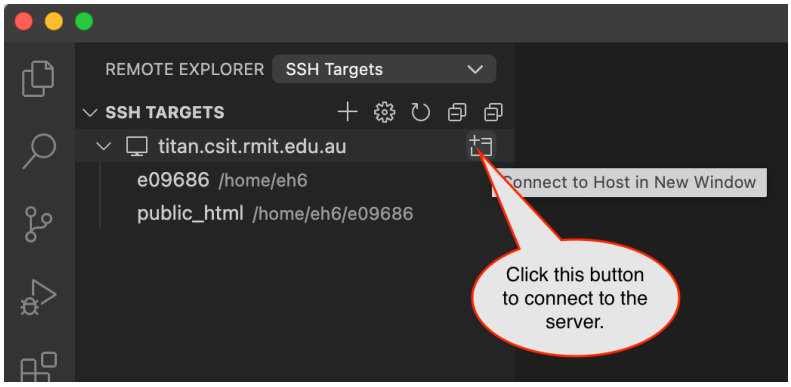


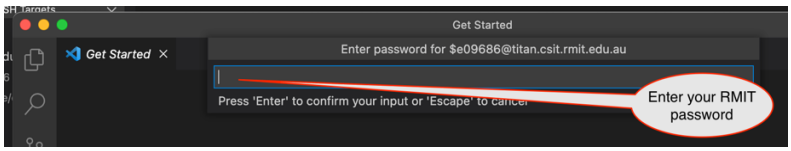


1.1.2.3 Configure your ssh connection

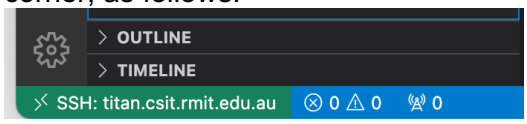
Enter server name and your RMIT credentials to access one of the core teaching servers.
Server name: titan.csit.rmit.edu.au or Jupiter.csit.rmit.edu.au or Saturn.csit.rmit.edu.au







If you are successfully connected, you should see the Green connection icon at the bottom left corner, as follows:

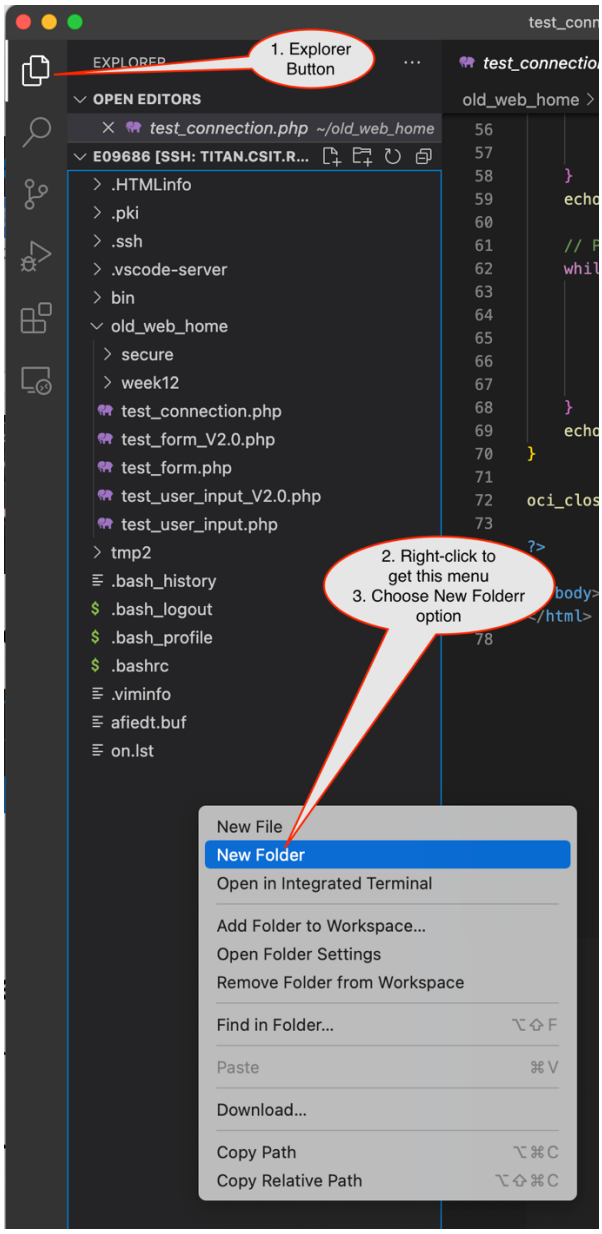


Now your VS Code is ready to start coding your web application.

1.1.2.4 Set up web hosting directory.

On core teaching servers, the WEB_HOME is the public_html directory. If this directory does not exist, create it as follows:

On the left-hand menu, click ‘Explorer’ button. Then, on the server list, under titan.csit.rmit.edu.au area, right-click and choose ‘Add Folder’ option from the pop-up menu.



Add a folder named ‘public_html’.

Then, get inside (currently empty) ‘public_html’ folder.

Add a new file, named ‘index.html’, and, enter some simple HTML code to test if your setup is working as expected.

```
<html>
<body>
Hello!
</body>
</html>
```

After saving it, to test it, open a web browser, and enter the URL of your new website:
http://titan.csit.rmit.edu.au/~s1234567
You should be able to see your first web page.

Now you are ready to start the next step.

Your Lab assistant will demonstrate you how to configure and use VS Code to access and edit your web application.

1.1.3 Test Oracle Database Connectivity.

In this step, we check if the web server (with PHP) can communicate with the backend database server. PHP and Oracle use a suite of APIs (called oci8 API) to communicate each other. If you haven't done yet in your pre-lecture activities, please visit:
https://www.php.net/manual/en/book.oci8.php
for more information on oci8 suite.

Among many functions, there are four core functions that you will always execute to fetch data from an Oracle database.
`oci_connect()` -- Connects to an Oracle database. Returns a connection identifier needed for most other OCI8 operations.

`oci_parse()` -- Prepares an Oracle statement for execution

`oci_execute()` -- Executes a statement. After execution, statements like *INSERT* will have data committed to the database by default. For statements like *SELECT*, execution performs the logic of the query. Query results can subsequently be fetched in PHP with functions like oci_fetch_array().

`oci_fetch_array()` -- Returns an array containing the next result-set row of a query. Each array entry corresponds to a column of the row. This function is typically called in a loop until it returns **FALSE**, indicating no more rows exist.

Download `test_connection.php` script from Canvas (in Week 2 Learning Resources area) into a folder in your computer. From there, you can drag and drop it into the public_html folder in VS Code.

This script is your starting point in writing PHP scripts that embed oci8 functions. This sample php script runs two SQL statements: (1) shows Oracle server's system date; (2) second SQL statement fetches data from movie table. Finally, a mix of PHP and HTML code is used to format the result and present it back to the client.

Make sure this file is in public_html folder in your directory tree.

Note down the following PHP statements in the script:

```
$username = 'YOUR RMIT login s1234567';
$password = 'YOUR ORACLE PASSWORD';
$servername = 'talsprddb01.int.its.rmit.edu.au';
$servicename = 'CSAMPR1.ITS.RMIT.EDU.AU';
$connection = $servername. "/" .$servicename;
```

```
$conn = oci_connect($username, $password, $connection);
```

Next, SQL statement is parsed and attached to a PHP variable.

```
$stid = oci_parse($conn, 'SELECT * FROM movie');
```

Now, it is ready to execute.

```
oci_execute($stid);
```

Finally, the rows are fetched using a loop.

```
while ($row = oci_fetch_array($stid,
                    OCI_ASSOC+OCI_RETURN_NULLS))
{
  ...
}
```

Test it on a browser, using the URL:
`titan.csit.rmit.edu.au/~s1234567/test_connection.php`

If it is correctly deployed on the browser, further explore the contents in this php script.

Make changes to the `test_connection.php` script to:

1. Change the username to a non-existant username (say, change s1234567 to t1234567.) What was the error message you got when you run it on the browser?
2. Change the password to an incorrect password. What was the error message you got when you run it on the browser?
3. Change the SQL statement to display titles, names of directors and year made for all movies.
4. What is the major security issue in this script? Can you propose a workable solution to this drawback?

1.1.4 Feeding user input into a PHP script.

In the above PHP script, you fetch all rows from the movie table. You were not required to enter any user inputs.
In most of the web database applications, the client requires to interact with the applications, say by inputting his/ her choices. For example, let's suppose the client likes to see movies directed by his/ her favourite director. Two scripts are required: (1) a script to render a HTML form; (2) a PHP script to receive form inputs and embed them in the corresponding SQL statements to filter data as client wants. (sometimes, programmers merge these two into one script, but, logically they are two components.)

In the following example, the simple HTML form has two fields, dir_fname and dir_surname. The action on form Submit button states which PHP script to be executed when the form is submitted. In that script, two PHP variables are used to accept form inputs and use them within the PHP script. Form input for dir_fname can be referred by $_POST["dir_fname"] PHP variable.

These PHP variables can then be used to form the SQL statement, as follows:

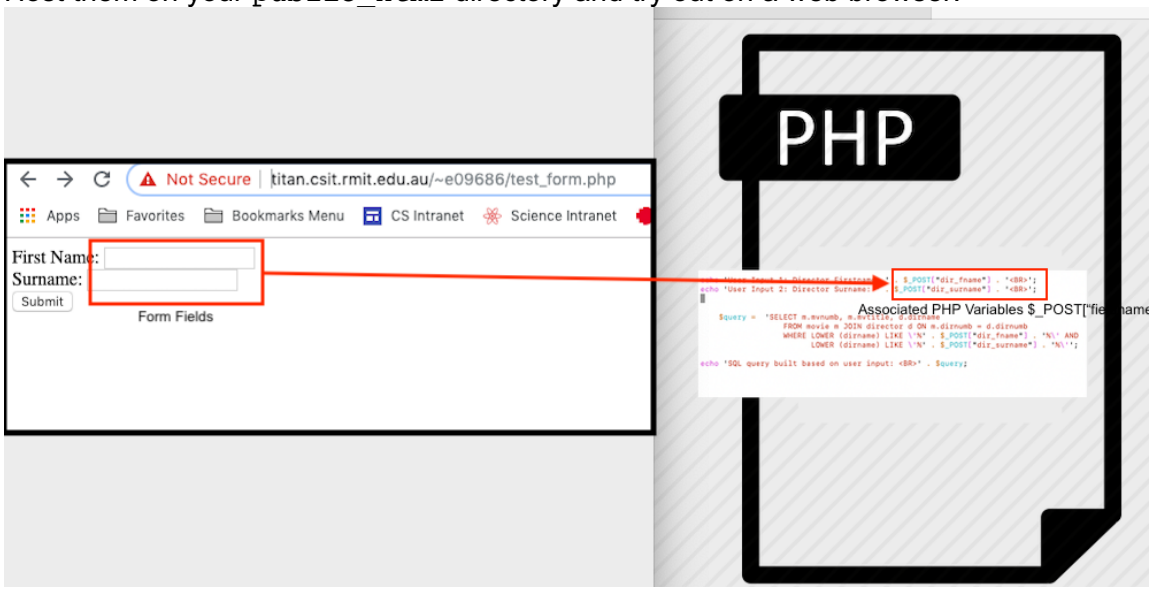
```
$query = 'SELECT m.mvnumb, m.mvtitle, d.dirname
            FROM movie m JOIN director d ON m.dirnumb = d.dirnumb
```

```
WHERE LOWER (dirname) LIKE \'%' . $_POST["dir_fname"] . '%\' AND
      LOWER (dirname) LIKE \'%' . $_POST["dir_surname"] . '%\'';
```

(The above is concatenating strings with string variables. Note that . operator in PHP is used to concatenate strings. Also note that single semicolons (that are part of the SQL) are to be escaped with a backslash (\). If you don't escape, PHP incorrectly assumes that's the end of the string.)

Download `test_form.html` script and `test_user_input.php` script from Canvas (in Week 2 Learning Resources area).

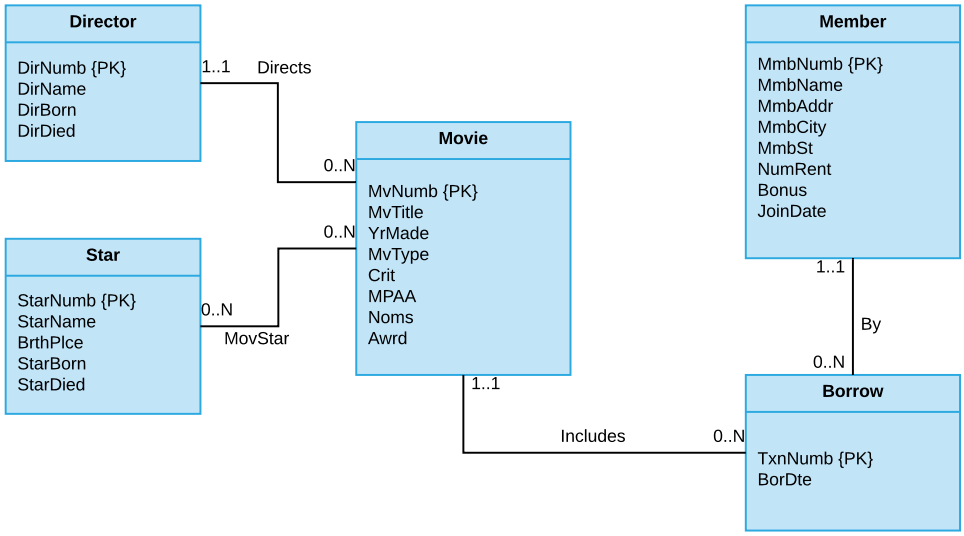
Host them on your `public_html` directory and try out on a web browser.



2 Web Database Applications Question

The movies database is built to store information about movies, directors, and stars (i.e. actors and actresses) in a video store. In addition, it stores information on members (who got active memberships with the video store) and their borrowing transactions.

A sketchy ER diagram for the movies database is given below.



In last week's lab, we developed SQL statements to answer the queries the client asked. In the middle of the dialogue the client asked:

"Can you show me the full list of directors, with all the movies they have directed..."

This is a situation where a simple join does not produce the required output. A simple join will show the list of directors with their movies, but, it will miss the directors who haven't directed a movie. So, you will not get the "full list" of directors. What you must use here is LEFT OUTER JOIN which displays all rows in the table in the left of the join. So, use Director in the left of the join, and Movie on the right side. Then, the query will show the desired result with NULL values to pad up Director rows with no associated movies.

The final query will have the following form:

```
SELECT dirname, mvtitle
    FROM director d LEFT OUTER JOIN movie m
         ON d.dirnumb = m.dirnumb
```

However, if you run this on SQL Developer, the output will be similar to:

DIRNAME	MVTITLE
Allen, Woody	Interiors
Allen, Woody	Manhattan
Allen, Woody	Annie Hall
Hitchcock, Alfred	North by Northwest
Hitchcock, Alfred	Rope
Hitchcock, Alfred	Psycho
Hitchcock, Alfred	Rear Window
Hitchcock, Alfred	The Birds
Hitchcock, Alfred	Vertigo
De Mille, Cecil B	Samson and Delilah
De Mille, Cecil B	The Ten Commandments
Kramer, Stanley	Inherit the Wind
Kramer, Stanley	Judgment at Nuremberg

While this is correct, it doesn't look professional to show director names multiple times. A more professional-looking output will be similar to:

DIRNAME	MVTITLE
Allen, Woody	Interiors
	Manhattan
	Annie Hall
Hitchcock, Alfred	North by Northwest
	Rope
	Psycho
	Rear Window
	The Birds
	Vertigo
De Mille, Cecil B	Samson and Delilah
	The Ten Commandments
Kramer, Stanley	Inherit the Wind
	Judgment at Nuremberg

A simple SQL query cannot produce this output. However, it is possible to write a PHP application to produce the desired output in a web database application.

1. What's your algorithm/ approach you would use in your PHP application?
2. Write a PHP script to produce a report to list names of all directors with the movie titles they have directed. Your output should be similar to the second report above.