|  |
| --- |
| Colchester_Institute_Logo.png |
| Webfolio Project – Part 1 |
| Web Programming |
|  |
| **117906** |
| **21/11/14** |

**Tutor: Julia Hunter**

|  |
| --- |
| This report contains the planning and back-end design for a website that will be used to create and house a collection of student webfolios. This document contains a functional specification, information on the design and construction of the database, details concerning the security of the database and the scripting required to test the database. All scripts can be accessed at: https://github.com/AdrianLeach/WebProg |

Contents

[1. Introduction 5](#_Toc404331631)

[2. Reflection 6](#_Toc404331632)

[3. Functional Specification 7](#_Toc404331633)

[o Main Flow 7](#_Toc404331634)

[ Alternate Flow 2a: User Performs a Search 7](#_Toc404331635)

[ Alternate Flow 4a: User Clicks on Register Button 7](#_Toc404331636)

[ Alternate Flow 6a: User Cancels Log-In Operation 8](#_Toc404331637)

[ Alternate Flow 6b: User Enters Facebook Credentials 8](#_Toc404331638)

[ Alternate Flow 8a: User Enters Invalid Log-on Details 8](#_Toc404331639)

[ Alternate Flow 9a: User clicks on the ‘Edit Content’ control 8](#_Toc404331640)

[ Alternate Flow 9b: User Clicks on ‘Delete Topic’ Button 9](#_Toc404331641)

[ Alternate Flow 12a: Data Entered Does Not Pass Safety Check 9](#_Toc404331642)

[o Diagram Illustrating Application Flow 10](#_Toc404331643)

[4. Conceptual Database Design 11](#_Toc404331644)

[o Requirements 11](#_Toc404331645)

[ Purpose 11](#_Toc404331646)

[ Access Levels 11](#_Toc404331647)

[ User Training 11](#_Toc404331648)

[ Content 11](#_Toc404331649)

[o Database Rules 12](#_Toc404331650)

[o Entity Relationship Diagram 13](#_Toc404331651)

[ ERD Summary 14](#_Toc404331652)

[ ERD Description 14](#_Toc404331653)

[5. Building the Database 15](#_Toc404331654)

[ Creating a new table: 15](#_Toc404331655)

[ Inserting Records Into a Table: 15](#_Toc404331656)

[ Updating Records: 16](#_Toc404331657)

[ Changing the Datatype of an Attribute 17](#_Toc404331658)

[ Adding a New Column 17](#_Toc404331659)

[ Updating Records 18](#_Toc404331660)

[ Arranging Columns 18](#_Toc404331661)

[ Enforcing Referential Integrity 18](#_Toc404331662)

[6. Error Checking 21](#_Toc404331663)

[o Connection 21](#_Toc404331664)

[o Query 22](#_Toc404331665)

[7. Security 23](#_Toc404331666)

[o Regular Expressions 23](#_Toc404331667)

[o Prepared Statements 24](#_Toc404331668)

[o HTML Entities 25](#_Toc404331669)

[o Trim 26](#_Toc404331670)

[8. Assignment Queries 26](#_Toc404331671)

[o Display Webfolios 26](#_Toc404331672)

[o Show Webfolio Content 28](#_Toc404331673)

[o Search Registered Users 30](#_Toc404331674)

[o Add a New Topic 32](#_Toc404331675)

[o Register a New Student 35](#_Toc404331676)

[ Testing 40](#_Toc404331677)

[o Delete A Topic 41](#_Toc404331678)

[o Update a Topic 44](#_Toc404331679)

[9. Future Development 48](#_Toc404331680)

[10. References 49](#_Toc404331681)

[11. Appendix 1: Logical Database Design 50](#_Toc404331682)

[o Database Dictionary 50](#_Toc404331683)

[ Student 50](#_Toc404331684)

[ Course 50](#_Toc404331685)

[ Register 50](#_Toc404331686)

[ Content 50](#_Toc404331687)

[ Facebook 51](#_Toc404331688)

[o Normalisation 51](#_Toc404331689)

[ Un-normalised Form 51](#_Toc404331690)

[ First Normal Form 52](#_Toc404331691)

[ Second Normal Form 53](#_Toc404331692)

[ Third Normal Form 54](#_Toc404331693)

[12. Appendix 2: Additional Scripts 56](#_Toc404331694)

[o Login 56](#_Toc404331695)

[o Constants 56](#_Toc404331696)

[o An Insert Query with a Prepared Statement Using PDO 56](#_Toc404331697)

[13. Appendix 3: MySQL Code 57](#_Toc404331698)

# Introduction

This assignment is the first of a two-part project that will develop a tool that allows a student to publish their webfolio online. The webfolio features persistent storage due to being integrated with a MySQL database. The webfolio will be viewable by both the student and other visitors so needs to be designed and implemented accordingly. This part of the assignment looks at the planning of the project as a whole and also the necessary steps to organise and develop the back-end of the webfolio tool. As such, a functional description will be completed, followed by a relational database. Next, a variety of PHP scripts will be written. These scripts will be tested and ultimately allow the user to interact with their webfolio and provide all users of the webfolio tool with an interactive experience. Finally, the matter of SQL injection will be considered and a robust security system put in place to ensure the data stored is not compromised.

The software used to create this assignment includes:

* Sublime Text 2 (all scripting)
* Snipping Tool (screenshots)
* Word (Assignment write-up)
* Firefox (Testing scripts and accessing the database)
* XAMPP (Provide a means for local testing)
* PHPMyAdmin

All scripts will be submitted within a compressed file along with this report, and are also available for access at the following URLs:

<https://github.com/AdrianLeach/WebProg>

|  |  |
| --- | --- |
| Script | URL |
| View All Webfolios | http://adrianl8113.ccacolchester.com/webprog/display\_webfolios.php |
| View Content of a Webfolio | http://adrianl8113.ccacolchester.com/webprog/show\_content.php |
| Insert a New User | http://adrianl8113.ccacolchester.com/webprog/insert\_user.php |
| Insert a New Topic | http://adrianl8113.ccacolchester.com/webprog/add\_topic.php |
| Update a Topic | http://adrianl8113.ccacolchester.com/webprog/update\_topic.php |
| Delete a Topic | http://adrianl8113.ccacolchester.com/webprog/delete\_topic.php |
| Search for a User | http://adrianl8113.ccacolchester.com/webprog/search\_users.php |
| Login File | http://adrianl8113.ccacolchester.com/webprog/login.php |
| Constants File | http://adrianl8113.ccacolchester.com/webprog/constants.php |

Figure 1: List of URLs for Scripts

# Reflection

This was a very interesting assignment as it required sound knowledge across a variety of areas. Designing the database illustrated how important it is to carefully analyse exactly how a system will be used, as strictly adhering to the theoretical constraints of 3NF would have actually lowered the effectiveness of the application in this case. After much effort and thought I realised that there are times where a developer needs to focus specifically on the application (and its intended use) at hand as opposed to following the default approach in every situation. I feel this is a valuable lesson.

Security was an area that I spent considerable time on. Comparing and contrasting the various techniques used to prevent malicious code from entering the database (or a user’s computer) was an interesting exercise. I have tried to use the most effective form of protection in each script, paying attention to whether it is the database or the end user who would be most at risk. With my current knowledge I feel I have settled on the solution that best suits the needs of the database but am aware that there is an incredible amount left to learn in this area.

There are many challenges moving into Assignment 2. Integrating Facebook users into my database successfully will be difficult due to my inexperience with the Facebook API but also Facebook itself. In order for my database to function correctly it is important that each user has a unique ID, so I hope to mitigate any issues by asking Facebook users to create an account with the webfolio tool after logging in. From here it should be a case of matching a registered user’s details with those from the Facebook API.

Other areas that will require attention will be correctly storing user’s images using Dreamhost (details of how I propose to do this are contained in this assignment) and identifying which topic has been selected by a user for editing where the user has multiple topics. Additionally, structuring the front-end will be an interesting task as it will require working with the JSON output from this assignment. It may also involve revisiting my security procedures, as using htmlentities or mysqli\_real\_escape\_string could potentially interfere with file paths required for image hosting.

During my research on prepared statements I noticed the prevalence of PDO within the code developers use to interact with their databases. After investigation I feel it is both more logical and more efficient than MySQLi, as well as offering greater compatibility with databases built on a non-MySQL platform. Therefore I will be learning more about PDO moving forward. A brief example of a prepared statement constructed using PDO can be found in the Appendix.

I also found keeping the content of this assignment down to a reasonable level to be a challenge. I felt that many of the areas covered by this assignment - such as database design and security - could warrant an assignment of their own. However I tried to ensure that only the data specifically relevant to the assignment remained in the body of the report, with the supporting documentation placed into the appendix, in order to balance readability with my desire to achieve the best mark I could.

# Functional Specification

A functional specification describes the behaviour of a system from the perspective of the user. As such it can be considered a summary of the user’s interaction with the system. Below is the functional specification for this particular application. It is presented in the style of a use case, in order to clearly illustrate the flow of the program and ensure all eventualities are accounted for.

## Main Flow

1. The user opens the index page which contains links to the student webfolios as well as several controls. These controls allow the user to log-in to either the webfolio site or Facebook and also to search the site.
2. The user selects a specific webfolio from the list of student webfolios.
3. The system displays the contents of the selected webfolio.
4. The user clicks on the control marked log-in
5. The system displays a page that asks for the user’s log-in credentials, providing the opportunity to enter either student log-in details or Facebook authentication information.
6. The user enters their student log-on details
7. The system validates the user’s log-in details by matching the input with the data stored for that individual
8. The system displays the webfolio owned by the student who successfully logged in, comprising the details they uploaded and also controls that facilitate adding, editing or removing data
9. The student clicks the ‘Add Topic’ button
10. The system displays the ‘Add Topic’ page, comprising fields that enable the student to enter their topic title and the actual content
11. The student adds their information then clicks Submit to send the data to the system
12. The system performs checks to verify that the data is not harmful
13. The user receives a message: “Thank you. Your topic has been created, and you will now be returned to your webfolio”. The student’s new topic is saved to the system
14. The system displays the student’s page, complete with updated content

### Alternate Flow 2a: User Performs a Search

1. The user enters their search criteria into the search field
2. The system displays all registered users who match the search criteria
3. The user selects one result and is taken to the corresponding webfolio
4. Return to Main Flow – Step 3

### Alternate Flow 4a: User Clicks on Register Button

1. The user is presented with a form comprising 7 fields – Desired Username, Desired Password, Repeat Password, First Name, Last Name, Date of Birth, Email Address
2. The user completes the form and passes validation tests
3. The user’s details are recorded
4. Return to Main Flow – Step 8

#### Alternate Flow 2a: User Fails Validation Tests

1. The user is provided with a message specifying which field caused the form to fail validation: “Your name input was not valid. Please do not use symbols or numbers.” || "Your username or password was not accepted. Please use only letters, numbers and underscores." ||“Sorry your passwords did not match. Please try again.” || “Your date of birth was not accepted. The format required is YYYY-MM-DD.” || “Your email address was not accepted. Please try again.”
2. User is returned to Step 1

#### Alternate Flow 2c: User Clicks on Cancel Button to End Registration

1. Return to Main Flow – Step 1

### Alternate Flow 6a: User Cancels Log-In Operation

1. The user is returned to the main index page
2. Return to Main Flow – Step 1

### Alternate Flow 6b: User Enters Facebook Credentials

1. System validates entered details by checking with Facebook’s records
2. System stores the user’s Facebook authentication details
3. System attempts to match Facebook details stored for first and last name with stored data
4. System matches Facebook details with a registered student
5. Return to Main Flow – Step 8

#### Alternate Flow 1a: Facebook does not recognise the user’s details

1. The user is presented with a message: “Unfortunately we were unable to log you into Facebook. Please check your log-in details and try again.”
2. Return to Main Flow – Step 1

#### Alternate Flow 3a: System is not able to find a match between Facebook details and student information

1. System displays a message: “Thank you for logging in with Facebook. Please complete the following short form to gain access to the full functionality of this site.”
2. Return to Alternate Flow 4a – Step 1

### Alternate Flow 8a: User Enters Invalid Log-on Details

1. System displays a message: “Unfortunately the details you entered did not match our records. Please check and try again.”
2. Return to Step 1

### Alternate Flow 9a: User clicks on the ‘Edit Content’ control

1. The system displays the Edit Topic page and associated controls
2. The user updates their content and clicks Submit
3. The system checks the content for any undesirable input
4. The system updates records and displays a message: “Thank you for updating this section of your webfolio. Your changes have been successfully saved. We will now return you to your main page.”
5. Return to Main Flow – Step 12

#### Alternate Flow 3a: Dangerous Code Discovered in Updated Content

1. System displays message: “Regrettably our security systems discovered unwanted code within your content and it has therefore not been saved. You will now be returned to your Webfolio.”
2. Return to Main Flow – Step 8

### Alternate Flow 9b: User Clicks on ‘Delete Topic’ Button

1. The system displays a message box with two choices underneath – Yes and No. The message is: “Are you sure you want to permanently remove this topic and all its associated content? Once performed this action cannot be undone.”
2. The user clicks on the ‘Yes’ button
3. The system displays a message: “The topic and its contents have been deleted. You will now be returned to your webfolio.”. The system removes the selected topic and content
4. Return to Main Flow – Step 14

#### Alternate Flow 2a: User Clicks on the ‘No’ Button

1. A message is displayed: “No content has been removed. You will not be returned to your Webfolio.”
2. Return to Main Flow – Step 8

### Alternate Flow 12a: Data Entered Does Not Pass Safety Check

1. Display a message: “Regrettably our security systems discovered unwanted code within your content and it has therefore not been saved. You will now be returned to your Webfolio.”
2. Return to Main Flow – Step 8

## Diagram Illustrating Application Flow

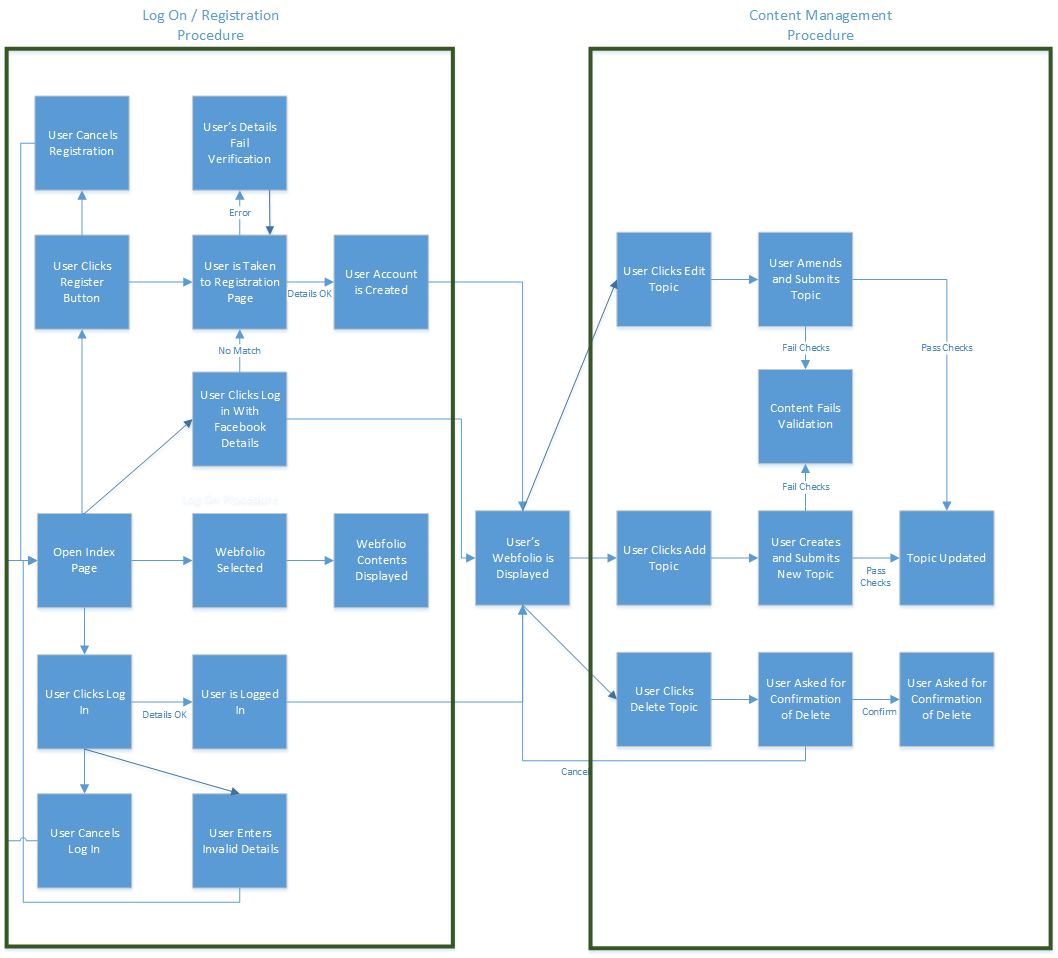


Figure 2: Design Flow of Functional Specification

# Conceptual Database Design

Before the actual development of the database can begin it is vital to first understand exactly what information will be stored in it. This will help to shape the purpose of the database.

## Requirements

### Purpose

The database will serve as a storage facility for the data needed by the Webfolio application. The database will be designed in such a way that there is as little data redundancy as possible and errors are kept to an absolute minimum.

Without an efficient database the webfolio tool would suffer considerably. There would be a very real chance that data would be lost forever, be it student details or their uploaded content. Also, users of the webfolio tool are not necessarily IT-literate, and so the manner in which they interact with their data needs to be as streamlined and intuitive as possible. Using a well-structured database should help as it will enable succinct PHP queries to be written. In turn, these queries will be used by the tool by way of interactive buttons that should be simple to understand and operate.

### Access Levels

Users will be able to interact with the database to both store and retrieve data, although levels of access apply. Students who have logged in to the Webfolio tool will be able to retrieve all data pertinent to themselves, and will also have clearance to add their own data to the database. Users who have not logged in will have access to all public-facing data, so they will be able to view all webfolios, but will not be able to make any changes. Users who log in via Facebook but have no student log-in will share the same access privilege as users who have not logged in, except they will now be able to make use of the Facebook controls contained within the site. This will enable them to share a student’s portfolio with their social network contacts, for example. Facebook users who do have a student log-in will have the same level of access as logged-in student users, but again have access to the social media tools.

### User Training

In order to keep the tool as accessible as possible, it is assumed that each user has received no training whatsoever on database management, and possesses only basic IT skills such as being able to navigate web pages competently.

### Content

The database will be used to house text only. Images that are uploaded will be stored within a dedicated file on Dreamhost, with only a reference to that file stored on the database.

## Database Rules

* The database must be hidden from users – there should be no reference to where data is stored
* The database must be easy to query – users must be able to interact with it by simply clicking a button or completing a form
* User data must be kept secure – a student’s private information can only be accessed by that student
* The database must be safe from all known methods of attack, including SQL Injections
* Students must be able to add, edit and remove their own webfolio content from the database
* Students must be able to log in using their Facebook credentials
* Every user who logs in must be issued with a Student ID and password for the Webfolio site
* Facebook integration must be achieved so that it is possible to match Facebook information with database records
* Each student must have a unique reference number that persists even if the student is removed from the database
* Changes to course details and content must be possible without affecting student data in any way
* A unique reference number must be given to each piece of content uploaded to the database
* Course tutors must be able to teach across multiple faculties without compromising the data stored
* Students can enrol on any number of courses without creating any errors within the database
* Each piece of content uploaded to the database must be linked to the student who uploaded it

## Entity Relationship Diagram

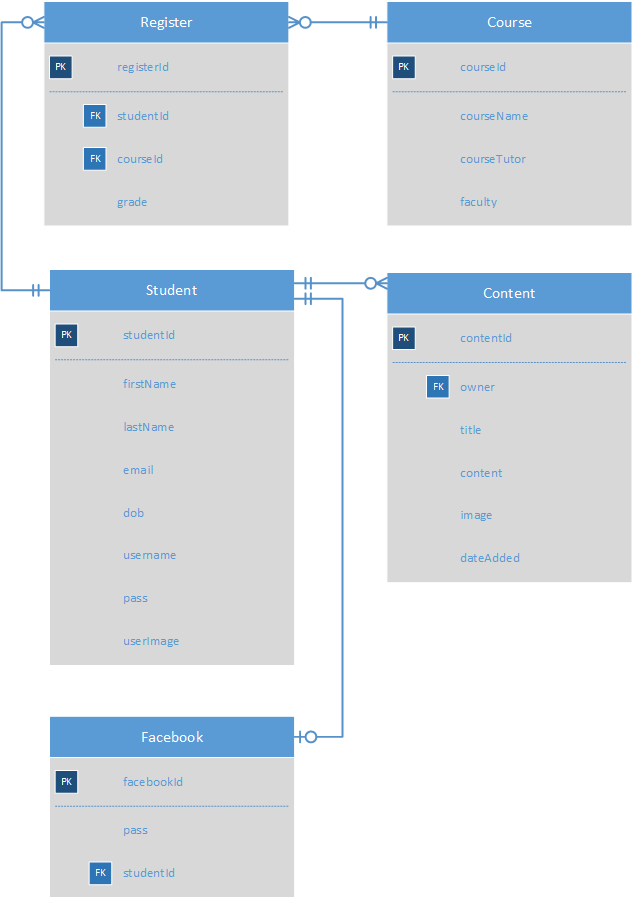


Figure 3: ERD for this Application

### ERD Summary

Each student can register on a course zero to many times. Each registration ID must belong to one, and only one, student. Each course can appear on the Register zero to many times, but each register ID must have exactly one course assigned to it.

A student can post zero to many items of content to a webfolio, but each Content ID must have exactly one student associated to it.

A student can have zero to one Facebook ID, but each Facebook ID must have exactly one match in the Student table.

### ERD Description

The database contains a total of 5 tables. The Student table contains the personal details for each student, including their name, email address, date of birth and log-in credentials to the webfolio site.

The Course table contains a list of courses, as well as the tutors assigned to them and the faculty the course belongs to. As it is possible that a student could register for many courses and a course could be allocated to many students it was necessary to create an additional table with which to join these relations, thereby breaking the many-to-many relationship. The Register table was designed with this function in mind. Each student can have multiple registrations (allowing them to sign onto many courses) but each registration can only belong to one student. Similarly, each course can be entered onto the Register table many times (so more than one student can attend it) but each registration can only be for one course. In order to successfully join the two relations a foreign key referencing each of them is included in the Register table. Finally, this table includes a grade attribute, so that the students overall grade for that particular course can be recorded in a manner that is unique to the student / course combination represented by the registerId attribute.

Next, a table called Content was created to house the data required for the webfolios to function. This table records the content title, the actual data, the date it was added and, critically, who the content belongs to. A reference to the studentId field in the Student table was used for this purpose. Facebook users will be assigned a studentId, so that it will not be possible for any content to be lost or appended to the incorrect webfolio.

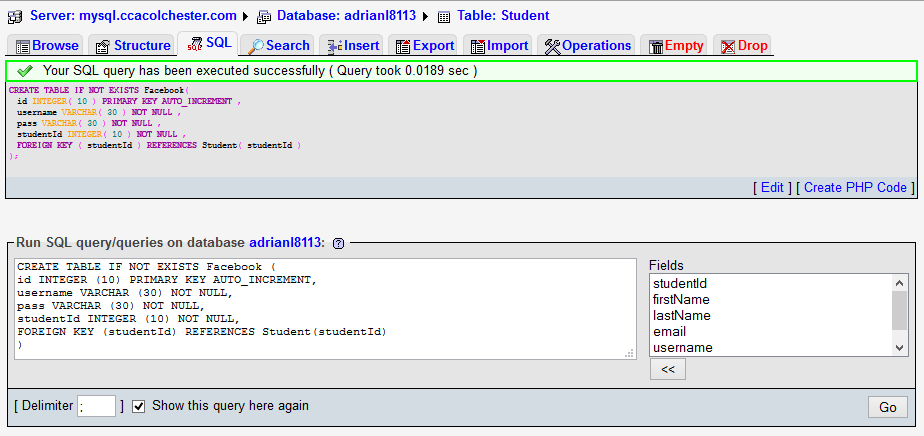
Finally, to enable integration with Facebook users, a table called Facebook was created. This holds data pertinent to Facebook’s log-on details, but also references the Student table. Students who have a website log-in already will be matched to their profile (by comparing the values for the names held in both relations), and Facebook users who do not have a profile will be asked to create one.

**Note**: Please refer to Appendix 1: Logical Database Design to view full details as to how this database application was designed.

# Building the Database

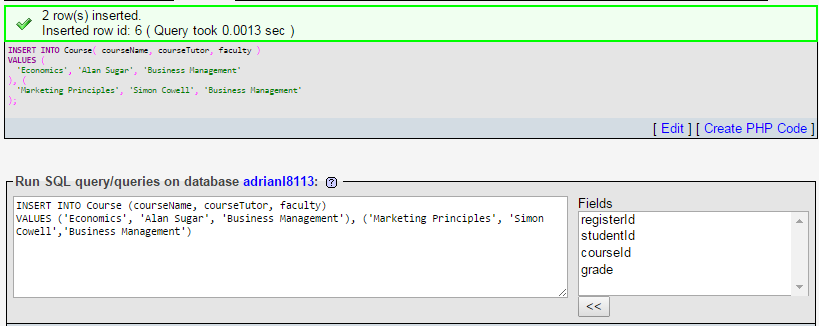
### Creating a new table:

|  |
| --- |
| CREATE TABLE IF NOT EXISTS Facebook (  id INTEGER (10) PRIMARY KEY AUTO\_INCREMENT,  username VARCHAR (30) NOT NULL,  pass VARCHAR (30) NOT NULL,  studentId INTEGER (10) NOT NULL,  FOREIGN KEY (studentId) REFERENCES Student(studentId)  ); |



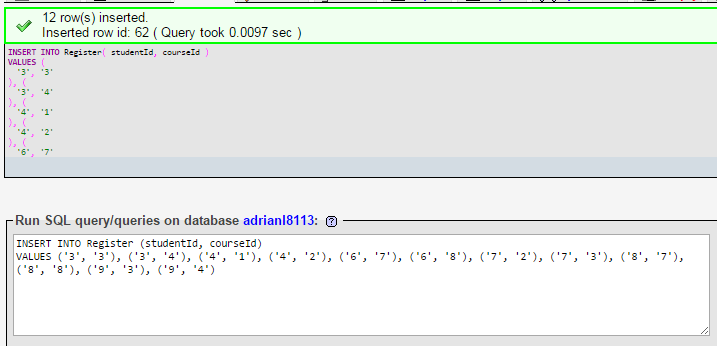
### Inserting Records Into a Table:

|  |
| --- |
| INSERT INTO Course (courseName, courseTutor, faculty)  VALUES ('Economics', 'Alan Sugar', ‘Business’), ('Marketing Principles', 'Simon Cowell','BusinessManagement'); |



The following query is how user’s student ID and course name details are combined to give them a unique registration ID for each course they enrol on.

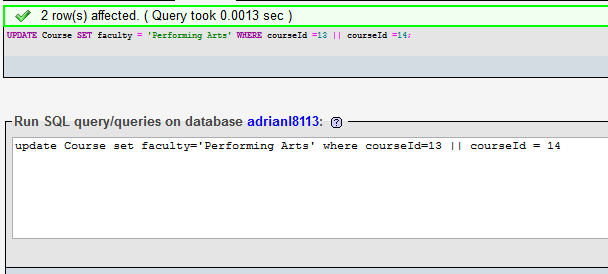
|  |
| --- |
| INSERT INTO Register (studentId, courseId)  VALUES ('3', '3'), ('3', '4'), ('4', '1'), ('4', '2'), ('6', '7'), ('6', '8'), ('7', '2'), ('7', '3'), ('8', '7'), ('8', '8'), ('9', '3'), ('9', '4') |



### Updating Records:

An error occurred when updating the Course table, as some of the values entered in the ‘faculty’ attribute did not append to the correct records. The reason for this is that faculty was set up using the enumeration data type, and would therefore only recognise values that were predefined. After modifying the attribute, the following statement was used to update two records to ensure they had the correct value stored:

|  |
| --- |
| UPDATE Course SET faculty='Performing Arts' WHERE courseId=13 || courseId = 14; |



|  |
| --- |
| UPDATE Course  SET faculty='Computing'  WHERE courseID >= 1 AND courseId <= 4; |

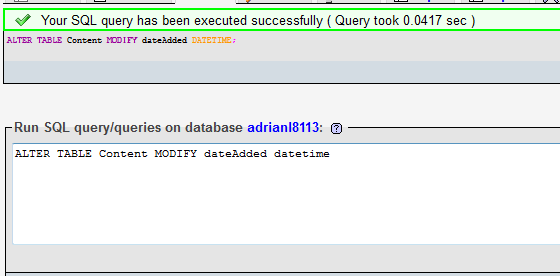
Many records were also inserted by using a URL in conjunction with a PHP script using $\_GET requests to add records programmatically. An example of this is below:

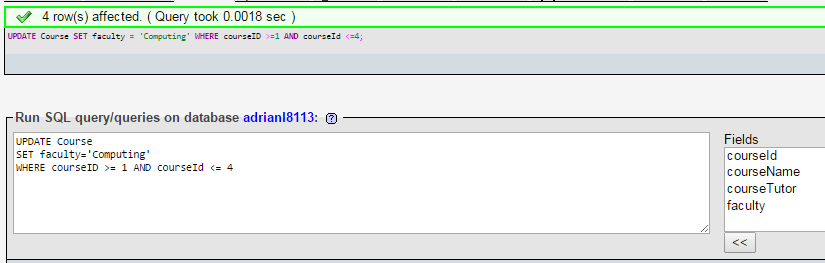
<http://adrianl8113.ccacolchester.com/webprog/register_user.php?firstName=Shigeru&lastName=Miyamoto&email=stillgreat@nintendo.co.jp&username=mario&pass=bowser&dob=1969-11-06>

### Changing the Datatype of an Attribute

This change was made to enable the database to record the time and date a piece of content was updated automatically.

|  |
| --- |
| ALTER TABLE Content MODIFY dateAdded datetime |





### Adding a New Column

It was decided that users should have the ability to upload one image per topic on their webfolio. To support this a new field was required within the Content table.

|  |
| --- |
| ALTER TABLE Content ADD image varchar(100); |



### Updating Records

It was necessary to amend some records, either because the values were not inserted along with the rest of the data, or because the stored values needed to be altered. Below is an example of a hypothetical situation where a user requested to change their username:

|  |
| --- |
| UPDATE Student SET username='issamemario' WHERE studentId=22; |



### Arranging Columns

An image column was added to the Content table after the table had been created. This meant the column was added to the end of the table. To move it to a more sensible position, the following statement was used:

|  |
| --- |
| ALTER TABLE Content MODIFY image varchar(100) AFTER content; |

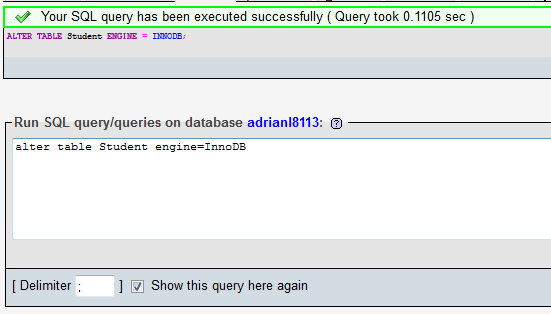
### Enforcing Referential Integrity

By default, the engine used by MySQL, MyISAM, does not support referential integrity. Although it allows the code pertaining to this concept to be entered (for example defining primary and foreign keys) it does not actually place any constraints on the data. As such the database is at risk from multiple errors, not least because any developer working with it might assume referential integrity was in force and protecting the data.

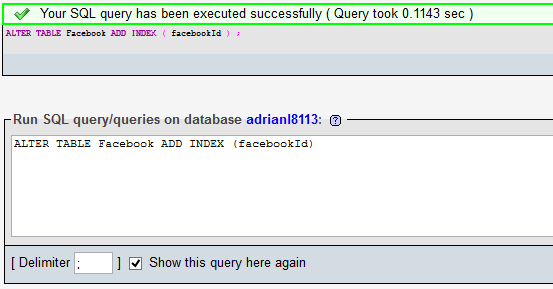
The answer to this problem is to switch the engine from MyISAM to InnoDB. The latter fully supports referential integrity. MyISAM is slightly faster than InnoDB in some circumstances, as it offers increased performance for reading data from a database. However, this application is not read-only, but has several instances where data will be written to the database. With these reasons in mind it was decided that InnoDB was the better solution for this application.

To successfully introduce the InnoDB engine it was necessary to take the following actions:

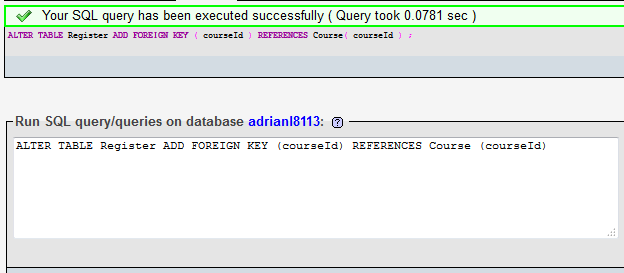
**Step 1: Update the engine for each relation.**



**Step 2: Create an index for the relation.**



**Step 3: Add Foreign Keys to Relations**



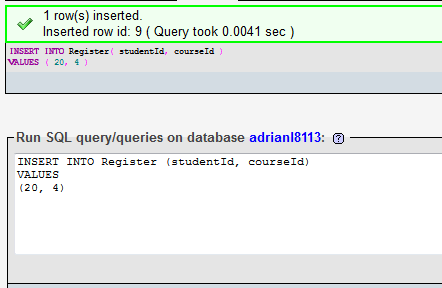
#### Testing

Before testing to check whether converting the database to run using the InnoDB engine was successful it is necessary to examine the default behaviour of the MyISAM engine. Below is the Student table:



It can be seen that, at this point, there are a total of four students.

Below is the result of trying to add a Student who doesn’t exist into the Register database. This is not something expected in a real life situation – a student who is completely unknown to the college cannot enrol on a course – but below it is evident this exact situation has occurred.



A student with the ID of 20 has managed to enrol on the Work Based Learning course, despite the known Student IDs only ranging from 1 to 4. Clearly this is an insert anomaly, and despite normalisation and setting up the relationships between the Student, Course and Register tables it has slipped through.

Following the conversion procedure outlined above, the exact same insert statement was given to the Register table. The outcome of the statement is below:



This time, MySQL was able to detect that something very wrong was happening. The rules of referential integrity state that any change to a foreign key must also be reflected in the table where that foreign key is the primary key. Of course, this cannot happen as a studentId of 20 does not exist in the Student table. As a result, a message appears informing the user that a foreign key constraint has prevented the insert statement from completing. As such it can be seen referential integrity is now in effect.

# Error Checking

Before the database can be tested for functionality it was necessary to develop a system for reporting errors should a problem arise.

To evaluate whether or not the database is accessible a test script was designed. This code is included in all query scripts, so that the user is able to receive feedback should an error occur.

## Connection

The first thing to confirm is whether it is possible to connect to the database. The following code provides this functionality:

|  |
| --- |
| $db = mysqli\_connect($server, $user, $pass, $database)  or trigger\_error("The connection was not successful. The returned error is: <BR>". mysqli\_connect\_error(), E\_USER\_ERROR); |

Here the function used to connect to the database (mysqli\_connect()) is stored in a variable. However, should the function prove false (by way of the OR operator) the error will be displayed to the user.

Although a more simple if statement could have been used here, this code was designed so that the user would receive a customised error message should connection not be possible which would include the actual error itself. It will also terminate the program upon error discovery. Otherwise the user would receive many errors (for example, the query would also generate an error as it would not run successfully with no connection) and thus encounter more difficulty when trying to resolve the problem.

It is worth noting that the variables used to create the connection are not stored locally within each script, but rather in a separate PHP file named login.php. This increases the security of the scripts and also helps to save time as it is not necessary to create and define the variables on every script.

## Query

Assuming the connection is successful, the next part of the error checking script relates to the query used. Again it will provide the user with feedback upon an error.

|  |
| --- |
| $result = mysqli\_query($db, $query)  or trigger\_error("The query failed! The SQL was:<BR/> $query.<BR/><BR/>It returned the error: "  . mysqli\_error($db) , E\_USER\_ERROR); |

A variable is created that saves the outcome of the query (which itself has been stored in a variable named $query). If the mysqli\_query() function is false an error message will be presented. This message is customised, so that the user can see the actual MySQL code that was entered, as well as the error generated. As with the connection error check, this code will terminate the program upon discovery of an error.

Trigger\_error was used in place of die() as the latter returns information that is simply not useful to the end user, such as “Cannot connect to database”. Trigger\_error enables the developer to provide the exact message they want to convey, which improves the chances of both error resolution and user experience. As mentioned above, E\_USER\_ERROR is used to stop the program at the point the error is encountered. Another alternative would be to throw an exception at this point, however in this application it was intended that the user should not be able to proceed in light of an error. As such throwing an exception would not be a practical choice.

Below is an example of the error checking providing some useful feedback as mentioned previously. In this case, while the insert query worked correctly the query to return the studentId value did not work as expected. By comparing the SQL returned with the target query (the exact same query proven to work using the database directly) it was easy to spot the error – the values used in the studentID query were not surrounded in apostrophes or speech marks and so were being treated as an attribute name instead of a value.

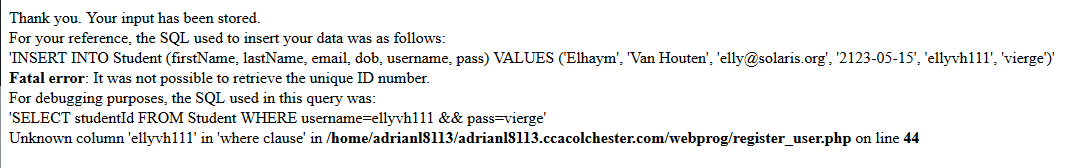


Figure 4: Output showing error message

More specific error-checking will be included on each script, but the above tests illustrate the basic error checking is functioning.

# Security

This section is dedicated to explaining the various forms of defence employed to prevent attacks on the database from corrupting either the database itself or another user’s computer.

SQL injection is a form of attack that pushes malicious SQL code into a database. These attacks are most often accomplished by way of a form. Assembling an SQL query as a string containing user input is the main cause of SQL injection vulnerability. (Clarke, 2009). For example, a hacker might enter something like: 1; drop table Student; in the final form on a page. When the query is run it would actually contain two queries – the one intended to be completed an additional – and unwanted – delete table query. Another popular injection is to provide a statement that will always prove true, then another statement to follow.

Malicious code can also be inserted with a view to harming a user, not the database. For example, some JavaScript could be entered on the name field of a form. This would lie dormant in the database, but when a user requested that name the server would send the values as HTML, and the JavaScript would activate.

Clearly these situations are something that should be avoided at all costs. Fortunately, there are many ways to combat hackers who seek to compromise the integrity of a database. Some of these are listed below.

## Regular Expressions

Regular expressions are a very powerful and flexible solution. In previous assignments these have been used for form validation, and in this case it was decided they would be useful in validating the input before it reached the query. The check below would flag any non-alphabetical character (except space and an apostrophe) present in either first name or last name, and exit from the program upon detection. Alternatively regular expressions can be used in conjunction with the preg\_replace() function, whereby the code is allowed to run but any matching characters are replaced by another character of the developer’s choosing.

|  |
| --- |
| $test\_array\_names = array($first,$last);  $regex = "/[^a-zA-Z ']+/";  $errorMsg = "Your name input was not valid. Please do not use symbols or numbers.<BR/>";    foreach ($test\_array\_names as $check)  {  if(preg\_match($regex, $check))  {  echo $errorMsg;  exit("You will now be returned to the previous page.");  }  } |

Regular expressions will also be used for the second part of the assignment in order to provide error message to users without having to send the form to the server. However, this is best considered a system to provide a better user experience as opposed to improved security. As such it is no replacement for robust server-side security.

## Prepared Statements

Prepared statements (also known as parameterised input) is another excellent way to protect against SQL injection. This technique separates the query from the user’s data completely. In this way the data entered by the user cannot interact with the query at all, rendering SQL injection very difficult. This is because the data entered is just that – data – and is not treated as code. To utilise this technique involves preparing a statement that contains the desired query, but using placeholders instead of data. After this, the data is sent to replace the placeholders, before finally the query is executed. As well as being more secure prepared statements also offer a performance advantage over regular queries, as the server only needs to parse the query once and it is only the data that is sent with each query. Additionally the code is faster to compile, as the query needs not be compiled into binary each time it is used. (Schwartz, 2012).

Below is an example of a prepared statement that also features error checking at each step. These feature in every script that involves user input.

The online blog by Mark Gosselin at <http://markonphp.com/simple-insert-mysqli/> was the single most helpful resource when learning this technique.

|  |
| --- |
| $contentId = $\_GET['content'];  $query = "DELETE FROM Content WHERE contentId = ?";  $stmt = mysqli\_stmt\_init($db);  //error check to test preparation was successful  if(!mysqli\_stmt\_prepare($stmt, $query))  {  echo "It was not possible to prepare the statement.<BR/>";  exit("The program will now exit.");  }  //i used to signify contentId is a numeric value  $bind = mysqli\_stmt\_bind\_param($stmt, "i", $contentId);  //error check to ensure bind was successful  if($bind === false)  {  trigger\_error("Error encountered during binding of parameters.", E\_USER\_ERROR);  }  //execute and check for error  $execute = mysqli\_stmt\_execute($stmt);  if($execute === false)  {  trigger\_error("Error during execution of statement." . mysqli\_stmt\_error($stmt), E\_USER\_ERROR);  }  echo "The topic and its contents have been deleted. <BR/>  You will now be returned to your webfolio.";  //close the statement  mysqli\_stmt\_close($stmt); |

## HTML Entities

In contrast to the two previous methods, HTML entities does little to protect against SQL injection or indeed an attack on the database. Instead, it converts tags and other characters that have special significance in HTML into text. The reason this defence is used is to protect users from XSS (cross-site scripting) when they perform a request from the database. XSS involves running malicious script on a user’s computer and can be used for a wide variety of attacks, from retrieving sensitive data from cookies to forcing an unwanted pop-up.

HTML entities helps by changing potentially dangerous characters into harmless text. For even greater security this is done when saving data to the database, but critically also when outputting data from it. HTML Special Characters is another option, however is more limited and therefore was not used.

To further increase the effectiveness of the function, the ENT\_QUOTES flag was used. This ensures all single and double quote marks are also converted, eliminating another source of injection. The alternative quote-converting flags, ENT\_COMPAT and ENT\_NOQUOTES are not as flexible and so were not used in this application. (Snyder, 2005)

Below is an image showing the impact of HTML entities. This is a test script that adds content to the database and uses a prepared statement. First, the form was completed using potentially dangerous characters and just the prepared statement. Next, the exact same data was inserted again, but this time with the additional protection of HTML entities and Trim:

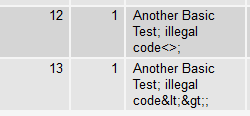


Figure 5: HTML entities in effect

Here we can see that while both uploads were successful, the second adds an extra layer of security by converting the unwanted and potentially dangerous characters into harmless text.

The full record is below:



Figure 6: The record containing sanitised data

## Trim

Trim is another function used to provide additional protection.. It works by removing leading and trailing spaces from user input. The significance of this is that some SQL injection attacks rely on using a comment to mask the code, and to make a comment in SQL a space must be included after the double-hyphen (-- ). By including this small piece of code the security of the script is further enhanced.

# Assignment Queries

This section contains information on the queries produced for this assignment.

## Display Webfolios

This PHP script is used on the main index page that is seen as soon as a user visits the site. It contains a list of all students who possess a webfolio, and also the course area that they are studying. After the query takes the required data from the database it is displayed in JSON format.

This script features a degree of security. While it is not necessary to be concerned with SQL injection as there is no user input, there is still the possibility that some malicious code could be residing in the database ready to act when output onto a user’s computer. To counter this the result set from the query is passed through the HTML Entities function. This will convert any HTML chartacters to text. Even if a script had somehow found its way into the database, it would not be able to run on the user’s computer as the code would have no tags. <script> would be read as &ltscript&gt, for example.

The URL for this file is: [www.adrianl8113.ccacolchester.com/webprog/display\_webfolios.php](http://www.adrianl8113.ccacolchester.com/webprog/display_webfolios.php)

It is used in the Functional Specification’s Main Flow at Step 1.

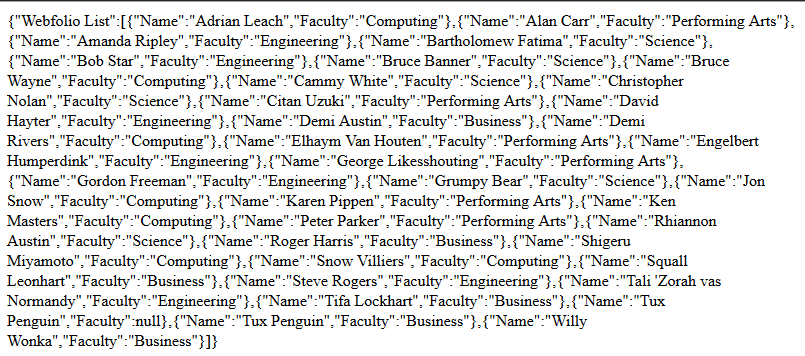


Figure 7: The output of this script in JSON format

The code for the query is below:

|  |
| --- |
| <!-- Adrian Leach - Web Programming Assignment 1 - November 2014  This script connects to the database, then uses a query to return a result set consisting of all  student's names and their course area. The students first and last names are concatenated and the  results are ordered by name. The query will only include each student once, regardless of if they  are studying multiple modules.  This file contains error checking for the connection and the query as commented below.  The result set from the query is output in JSON format -->  <html>  <body>  <?php  //include file containing log-in credentials  require\_once 'login.php';  //function to sanitise output before it reaches the user  function cleanOutput($data)  {  htmlentities(trim($data), ENT\_QUOTES);  return $data;  }  //store the connection in a variable. connection requires 4 arguments - server(host), user, password and database name  $db = mysqli\_connect($server, $user, $pass, $database)  //if not successful, display the actual error, and then terminate the program so as not to reveal further (irrelevant) error messages  or trigger\_error("The connection was not successful. The returned error is: <BR>". mysqli\_connect\_error(), E\_USER\_ERROR);  $query = "SELECT DISTINCT CONCAT(firstName, ' ', lastName) AS Name, Faculty  FROM Student  LEFT JOIN Register  ON Student.studentId = Register.studentId  LEFT JOIN Course  ON Course.courseId = Register.courseId  ORDER BY Name ASC  LIMIT 0, 50";  //call function  cleanOutput($query);  //save the results of the query OR return a comprehensive error message including SQL used if the query is unsuccessful.  $result = mysqli\_query($db, $query)  or trigger\_error("The query failed! The SQL was:<BR/> $query.<BR/><BR/>It returned the error: "  . mysqli\_error($db) , E\_USER\_ERROR);  //create an array, fetch each record in turn and append it to the array  $rows = array();  while($row = mysqli\_fetch\_assoc($result))  {  $rows['Webfolio List'][] = $row;  }    //format mysql result set as javascript online notation  print json\_encode($rows);    ?>  </body>  </html> |

## Show Webfolio Content

This script outputs the contents of a student’s webfolio as stored within the database. It functions in a very similar way to the previous script, connecting to the database, running a query, then saving the result set before outputting the data in JSON format. Again htmlentities was used to provide protection from malicious code. The results of this can be seen when examining the file path for the images stored on Dreamhost. However, contrary to expectations the htmlentities seems to be escaping the forward slash, rather than converting it into text.

It should be noted that hard-coding the data for student ID is a temporary measure used for testing purposes. The final application will use $\_SESSION to pull the value stored for the student’s ID, then use that information in the query. Also, to enable the session to be initialised it was necessary to place the session\_start() function on Line 1 of the file.

Below is an example of the output generated, in JSON format, when the file is loaded within a browser.

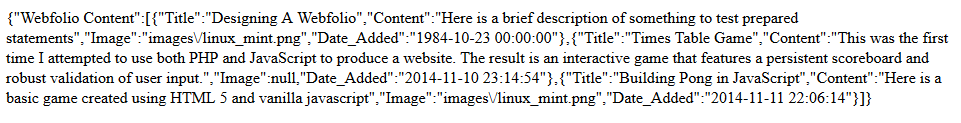


Figure 8: JSON output as produced by the Show Content script

This script will be used in the Functional Specification’s Main Flow, at Steps 8 and 14.

Below is the script for this query:

|  |
| --- |
| <?php session\_start();  ?>  <!-- Adrian Leach - Web Programming Assignment 1 - November 2014  This script is designed to allow a user to view all of the content contained within their own  webfolio. To achieve this the script will eventually use $\_SESSION to pull the user's stored  student ID, for now though it is being tested with hard-coded values.  -->  <html>  <body>  <?php  //include file containing log-in credentials  require\_once 'login.php';  //function to clean any output before it is sent - converts any HTML chars into text, including and quote marks. also removes whitespace before and after input  function cleanOutput($data)  {  htmlentities(trim($data), ENT\_QUOTES);  return $data;  }  //store the connection in a variable. connection requires 4 arguments - server(host), user, password and database name  $db = mysqli\_connect($server, $user, $pass, $database)  //if not successful, display the actual error, and then terminate the program so as not to reveal further (irrelevant) error messages  or trigger\_error("The connection was not successful. The returned error is: <BR>". mysqli\_connect\_error(), E\_USER\_ERROR);  //the final application will use $\_SESSION to retrieve the stored studentId  //$owner = $\_SESSION['studentId'];  $owner = 1;  $query = "SELECT title AS Title, content AS Content, image AS Image, dateAdded AS Date\_Added  FROM Content  WHERE owner = $owner  ORDER BY dateAdded ASC";  cleanOutput($query);  //$safeQuery = strip\_tags($query);  //save the results of the query OR return a comprehensive error message including SQL used if the query is unsuccessful.  $result = mysqli\_query($db, $query)  or trigger\_error("The query failed! The SQL was:<BR/> $query.<BR/><BR/>It returned the error: "  . mysqli\_error($db) , E\_USER\_ERROR);  $rows = array();  while($row = mysqli\_fetch\_assoc($result))  {  $rows['Webfolio Content'][] = $row;  }    //format mysql result set as javascript online notation  print json\_encode($rows);    ?>  </body>  </html> |

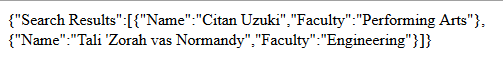
## Search Registered Users

This script provides users of the site with a way to search for a student by name. The query makes use of a wildcard search which places any input from the user between two wildcard characters. This allows the user to be as vague or as specific as they like when searching.

The script contains some security to prevent malicious code being returned along with the results of the search. It outputs the result in JSON. Below is the result of a search for ‘shaun’:



Here are the results for ‘z’:



The script will be used in the Functional Specification’s Alternate Flow 2a.

Below is the code for this script:

|  |
| --- |
| <!-- Adrian Leach - Web Programming Assignment 1 - November 2014  This script is used to find registered students. A user can enter enter their search criteria  which is pulled through to this PHP script and then used in the MySQL query.  The output is encoded in JSON format. -->  <html>  <body>  <?php  //include file containing log-in credentials  require\_once 'login.php';  //store the connection in a variable. connection requires 4 arguments - server(host), user, password and database name  $db = mysqli\_connect($server, $user, $pass, $database)  //if not successful, display the actual error, and then terminate the program so as not to reveal further (irrelevant) error messages  or trigger\_error("The connection was not successful. The returned error is: <BR>". mysqli\_connect\_error(), E\_USER\_ERROR);  //temporary storage - user can enter any search criteria via URL for now, will use $\_POST for app  $searchTerm = $\_GET['search'];  $query = "SELECT DISTINCT CONCAT( firstName, ' ', lastName ) AS Name, Faculty  FROM Student  LEFT JOIN Register ON Student.studentId = Register.studentId  LEFT JOIN Course ON Course.courseId = Register.courseId  WHERE firstName LIKE '%$searchTerm%'  OR lastName LIKE '%$searchTerm%'  ORDER BY Name ASC  LIMIT 0 , 50";  //convert any HTML characters (e.g. tags) to text, rendering any malicious code unable to run  $safeQuery = htmlentities(trim($query));  //save the results of the query OR return a comprehensive error message including SQL used if the query is unsuccessful.  $result = mysqli\_query($db, $safeQuery)  or trigger\_error("The query failed! The SQL was:<BR/> $query.<BR/><BR/>It returned the error: "  . mysqli\_error($db) , E\_USER\_ERROR);  //cretae an array, fetch each record in turn and append it to the array  $rows = array();  while($row = mysqli\_fetch\_assoc($result))  {  $rows['Search Results'][] = $row;  }    //format mysql result set as javascript online notation  print json\_encode($rows);    ?>  </body>  </html> |

## Add a New Topic

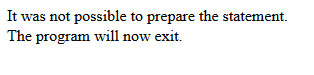
This script is used to add a new topic to a registered (and logged in) student’s webfolio. It makes a connection to the database, then pulls the required data from the user. It should be noted that the actual webfolio tool will use a form to capture input from the user. However, because a form for this purpose does not exist as yet an alternative method was used for testing. $\_GET was used in place of $\_POST, and the PHP script was written with this in mind. In terms of sending data to the server the desired values were entered into the URL as in the example below:

[http://adrianl8113.ccacolchester.com/webprog/add\_topic.php?title=Designing A Webfolio&content=Here is a brief description of something to test prepared statements](http://adrianl8113.ccacolchester.com/webprog/add_topic.php?title=Designing%20A%20Webfolio&content=Here%20is%20a%20brief%20description%20of%20something%20to%20test%20prepared%20statements)

In order to defend against malicious code (please refer to the Security section of the report for more detail) a combination of defensive techniques were employed so safeguard the database and any users connected to it. The solution decided upon involved combining three powerful forms of defence – HTML entities, trim and prepared statements – to reduce the possibility of a successful attack to the absolute minimum.

First, the input from the user is sanitised. To do this, a combination of two functions was used within one statement. HTML entities acts to convert any potentially dangerous characters present in the input into harmless text. Next, trim cuts any spaces before or after the content. The result of this sanitation is saved as a new variable. Using prepared statements alone is a very secure defence against SQL injection, but to illustrate the effectiveness of the additional security measures employed please refer to the Security section of the report.

The next stage in the security process is to initialise a prepared statement. A query is defined, using placeholders instead locally-stored content. A new variable is created to store the statement, before being initialised and then prepared. This stage adds the query to the statement. The next step is to bind the actual values (the sanitised input along with any pre-determined data) to the query. The statement is then executed. All being well a message will advise the user their data was saved. If not, the prepared statement has a check at each stage so the user will be advised of exactly where the error occurred. An example of this is below, where the statement was not prepared. This was a result of a syntax error:



Conversely, if the script runs to completion the following message is displayed:



Below is the content now present on the database:

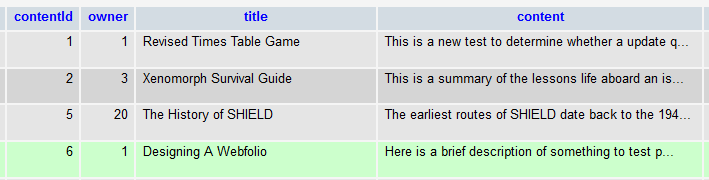


Figure 9: The uploaded content has been stored in the database

This script will be used in the Functional Specification’s Main Flow at Steps 11, 12 and 13.

Below is the code for this script:

|  |
| --- |
| <!-- Adrian Leach - Web Programming Assignment 1 - November 2014  This script lets a user upload new content to their webfolio.  As a user is free to enter their own data, the database is vulnerable to SQL injection, as well  as malicious code that attempts to run on other users' machines. Therefore considerable effort was  made to ensure the script was secure from attack. Output is a message advising the user their entry  was successful - or an error at the stage of the process where the problem occurred.  -->  <html>  <body>  <?php //insert query to add a new content topic to a user's webfolio using a prepared statement    //include file containing log-in credentials  require\_once 'login.php';  //store the connection in a variable. connection requires 4 arguments - server(host), user, password and database name  $db = mysqli\_connect($server, $user, $pass, $database)  //if not successful, display the actual error, and then terminate the program so as not to reveal further (irrelevent) error messages  or trigger\_error("The connection was not successful. The returned error is: <BR>". mysqli\_connect\_error(), E\_USER\_ERROR);  //the actual application would use $\_SESSION to retain the logged in user's ID  //$owner = $\_SESSION['studentId'];  $owner = 1;  $title = $\_GET['title'];  $content = $\_GET['content'];  $image = "images/linux\_mint.png";    //sanitise input by converting HTML characters to text and removing spaces before and after input  //without spaces an SQL comment cannot be made, which protects against malicious code that is in  //comments from avoiding detection  $safeTitle = htmlentities(trim($title));  $safeContent = htmlentities(trim($content));  //define the query to be used. dateAdded is given a value, all other fields are given placeholders  //date needs to be set here - now() will be mistaken by PHP as a function if assigned to a variable  $query = "INSERT INTO Content (owner, title, content, image, dateAdded)  VALUES (?, ?, ?, ?, NOW())";    //create a variable to hold the statement  $stmt = mysqli\_stmt\_init($db);  //prepare the statement, return error if this is not successful  if(!mysqli\_stmt\_prepare($stmt, $query))  {  echo "It was not possible to prepare the statement.<BR/>";  exit("The program will now exit.");  }    /\*bind parameters required by the query to the statement variable  requires 3 parameters - name of statement variable, datatype of each value (in order), values themselves  datatype of variable - string = s, integer = i, blob = b, double = d  ref - http://markonphp.com/simple-insert-mysqli/ \*/  $bind = mysqli\_stmt\_bind\_param($stmt, "isss", $owner, $safeTitle, $safeContent, $image);  //error check to ensure bind was successful  if($bind === false)  {  trigger\_error("Error encountered during binding of parameters.", E\_USER\_ERROR);  }  //execute statement and check for error  $execute = mysqli\_stmt\_execute($stmt);  if($execute === false)  {  trigger\_error("Error during execution of statement." . mysqli\_stmt\_error($stmt), E\_USER\_ERROR);  }  echo "Thank you. Your topic has been created, and you will now be returned to your webfolio.";  //close the statement  mysqli\_stmt\_close($stmt);  ?>  </body>  </html> |

## Register a New Student

This script will be used when it is necessary to insert details of a previously-unregistered user into the database. This would happen if they clicked the Register Button then submitted their details, or if the user logged in via a Facebook account that had no match in the Student table.

This script has a lot in common with the previous script. Again $\_GET was used in place of $\_POST, and the PHP script was written with this in mind. In terms of sending data to the server the desired values were entered into the URL as in the example below:

<http://adrianl8113.ccacolchester.com/webprog/insert_user.php?firstName=Shaun&lastName=Sheep&email=shaun@baabaa.com&dob=2000-10-19&username=wooly1&pass=shaunsheep>

As with the Add Content query, this one is reliant on the user to enter data it is extremely vulnerable to attacks via SQL injection. Therefore it was necessary to take precautions to ensure that the database itself was not compromised by SQL injection, and that no rogue code made it into the database to mitigate the risk of users receiving unwanted code with their requested data.

The solution decided upon involved combining four powerful forms of defence – HTML entities, trim, regular expressions and prepared statements – to reduce the possibility of a successful attack to the absolute minimum.

It could be argued that this level of security is unnecessary, as in most cases using prepared statements alone should be sufficient. However, although it’s true the complexity of the script would have a negative impact on performance it is a matter of milliseconds in difference. As such it was decided that stronger security was worth the slight performance penalty.

The code for this query is below:

|  |
| --- |
| <?php session\_start();  //session started to enable studentId to be captured for use in other scripts.  ?>  <!-- Adrian Leach - Web Programming Assignment 1 - November 2014  This script inserts a new user into the Student table.  This script contains a very robust defence against SQL injection and malicious code input.  Output is provided to the user to advise their data was stored.  Error checking has been incorporated into the script to provide feedback in the event of a problem.  -->  <html>  <body>  <?php  //include files containing log-in credentials and constants  require\_once 'login.php';  //require\_once 'constants.php'  //store the connection in a variable. connection requires 4 arguments - server(host), user, password and database name  $db = mysqli\_connect($server, $user, $pass, $database)  //if not successful, display the actual error, and then terminate the program so as not to reveal further (irrelevent) error messages  or trigger\_error("The connection was not successful. The returned error is: <BR>". mysqli\_connect\_error(), E\_USER\_ERROR);  //store raw user input in each field  $firstName\_get = $\_GET['firstName'];  $lastName\_get = $\_GET['lastName'];  $email\_get = $\_GET['email'];  $dob\_get = $\_GET['dob'];  $username\_get = $\_GET['username'];  $pass\_get = $\_GET['pass'];  //pull image data from $\_FILES superglobal  //$image = $\_FILES['userImage']['name'];  //set the path to the permanent storage location using the constant defined in constants.php  //$path = UPLOADPATH.$userImage;  //use temp name of image to identify and locate it, then move it to the images folder on the server  //$\_FILES['userImage']['tmp\_name'],$path;  $userImage = $\_GET['userImage'];  //begin comprehensive sanitation. First, remove all dangerous characters from string, plus spaces before and after input  $first = mysqli\_real\_escape\_string($db, trim($firstName\_get));  $last = mysqli\_real\_escape\_string($db, trim($lastName\_get));  $email = mysqli\_real\_escape\_string($db, trim($email\_get));  $dob = mysqli\_real\_escape\_string($db, trim($dob\_get));  $username = mysqli\_real\_escape\_string($db, trim($username\_get));  $pass = mysqli\_real\_escape\_string($db, trim($pass\_get));  //Next, run current input (after being escaped and trimmed) through a regular expression  //create an array to test similar strings  $test\_array\_names = array($first,$last);  //define regex for each test, and error messages to display if a match is found  //any alphabetical character, space and apostrophe  $regex = "/[^a-zA-Z '-]+/";  //any digit and a hyphen  $regex\_dob = "/[^\d-]+/";  //any alphabetical character, any number, underscore...also allows #, WHY?  $regex\_user = "/[^a-zA-Z0-9\_]+/";  $errorMsg = "Your name input was not valid. Please do not use symbols or numbers.<BR/>";  $errorEmail = "Your email address was not accepted. Please try again.<BR/>";  $errorDob = "Your date of birth was not accepted. The format required is YYYY-MM-DD.<BR/>";  $errorUser = "Your username or password was not accepted. Please use only letters, numbers and underscores.<BR/>";  //begin testing - first, user input for first name and last name.  foreach ($test\_array\_names as $check)  {  if(preg\_match($regex, $check))  {  echo $errorMsg;  exit("You will now be returned to the previous page.");  }  }  //filter\_var function used in place of regex due to the complication of checking for a valid email address  //this function actually uses a regex defined specfically for eliminating false email addresses whilst reducing  //false-positives (addresses that are genuine but flagged as invalid) as possible - https://fightingforalostcause.net/content/misc/2006/compare-email-regex.php  if (!filter\_var($email, FILTER\_VALIDATE\_EMAIL))  {  echo $errorEmail;  exit("You will now be returned to the previous page.");  }  //test dob (need to find out how to test for format - currently DD-MM-YYYY will work)  if(preg\_match($regex\_dob, $dob))  {  echo $errorDob;  exit("You will now be returned to the previous page.");  }  //test username and password  $test\_array\_user = array($username,$pass);    foreach ($test\_array\_user as $check\_user)  {  if(preg\_match($regex\_user, $check\_user))  {  echo $errorUser;  exit("You will now be returned to the previous page.");  }  }  //check to see if an image was uploaded, if so, check that there is actually a file present and that the size is larger than 0  if(isset($userImage))  {  if(!is\_file($userImage) || filesize($userImage) <= 0)  {  echo "Please use a valid image for the file upload.";  exit("You will now be returned to the previous page.");  }  }  //only when all the above is verified will the query be concatenated  $query = "INSERT INTO Student (firstName, lastName, email, dob, username, pass, userImage)  VALUES (?, ?, ?, ?, ?, ?, ?)";  //create a variable to hold the statement  $stmt = mysqli\_stmt\_init($db);  //prepare the statement, return error if this is not successful  if(!mysqli\_stmt\_prepare($stmt, $query))  {  echo "It was not possible to prepare the statement.<BR/>";  exit("The program will now exit.");  }    /\*bind parameters required by the query to the statement variable  requires 3 parameters - name of statement variable, datatype of each value (in order), values themselves  datatype of variable - string = s, integer = i, blob = b, double = d  ref - http://markonphp.com/simple-insert-mysqli/ \*/  $bind = mysqli\_stmt\_bind\_param($stmt, "sssssss", $first, $last, $email, $dob, $username, $pass, $userImage);  //error check to ensure bind was successful  if($bind === false)  {  trigger\_error("Error encountered during binding of parameters.", E\_USER\_ERROR);  }  //execute statement and check for error  $execute = mysqli\_stmt\_execute($stmt);  if($execute === false)  {  trigger\_error("Error during execution of statement." . mysqli\_stmt\_error($stmt), E\_USER\_ERROR);  }  echo "Thank you. Your registration is now complete.<BR/> You will now be taken to your new webfolio.";  //close the statement  mysqli\_stmt\_close($stmt);  //the following section is required to provide the $\_SESSION array with the new user's ID  $get\_studentId = "SELECT studentId FROM Student WHERE username='$username' && pass='$pass'";  $sid\_result = mysqli\_query($db, $get\_studentId)  or trigger\_error("It was not possible to retrieve the unique ID number. <BR/>  For debugging purposes, the SQL used in this query was:  <BR/> '$get\_studentId' <BR/>" . mysqli\_error($db) , E\_USER\_ERROR);    while($row = mysqli\_fetch\_array($sid\_result))  {  $studentId = $row['studentId'];  }  //$\_SESSION['studentId'] = $studentId;  //below used for testing purposes to ensure correct value is passed into $studentId  echo "<br/>Your new user ID is: " . $studentId . " .";  ?>  </body>  </html> |

Finally, output is given to the user in the form of a short message. An example of this is below:

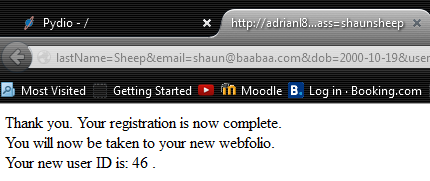


Figure 10: Message shown to the user after a successful upload

The comment about user ID will not feature in the final application, however proved very useful for testing purposes.

To be certain the data was stored correctly a quick check on the database was conducted:



The data was indeed stored successfully on the database, and it is possible to verify that the student ID identified by the script is also correct.

It can be seen that where possible each step of the script has had some form of error checking attached. This should help to ensure if an issue arises it is obvious as to the cause. One part of the script that does not have error checking is the query itself. An argument could be made for a defence utilising mysqli\_real\_escape\_string(), as this would allow the more comprehensive error message to be displayed in relation to the query. However, whilst that information was very useful during the initial tests of the query, once the statement had been successfully stored it served little purpose as the end user should never see it. Instead, the error checking concentrates on the regular expressions and prepared statement, so that if an issue does occur the problem is relatively easy to pinpoint. With this in mind it was decided there was no reason not to take advantage of the greater security offered by prepared statements.

This script would be used in the Functional Specification’s Alternate Flow 4a at Step 2.

### Testing

This is a complex script that has a fairly large number of possible outcomes due to the number of fields that will be present in the form. Below is a table that illustrates how testing was conducted to ensure the script was working as intended:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | Input | Expected Output | Actual Output | Notes |
| Valid details are approved and a student ID is issued | All fields completed with valid details | Thank you message, student ID issued | Thank you. Your registration is now complete. You will now be taken to your new webfolio. Your new user ID is: 51 . | Test Passed |
| First name contains numbers | firstName=Sh4un | Name error message displayed, script stopped | Your name input was not valid. Please do not use symbols or numbers. You will now be returned to the previous page. | Test Passed |
| First name contains symbols | firstName=Sh@un | Name error message displayed, script stopped | Your name input was not valid. Please do not use symbols or numbers. You will now be returned to the previous page. | Test Passed |
| Last name contains numbers | lastName=Sh3ep | Name error message displayed, script stopped | Your name input was not valid. Please do not use symbols or numbers. You will now be returned to the previous page | Test Passed |
| Last name contains symbols | lastName=She£p | Name error message displayed, script stopped | Your name input was not valid. Please do not use symbols or numbers. You will now be returned to the previous page | Test Passed |
| Email field contains an invalid address | email=shaun@baabaa | Email address error displayed, script stopped | Your email address was not accepted. Please try again. You will now be returned to the previous page. | Test Passed |
| Date of Birth contains an illegal character | dob=20A0-10-19 | DOB error message displayed, script stopped | Your date of birth was not accepted. The format required is YYYY-MM-DD. You will now be returned to the previous page. | Test Passed |
| Username contains illegal character | username=w!oly1 | Username error message displayed, script stopped | Your username or password was not accepted. Please use only letters, numbers and underscores. You will now be returned to the previous page. | Test Passed |
| Password contains illegal characters | pass=sh@unsheep | Username (same rules as password) error displayed, script stopped | Your username or password was not accepted. Please use only letters, numbers and underscores.You will now be returned to the previous page. | Test Passed |

Figure 11: Test Table used to ensure the Insert User script was working correctly

After the testing above it was concluded the script is working as intended. Below is the user created during the first step of the testing:



## Delete A Topic

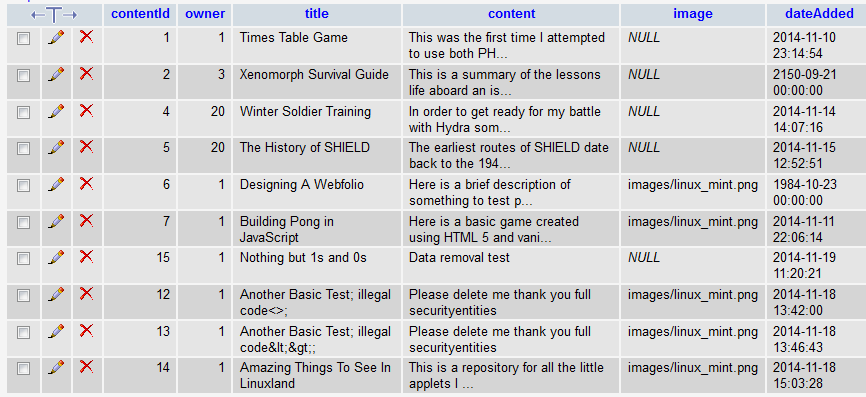
This script will be used to remove a selected topic within a user’s webfolio. The most difficult part of this script was deciding how the script would be able to determine exactly which topic was to be deleted in a situation where a user has multiple topics. The method that was decided on involves providing the choice to the user via the front-end, as opposed to writing a complex script server-side. The actual implementation will involve a radio box, and a user will be able to select which topic they would like to delete. This information will be passed to the script via $\_POST, although for testing again $\_GET was used. This will involve matching a topic title with the contentId field within the database, however it is thought this should be possible.

Although there is only a minimal security threat related to this script, as although data is sent from the user to the server it cannot be modified and will simply be a tiny integer that corresponds to the contentId of the topic. However, it was decided that there was no reason not to use a prepared statement, since it is just as performant (and, in some cases, faster) and not a complicated addition to the code, following the same basic format that it has taken in the other scripts.

Below is an example of the URL used to remove a topic. In this URL the value of the contentId is set to 15.

<http://adrianl8113.ccacolchester.com/webprog/delete_topic.php?content=15>

Here is an image of the Content table before the script was launched:



Following the script, the user is presented with the following message:

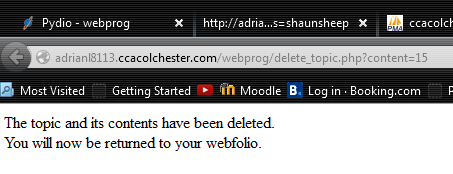
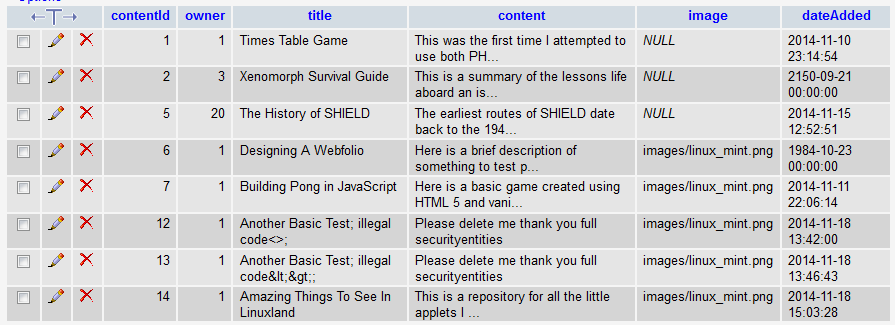


Figure 12: Message displayed to users to confirm the data is deleted

To ensure that the message is accurate, thought, the database was examined:



As can be seen, the record with contentID = 15 has indeed been deleted. As such we can conclude that the script is working as it should.

This code will be used in the Alternate Flow 9b at Step 3.

Below is the code for this query:

|  |
| --- |
| <?php session\_start();  ?>  <!-- Adrian Leach - Web Programming Assignment 1 - November 2014  This script deletes a topic from the Content table.  A prepared statement is used to protect against SQL injection.  Output is provided to the user to advise the content was removed.  Error checking in the script is limited to the connection and the prepared statement as there is not  anything else that should produce an issue.  -->  <html>  <body>  <?php //join query on database with error checking, output results in JSON format  //include file containing log-in credentials  require\_once 'login.php';  //store the connection in a variable. connection requires 4 arguments - server(host), user, password and database name  $db = mysqli\_connect($server, $user, $pass, $database)  //if not successful, display the actual error, and then terminate the program so as not to reveal further (irrelevant) error messages  or trigger\_error("The connection was not successful. The returned error is: <BR>". mysqli\_connect\_error(), E\_USER\_ERROR);  //the final application will use $\_SESSION to retrieve the stored contentId.  //a radio button will be included on the page to enable users to select which topic they want to remove  //for now, a $\_GET request is used to mimic this functionality  //$contentId = $\_SESSION['contentId'];  $contentId = $\_GET['content'];  $query = "DELETE FROM Content WHERE contentId = ?";  $stmt = mysqli\_stmt\_init($db);  //error check to test preparation was successful  if(!mysqli\_stmt\_prepare($stmt, $query))  {  echo "It was not possible to prepare the statement.<BR/>";  exit("The program will now exit.");  }  //i used to signify contentId is a numeric value  $bind = mysqli\_stmt\_bind\_param($stmt, "i", $contentId);  //error check to ensure bind was successful  if($bind === false)  {  trigger\_error("Error encountered during binding of parameters.", E\_USER\_ERROR);  }  //execute and check for error  $execute = mysqli\_stmt\_execute($stmt);  if($execute === false)  {  trigger\_error("Error during execution of statement." . mysqli\_stmt\_error($stmt), E\_USER\_ERROR);  }  echo "The topic and its contents have been deleted. <BR/>  You will now be returned to your webfolio.";  //close the statement  mysqli\_stmt\_close($stmt);  ?>  </body>  </html> |

## Update a Topic

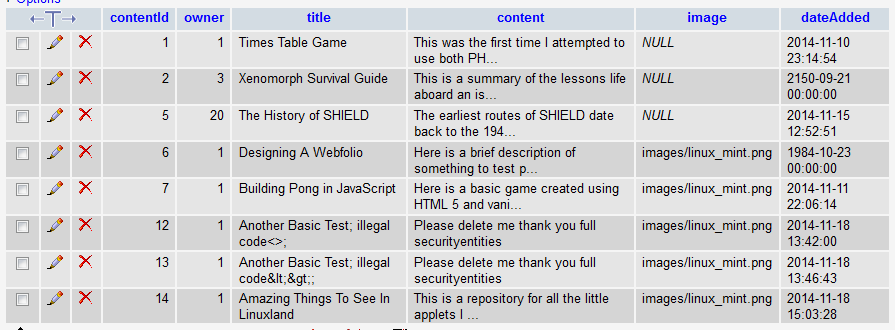
This script allows a user to update a topic within their webfolio. The finished version of this script will display to the user their current post, thereby enabling them to edit it before storing it again. Currently this script actually replaces the existing content. It is possible to concatenate the records by taking the old data and adding it to the new input, however this is really not the solution required. As such the script has been prepared for part 2 of the assignment but lacks full functionality. Similarly, and as with other scripts, it is not possible to upload images at this point – but provisions for this service have been made so that it should be relatively straightforward to implement for the second part of this assignment.

As with other scripts containing user input prepared statements have been used to provide defence against SQL injection. Regular expressions have not been used in this script as they would simply be too difficult to customise for a content field that may well contain a wide variety of symbols.

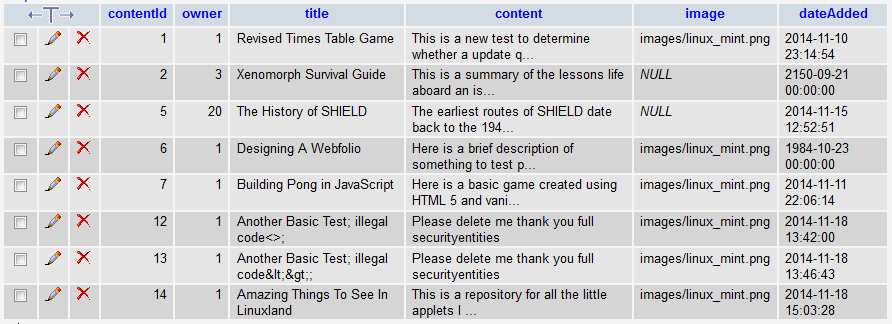
Below is an example of a URL used to test this query:

adrianl8113.ccacolchester.com/webprog/update\_content.php?contentId=1&title=Revised Times Table Game&content=This is a new test to determine whether a update query is working correctly

To ensure the query is working as hoped screenshots were taken before and after running the query above. Below is the screenshot taken before the script ran:



Here is the updated screenshot:



Interestingly, the script would not function correctly until mysqli\_escape\_string() was changed to htmlentities(). Further research will be performed as to why.

The output given to the user following a successful update is below:

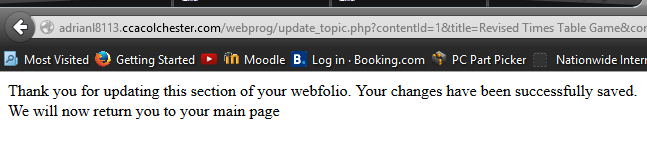


Figure 13: Message displayed after a successful update

This script will be used in the Functional Specification’s Alternate Flow 9a.

The code for this script is below:

|  |
| --- |
| <?php session\_start();  //session started to enable studentId to be captured for use in other scripts.  ?>  <!-- Adrian Leach - Web Programming Assignment 1 - November 2014  This script allows a user to edit the content of a topic within the Content table. This script is a basic  version of the finalised one, as eventually it will post data to the user, allowing them to see their current  topic information. They will then be able to edit and save the record. Currently, this script simply overwrites  the stored information as nothing has been presented to the user.  As the script contains user input robust defence against SQL injection and malicious code input has been included.  Output is provided to the user to advise their updated data was stored.  Error checking has been incorporated into the script to provide feedback in the event of a problem.  -->  <html>  <body>  <?php  //include files containing log-in credentials and constants  require\_once 'login.php';  //require\_once 'constants.php'  //store the connection in a variable. connection requires 4 arguments - server(host), user, password and database name  $db = mysqli\_connect($server, $user, $pass, $database)  //if not successful, display the actual error, and then terminate the program so as not to reveal further (irrelevent) error messages  or trigger\_error("The connection was not successful. The returned error is: <BR>". mysqli\_connect\_error(), E\_USER\_ERROR);  //store raw user input in each field  //application will use a radio button interface, with the titles of user's topics displayed  //each choice will be named according to the contentId value stored in the database.  $contentId = $\_GET['contentId'];  $title = $\_GET['title'];  $content\_get = $\_GET['content'];  /\*pull image data from $\_FILES superglobal  //$image = $\_FILES['userImage']['name'];  //set the path to the permanent storage location using the constant defined in constants.php  //$path = UPLOADPATH.$userImage;  //use temp name of image to identify and locate it, then move it to the images folder on the server  //$\_FILES['userImage']['tmp\_name'],$path; \*/  $image\_get = "images/linux\_mint.png";  //begin sanitation. First, remove all dangerous characters from string, plus spaces before and after input  $title = mysqli\_real\_escape\_string($db, trim($title));  //mysqli\_real\_escape\_string removed the data from the content field  $content = htmlentities(trim($content\_get));  $image = mysqli\_real\_escape\_string($db, trim($image\_get));    /\*check to see if an image was uploaded, if so, check that there is actually a file present and that the size is larger than 0  if(isset($userImage))  {  if(!is\_file($userImage) || filesize($userImage) <= 0)  {  echo "Please use a valid image for the file upload.";  exit("You will now be returned to the previous page.");  }  } \*/  //query to be used within prepared statement  $query = "UPDATE Content SET title=?, content=?, image=?  WHERE contentId= ?";  //create a variable to hold the statement  $stmt = mysqli\_stmt\_init($db);  //prepare the statement, return error if this is not successful  if(!mysqli\_stmt\_prepare($stmt, $query))  {  echo "It was not possible to prepare the statement.<BR/>";  exit("The program will now exit.");  }    /\*bind parameters required by the query to the statement variable  requires 3 parameters - name of statement variable, datatype of each value (in order), values themselves  datatype of variable - string = s, integer = i, blob = b, double = d  ref - http://markonphp.com/simple-insert-mysqli/ \*/  $bind = mysqli\_stmt\_bind\_param($stmt, "sssi", $title, $content, $image, $contentId);  //error check to ensure bind was successful  if($bind === false)  {  trigger\_error("Error encountered during binding of parameters.", E\_USER\_ERROR);  }  //execute statement and check for error  $execute = mysqli\_stmt\_execute($stmt);  if($execute === false)  {  trigger\_error("Error during execution of statement." . mysqli\_stmt\_error($stmt), E\_USER\_ERROR);  }  echo "Thank you for updating this section of your webfolio. Your changes have been successfully saved.<BR/>  We will now return you to your main page";  //close the statement  mysqli\_stmt\_close($stmt);  ?>  </body>  </html> |

# Future Development

Due to time constraints it was not possible to include as much content in this assignment as I would have liked. However, as a lot of the desired functionality will be introduced in Part 2 of the assignment this section is a little difficult to write.

To start, I would have liked to have presented a more detailed plan of how the front-end and back-up would converge but decided this would simply be too time-consuming at this stage of the assignment. I also wanted to display my plan for the website in slightly more detail, but again felt this was probably best left for the second part of the assignment.

I also feel that while I can implement a variety of security measures to prevent SQL injection and other malicious code insertion I have not spent long enough using the various techniques to develop a sound understanding of their workings. For example, I am interested in exactly how using prepared statements prevents malicious code from working, apart from the mechanisms described in the security section of this report. Similarly, whilst I am growing more used to regular expressions and their uses I still have much to learn. Increasing my knowledge in these two areas would make this application even more robust.

I also would have liked to investigate the Facebook API. While not a requirement for this assignment it would have made planning for its integration easier as I would have developed a better understanding as to how to best utilise the API for my purposes.

Along those lines, I would also like to add more Facebook functionality, so that users who log in with the service are presented with options such as ‘Comment’ or ‘Share’.

Another improvement that I would like to make is to create different levels of user access. This would enable administrators to have full control over the entire webfolio (editing / removing content as appropriate) while normal, logged in users would only have access to their own webfolio. Currently users have to be logged in to create/edit/delete topics, and can only do so on their own webfolio. In the future it may be the case that a student also requires administrator priviledges, however for now I do not currently have any need for further administrators, so it was decided this improvement could be made after more critical issues had been dealt with.

I was also planning to allow users to change both their usernames and passwords. I hope to be able to include this feature within Part 2 of the assignment. Similarly, I had planned to include a password field twice on the insert\_user.php script, so that I could check that both fields contained identical input. This is another feature that will be added as soon as possible.

During my research on prepared statements I noticed that many developers much prefer the use of PDO as opposed to MySQLi. Having tried both methods, I feel PDO is more efficient and logical. Additionally, PDO offers greater compatibility. As such I intend to practice using PDO when working with database applications. However, for this assignment I elected to use the same style throughout.

# References

* Captain, F. (2013). *Six-Step Relational Database Design: A Step by Step Approach to Relational Database Design and Development*. p167
* Carpenter, T. (2013). *Microsoft SQL server 2012 administration*. Indianapolis, Ind.: Sybex. p288
* Clarke, J. (2009). *SQL injection attacks and defense*. Burlington, MA: Syngress Pub.
* Gosselin, M (2014). *Simple Insert with MySQLi & Prepared Statement* (Online). Available at: <http://markonphp.com/simple-insert-mysqli/> [Accessed 17 November 2014]
* Powell, G. (2006). *Beginning database design*. Indianapolis, IN: Wiley. p74
* Ricardo, C. (2012). *Databases illuminated*. Sudbury, MA: Jones & Bartlett Learning. p219
* Schwartz, B. et al (2012*). High Performance MySQL: Optimisation, Backups and Replication*. Sebastopol, CA: O’Reilly. p291
* Snyder, C., Myer, T. and Southwell, M. (2010). *Pro PHP Security: From Application Security Principles to the Implementation of XSS Defenses*. Berkeley, CA: Apress. p52

# Appendix 1: Logical Database Design

## Database Dictionary

### Student

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | Data Type | Format | Comments |
| studentId | Integer | 11, Auto Increment | Primary Key |
| firstName | Varchar | 20 | Not Null |
| lastName | Varchar | 30 | Not Null |
| email | Varchar | 40 | Not Null |
| dob | Date | YYYY-MM-DD | Not Null |
| username | Varchar | 20 | CI username |
| pass | Varchar | 30 | CI password |
| userImage | Varchar | 100 | Uploaded image |

### Course

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | Data Type | Format | Comments |
| courseId | Integer | 11, Auto Increment | Primary Key |
| courseName | Varchar | 30 | Not Null |
| courseTutor | Varchar | 40 | Not Null |
| faculty | Varchar | ENUM | Faculty name, Not Null |

### Register

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | Data Type | Format | Comments |
| registerId | Integer | 11, Auto Increment | Primary Key |
| studentId | Integer | 11 | Foreign Key – references Student(studentId) |
| courseId | Integer | 11 | Foreign Key – references Course(courseId) |
| grade | Integer | 3 | Student’s module grade |

### Content

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | Data Type | Format | Comments |
| contentId | Integer | 11, Auto Increment | Primary Key |
| owner | Integer | 11 | Foreign Key – references Student(studentId) |
| title | Varchar | 20 | Title of post |
| content | Varchar | 1000 | Actual content of the post, Not Null |
| image | Varchar | 100 | Image for the topic |
| dateAdded | datetime | YYYY-MM-DD | Date of post, uses now() |

### Facebook

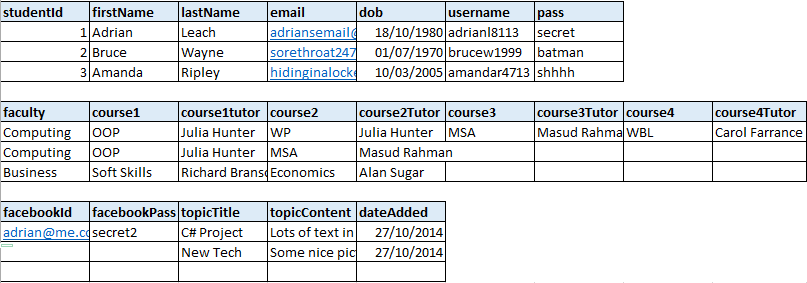
|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | Data Type | Format | Comments |
| facebookId | Varchar | 50 | Facebook username, Not Null |
| studentId | Integer | 11 | Foreign Key – references Student(studentId) |
| pass | Varchar | 30 | Password for Facebook |

## Normalisation

Normalisation is the process undertaken to evaluate and restructure databases into sensible relations. The goal is to reduce or remove data anomalies and redundancy, resulting in a more organised database. [Powell, 2006]

### Un-normalised Form

The un-normalised form (UNF) contains all the attributes within the entire database. Effectively this is a single, large table. Below is a representation of this table, however in order to fit the table into this document the attributes have been organised into three rows. In the actual table the attributes would move from left to right in normal fashion, and the table would comprise of 4 rows in total (one for the attribute names and three for the actual data).



#### Outstanding Problems

The first issue is that this table serves many purposes – it provides information on the student, the course as well as attributes relating to content for the webfolio. In addition there are columns that repeat, such as course and courseTutor. Overall this is very poor design, as the database is prone to all manner of errors.

For example, a delete error would occur should ‘Amanda’ quit the college – when her record was deleted we would also lose all data relating to the ‘Business’ faculty, and the courses she had taken as they are not featured elsewhere in the database.

An insert error could occur if we wanted to add a new course to the database as, unless we had a new student who had enrolled on it, there would be no primary key for the record and as such the entry would be permitted.

An update anomaly could occur if we needed to modify the contents of one of the repeating fields. For example, perhaps it was necessary to change ‘OOP’ to ‘Object Oriented Programming’. Every instance of ‘OOP’ would need to be amended, which might not impact the current database greatly – but would certainly do so if it had more records.

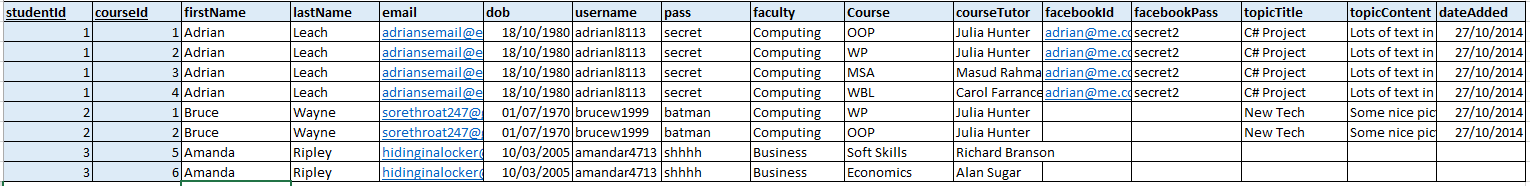
Finally the database is also difficult to query. For example, if a user wanted to check which students were enrolled on the ‘OOP’ course, they would need to check each and every course attribute.

### First Normal Form

The database must be a relational table, containing only atomic values and there can be no repeating columns. Every attribute must only have one value for each row. [Ricardo, 2012]

The database does not contain any non-atomic values (where more than one value is stored per field) and is a relational table with a Primary Key. However, it does contain columns that repeat.

To correct this the multiple course and courseTutor columns were replaced with one column for course and another for courseTutor. The results of this operation are below, Primary key columns are shaded in light blue for easier recognition.



#### Problems Solved

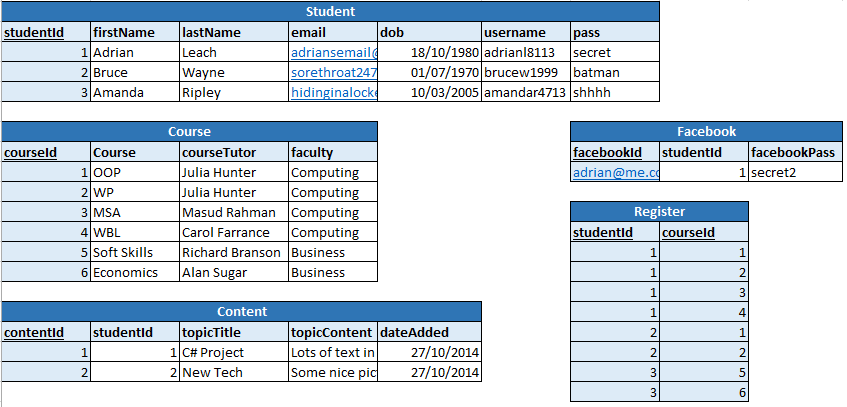
Repeating columns have been eliminated, allowing queries relating to those columns to be much more efficient.

#### Outstanding Problems

To accommodate for the change there are now multiple rows for each student. Also, a new column was added – courseId – which uniquely identifies each record in the Course table. It could be argued that the course name alone is sufficient for this, however by using a synthetic key it is possible to change the details of a course (for example, changing ‘MSA’ to ‘Modern Systems Architecture’) without causing errors. Similarly, it makes inserting and updating records less prone to error as referring to a course is done by using a single integer, rather than typing out a word. In order to make sure every record is unique a composite key is created by combining studentId and courseId. Although this does uniquely identify each record there is still a large amount of data redundancy present. In fact, update anomalies are now more of an issue than previously as there are more cases of repetitive data. Similarly, insert anomalies could also cause a serious issue as now two pieces of information (studentId and courseId) must be present in order for a new record to be entered.

### Second Normal Form

The database should be in 1NF and all should exhibit entity integrity – all non-key attributes should be dependent on the table’s primary key, and every row must be unique. [Captain, 2013]

To take the database to the second normal form (2NF) it is necessary to remove any non-key attributes. Where an attribute has dependency on only one part of the composite key, that attribute (and a copy of the key it is dependent on) are moved to a new table. The results of this process are below. 

There are now a total of 5 tables. The Student table contains personal details of each student, so every attribute is dependent only on the studentId primary key. The Course table has the attributes that were previously dependent only on the courseId element of the former composite key. As such, they have been moved along with the courseId attribute. Similarly, there are a number of attributes that are related to the content of a webfolio. These were dependent on studentId - but not courseId - so have been moved accordingly. A new synthetic key – contentId – has been created to guarantee each record can be uniquely identified.

The Facebook details are also not dependent on courseId, so have been moved to a new table with the part of the previous composite key – studentId – that they are dependent on. In this case, however, the Facebook ID is the primary key as it can uniquely identify all records.

The final table holds the data that forms the relationship between the Student table and the Course table. This is a many-to-many relationship, and the purpose of this intersection table is to break that relationship but enable a bridge between the tables.

2NF has been achieved, with every table dependent on its primary key and serving a particular purpose.

#### Problems Solved

At this stage the database is, practically speaking, workable. Update anomalies have been reduced as far as possible. If the tutor for OOP changed, for example, rather than needing to change the details for every student enrolled it would be a matter of updating one field.

Delete anomalies are also now not a concern as only the data directly relevant to each field would be removed. Deleting ‘Amanda’ now would only remove her personal information, and not any details pertaining to the courses she had enrolled on.

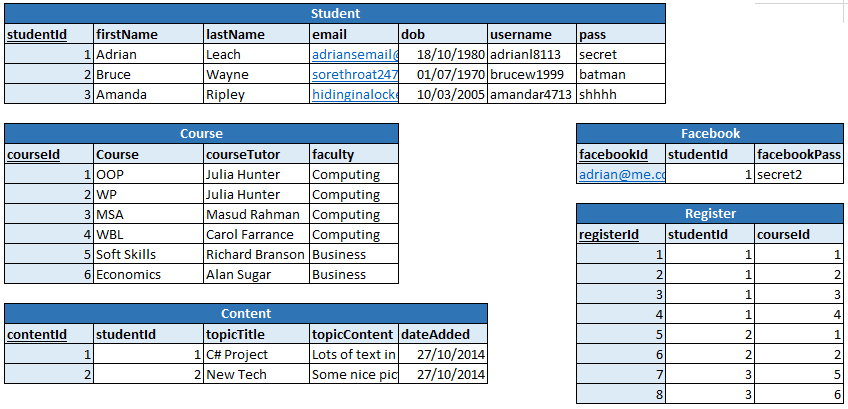
Problems arising from insert anomalies are also mitigated as it is possible to add to any of the relations without requiring extraneous information. Returning to the example given previously, it would now be possible to add a new course simply by inserting the relevant data into the course table – regardless of whether a particular student (or any student) was enrolled. Finally, the efficacy of a query should now be dependent on the script used, as opposed to the structure of the database.

#### Outstanding Problems

The Register table is currently comprised of a composite key and therefore no clear way to identify each individual record cleanly.

### Third Normal Form

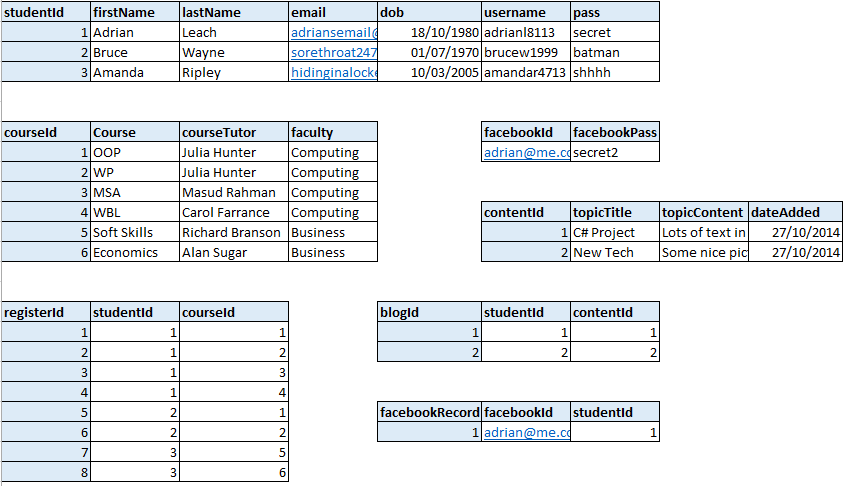
To move to third normal form (3NF) the tables must be modified in such a way that there is non-transitive dependency between any non-key attribute and the primary key in each table – every attribute must be directly dependent on the primary key. [Carpenter, 2013]. To this end a synthetic key was created for the Register table.



At this stage every attribute in the database is dependent on its table’s primary key and nothing else. Although it could be argued there is a degree of transitive dependency in the Course table, as the faculty could be derived from the tutor as well as the course, the tables have been constructed in this manner deliberately to allow tutors to have the flexibility to teach across different faculties. As such the faculty is indeed fully dependent on the course.

There are still matters that draw attention at this point during the normalisation process. Namely, there are a number of attributes in some relations that are not directly dependent on the primary key. In the Facebook table, for example, the studentId (a foreign key that references the Student table) attribute is not dependent on the PK for that relation, facebookId. Likewise, the same attribute in the Content table has the same issue.

However, despite these theoretical shortcomings, the database is now fully functional and, from a practical perspective, working correctly and according to specification. To be fully compliant with 3NF it would be necessary to introduce two more tables, as per the draft design below:



This introduces an unwarranted level of complexity to the database. It does not help to reduce any form of anomaly, or improve user experience. The two new intersection tables are not breaking many-to-many relationships, and in fact pose new challenges in terms of relationship definition. As such, it was decided in this case that rigidly adhering to the theoretical rules of normalisation was not in the best interests of this database. Instead a design that prioritises practical effectiveness and usability without sacrificing data integrity was chosen.

One final potential issue lies in the Facebook table. Users who choose to register using Facebook may not possess a studentId. Initially this was considered a positive point as it would allow Facebook users who are not students to log-in to the site and use their Facebook connection to share student webfolios etc within their social network. However, in order to allow this, the studentId attribute in the Facebook relation must allow null entries, or else a Facebook user who has no studentId could not register and log-in. Null entries are typically avoided as they can cannot be operated on (except for IS and IS NOT) and can produce undesired results when sorting data.

It is therefore tempting to remove the studentId attribute from the relation, but in that case it would not be possible to tie a user’s Facebook log-in with their studentId, which would be required should the Facebook user be a student who wants to create/edit a webfolio. To overcome this difficulty the relation will be left as it is, however users who choose to log in with Facebook details will be asked to register for the Webfolio site. At this stage they will be issued with both an ID and password, enabling full use of the site.

Normalisation has now been completed to a satisfactory level.

# Appendix 2: Additional Scripts

This section contains some additional code that may have been used in the assignment or possibly completed for testing purposes.

## Login

This small file is what is used on each script to provide the details needed to connect to the database.

|  |
| --- |
| <html>  <body>  <?php  $server = "mysql.ccacolchester.com";  $user = "adrianl8113";  $pass = "1058113";  $database = "adrianl8113";  //this section allows for quick changing to local details for testing  //$localServer = "localhost";  //$localUser = "root";  //$localPass = "";  //$localDatabase = "insert\_name\_of\_database\_here";  ?>  </body>  </html> |
|  |
|  |

## Constants

This file does not serve a real purpose as yet, however will be used in the upcoming assignment to store values that are unchanging and used across different pages. For example, the path that temporary images will be sent to for persistent storage is defined here.

|  |
| --- |
| <?php  //define application constants  define('UPLOADPATH', 'images/');  ?> |

## An Insert Query with a Prepared Statement Using PDO

Below is a brief example of how a prepared statement could be performed with PDO. It is more logical than MySQLi, but does require a little extra code initially to initialise PDO. This was a test script and has not been used in the assignment, but an interesting experiment and one that has shown that further research into PDO is definitely required.

|  |
| --- |
| <?php  //constant to tell PDO what type of database it will connect to  define('DB\_DRIVER', "mysql");  require\_once 'login.php';    $db = mysqli\_connect($server, $user, $pass, $database)  or trigger\_error("The connection was not successful. The returned error is: <BR>"  . mysqli\_connect\_error(), E\_USER\_ERROR);  //Allows PDO to throw exceptions in the event of an error  $db->setAttribute(PDO::ATTR\_ERRMODE, PDO::ERRMODE\_EXCEPTION);  $first = $\_GET['firstName'];  $last = $\_GET['lastName'];  $email = $\_GET['email'];  //using try/catch to expose errors if encountered  try  {  //prepare the statement using named placeholders  $stmt = $db->prepare("INSERT INTO Student(firstName, lastName, email) VALUES (:first, :last, :email)");  //bind parameters to placeholders. 4 arguments - name of placeholder, value, data type of value and length.  $stmt->bindParam(':first', $first, PDO::PARAM\_STR, 30);  $stmt->bindParam(':last', $last, PDO::PARAM\_STR, 40);  $stmt->bindParam(':email', $email, PDO::PARAM\_STR, 50);  $count = $stmt->execute();  $db = null;  } catch(PDOException $e) {  trigger\_error('Error occured while trying to insert into the DB:' . $e->getMessage(), E\_USER\_ERROR);  }  ?> |

# Appendix 3: MySQL Code

This section contains a printout of the code used in the MySQL database. It was captured via the export function.

|  |
| --- |
| -- phpMyAdmin SQL Dump  -- version 3.3.10.4  -- http://www.phpmyadmin.net  --  -- Host: mysql.ccacolchester.com  -- Generation Time: Nov 21, 2014 at 02:05 AM  -- Server version: 5.1.56  -- PHP Version: 5.3.27  SET SQL\_MODE="NO\_AUTO\_VALUE\_ON\_ZERO";  --  -- Database: `adrianl8113`  --  -- --------------------------------------------------------  --  -- Table structure for table `Content`  --  CREATE TABLE IF NOT EXISTS `Content` (  `contentId` int(11) NOT NULL AUTO\_INCREMENT,  `owner` int(11) DEFAULT NULL,  `title` varchar(50) NOT NULL,  `content` varchar(500) NOT NULL,  `image` varchar(100) DEFAULT NULL,  `dateAdded` datetime DEFAULT NULL,  PRIMARY KEY (`contentId`),  KEY `owner` (`owner`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO\_INCREMENT=17 ;  --  -- Dumping data for table `Content`  --  INSERT INTO `Content` (`contentId`, `owner`, `title`, `content`, `image`, `dateAdded`) VALUES  (1, 1, 'Revised Times Table Game', 'This is a new test to determine whether a update query is working correctly', 'images/linux\_mint.png', '2014-11-10 23:14:54'),  (2, 3, 'Xenomorph Survival Guide', 'This is a summary of the lessons life aboard an isolated space station have taught me.', NULL, '2150-09-21 00:00:00'),  (5, 20, 'The History of SHIELD', 'The earliest routes of SHIELD date back to the 1940s. During bitter battles the world over the allied forces understood something extraordinary was required.', NULL, '2014-11-15 12:52:51'),  (6, 1, 'Designing A Webfolio', 'Here is a brief description of something to test prepared statements', 'images/linux\_mint.png', '1984-10-23 00:00:00'),  (7, 1, 'Building Pong in JavaScript', 'Here is a basic game created using HTML 5 and vanilla javascript', 'images/linux\_mint.png', '2014-11-11 22:06:14'),  (12, 1, 'Another Basic Test; illegal code<>;', 'Please delete me thank you full securityentities', 'images/linux\_mint.png', '2014-11-18 13:42:00'),  (13, 1, 'Another Basic Test; illegal code&lt;&gt;;', 'Please delete me thank you full securityentities', 'images/linux\_mint.png', '2014-11-18 13:46:43'),  (14, 1, 'Amazing Things To See In Linuxland', 'This is a repository for all the little applets I find to enhance Mint 17.', 'images/linux\_mint.png', '2014-11-18 15:03:28');  -- --------------------------------------------------------  --  -- Table structure for table `Course`  --  CREATE TABLE IF NOT EXISTS `Course` (  `courseId` int(11) NOT NULL AUTO\_INCREMENT,  `courseName` varchar(30) NOT NULL,  `courseTutor` varchar(40) NOT NULL,  `faculty` enum('Computing','Business','Engineering','Science','Performing Arts') NOT NULL,  PRIMARY KEY (`courseId`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO\_INCREMENT=15 ;  --  -- Dumping data for table `Course`  --  INSERT INTO `Course` (`courseId`, `courseName`, `courseTutor`, `faculty`) VALUES  (1, 'Web Programming', 'Julia Hunter', 'Computing'),  (2, 'Modern System Architecture', 'Masud Rahman', 'Computing'),  (3, 'Object Oriented Programming', 'Julia Hunter', 'Computing'),  (4, 'Work Based Learning', 'Carol Farrance', 'Business'),  (7, 'Economics', 'Alan Sugar', 'Business'),  (8, 'Marketing Principles', 'Simon Cowell', 'Business'),  (9, 'Molecular Physics', 'Gordon Freeman', 'Science'),  (10, 'Xenobiology', 'Ellen Ripley', 'Science'),  (11, 'Automotive Aerodynamics', 'Gordon Murray', 'Engineering'),  (12, 'Zero-G Astroengineering', 'Isaac Clarke', 'Engineering'),  (13, 'Contemporary Dance', 'Brian Friedman', 'Performing Arts'),  (14, 'Vocal Acrobatics', 'Smooth McGroove', 'Performing Arts');  -- --------------------------------------------------------  --  -- Table structure for table `Facebook`  --  CREATE TABLE IF NOT EXISTS `Facebook` (  `facebookId` varchar(40) NOT NULL,  `pass` varchar(30) NOT NULL,  `studentId` int(10) NOT NULL,  PRIMARY KEY (`facebookId`),  KEY `facebookId` (`facebookId`),  KEY `studentId` (`studentId`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8;  --  -- Dumping data for table `Facebook`  --  -- --------------------------------------------------------  --  -- Table structure for table `Register`  --  CREATE TABLE IF NOT EXISTS `Register` (  `registerId` int(11) NOT NULL AUTO\_INCREMENT,  `studentId` int(11) NOT NULL,  `courseId` int(11) NOT NULL,  `grade` int(3) DEFAULT NULL COMMENT '(%)',  PRIMARY KEY (`registerId`),  KEY `studentId` (`studentId`),  KEY `courseId` (`courseId`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO\_INCREMENT=93 ;  --  -- Dumping data for table `Register`  --  INSERT INTO `Register` (`registerId`, `studentId`, `courseId`, `grade`) VALUES  (9, 1, 1, NULL),  (10, 1, 2, NULL),  (11, 1, 3, NULL),  (12, 2, 1, 80),  (13, 2, 3, 75),  (51, 3, 11, NULL),  (52, 3, 12, NULL),  (53, 4, 1, NULL),  (54, 4, 2, NULL),  (55, 6, 7, NULL),  (56, 6, 8, NULL),  (57, 7, 2, NULL),  (58, 7, 3, NULL),  (59, 8, 7, NULL),  (60, 8, 8, NULL),  (61, 9, 9, NULL),  (62, 9, 10, NULL),  (63, 10, 10, NULL),  (64, 11, 11, NULL),  (65, 12, 7, NULL),  (66, 13, 11, NULL),  (67, 14, 14, NULL),  (68, 15, 12, NULL),  (69, 16, 13, NULL),  (70, 17, 11, NULL),  (71, 18, 14, NULL),  (72, 19, 9, NULL),  (73, 20, 11, NULL),  (74, 21, 12, NULL),  (75, 32, 9, NULL),  (76, 31, 10, NULL),  (77, 23, 9, NULL),  (78, 29, 13, NULL),  (79, 25, 14, NULL),  (80, 42, 13, NULL),  (81, 30, 1, NULL),  (82, 22, 2, NULL),  (83, 24, 3, NULL),  (84, 26, 4, NULL),  (85, 27, 7, NULL),  (86, 33, 8, NULL),  (87, 33, 7, NULL),  (88, 32, 10, NULL),  (89, 25, 13, NULL),  (90, 22, 1, NULL),  (91, 46, 3, NULL),  (92, 51, 7, NULL);  -- --------------------------------------------------------  --  -- Table structure for table `Student`  --  CREATE TABLE IF NOT EXISTS `Student` (  `studentId` int(11) NOT NULL AUTO\_INCREMENT,  `firstName` varchar(20) NOT NULL,  `lastName` varchar(30) NOT NULL,  `email` varchar(40) DEFAULT NULL,  `username` varchar(20) DEFAULT NULL,  `pass` varchar(30) NOT NULL,  `dob` date NOT NULL,  `userImage` varchar(100) DEFAULT NULL,  PRIMARY KEY (`studentId`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO\_INCREMENT=52 ;  --  -- Dumping data for table `Student`  --  INSERT INTO `Student` (`studentId`, `firstName`, `lastName`, `email`, `username`, `pass`, `dob`, `userImage`) VALUES  (1, 'Adrian', 'Leach', 'adriansemail@email.com', 'adrianl8113', 'secret', '1980-10-18', 'images/linux\_mint.png'),  (2, 'Bruce', 'Wayne', 'alterego@gotham.com', NULL, '', '0000-00-00', NULL),  (3, 'Amanda', 'Ripley', 'hidinginalocker@sevastopol.com', NULL, '', '0000-00-00', NULL),  (4, 'Jon', 'Snow', 'coolguy@thewall.com', NULL, '', '0000-00-00', NULL),  (6, 'Roger', 'Harris', 'rogersemail@home.com', 'rogerh1234', '1234567', '1978-12-20', NULL),  (7, 'Elyssia', 'Rivers', 'name@address.com', 'demir1234', '', '0000-00-00', NULL),  (8, 'Demi', 'Austin', 'demi@address.com', 'demia1234', '', '0000-00-00', NULL),  (9, 'Rhiannon', 'Austin', 'rhitin@testing.com', 'rhiannona1234', '', '0000-00-00', NULL),  (10, 'Grumpy', 'Bear', NULL, NULL, '', '0000-00-00', NULL),  (11, 'Gordon', 'Freeman', NULL, NULL, 'password1', '1989-12-01', NULL),  (12, 'Willy', 'Wonka', NULL, NULL, 'password2', '1899-12-06', NULL),  (13, 'Engelbert', 'Humperdink', NULL, NULL, 'password4', '2000-06-27', NULL),  (14, 'George', 'Likesshouting', NULL, NULL, 'password6', '2000-08-25', NULL),  (15, 'David', 'Hayter', NULL, NULL, 'password7', '2000-08-25', NULL),  (16, 'Alan', 'Carr', NULL, NULL, 'comic1', '1970-05-25', NULL),  (17, 'Bob', 'Star', NULL, NULL, 'eagle1', '1975-05-29', NULL),  (18, 'Peter', 'Parker', NULL, NULL, 'spidey', '1985-05-29', NULL),  (19, 'Bruce', 'Banner', 'hulk@hulksmash.com', 'drbanner', 'green', '1985-05-29', NULL),  (20, 'Steve', 'Rogers', 'chargingstar@usa.com', 'thecaptain', 'bucky', '1918-07-04', NULL),  (21, 'Tali ''Zorah', 'vas Normandy', 'gettingvacinated@ssvnormandy.com', 'taliz', 'shepard', '2161-12-20', NULL),  (22, 'Shigeru', 'Miyamoto', 'stillgreat@nintendo.co.jp', 'issamemario', 'bowser', '1969-11-06', NULL),  (23, 'Christopher', 'Nolan', 'engrossed@comiccon.com', 'chrisn666', 'smvsbm', '1945-07-19', NULL),  (24, 'Snow', 'Villiers', 'hero@nora.org', 'papasnow', 'serahfarron', '2020-07-21', NULL),  (25, 'Elhaym', 'Van Houten', 'elly@solaris.org', 'ellyvh111', 'vierge', '2123-05-15', NULL),  (26, 'Squall', 'Leonhart', 'sullenbutcaring@balambgarden.edu', 'griever1', 'rinoa', '1999-01-12', NULL),  (27, 'Tifa', 'Lockhart', 'tifa@seventhheaven.com', 'tifal7777', 'smilecloud', '2002-03-16', NULL),  (29, 'Citan', 'Uzuki', 'doc@lahanvillage.com', 'hyuga', 'heimdal', '0000-00-00', NULL),  (30, 'Ken', 'Masters', 'ceo@masterscorp.com', 'kenmasters', 'shotokan', '1975-06-30', NULL),  (31, 'Cammy', 'White', 'killerbee@mi5.org', 'cammy', 'whykylie', '1985-11-11', NULL),  (32, 'Bartholomew', 'Fatima', 'bart@yggdrasil.com', 'princebart', 'andvari', '1990-04-12', NULL),  (33, 'Tux', 'Penguin', 'thetux@linux.org', 'mintytux', 'mint17', '1999-12-25', 'images/linux\_mint.png'),  (42, 'Karen', 'Pippen', 'karen@regex.com', 'jolly', 'beetle', '2000-11-19', ''),  (46, 'Shaun', 'Sheep', 'shaun@baabaa.com', 'wooly1', 'shaunsheep', '2000-10-19', NULL),  (51, 'Emily', 'Elephant', 'emily@baabaa.com', 'emily1', 'tusktusk', '2009-06-19', NULL);  --  -- Constraints for dumped tables  --  --  -- Constraints for table `Content`  --  ALTER TABLE `Content`  ADD CONSTRAINT `Content\_ibfk\_1` FOREIGN KEY (`owner`) REFERENCES `Student` (`studentId`);  --  -- Constraints for table `Facebook`  --  ALTER TABLE `Facebook`  ADD CONSTRAINT `Facebook\_ibfk\_1` FOREIGN KEY (`studentId`) REFERENCES `Student` (`studentId`);  --  -- Constraints for table `Register`  --  ALTER TABLE `Register`  ADD CONSTRAINT `Register\_ibfk\_1` FOREIGN KEY (`courseId`) REFERENCES `Course` (`courseId`),  ADD CONSTRAINT `Register\_ibfk\_2` FOREIGN KEY (`studentId`) REFERENCES `Student` (`studentId`); |