# Computing A2 Coursework

Imran Rahman

March 12, 2015

# Contents


**1 Analysis** **6**
1.1 Introduction . . . 6
1.1.1 Client Identification . . . 6
1.1.2 Define the current system . . . 7
1.1.3 Describe the problems . . . 7
1.1.4 Section appendix . . . 8
1.2 Investigation . . . 10
1.2.1 The current system . . . 10
1.2.2 The proposed system . . . 18
1.3 Objectives . . . 23
1.3.1 General Objectives . . . 23
1.3.2 Specific Objectives . . . 23
1.3.3 Core Objectives . . . 24
1.3.4 Other Objectives . . . 24
1.4 ER Diagrams and Descriptions . . . 26
1.4.1 ER Diagram . . . 26
1.4.2 Entity Descriptions . . . 27
1.5 Object Analysis . . . 27
1.5.1 Object Listing . . . 27
1.5.2 Relationship diagrams . . . 29
1.5.3 Class definitions . . . 30
1.6 Other Abstractions and Graphs . . . 31
1.7 Constraints . . . 32
1.7.1 Hardware . . . 32
1.7.2 Software . . . 32
1.7.3 Time . . . 32
1.7.4 User Knowledge . . . 32
1.7.5 Access restrictions . . . 33
1.8 Limitations . . . 33
1.8.1 Areas which will not be included in computerisation . . . 33
1.8.2 Areas considered for future computerisation . . . 33
1.9 Solutions . . . 34
1.9.1 Alternative solutions . . . 34

1.9.2 Justification of chosen solution . . . . . 34

**2 Design** **36**
2.1 Overall System Design . . . . . 36
2.1.1 Short description of the main parts of the system . . . . . 36
2.1.2 System flowcharts showing an overview of the complete system . . . . . 38
2.2 User Interface Designs . . . . . 46
2.3 Hardware Specification . . . . . 56
2.4 Program Structure . . . . . 56
2.4.1 Top-down design structure charts . . . . . 56
2.4.2 Algorithms in pseudo-code for each data transformation process . . . . . 58
2.4.3 Object Diagrams . . . . . 64
2.4.4 Class Definitions . . . . . 65
2.5 Prototyping . . . . . 68
2.6 Definition of Data Requirements . . . . . 72
2.6.1 Identification of all data input items . . . . . 72
2.6.2 Identification of all data output items . . . . . 73
2.6.3 Explanation of how data output items are generated . . . 74
2.6.4 Data Dictionary . . . . . 74
2.6.5 Identification of appropriate storage media . . . . . 76
2.7 Database Design . . . . . 77
2.7.1 Normalisation . . . . . 77
2.7.2 SQL Queries . . . . . 83
2.8 Security and Integrity of the System and Data . . . . . 85
2.8.1 Security and Integrity of Data . . . . . 85
2.8.2 System Security . . . . . 85
2.9 Validation . . . . . 85
2.10 Testing . . . . . 89
2.10.1 Outline Plan . . . . . 89
2.10.2 Detailed Plan . . . . . 89

**3 Testing** **103**
3.1 Test Plan . . . . . 104
3.1.1 Original Outline Plan . . . . . 104
3.1.2 Changes to Outline Plan . . . . . 105
3.1.3 Original Detailed Plan . . . . . 105
3.1.4 Changes to Detailed Plan . . . . . 118
3.2 Test Data . . . . . 144
3.2.1 Original Test Data . . . . . 144
3.2.2 Changes to Test Data . . . . . 145
3.3 Annotated Samples . . . . . 145
3.3.1 Actual Results . . . . . 145
3.3.2 Evidence . . . . . 146
3.4 Evaluation . . . . . 188

3.4.1 Approach to Testing . . . 188
3.4.2 Problems Encountered . . . 188
3.4.3 Strengths of Testing . . . 188
3.4.4 Weaknesses of Testing . . . 189
3.4.5 Reliability of Application . . . 189
3.4.6 Robustness of Application . . . 189

**4 System Maintenance 190**
4.1 Environment . . . 190
4.1.1 Software . . . 190
4.1.2 Usage Explanation . . . 190
4.1.3 Features Used . . . 191
4.2 System Overview . . . 191
4.2.1 Main window . . . 191
4.2.2 Adding Entries . . . 192
4.2.3 Updating Entries . . . 192
4.2.4 Deleting Entries . . . 192
4.2.5 Calculations . . . 193
4.2.6 Detailed Searches . . . 193
4.3 Code Structure . . . 193
4.3.1 Function - Adding an Item . . . 193
4.3.2 Function - Removing an Item . . . 194
4.3.3 Function - Generating an update window . . . 195
4.4 Variable Listing . . . 197
4.5 System Evidence . . . 200
4.5.1 User Interface . . . 200
4.5.2 ER Diagram . . . 222
4.5.3 Database Table Views . . . 223
4.5.4 Database SQL . . . 226
4.5.5 SQL Queries . . . 228
4.6 Testing . . . 237
4.6.1 Summary of Results . . . 237
4.6.2 Known Issues . . . 238
4.7 Code Explanations . . . 238
4.7.1 Difficult Sections . . . 238
4.7.2 Self-created Algorithms . . . 240
4.8 Settings . . . 241
4.9 Acknowledgements . . . 241
4.10 Code Listing . . . 243
4.10.1 Module 1 - Login Window and Database Creation . . . 244
4.10.2 Module 2 - Main Window and functionality for majority of system . . . 252
4.10.3 Module 3 - Initialising Main Menu Layout . . . 279
4.10.4 Module 4 - Initialising Menu Bar . . . 281
4.10.5 Module 5 - Initialising a Table Widget . . . 282
4.10.6 Module 6 - Initialising Add Customer Entry Window . . . 282

4.10.7 Module 7 - Initialising Update Customer Entry Window . 285
4.10.8 Module 8 - Initialising View Menu Layout . . . . . 287
4.10.9 Module 9 - Initialising Add Entry Window for non-customer entries . . . . . 288
4.10.10 Module 10 - Initialising Calendar Widget . . . . . 298
4.10.11 Module 11 - Initialising a Confirmation Dialog . . . . . 300
4.10.12 Module 12 - Initialising a dialog for viewing the Items, and creating calculations for the payments. . . . . 303
4.10.13 Module 13 - Initialising a dialog for viewing Royalties and Invoices . . . . . 308
4.10.14 Module 14 - Initialising a Search Window . . . . . 310
4.10.15 Module 15 - Creating a layout for displaying Search Results in the Main Window . . . . . 314
4.10.16 Module 16 - Creating a dialog for the User to choose to change Username or Password . . . . . 315
4.10.17 Module 17 - Initialising a dialog for changing the Username 316
4.10.18 Module 18 - Initialising a dialog for changing the Password 318

**5 User Manual 321**
5.1 Introduction . . . . . 321
5.2 Installation . . . . . 321
5.2.1 Prerequisite Installation . . . . . 321
5.2.2 System Installation . . . . . 322
5.2.3 Running the System . . . . . 331
5.3 Tutorial . . . . . 334
5.3.1 Introduction . . . . . 334
5.3.2 Assumptions . . . . . 334
5.3.3 Tutorial Questions . . . . . 335
5.3.4 Saving . . . . . 335
5.3.5 Limitations . . . . . 335
5.4 Error Recovery . . . . . 335
5.4.1 Error 1 . . . . . 335
5.4.2 Error 2 . . . . . 335
5.5 System Recovery . . . . . 335
5.5.1 Backing-up Data . . . . . 335
5.5.2 Restoring Data . . . . . 335

**6 Evaluation 336**
6.1 Customer Requirements . . . . . 337
6.1.1 Objective Evaluation . . . . . 337
6.2 Effectiveness . . . . . 337
6.2.1 Objective Evaluation . . . . . 337
6.3 Learnability . . . . . 337
6.4 Usability . . . . . 337
6.5 Maintainability . . . . . 337
6.6 Suggestions for Improvement . . . . . 337

6.7 End User Evidence . . . . . 337
6.7.1 Questionnaires . . . . . 337
6.7.2 Graphs . . . . . 337
6.7.3 Written Statements . . . . . 337

# Chapter 1

# Analysis

## 1.1 Introduction

### 1.1.1 Client Identification

My client, Shahida Rahman, is an Author, and the Director and Secretary of Perfect Publishers Ltd, which has been a self publishing company since 2005. This means that the author pays Perfect Publishers to publish their book. She published her first book through Perfect Publishers Ltd, and this was when the company was born. She is 42 years old and is a mother of 4 children. Shahida uses computers to deal with online enquiries and to publish books from all over the world. Furthermore, she also produces the royalty statements for each book twice yearly. Aside from this, she has little experience with computers. Shahida generally uses a computer for research, social networking and reading the news. Every book is outsourced to an Editor and a Cover Designer. When the book is fully edited and formatted to the right specifications, they return the ready to print files to Shahida, who sends the books off to print. They track the books and their details manually using a database on an Excel Spreadsheet. Currently, it is difficult to keep all the data up to date and it is rather disorganised. Shahida would like to be able to look up a book/number of books in the system by using the details, such as the Author/Title/Date etc. She would also like the new system to link this database with information about the royalties of each book, and when they are needed to be paid every six months. The system could send an email to her, updating her about these.

### 1.1.2 Define the current system

The system that is currently being used consists of Shahida entering the book and its details into the spreadsheet. These details are taken from the enquiries that she receives via email, and include; first name, last name, email, and vague details of the book. Then, the more accurate details such as; book title, size, number of pages, hardback/paperback, mat or gloss, crème or white paper, font and font size are discussed between Shahida and the customer. She also records their details in a separate spreadsheet, which includes their email, phone number, and address, upon the customer's consent. The spreadsheets are used to track current and previous customers and their books, as the details about the book and themselves are recorded in these. Subsequently, Shahida informs the customer of the price details, waits for full payment and then sends the customer an invoice. She then contacts her editor and her illustrator to start work on the book. Shahida refers to her company's website, where the calculated prices are ready for books, in order to correctly price the book, in accordance to the book's details. An ISBN number is assigned to the book, which is bought in bulks by Shahida from the ISBN Office. Once the book is finished, the book is sent off to print, and the author receives 25 copies.

### 1.1.3 Describe the problems

There are numerous problems with the current system. First of all, the usage of the spreadsheet makes it harder to find a customer and their details, and their book's details. This is because the spreadsheet is much disorganised. Furthermore, it is harder to keep track of the details of each book, meaning it is difficult to update the details of the book when necessary. Because there are a large number of books in the system, it is harder to find each book and then seperately find the customer that the book was written by, as they are in seperate spreadsheets, meaning that Shahida must manually search for them in both spreadsheets. Also, if the same author makes an enquiry about another book, their details must be entered into the spreadsheet again, because it is difficult to find where the customers details currently are due to there being many customers in the spreadsheet, having incorrectly entered their data from a given enquiry, or they might have been removed after having been there for a long time This could cause inconsistencies in the data, because for instance, the customer may move house, meaning their address would need changing, and it would be difficult to find and update all entries where their address is recorded.

### 1.1.4 Section appendix

Interview Questions

1. What current system is in place?
   - Excel spreadsheets
   - Holds details of the books and authors/customers
2. What are the problems with this system?
   Hard to keep track of everything
   Data is duplicated for existing customers
   Very disorganised
3. What data do you record?
   Details about the authors - Name, email, Phone number, Address
   Book details - Title, author, Pages, Hardcover Soft Back, matte gloss, colour, size, ISBN
4. How much data is duplicated for existing customers who send another enquiry?
   Every time an enquiry is sent, their details are added to the database
5. What should the new system accomplish?
   To create an organised database
   To avoid duplication
   To be able to calculate royalties
6. What will stay the same?
   Details that are stored
   Storing data electronically
7. How long will the data remain in the system?
   As long as necessary due to details required for paying royalties
8. Are hard copies of the data required?
   No, everything is conducted electronically
9. What computing resources do you have available to you?
   Laptop - windows 7
   - microsoft office
10. Is security an issue?
    Very secure - Data isn't shared with other 3rd parties
    - Kept confidential

Figure 1.1: Interview Questions: Page 1

11. Who will be using the data?

Just Shahida, and the customer it belongs to

12. Do you have a particular system in mind?

Anything that is efficient
avoids duplication of data
easy to keep track of data

I confirm that I answered these questions to help Imran Rahman investigate my current system to help with the design of his new system.

Signed: IRahman

Figure 1.2: Interview Questions: Page 2

## 1.2 Investigation

### 1.2.1 The current system

#### Data sources and destinations

In the current system there are three key data sources that are used. These are Shahida herself, the customer and the spreadsheets. The Customer sends the enquiry which is sent via email. After the details about the book have been discussed, they are stored in one spreadsheet, and the details of the customer are stored in a seperate spreadsheet, linked to the details of the book. These details are agreed seperately from the enquiry, between Shahida and the customer. Details such as the book size, page number, hardback/softback and paper type are used to calculate the cost for the customer, which is used to create an invoice which is sent to the customer. This is the first output of the system. A copy of every invoice is stored on Shahida's computer in a special folder just for invoices. Once Shahida receives full payment, the work is conducted and completed. If the customer wishes to publish another book, they send another enquiry, and their personal data is duplicated because of the details of the new book which are added. Every six months after the book has been published, the royalties must be paid to the author. The royalties are the profit that the author makes from sales of her book from bookshops. A royalty statement is created and stored in a special folder just for royalties.

| **Source** | **Data** | **Example Data** | **Destination** |
|---|---|---|---|
| Customer Enquiry | Forename | Peter | Shahida |
| Customer Enquiry | Surname | Parker | Shahida |
| Customer Enquiry | Email | mail@example.com | Shahida |
| Customer | Address | 1 Example Road | Shahida |
| Customer | Postcode | AB1 2CD | Shahida |
| Customer | Phone Number | 07123456789 | Shahida |
| Customer | Book Title | The Hobbit | Shahida |
| Customer | Size | Large | Shahida |
| Customer | Number of Pages | 395 | Shahida |
| Customer | Hardback/Paperback | Paperback | Shahida |
| Customer | Mat/Gloss | Gloss | Book Database |
| Customer | Creme/White Paper | White Paper | Shahida |
| Customer | Font | Times New Roman | Shahida |
| Customer | Font Size | 12 | Shahida |
| Shahida | Book Title | The Hobbit | Book Database |
| Shahida | Size | Large | Book Database |
| Shahida | Number of Pages | 395 | Book Database |
| Shahida | Hardback/Paperback | Paperback | Book Database |
| Shahida | Mat/Gloss | Gloss | Book Database |
| Shahida | Creme/White Paper | White Paper | Book Database |
| Shahida | Font | Times New Roman | Book Database |
| Shahida | Font Size | 12 | Book Database |
| Shahida | Forename | Peter | Author Database |
| Shahida | Surname | Parker | Author Database |
| Shahida | Email | mail@example.com | Author Database |
| Shahida | Address | 1 Example Road | Author Database |
| Shahida | Postcode | AB1 2CD | Author Database |
| Shahida | PhoneNumber | 07123456789 | Author Database |
| Shahida | Invoice | - | Invoice Folder |
| Shahida | Invoice | - | Customer |
| Shahida | ISBN | 9780007525492 | Book Database |
| Shahida | Date Published | 23/10/2014 | Book Database |

| **Source** | **Data** | **Example Data** | **Destination** |
|---|---|---|---|
| Shahida | Price | £12.99 | Book Database |
| Customer | Payment | £1000 | Shahida |
| Shahida | Cover Preferences | - | Cover Designer |
| Shahida | Book | - | Editor |
| Editor | Completed Book | - | Shahida |
| Cover Designer | Completed Cover | - | Shahida |
| Shahida | Royalty Statement | - | Royalty Statement Folder |
| Shahida | Royalty Statement | - | Customer |
| Shahida | Royalties | £211.20 | Customer |

**Algorithms**

In the current system there are two Algorithms which are being used. The first sends an invoice to the customer and checks whether Shahida has recieved full payment. Once Shahida has received full payment, she, her cover designer and her editor can begin working on the book. The second algorithm consists of completing the work that is needed to be done, and checks whether the work has been completed, so that the completed book can then be sent off for printing.

**Algorithm 1** First Algorithm - Sending an invoice and Checking for Payment

```
SET Payment TO false

    Check Website for Price
    Create Invoice
    Send Invoice
WHILE Payment = false DO
    Check For Payment
    IF PaymentReceived THEN
    Payment = true
    END IF
END WHILE
```

**Algorithm 2** Second Algorithm - Completing Work and Checking If Work is Completed

```
SET WorkComplete TO false
SET CoverComplete TO false
SET BookComplete TO false

WHILE WorkComplete = false DO
    Get Completed Cover from Cover Designer
    SET CoverComplete TO true
    Get Completed Book from Editor
    SET BookComplete TO true
    IF BookComplete and CoverComplete THEN
        SET finished TO true
    END IF
END WHILE
```

## Data flow diagrams

Figure 1.3: Data Flow Diagrams

**Input Forms, Output Forms, Report Formats**

The current system has just one input form. This is the Enquiry that is sent to the company, from an author. Also, The current system has two different output forms - The Invoice and The Royalty Statement.

The enquiry is received via email, which is sent using the company's website. The email will look like this when received:

First Name:

Last Name:

Email:

Question/Comment:

The following image is an example of the first output form, an Invoice.

PERFECT PUBLISHERS Ltd.23 MAITLAND AVENUE, CAMBRIDGE, CB4 1TA, UK.
Tel: +44 (0) 1223 424422 Fax: +44 (0) 1223 424414

Perfect Publishers Ltd

**INVOICE** ***"Fulfil your books' potential"***

**Invoice Date: 27-05-14**

**Author Name: Svagito Liebermeister**

**Title of Book: Osho Therapy**

**ISBN: 978-1905399-9-25**

**Shipping details:**
**Pratibha de Stoppani**
**via Al Marcadello 2**
**CH-6988 Ponte Tresa**
**Switzerland**

| **Order Description** | | **PRICE** |
|---|---|---|
| 12 Osho Therapy @ 50% discount. Retail price £25.99 | | £155.94 |
| **Shipping** | Premium | £10.06 |
| **TOTAL** | | **£166.00** |

**Account details: BIC: LOYDGB21206 IBAN: GB33 LOYD 3091 7402 3570 35**

**SORT CODE: 30-91-74 ACCOUNT NUMBER: 02357035**

**Payment within 14 days. Late payment will incur a fixed penalty of £20 for the first month and £50 per month thereafter.**

Company Number 5429532. VAT Number: 857 5975 58. Registered in England.
**www.perfectpublishers.co.uk**

Figure 1.4: Invoice Example

The following image is an example of the second output form, a Royalty Statement.

**23 Maitland Avenue**
**Cambridge**
**CB4 1TA**
**United Kingdom**
enquiries@perfectpublishers.co.uk

ROYALTY STATEMENT

Date: 01-01-14 – 30-06-14

| AUTHOR | TITLE | ISBN (13-DIGIT) |
|---|---|---|
| **Andre Corrie** | **Into The Mourning Light** | **9781905399895** |

| LIST PRICE | DISCOUNT | WHOLESALE PRICE | QUANTITY | NET SALES | PRINT COST | NET PUB COMP |
|---|---|---|---|---|---|---|
| **$15.99** | **40%** | **$9.59** | 19 | $182.21 | $96.00 | $91.01* |
| **£9.99** | **40%** | **£5.99** | 69 | £413.31 | £233.10 | £158.01 |
| | | | | **TOTAL** | | **£211.20** |

**PRINT COST PER BOOK: £3.70 UK AND $4.80 US**
***$91.01 = £53.19**

**1 GBP = 1.71565 USD 09-07-14**

**Company Number: 5429532. VAT Number: 857 5975 58. Registered in England.**
www.perfectpublishers.co.uk | www.facebook.com/ppublishers | www.twitter.com/ppublishers

Figure 1.5: Royalty Statement Example

### 1.2.2 The proposed system

In the proposed system the Customer's information will still be received through the online form on the company's website, which Shahida receives via email. She will then enter this into the system using a new interface that will ask her for the details. This will be placed into a database. Each Customer's book will have a primary key, the ISBN number which Shahida assigns to the book. In a seperate database, the author's details will be stored and the author will have a special ID number which is used only in the databases. Every book that is published by the same author will have an attribute which is the author's ID. The ID will just be a 3 digit number. The system's interface will have a search feature, which can search for book titles, authors, and author IDs.

**Data sources and destinations**

| Source | Data | Data Type | Destination |
|---|---|---|---|
| Customer Enquiry | Forename | String | Shahida |
| Customer Enquiry | Surname | String | Shahida |
| Customer Enquiry | Email | String | Shahida |
| Customer | Address | String | Shahida |
| Customer | Postcode | String | Shahida |
| Customer | Phone Number | String | Shahida |
| Customer | Book Title | String | Shahida |
| Customer | Size | String | Shahida |
| Customer | Number of Pages | 395 | Shahida |
| Customer | Hardback/Paperback | Paperback | Shahida |
| Customer | Mat/Gloss | Gloss | Shahida |
| Customer | Creme/White Paper | White Paper | Shahida |
| Customer | Font | Times New Roman | Shahida |
| Customer | Font Size | 12 | Shahida |
| Shahida | Book Title | The Hobbit | Database |
| Shahida | Size | Large | Database |
| Shahida | Number of Pages | 395 | Database |
| Shahida | Hardback/Paperback | Paperback | Database |
| Shahida | Mat/Gloss | Gloss | Database |
| Shahida | Creme/White Paper | White Paper | Database |
| Shahida | Font | Times New Roman | Database |
| Shahida | Font Size | 12 | Database |
| Shahida | Forename | String | Database |
| Shahida | Surname | String | Database |
| Shahida | Email | String | Database |
| Shahida | Address | String | Database |
| Shahida | Postcode | String | Database |
| Shahida | PhoneNumber | String | Database |
| Shahida | ISBN | String | Database |
| Shahida | Date Published | Date | Database |
| Shahida | Price | Real | Book Database |
| Database* | Author ID | Integer | Shahida |
| Shahida | Invoice | String | Invoice Folder |
| Shahida | Invoice | String | Customer |

| **Source** | **Data** | **Data Type** | **Destination** |
|---|---|---|---|
| Customer | Payment | Real | Shahida |
| Shahida | Cover Details | String | Cover Designer |
| Shahida | Book | String | Editor |
| Editor | Completed Book | String | Shahida |
| Cover Designer | Completed Cover | Image | Shahida |
| Shahida | DatePublished | Date | Database |
| Shahida | Royalty Statement | - | Royalty Statement Folder |
| Shahida | Royalty Statement | - | Customer |
| Shahida | Royalties | Real | Customer |

*The Database will create a number and assign that number as an Author ID

## Data flow diagram

Figure 1.6: Data Flow Diagram

**Data dictionary**

| Name | Data Type | Length | Validation | Example Data |
|---|---|---|---|---|
| FirstName | String | 2-20 Characters | Length | Jo |
| LastName | String | 2-20 Characters | Length | Williamson |
| Email | String | 7-30 Characters | Length | mail@example.com |
| PhoneNumber | String | 9-15 Characters | Format | 07123456789 |
| Address | String | 5-64 Characters | Length | Example Road |
| Postcode | String | 7 Characters | Format | AB1 2CD |
| Author ID | Integer | 1-255 | Range | 17 |
| ISBN | String | 13 Characters | Length | 9780007525492 |
| BookTitle | String | 1-127 Characters | Length | The Hobbit |
| NoOfPages | Integer | 1-1023 | Range | 395 |
| Size | String | 5 | Existence | Large |
| Back | String | 8 or 9 Characters | Existence | Paperback |
| Cover | String | 3 or 5 Characters | Existence | Gloss |
| Paper | String | 11 Characters | Existence | White Paper |
| Font | String | 1-64 Characters | Length | Arial |
| FontSize | Real | 8-64 | Numbers only | 12.5 |
| DatePublished | Date | dd/mm/yyyy | Range | 23/10/2014 |
| Price | Real | Numbers only | £12.99 | |

**Volumetrics**

I have conducted calculations to calculate the maximum possible size of 1 customer and book record, which is 275 Bytes. However, when a customer wishes to publish more than one book, more book records are required. As the most amount of books one customer has published with the company is 3, we can have 4 book records per customer record.

Each ASCII Character is 1 byte, each number up to 255 is 1 byte, and each number between 256 and 32768 is 2 bytes. Real Numbers such as 12.5 are 2 bytes, and a Date is 3 bytes.

Firstly, I have worked out the size of the customer record, which is 157 Bytes.

FirstName (20) + LastName (20) + Email (30) + PhoneNumber (15) + Address (64) + Postcode (7) + Author ID (1) = 157 Bytes.

I have then calculated the size of one book record, which is 118 Bytes. Book

Title (1), NoOfPages (2), Size (5), Back (9), Cover (5), Paper (11), Font (64), FontSize (2) + ISBN (13) + DatePublished (3) + Price (2) + Author ID (1) = 118 Bytes

If we have 4 book records per customer record, that would mean that the size for 1 customer with 4 books would be 157 + (5 * 118) = 747 Bytes.

I have chosen to use a size of 100 different customer records, which would be equivalent to 74700 bytes, and 74700 / 1024 = 72.9 Kilobytes. This is because the company rarely have more than 20 enquiries in a year. This would be a suitable number of customer records as it will last a few years before it may require resizing, which can be conducted at a later date when necessary. 72.9 KB will be not be difficult for Shahida's PC to hold, as it is a very small size.

## 1.3 Objectives

### 1.3.1 General Objectives

The general objectives are:

- Organised layout for the database.
- Prevention of unnecessary duplication of data.
- Simple interface for entering data, meaning it can be conducted quickly.
- Search function to find a specific customer in the database.
- Ability to edit existing data easily and quickly.

The System must be able to prevent unecessary duplication of data, and be able to organise data well, and this will be a priority.

### 1.3.2 Specific Objectives

Organising a new layout for the database:

- Be able to sort by date (ascending and descending)
- Clear tables and fields for each entity and attribute

Preventing Duplication:

- Checks to see if the data already exists
- Use of Author ID to ensure it will only be entered once

Simple interface for entering data:

- As little amount of boxes as possible

- Clearly label entry boxes

Search function and editing data:

- Data can be found using the Author ID, Author Name, or Book Title
- Can be edited upon finding the desired data

### 1.3.3 Core Objectives

- Organising the data using certain attributes
- Preventing Duplication

### 1.3.4 Other Objectives

- Searching for data using attributes
- Editing data in the database

## 1.4 ER Diagrams and Descriptions

### 1.4.1 ER Diagram

Figure 1.7: ER Diagram

### 1.4.2 Entity Descriptions

Customer(Author ID, *Email*, Forename, Surname, Address, Postcode, Phone Number)

Enquiry(Email, *Author ID*, Forename, Surname)

Invoice(Author ID, *ISBN Number*, Book title, Price, Forename, Surname, Address, Postcode)

Royalties(AuthorID, *ISBN Number*, Book title, Price, Forename, Surname, Address, Postcode)

Book(ISBN Number, *AuthorID*, Book Title, Pages, Size, Cover type, Colour, Back Type, Paper, Font, Font size, Date published, Price)

The database will only store data about the customers and their books, as the enquiries give details about the books and customers, and the royalties and invoices are stored seperately from the database.

## 1.5 Object Analysis

### 1.5.1 Object Listing

- Shahida
- Customer
- Editor
- Cover Designer
- Spreadsheet

### 1.5.2 Relationship diagrams

Figure 1.8: Relationship Diagram

### 1.5.3 Class definitions

Key:

| **Label** |
|---|
| Attributes |
| Behaviours |

| **Customer** |
|---|
| Author ID<br>ForeName<br>Surname<br>Email<br>Address<br>Postcode<br>Phone Number |
| Add ForeName<br>Edit Forename<br>Add Surname<br>Edit Surname<br>Add Email<br>Edit Email<br>Add Address<br>Edit Address<br>Add Postcode<br>Edit Postcode<br>Add Phone Number<br>Edit Phone Number |

| **Book** |
| --- |
| Title<br>ISBN<br>Pages<br>Size<br>Cover Type<br>Back Type<br>Paper<br>Font<br>Font Size<br>Price<br>Date Published |
| Add Title<br>Edit Title<br>Add ISBN<br>Edit ISBN<br>Add Pages<br>Edit Pages<br>Add Size<br>Edit Size<br>Add Cover Type<br>Edit Cover Type<br>Add Back Type<br>Edit Back Type<br>Add Paper<br>Edit Paper<br>Add Font<br>Edit Font<br>Add Font Size<br>Edit Font Size<br>Add Date Published<br>Edit Date Published<br>Add Price<br>Edit Price |

## 1.6 Other Abstractions and Graphs

Graphs not required.

## 1.7 Constraints

### 1.7.1 Hardware

Shahida uses her laptop to run the company from home. The new system will need to be able to run on this machine.

Computer Specifications:

- 15.6" Display
- AMD Quad-Core A4-5000M APU (1.5GHz, 2MB cache)
- 4 GB DDR3 RAM
- 750 GB HDD, 5400 rpm
- AMD Radeon HD 8330 Graphics Card

The proposed system will have no problem with running on this machine, as it uses a small amount of CPU usage. A constraint would be the size of the screen. This is because the system will need to be based around the screen size of her laptop. As her laptop is portable, portability is not a constraint. The laptop will need enough RAM to hold the system. However, Shahida's laptop has more than enough memory for this, meaning this will not be a problem.

### 1.7.2 Software

Shahida would prefer that the system will run on Windows 7, as she uses this operating system for her laptop. Changing the operating system will cause difficulties, meaning it is best for the system to run on Windows 7, suiting her needs.

### 1.7.3 Time

Shahida does not need this system to be built quickly, but she would like it to be complete as soon as reasonably possible. Otherwise, the only deadline for this project is April 2015, which has been set by my teacher.

### 1.7.4 User Knowledge

Having worked in the publishing industry beforehand, being an author has also given Shahida the knowledge of how to run her current company. Aside from being able to perform basic tasks on a computer, browsing the internet and using social media, Shahida has small experience with computers.

### 1.7.5 Access restrictions

Shahida will be the only person who will have full access to all the data in the proposed system, and she will be the only one who can access it. This can be password protected for security reasons, meaning that only she can gain access to the database. This is also because she is the only necessary person to view, enter and edit data in the system, as her Editor and her Cover Designer do not need to use the database. As she is the only user of the database, it will be easier to keep secure. The authors will be able to make requests about personal data, such as having it removed, or receiving a copy of the personal data about them. The database will comply to the Data Protection Act 1998, as the company already does so with their current system.

## 1.8 Limitations

### 1.8.1 Areas which will not be included in computerisation

Generally, all actions require the use of a computer in the company. However, rarely, a customer does call Shahida about an enquiry, as this customer may not be so computer literate. In this case, Shahida will note down the details of the enquiry, and will enter it into the database.

### 1.8.2 Areas considered for future computerisation

The database could be used online, so that the authors can use their Author ID to log in and see just their details on the database. This would mean that the customers would not have to contact Shahida to receive the data held about them, as they can see the data by theirselves. They will also be able to access this data from anywhere where they have access to the internet. This could also enable Shahida to access the data from other machines aside from her laptop.

## 1.9 Solutions

### 1.9.1 Alternative solutions

| Solution | Advantages | Disadvantages |
|---|---|---|
| Re-organisation of the current spreadsheet | No changes to current operating system and software required, will not cost | Current problems will still occur, Difficult to keep organised as it will require more maintenance to do so |
| Python Desktop Application with GUI | User Friendly, Clear and easy to interpret, Layout can be designed specificly for the client, Usage of buttons simplifies tasks, Minimal training needed for most levels of experience | Takes up more memory, Takes longer to create the application |
| Filing system | No electronics needed, Costs less, Minimal training needed for most levels of experience | Difficult to back up the data due to it being held on paper, data will have to be sent via post when necessary, Lots of physical space is required, more prone to damage and deteriation due to more movement |

### 1.9.2 Justification of chosen solution

I have chosen to use the Python Desktop Application with GUI as my solution. This is because:

- I am already familiar with the Python Programming Language, whereas I have little knowledge of how to manage a Paper Filing system or with creating advance spreadsheets.
- This will keep the system using computers and software, meaning there will not be a drastic change.
- Using the application will take less time than manually entering everything into a spreadsheet.

- This will also take less time and physical space than writing details down on paper.

# Chapter 2

# Design

## 2.1 Overall System Design

### 2.1.1 Short description of the main parts of the system

- Log In Window
- Main Database Interface
- Adding/Removing/Editing Customers and Entries
- Calender Interface
- Changing Password
- Search Window

Log In Window

- A window is displayed which prompts the user to input their ID and password.
- Checks the entered valiues with the database to identify whether the user's credentials are correct.
- Once a correct set of values are entered, the user will be granted access to the database.
- A link will be at the bottom which says "Forgotten password?". This can be clicked on and then the user wil be prompted for the email address, and the corresponding password for the email address entered will be sent to that email.

- If there is no record of an email and password then the user will be prompted to create one for their corresponding email.

Main Database Interface

- This will be the "home" interface.
- A view of the Customer details in the database will be available.
- The user can select an author from the basic view of the database, and click view
- A user interface is presented with a set of options which are: View, Search Database, Add Entry, Remove an Entry, Edit an Entry, Change Password, and Log out.
- Clicking the Search Database Button will prompt a seperate interface to open, and shows details which can be used to search for specific items in the database.
- Customers can be searched for quickly using their first name on this screen.

View Screen

- Clicking view after having selected an customer will open a new window which will show a more in depth view of it. It will show the books that have been published with them.
- There will be buttons to expand on certain fields, including Royalties, Publishing Invoices and Book Invoices. These will show in new windows. The user can see the breakdown of various parts of the database, such as the royalties and invoices. The user will have the option to add and remove royalties and invoices.

Adding/Removing/Editing Customers and Entries

- Clicking the Add Entry Button will prompt a seperate interface to open, and contains a layout of entry boxes for required fields for entering details about the customer. After this, the user can click on the customer's new record from the menu and click edit or a book/royalties/royalty items/-book invoice/book invoice items/publication invoice, dependent on which has been selected. An existing customer can be selected using the search function.
- If a customer already exists, and details are needed to be edited or deleted, a search can be conducted to find that customer.
- Clicking the Remove Entry Button will prompt a seperate interface to open, which contains a view of the database, consisting of all the customers. Three search boxes can be used for searching for their forename,

surname or AuthorID. If an entry was selected beforehand, then upon clicking Remove Entry, The user will be prompted for confirmation, then asked to enter their password.

- Clicking the Edit Entry button will prompt a seperate interface to open, and will contain a view of the database. An entry can be searched for using the search, selected, and once the user clicks "Edit", the user will be prompted with a text box, asking for the user to enter text. Upon confirming what the user wants to enter, they are required to enter their password. This will then be saved. If a customer entry was selected before hand, a new window for adding entries will open first upon clicking Edit Entry, and the data about the customers will be in the fields already, ready for editing. Then, the data can be edited and saved, and the user will be prompted for confirmation then asked to enter their password.

Changing Password

- An interface will open, which will prompt the user to enter their Email, Old Password, and Dhen the new password twice for confirmation.
- Once this has been confirmed, the interface will close, resorting back to the log in window.

Search Window

- Clicking the Search Database Button will open a seperate interface, which contains a set options which the user can choose from in order to conduct a search.
- Once the Search Button is clicked, a list of all the data entries that match the search criteria will come up in a list in the Main Window or the Editing Window.
- The search will only use values and text strings that the user is familiar with, and will not require the user to know obscure pieces of data.

### 2.1.2 System flowcharts showing an overview of the complete system

The following is a flowchart representing a summary of the complete system.

Figure 2.1: Flowchart 1

Figure 2.2: Flowchart 2

Figure 2.3: Flowchart 3

Figure 2.4: Flowchart 4

Figure 2.5: Flowchart 5

Figure 2.6: Flowchart 6

Figure 2.7: Flowchart 7

## 2.2 User Interface Designs

Figure 2.8: Login Screen and Main Menu

Figure 2.9: View Menu and Royalties

Figure 2.10: RoyaltiesItems and BookInvoice and Items

Figure 2.11: Publishing Invoice and Adding Entries

Figure 2.12: Search

Figure 2.13: Editing Screens

Figure 2.14: Remove Entry and Change Password

Figure 2.15: Calculations

Figure 2.16: Adding Publishing Invoices

Figure 2.17: Add Book

## 2.3 Hardware Specification

The system needs to be able to run on a laptop with a 1366 x 768, 16:9 aspect ratio screen which runs on Windows 8. This is imperative to the size of the application I will be creating because it must fit on the given screen size and can't be resizable. A mouse or touchpad will be used for navigational purposes, and for confirmation of entries. Also, a keyboard will be used for inputting information into fields for entering and editing information. All the data used by the program and its database will be held on a local hard drive, and a display is needed for the outputs of the program. These are all suitable for their purposes required, and are available to my client, as she already has all of the required input and output devices, software and hardware.

## 2.4 Program Structure

### 2.4.1 Top-down design structure charts

.

Figure 2.18: Top-down design structure chart

### 2.4.2 Algorithms in pseudo-code for each data transformation process

**Algorithm 3** Add Customer Entry

```
function AddEntry(CustomerTable) Loop = 1
    SET GetNewAuthorID TO False
    SET ConfirmClicked TO False NewAuthorID = 0
    WHILE GetNewAuthorID = false DO
     NewAuthorID = loop Get AuthorID[loop] from CustomerTable
       IF NewAuthorID = AuthorID[loop] THEN
     loop = loop +1
       ELSE
           SET GetNewAuthorID TO True
       END IF
    END WHILE
     FirstName = null
     LastName = null
     Email = null
     PhoneNumber = null
     Address = null
     Postcode = null
    WHILE len(FirstName) <= 0 DO
     INPUT FirstName
    END WHILE
    WHILE len(LastName) <= 0 DO
     INPUT LastName
    END WHILE
    WHILE len(Email) <= 0 DO
     INPUT Email
    END WHILE
    WHILE len(PhoneNumber) <= 0 DO
     INPUT PhoneNumber
    END WHILE
    WHILE len(Address) <= 0 DO
     INPUT Address
    END WHILE
    WHILE len(Postcode) <= 0 DO
     INPUT Postcode
    END WHILE
    IF ConfirmClicked == True THEN
     CONNECT to Customer Database
    END IF
END function
```

**Algorithm 4** Add and Calculate Royalty Items- part 1

```
SET CalculateClicked TO False
SET AddToDatabaseClicked TO False
    Date = null
    WholeSalePrice = null
    QuantitySold = null
    Currency = null
    ExcRateFromGBP = null
    ISBN = null
    RoyaltyDiscount = null
WHILE len(Date) <= 0 DO
    INPUT Date
END WHILE
WHILE len(WholeSalePrice) <= 0 DO
    INPUT WholeSalePrice
END WHILE
WHILE len(QuantitySold) <= 0 DO
    INPUT LastName
END WHILE
WHILE len(Currency) <= 0 DO
    INPUT Currency
END WHILE
IF Currency <> $ THEN
    WHILE len(ExcRatefromGBP) <= 0 DO
    INPUT ExcRateFromGBP
    END WHILE
END IF
WHILE len(ISBN) <= 0 DO
    INPUT ISBN
END WHILE
WHILE len(RoyaltyDiscount) <= 0 DO
    INPUT RoyaltyDiscount
    IF RoyaltyDiscount > 100 THEN
    RoyaltyDiscount = null
    END IF
    IF RoyaltyDiscount < 0 THEN
    RoyaltyDiscount = null
    END IF
END WHILE
```

**Algorithm 5** Add and Calculate Royalty Items- part 2

```
IF Cover == Colour THEN
    Size = Small
    PagePrice = 0.06 * NoOfPages
  IF Back == Hard THEN
    CoverPrice = 5
  END IF
  IF Back == Soft THEN
    CoverPrice = 3
  END IF
END IF
IF Cover == Black/White THEN
  IF Size == Large THEN
    PagePrice = 0.015 * NoOfPages
    IF Back == Hard THEN
    CoverPrice = 5
    END IF
    IF Back == Soft THEN
    CoverPrice = 1
    END IF
  END IF
  IF Size == Small THEN
    PagePrice = 0.01 * NoOfPages
    IF Back == Hard THEN
    CoverPrice = 4
    END IF
    IF Back == Soft THEN CoverPrice = 0.7
    END IF
  END IF
END IF
    PrintCost = (PagePrice + CoverPrice) * RoyaltyQuantity
IF CalculateClicked == True THEN
    NetSales = WholeSalePrice * RoyaltyQuantity
    RoyaltyPayment = NetSales - PrintCost
    OUTPUT RoyaltyPayment
END IF
IF AddToDatabaseClicked == True THEN
    CONNECT to RoyaltyItems Database
END IF
```

**Algorithm 6** Add and Calculate Invoice Items

```
function AddInvoiceItem(BookTable)
    SET CalculateClicked TO False
    SET AddToDatabaseClicked TO False
      Date = null
      QuantityBought = null
      InvoiceDiscount = null
      ShippingType= null
      ShippingPrice= null
      ISBN = null
    WHILE len(Date) <= 0 DO
      INPUT Date
    END WHILE
    WHILE len(QuantityBought) <= 0 DO
      INPUT QuantityBought
    END WHILE
    WHILE len(InvoiceDiscount) <= 0 DO
      INPUT InvoiceDiscount
        IF InvoiceDiscount > 100 THEN
      InvoiceDiscount = null
        END IF
        IF InvoiceDiscount < 0 THEN
      InvoiceDiscount = null
        END IF
    END WHILE
    WHILE len(ISBN) <= 0 DO
      INPUT ISBN
    END WHILE
    WHILE len(ShippingType) <= 0 DO
      INPUT ShippingType
    END WHILE
    WHILE len(ShippingPrice) <= 0 DO
      INPUT ShippingPrice
    END WHILE
    IF CalculateClicked == True THEN
      InvoicePayment = (QuantityBought * Price * InvoiceDiscount) + Shipping Price
      OUTPUT InvoicePayment
    END IF
    IF AddToDatabaseClicked == True THEN
      CONNECT to Invoice Database
    END IF
END function
```

**Algorithm 7** Add Book

```
function AddInvoiceItem(CustomerTable)
    SET AddToDatabaseClicked TO False
      DatePublished = null
      ISBN = null
      AuthorID = null
      BookTitle = null
      NoOfPages = null
      Size = null
      Back = null
      Cover = null
      Paper = null
      Font = null
      FontSize = null
      Price = null
    WHILE len(DatePublished) <= 0 DO
      INPUT DatePublished
    END WHILE
    WHILE len(ISBN) <= 0 DO
      INPUT ISBN
    END WHILE
    WHILE len(BookTitle) <= 0 DO
      INPUT BookTitle
    END WHILE
    WHILE len(Size) <= 0 DO
      INPUT Size
    END WHILE
    WHILE len(BackType) <= 0 DO
      INPUT BackType
    END WHILE
    WHILE len(NoOfPages) <= 0 DO
      INPUT NoOfPages
    END WHILE
    WHILE len(Paper) <= 0 DO
      INPUT Paper
    END WHILE
    WHILE len(Font) <= 0 DO
      INPUT Font
    END WHILE
    WHILE len(FontSize) <= 0 DO
      INPUT FontSize
    END WHILE
    WHILE len(Price) <= 0 DO
      INPUT Price
    END WHILE
    IF AddToDatabaseClicked == True THEN
      CONNECT to Book Database
    END IF
END function
```

**Algorithm 8** Add and Publishing Invoice

```
function AddInvoiceItem(BookTable, CustomerTable)
    SET AddToDatabaseClicked TO False
      Date = null
      AuthorID = null
      ServiceType = null
      InvoicePayment= null
      ISBN = null
    WHILE len(Date) <= 0 DO
      INPUT Date
    END WHILE
    WHILE len(AuthorID) <= 0 DO
      INPUT AuthorID
    END WHILE
    WHILE len(InvoicePayment) <= 0 DO
      INPUT InvoicePayment
    END WHILE
    WHILE len(ISBN) <= 0 DO
      INPUT ISBN
    END WHILE
    WHILE len(ServiceType) <= 0 DO
      INPUT ServiceType
    END WHILE
    IF AddToDatabaseClicked == True THEN
      CONNECT to PubInvoice Database
    END IF
END function
```

### 2.4.3 Object Diagrams

Figure 2.19: Object Diagram

### 2.4.4 Class Definitions

Key:

| **Label** |
|---|
| Attributes |
| Behaviours |

| **Customer** |
|---|
| Author ID<br>ForeName<br>Surname<br>Email<br>Address<br>Postcode<br>Phone Number |
| Add ForeName<br>Edit Forename<br>Add Surname<br>Edit Surname<br>Add Email<br>Edit Email<br>Add Address<br>Edit Address<br>Add Postcode<br>Edit Postcode<br>Add Phone Number<br>Edit Phone Number |

| **Book** |
| --- |
| ISBN<br>AuthorID<br>BookTitle<br>NoOfPages<br>Size<br>Cover<br>Back<br>Paper<br>Font<br>FontSize<br>Price<br>DatePublished |
| Add ISBN<br>Edit ISBN<br>Add AuthorID<br>Edit AuthorID<br>Add BookTitle<br>Edit BookTitle<br>Add NoOfPages<br>Edit NoOfPages<br>Add Size<br>Edit Size<br>Add Cover<br>Edit Cover<br>Add Back<br>Edit Back<br>Add Paper<br>Edit Paper<br>Add Font<br>Edit Font<br>Add FontSize<br>Edit FontSize<br>Add DatePublished<br>Edit DatePublished<br>Add Price<br>Edit Price |

| **Royalties** |
|---|
| RoyaltiesID<br>AuthorID<br>RoyaltyPayment<br>RoyaltiesDate |
| Add AuthorID<br>Edit AuthorID<br>Add RoyaltiesDate<br>Edit Royalties Date |

| **Royalties Items** |
|---|
| RoyaltiesItems<br>RoyaltiesID<br>ISBN<br>Currency<br>RoyaltyDiscount<br>WholesalePrice<br>RoyaltyQuantity<br>NetSales<br>PrintCost<br>ExcRateFromGBP |
| Add RoyaltiesID<br>Edit RoyaltiesID<br>Add ISBN<br>Edit ISBN<br>Add Currency<br>Edit Currency<br>Add RoyaltyDiscount<br>Edit RoyaltyDiscount<br>Add WholesalePrice<br>Edit WholesalePrice<br>Add RoyaltyQuantity<br>Edit RoyaltyQuantity<br>Add ExcRateFromGBP<br>Edit ExcRateFromGBP |

| **BookInvoice** |
|---|
| BookInvoiceID<br>AuthorID<br>BookInvoicePayment<br>BookInvoiceDate |
| Add AuthorID<br>Edit AuthorID<br>Add BookInvoiceDate<br>Edit BookInvoiceDaate |

| **BookInvoiceItems** |
|---|
| BookInvoiceItems<br>BookInvoiceID<br>ISBN<br>BookInvoiceQuantity<br>BookInvoiceDiscount<br>ShippingType<br>ShippingPrice |
| Add BookInvoiceID<br>Edit BookInvoiceID<br>Add ISBN<br>Edit ISBN<br>Add BookInvoiceQuantity<br>Edit BookInvoiceQuantity<br>Add BookInvoiceDiscount<br>Edit BookInvoiceDiscount<br>Add ShippingType<br>Edit Shipping Type<br>Add ShippingPrice<br>Edit ShippingPrice |

| **PubInvoice** |
|---|
| PubInvoiceID<br>AuthorID<br>ISBN<br>PubInvoiceDate<br>PubInvoiceService<br>PubInvoicePayment |
| Add AuthorID<br>Edit AuthorID<br>Add ISBN<br>Edit ISBN<br>Add PubInvoiceDate<br>Edit PubInvoiceDate<br>Add PubInvoiceService<br>Edit PubInvoiceService<br>Add PubInvoicePayment<br>EditPubInvoicePayment |

## 2.5 Prototyping

It will be helpful to create a prototype of the main menu to make sure that different windows can be navigated through without any difficulty. This would give me a good idea of the flow of control between interfaces. Also, I plan to

prototype a log in screen, because this would help me identify the difficulties involved in moving to and from the main menu and log in screen.

Furthermore, I plan to prototype the adding, editing and removal of data to and from the database. I am going to prototype this in order to make sure that data can be successfully added, edited and removed to and from the database so that it can be confirmed that this can be conducted upon creation of the program.

I have already prototyped the calendar interface used for inputting the date in a box. This is just a prototype of the basic interface, which is for selecting a date and placing it in the text box. It can switch between months with a dropdown list, or using the arrows and also years.

Figure 2.20: Calendar Interface

Figure 2.21: Calendar - Date Selection

## 2.6 Definition of Data Requirements

### 2.6.1 Identification of all data input items

- First Name
- Last Name
- Email
- Phone Number
- Address
- Postcode
- ISBN
- Book Title
- Number of pages
- Size
- Back type
- Cover type
- Paper type
- Font
- Font size
- Date published
- Price
- Currency
- Royalties date
- Royalty discount
- Wholesale price
- Royalty Quantity
- Exchange rate from pounds
- Publishing invoice date
- Publishing invoice service
- Publishing invoice payment
- Book invoice date

- Book invoice discount
- Book invoice quantity
- Shipping price
- Shipping type

### 2.6.2 Identification of all data output items

- Author ID
- Royalties ID
- Publishing Invoice ID
- Books Invoice ID
- Net Sales
- Print Cost
- Royalty Payment
- Net Publishing Compensation
- Book Invoice Payment
- Royalties Items
- Book Invoice Items

### 2.6.3 Explanation of how data output items are generated

| **Output** | **How it is produced** | **Input Items Required to Produce** |
|---|---|---|
| Author ID | Generated by program, using first available unused number | |
| Royalties ID | Generated by program, using first available unused number | |
| Publishing Invoice ID | Generated by program, using first available unused number | |
| Book Invoice ID | Generated by program, using first available unused number | |
| Net Sales | Calculated using WholesalePrice x Quantity | Wholesale Price, Royalty Quantity |
| Print Cost | Calculated by program using certain criteria for some details of the book | Number of Pages, Size, Back type, Cover type, Royalty Quantity |
| Royalty Payment | Calculated by program using calculations of all payments under the same RoyaltiesID | |
| Book Invoice Payment | Calculated by program using calculations of all payments under the same BookInvoiceID | Book Invoice Quantity, Book Invoice Discount, Price, Shipping Price |
| Royalties Items | Generated by program, using first available unused number | |
| Book Invoice Items | Generated by program, using first available unused number | |

### 2.6.4 Data Dictionary

There have been some changes to my data dictionary as a number of elements have been identified that need to be added to the system.

| **Name** | **Data Type** | **Length** | **Validation** | **Example Data** |
|---|---|---|---|---|
| Firstname | String | 2-20 Characters | Length | Jo |
| Lastname | String | 2-20 Characters | Length | Williamson |
| Email | String | 7-30 Characters | Length | jo@mail.com |
| Phonenumber | String | 9-15 Characters | Format | 07123456789 |
| Address | String | 5-64 Characters | Length | Example Road |
| Postcode | String | 7 Characters | Format | AB1 2CD |
| AuthorID | Integer | 1-255 | Unique in table, not Null | 17 |
| ISBN | String | 13 Characters | Length | 9780007525492 |
| BookTitle | String | 1-127 Characters | Length | The Hobbit |
| NoOfPages | Integer | 1-1023 | Range | 395 |
| Size | String | 5 | Existence | Large |
| Back | String | 8 or 9 Characters | Existence | Paperback |
| Cover | String | 3 or 5 Characters | Existence | Gloss |
| Paper | String | 11 Characters | Existence | White Paper |
| Font | String | 1-64 Characters | Length | Arial |
| FontSize | Real | 8-64 | Numbers only | 12.5 |
| DatePublished | Date | dd/mm/yyyy | Range | 23/10/2014 |
| Price | Real | 0.01-63.00 | Numbers only | £12.99 |
| RoyaltiesID | Integer | 1-255 | Unique in table, not Null | 123 |
| RoyaltiesItems | Integer | 1-511 | Unique in table, not Null | 12 |
| Currency | String | 1 Character | Pound, Dollar or Euro sign | £ |
| RoyaltyPayment | Real | 1-32767 | Numbers only | 489.92 |
| RoyaltiesDate | Date | dd/mm/yyyy | Range | 03/12/2014 |

| **Name** | **Data Type** | **Length** | **Validation** | **Example Data** |
|---|---|---|---|---|
| RoyaltyDiscount | Real | 0-100 | Numbers only | 40 |
| WholeSalePrice | Real | 0.01-63.00 | Numbers only | 7.99 |
| RoyaltyQuantity | Integer | 1- 2047 | Numbers only | 192 |
| NetSales | Real | 0.01-32767.00 | Numbers only | 900.00 |
| PrintCost | Real | 0.01-32767.00 | Numbers only | 800.00 |
| ExcRateFromGBP | Real | 0-1027 | Numbers only | 1.67 |
| PubInvoiceID | Integer | 1-511 | Unique in table, not Null | 123 |
| PubInvoiceDate | Date | dd/mm/yyy | Range | 23/05/2014 |
| PubInvoiceService | String | 1-127 Characters | Length | Standard |
| PubInvoicePayment | Real | 0.01-32767.00 | Numbers only | 700.00 |
| BookInvoiceID | Integer | 1-255 | Unique in table, not Null | 99 |
| BookInvoiceItems | Integer | 1-255 | Unique in table, not Null | 2 |
| BookInvoicePayment | Real | 0.01-32767.00 | Numbers only | 600.00 |
| BookInvoiceDate | Date | dd/mm/yyyy | Range | 01/01/2014 |
| BookInvoiceDiscount | Real | 0-100 | Numbers only | 50 |
| BookInvoiceQuantity | Integer | 1-2047 | Numbers only | 777 |
| Shipping Type | String | 7-8 Characters | Range | Premium |
| Shipping Price | Real | 0.01-64.00 | Numbers only | 25.00 |

### 2.6.5 Identification of appropriate storage media

A hard drive, such as the hard drive in my client's laptop, would be an example of appropriate storage media. For back up purposes, an external hard disk would be the optimal solution, because it is portable, meaning it can be kept

off site, as well as being able to hold all data files on it. The database will be needed to be kept for long term storage. This is the data will be required to be accessed a number of times.

## 2.7 Database Design

### 2.7.1 Normalisation

**ER Diagrams**

Figure 2.22: ER Diagram

**Entity Descriptions**

Customer(Author ID, FirstName, LastName, Email, Address, Postcode, Phone Number)

Book(ISBN, *AuthorID*, Book Title, NoOfPages, Size, Cover, Paper, Back, Paper, Font, FontSize, DatePublished, Price)

BookInvoice(BookInvoiceID, *AuthorID*, BookInvoiceDate, BookInvoicePayment)

BookInvoiceItems(BookInvoiceItems, *BookInvoiceID*, *ISBN*, BookInvoiceQuantity, BookInvoiceDiscount, ShippingPrice, ShippingType)

Royalties(RoyaltiesID, *AuthorID*, RoyaltiesDate, RoyaltyPayment)

RoyaltiesItems(RoyaltiesItems, *RoyaltiesID*, *ISBN*, Currency, RoyaltyDiscount, WholesalePrice, RoyaltyQuantity, NetSales, PrintCost, ExcRateFromGBP)

PubInvoice(PubInvoiceID, *AuthorID*, *ISBN*, PubInvoiceDate, PubInvoiceOrder, PubInvoicePayment)

**UNF to 3NF**

Key:

**Bold Font** = Primary Key

*Italics* = Foreign Key

Each Column represents a new group.

First of all, I have started with the data in its unnormalised form.

| |
|---|
| FirstName |
| LastName |
| Email |
| PhoneNumber |
| Address |
| PostCode |
| AuthorID |
| ISBN |
| BookTitle |
| NoOfPages |
| Size |
| Back |
| Cover |
| Paper |
| Font |
| FontSize |
| DatePublished |
| Price |
| RoyaltiesID |
| RoyaltiesItems |
| Currency |
| RoyaltyPayment |
| RoyaltiesDate |
| RoyaltyDiscount |
| WholeSalePrice |
| RoyaltyQuantity |
| NetSales |
| PrintCost |
| ExcRateFromGBP |
| PubInvoiceID |
| PubInvoicePayment |
| PubInvoiceDate |
| PubInvoiceService |
| BookInvoiceID |
| BookInvoiceItems |
| BookInvoicePayment |
| BookInvoiceDate |
| BookInvoiceDiscount |
| BookInvoiceQuantity |
| ShippingType |
| ShippingPrice |

Then, I put it into the first normal form, seperating the repeating attributes and the non-repeating attributes.

| | |
|---|---|
| **AuthorID** | **ISBN** |
| FirstName | **AuthorID** |
| LastName | BookTitle |
| Email | NoOfPages |
| PhoneNumber | Size |
| Address | Back |
| PostCode | Cover |
| | Paper |
| | Font |
| | FontSize |
| | DatePublished |
| | Price |
| | RoyaltiesID |
| | RoyaltiesItems |
| | Currency |
| | RoyaltyPayment |
| | RoyaltiesDate |
| | RoyaltyDiscount |
| | WholeSalePrice |
| | RoyaltyQuantity |
| | NetSales |
| | PrintCost |
| | ExcRateFromGBP |
| | PubInvoiceID |
| | PubInvoiceDate |
| | PubInvoiceService |
| | PubInvoicePayment |
| | BookInvoiceID |
| | BookInvoiceItems |
| | BookInvoicePayment |
| | BookInvoiceDate |
| | BookInvoiceDiscount |
| | BookInvoiceQuantity |
| | ShippingType |
| | ShippingPrice |

After that, I put it into the second normal form.

| | | |
|---|---|---|
| **AuthorID** | **ISBN** | **ISBN** |
| FirstName | **AuthorID** | BookTitle |
| LastName | RoyaltiesID | NoOfPages |
| Email | Currency | Size |
| PhoneNumber | RoyaltyPayment | Back |
| Address | RoyaltiesDate | Cover |
| PostCode | RoyaltyDiscount | Paper |
| | WholeSalePrice | Font |
| | RoyaltyQuantity | FontSize |
| | NetSales | DatePublished |
| | PrintCost | Price |
| | ExcRateFromGBP | |
| | PubInvoiceID | |
| | PubInvoiceDate | |
| | PubInvoiceService | |
| | PubInvoicePayment | |
| | BookInvoiceID | |
| | BooInvoiceItems | |
| | BookInvoicePayment | |
| | BookInvoiceDate | |
| | BookInvoiceDiscount | |
| | BookInvoiceQuantity | |
| | ShippingType | |
| | ShippingPrice | |

Finally, I put the data into its third normal form.

| **AuthorID** | **ISBN** | **RoyaltiesID** | **RoyaltiesItems** |
|---|---|---|---|
| FirstName | *AuthorID* | *AuthorID* | *RoyaltiesID* |
| LastName | BookTitle | RoyaltyPayment | *ISBN* |
| Email | NoOfPages | RoyaltiesDate | Currency |
| PhoneNumber | Size | | RoyaltyDiscount |
| Address | Back | | WholeSalePrice |
| PostCode | Cover | | RoyaltyQuantity |
| | Paper | | NetSales |
| | Font | | PrintCost |
| | FontSize | | ExcRateFromGBP |
| | DatePublished | | |
| | Price | | |

| **PubInvoiceID** | **BooksInvoiceID** | **BooksInvoiceItems** |
|---|---|---|
| *AuthorID* | *AuthorID* | *BooksInvoiceID* |
| *ISBN* | BookInvoicePayment | *ISBN* |
| PubInvoiceDate | BookInvoiceDate | BookInvoiceQuantity |
| PubInvoiceService | | BookInvoiceDiscount |
| PubInvoicePayment | | ShippingType |
| | | ShippingPrice |

### 2.7.2 SQL Queries

I am using Python to format the SQL query text strings.

| **SQL** | **Descriptions** |
|---|---|
| """insert into Customer(FirstName, LastName, Email, PhoneNumber, Address, Postcode) values (Ex, ample, example@mail.com, 07123456789, 1 example road, AB1 2CD) """ | An example of an SQL statement which adds customer records to the database. Here, it is entering a new customer record with the attributes: Firstname, Lastname, Email, Phonenumber, Address and Postcode. |
| """create table RoyaltiesItems( RoyaltiesID INTEGER, Currency REAL, RoyaltyDiscount STRING, WholeSalePrice REAL, RoyaltyQuantity INTEGER, NetSales REAL, PrintCost REAL, ExcRateFromGBP STRING PRIMARY KEY(RoyaltiesItems) FOREIGN KEY(RoyaltiesID) REFERENCES Royalties(RoyaltiesID) """ | An example of an SQL statement that creates a new table for the Royalties. There is a primary key which is RoyaltiesItems, and there is one foreign key, which is RoyaltiesID. |
| """select LastName, BookTitle from Customer, Book where Price = 13.00 and Back = Paperback """ | This statement will return all the LastNames and the BookTitles from the Customer table and the Book table whose book is paperback and costs £13. |
| """update Customer set Firstname = 'John', Lastname = 'Smith' where AuthorID = 1 """ | This statement will update the Firstname to 'John' and Lastname to 'Smith' of an entry in the Customer table where the AuthorID = 1. |
| """delete from Customer where AuthorID = 1 """ | This statement will delete the entry in the Customer table where the AuthorID = 1 |
| """select * from RoyaltiesItems where Currency = '£' and Firstname from Customer = 'John"""" | This statement will fetch all the RoyaltiesItems where the Currency is in GBP, and the firstname of the corresponding author is John. |

## 2.8 Security and Integrity of the System and Data

### 2.8.1 Security and Integrity of Data

The system will store personal data about the customers and will comply to the Data Protection Act. This means that the data requires sufficiently frequent checks, so there will be a way to edit and change the information using the program. All the data in the database must be kept securely, so that it can only be granted access to someone with the use of a password. To ensure that all data stored is valid, everytime the user adds data to the database, a check will be conducted to ensure that the data is valid. I need to keep referential integrity in the database so that there are never any records with missing key data upon any removals and the database will be encrypted.

### 2.8.2 System Security

The database will be password protected to make sure that only users who know the password can access the database. This will keep the number of users of the database to a minimum. This can prevent the data from being tampered with or stolen. The database will be encrypted in order to avoid unwanted people having access to the data without the use of the system, therefore the number of people who can access the data can be determined beforehand.

## 2.9 Validation

The system will check to make sure each entry is valid, in order to avoid any invalid entries into the database.

| **Item** | **Example** | **Validation** | **Comments** |
|---|---|---|---|
| FirstName | John | Presence Check, Type Check | Makes sure that a name is entered, and that the name is just letters. |
| LastName | Smith | Presence Check, Type Check | Makes sure that a name is entered, and that the name is just letters. |
| Email | test@mail.com | Presence Check, Type Check | Makes sure that an email is entered and that the @ symbol comes before the '.' in the email. |
| PhoneNumber | 07123456789 | Presence Check | Makes sure that a phone number is entered. |
| Address | 1 Example Road | Presence Check | Makes sure that an address is entered. |
| Postcode | AB1 2CD | Presence Check | Makes sure that a Postcode is entered. |
| ISBN | 9780748782987 | Presence Check, Length Check | Makes sure that an ISBN is entered and is not longer than 13 digits. |
| BookTitle | Computing A2 Text Book | Presence Check | Makes sure that a Book Title has been entered. |
| NoOfPages | 395 | Presence Check, Type Check | Makes sure that a positive integer has been entered. |
| Size | Large | Presence Check | Makes sure that a Size has been entered. |
| Back | Hard | Presence Check | Makes sure that a Back type has been entered. |
| Cover | Colour | Presence Check | Makes sure that a Cover type has been entered. |
| Paper | White | Presence Check | Makes sure that a Paper type has been entered. |
| Font | Microsoft Sans Serif | Presence Check | Makes sure that a Font has been entered. |

| **Item** | **Example** | **Validation** | **Comments** |
|---|---|---|---|
| FontSize | 12 | Presence Check, Type Check | Makes sure that a positive value has been entered, and just digits. |
| DatePublished | 25 Dec 2014 | Presence Check, Type Check | Makes sure that a Date has been entered. |
| Price | 8.99 | Presence Check, Type Check | Makes sure that a positive value has been entered, and just digits. |
| Currency | £ | Presence Check, Type Check | Makes sure that the correct symbol has been entered. |
| RoyaltiesDate | 25 Dec 2014 | Presence Check, Type Check | Makes sure that a Date has been entered. |
| RoyaltyDiscount | 50 | Presence Check, Type Check, Range Check | Makes sure that a positive value has been entered, just digits, and it must be between 0 and 100. |
| WholesalePrice | 5.99 | Presence Check, Type Check | Makes sure that a positive value has been entered, and just digits. |
| RoyaltyQuantity | 150 | Presence Check, Type Check | Makes sure that a positive integer has been entered. |
| ExcRate FromGBP | 1.67 | Presence Check, Type Check | Makes sure that a value has been entered. |
| Pub Invoice Date | 25 Dec 2014 | Presence Check, Type Check | Makes sure that a Date has been entered. |
| Pub Invoice Service | Standard | Presence Check | Makes sure that a Service has been entered. |
| Pub Invoice Payment | £1659.99 | Presence Check, Type Check | Makes sure that a positive value has been entered. |

| **Item** | **Example** | **Validation** | **Comments** |
|---|---|---|---|
| Book Invoice Date | 25 Dec 2014 | Presence Check, Type Check | Makes sure that a Date has been entered. |
| Book Invoice Discount | 50 | Presence Check, Type Check, Range Check | Makes sure that a positive value has been entered, just digits, and it must be between 0 and 100. |
| Book Invoice Quantity | 25 | PresenceCheck, Type Check | Makes sure that a positive integer has been entered. |
| Shipping Type | Premium | Lookup check | Makes sure that a Shipping type has been entered. |
| Shipping Price | 4.00 | Presence Check, Type Check | Makes sure that a positive value has been entered, and just digits. |
| AuthorID | 21 | Presence Check, Type Check, Look up Check | Makes sure that an integer has been entered, and will check the Customer Table to find the entry to confirm that it is valid. |
| RoyaltiesID | 21 | Presence Check, Type Check, Look up Check | Makes sure that an integer has been entered, and will check the Royalties Table to find the entry to confirm that it is valid. |
| PubInvoiceID | 21 | Presence Check, Type Check, Look up Check | Makes sure that an integer has been entered, and will check the PubInvoice Table to find the entry to confirm that it is valid. |
| BookInvoiceID | 21 | Presence Check, Type Check, Look up Check | Makes sure that an integer has been entered, and will check the Book Invoice Table to find the entry to confirm that it is valid. |

## 2.10 Testing

### 2.10.1 Outline Plan

| Test Series | Purpose of Test Series | Testing Strategy | Strategy Rationale |
|---|---|---|---|
| 1 | Testing the flow of control between user interfaces | Top-down Testing | |
| 2 | Testing the validation of input data | Bottom-up testing | All components are to be tested after development |
| 3 | Testing the algorithms' functionality | White box testing | |
| 4 | Testing that the information has been successfully stored, and in the right places | Black box testing | |
| 5 | Testing the system and whether it meets the requirements | System testing | |

### 2.10.2 Detailed Plan

| Test Series | Purpose of Test | Test Description | Test Data | Test Data Type (Normal/ Erroneous/ Boundary) | Expected Result | Actual Result | Evidence |
|---|---|---|---|---|---|---|---|

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1.1 | Test the Log in button on the log in sreen | This should check whether the password and email match and exist in a record | Click the log in button | Normal | If the email and password match, the main menu should open, else the program should prompt the user with an error | | |
| 1.2 | Test the View button on the main menu | This button links the main menu to the view menu. | Click the View Button | Normal | The program should open the View Menu in a new window | | |
| 1.3 | Testing the Log Out button on the Main Menu | This button links to the Login screen, where the user is required to log in again | Click the Log out button | Normal | The screen should switch to the log out screen | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1.4 | Testing the Search Database button on the Main Menu | This button should prompt a seperate interface to open, and show details which can be used to search for specific items in the database | Click the Search Database button | Normal | The program should open a new window consisting of the Search Database screen | | |
| 1.5 | Testing the Add Entry button on the Main Menu | This button should prompt a seperate interface to open and show the Add Entry screen | Click the Add Entry button | Normal | The program should open the Add Entry screen in a new window | | |
| 1.6 | Testing the Edit Entry button from the Main Menu | This button links to the Editing screen | Click the Edit Entry button | Normal | The program should switch to the Editing screen from the Main Menu | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1.7 | Testing the Edit Entry button after an entry has been selected beforehand | These conditions should open the Edit Entry screen upon clicking Edit Entry, with data on the selected entry already filled in on the grid | Click on an entry, and then click Edit Entry | Normal | The program should open the Edit Entry screen in a new window | | |
| 1.8 | Testing the Remove Entry Button after an entry has been selected beforehand | These conditions should prompt the user for verification on deleting a selected customer record | Click on an entry and then click Remove Entry | Normal | The program should open a Verification window, asking the user for confirmation and verification on removing the selected customer record | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1.9 | Testing the Change Password Button on the Main Menu | This button prompts the Change Password window to open | Click the Change Password button | Normal | The program should open the Change Password window, with fields required to be filled in in order to change the password | | |
| 1.10 | Testing the Quick Search button on the Main Menu | This button returns the customer that matches the entered AuthorID | Type in a Lastname and click QuickSearch | Normal | The program should return the customer that matches the entered Lastname and show it in the grid | | |
| 1.11 | Testing the back button on the View Menu | This button returns back to the main menu | Click Back | Normal | The program should return back to the main menu | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1.12 | Testing the Add Book Button on the View Menu | This button opens up a new window for adding a book | Click Add Book | Normal | The program should open a new window for adding a book | | |
| 1.13 | Testing the View Publishing Invoice button on the View Menu | This button opens up a new window displaying details about the selected book's publishing invoice | Click View Publishing invoice | Normal | The program should open a new window containing details about the selected book's publishing invoice | | |
| 1.14 | Testing the View Royalties button on the View Menu | This button opens up a new window displaying details about the selected book's royalties | Click View Royalties | Normal | The program should open a new window containing details about the selected book's royalties | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1.15 | Testing the View Book Invoice button on the View Menu | This button opens up a new window displaying details about the selected book's book invoices | Click View Book Invoices | Normal | The program should open a new window containing details about the selected book's book invoices | | |
| 1.16 | Testing the Delete book button on the View Menu | This button opens up a new window asking for verification on deleting a selected book record | Click Delete book | Normal | The program should open a Verification window, asking the user for confirmation and verification on removing the selected customer record | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2.1 | Verify that some criteria has been entered when using the search | At least one set of the input boxes and drop-down lists must have been filled in or selected from, else the program will prompt the user about the error | Selected from list with names and values entered<br>Selected from list, values en-teredand no names entered<br>Just values entered<br>Nothing | Normal<br>Normal<br>Erroneous<br>Erroneous | Accept<br>Accept<br>Error<br>Error | | |
| 2.2 | Verify that a valid email has been entered on the log in screen | The program will prompt the user telling them they have inputted an error | test@testmail.com<br>helloworld<br>test @test-mail.com<br>test.com@testmail | Normal<br>Erroneous<br>Erroneous<br>Erroneous | Accept<br>Error<br>Error<br>Error | | |
| 2.3 | Verify that a valid First-name has been entered when adding an entry | The user will be prompted with an error | John<br>Jo hn<br>Jo?hn<br>Jo2n | Normal<br>Erroneous<br>Erroneous<br>Erroneous | Accept<br>Error<br>Error<br>Error | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2.4 | Verify that a valid Lastname has been entered when adding an entry | The user will be prompted with an error | Smith<br>Sm ith<br>Smi!th<br>Smi1th | Normal<br>Erroneous<br>Erroneous<br>Erroneous | Accept<br>Error<br>Error<br>Error | | |
| 2.5 | Verify that a valid Phonenumber has been entered when adding an entry | The user will be prompted with an error | 07123456789<br>07123123.3<br>071CO2 | Normal<br>Erroneous<br>Erroneous | Accept<br>Error<br>Error | | |
| 2.7 | Verify that a valid Address has been entered when adding an entry | The user will be prompted with an error | 1 Example road<br>@@ £!<br>1231231 | Normal<br>Erroneous<br>Erroneous | Accept<br>Error<br>Error | | |
| 2.8 | Verify that a valid ISBN has been entered when adding a book | The user will be prompted with an error | 1234567890123<br>@@ £!<br>123<br>HelloWorld | Normal<br>Erroneous<br>Erroneous<br>Erroneous | Accept<br>Error<br>Error<br>Error | | |
| 2.9 | Verify that a valid NoOfPages has been entered when adding a book | The user will be prompted with an error | 123<br>123456789<br>2.3<br>@!<br>HelloWorld | Normal<br>Erroneous<br>Erroneous<br>Erroneous | Accept<br>Error<br>Error | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2.10 | Verify that a valid ISBN has been entered when adding a book | The user will be prompted with an error | 1234567890123<br>@@ £!<br>123<br>HelloWorld | Normal<br>Erroneous<br>Erroneous<br>Erroneous | Accept<br>Error<br>Error<br>Error | | |
| 3.1 | Verify that all fields required are entered when adding a book | If all fields are filled in correctly, a book will be successfully added to the database | Fill all Fields Correctly and then click Add to Database<br>Leave Fields Blank and then click Add to Database | Normal<br>Erroneous | Verification screen should open<br>User will be prompted with an error | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3.2 | Verify that all fields required are entered and calculations are complete when adding an Invoice Item | If all fields are filled in correctly and the calculations are complete, then the invoice item will be added to the database | Fill all fields correctly and click calculate and then click Add to Database<br>Leave Fields Blank and click Calculate<br>Fill all fields correctly and click add to database | Normal<br>Erroneuos<br>Erroneous | Verification screen should open<br>User will be prompted with an error<br>User will be prompted with an error | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3.3 | Verify that all fields required are entered and calculations are complete when adding a Royalty Item | If all fields are filled in correctly and the calculations are complete, then the royalty item will be added to the database | Fill all fields correctly and click calculate and then click Add to Database<br>Leave Fields Blank and click Calculate<br>Fill all fields correctly and click add to database | Normal<br>Erroneous<br>Erroneous | Verification screen should open<br>User will be prompted with an error<br>User will be prompted to click Calculate | | |
| 4.1 | Verify that all author data has been added to the author database | All the information should be added to the correct fields in the author table | Author Information | Normal | Added to the Author Table | | |
| 4.2 | Verify that all book data has been added to the book database | All the information should be added to the correct fields in the book table | Book Information | Normal | Added to the Book Table | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 4.3 | Verify that all royalty item data has been added to the book database | All the information should be added to the correct fields in the royalty item table | Royalty Items Information | Normal | Added to the Royalty Items Table | | |
| 4.4 | Verify that all invoice items data has been added to the Invoice Items database | All the information should be added to the correct fields in the Book Invoice Items table | Invoice Items Information | Normal | Added to the Invoice Items Table | | |
| 4.5 | Verify that all publishing invoice data has been added to the publishing invoice database | All the information should be added to the correct fields in the Publishing Invoice table | Publishing Invoice Information | Normal | Added to the Publishing Invoice Table | | |
| 4.6 | Verify that all Royalties data has been added to the Royalties database | All the information should be added to the correct fields in the Royalties table | Royalties Information | Normal | Added to the Royalties Table | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 4.7 | Verify that all Book Invoice data has been added to the Book Invoice database | All the information should be added to the correct fields in the Book Invoice table | Book Invoice Information | Normal | Added to the Book Invoice Table | | |
| 5 | Verify that the program meets the requirements given | Run the program testing all parts to make sure they meet all of the requirements | Add entries for all possible inputs in order to test them all, update all entries, remove the entries, conduct a search, view all windows, change password and log out. | Normal | Program is up to the required standards | | |

# Chapter 3

# Testing

## 3.1 Test Plan

### 3.1.1 Original Outline Plan

| Test Series | Purpose of Test Series | Testing Strategy | Strategy Rationale |
|---|---|---|---|
| 1 | Testing the flow of control between user interfaces | Top-down Testing | |
| 2 | Testing the validation of input data | Bottom-up testing | All components are to be tested after development |
| 3 | Testing the algorithms' functionality | White box testing | |
| 4 | Testing that the information has been successfully stored, and in the right places | Black box testing | |
| 5 | Testing the system and whether it meets the requirements | System testing | |

### 3.1.2 Changes to Outline Plan

| Test Series | Purpose of Test Series | Testing Strategy | Strategy Rationale |
|---|---|---|---|
| 1 | Testing the flow of control between user interfaces | Top-down Testing | |
| 2 | Testing the validation of input data | Bottom-up testing | All components are to be tested after development |
| 3 | Testing the algorithms' functionality | White box testing | |
| 4 | Testing that the information has been successfully stored, and in the right places | Black box testing | |
| 5 | Testing the system and whether it meets the requirements | System testing | |

### 3.1.3 Original Detailed Plan

| Test Series | Purpose of Test | Test Description | Test Data | Test Data Type (Normal/ Erroneous/ Boundary) | Expected Result | Actual Result | Evidence |
|---|---|---|---|---|---|---|---|

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1.1 | Test the Log in button on the log in screen | This should check whether the password and email match and exist in a record | Click the log in button | Normal | If the email and password match, the main menu should open, else the program should prompt the user with an error | | |
| 1.2 | Test the View button on the main menu | This button links the main menu to the view menu. | Click the View Button | Normal | The program should open the View Menu in a new window | | |
| 1.3 | Testing the Log Out button on the Main Menu | This button links to the Login screen, where the user is required to log in again | Click the Log out button | Normal | The screen should switch to the log out screen | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1.4 | Testing the Search Database button on the Main Menu | This button should prompt a seperate interface to open, and show details which can be used to search for specific items in the database | Click the Search Database button | Normal | The program should open a new window consisting of the Search Database screen | | |
| 1.5 | Testing the Add Entry button on the Main Menu | This button should prompt a seperate interface to open and show the Add Entry screen | Click the Add Entry button | Normal | The program should open the Add Entry screen in a new window | | |
| 1.6 | Testing the Edit Entry button from the Main Menu | This button links to the Editing screen | Click the Edit Entry button | Normal | The program should switch to the Editing screen from the Main Menu | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1.7 | Testing the Edit Entry button after an entry has been selected beforehand | These conditions should open the Edit Entry screen upon clicking Edit Entry, with data on the selected entry already filled in on the grid | Click on an entry, and then click Edit Entry | Normal | The program should open the Edit Entry screen in a new window | | |
| 1.8 | Testing the Remove Entry Button after an entry has been selected beforehand | These conditions should prompt the user for verification on deleting a selected customer record | Click on an entry and then click Remove Entry | Normal | The program should open a Verification window, asking the user for confirmation and verification on removing the selected customer record | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1.9 | Testing the Change Password Button on the Main Menu | This button prompts the Change Password window to open | Click the Change Password button | Normal | The program should open the Change Password window, with fields required to be filled in in order to change the password | | |
| 1.10 | Testing the Quick Search button on the Main Menu | This button returns the customer that matches the entered AuthorID | Type in a Lastname and click QuickSearch | Normal | The program should return the customer that matches the entered Lastname and show it in the grid | | |
| 1.11 | Testing the back button on the View Menu | This button returns back to the main menu | Click Back | Normal | The program should return back to the main menu | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1.12 | Testing the Add Book Button on the View Menu | This button opens up a new window for adding a book | Click Add Book | Normal | The program should open a new window for adding a book | | |
| 1.13 | Testing the View Publishing Invoice button on the View Menu | This button opens up a new window displaying details about the selected book's publishing invoice | Click View Publishing invoice | Normal | The program should open a new window containing details about the selected book's publishing invoice | | |
| 1.14 | Testing the View Royalties button on the View Menu | This button opens up a new window displaying details about the selected book's royalties | Click View Royalties | Normal | The program should open a new window containing details about the selected book's royalties | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1.15 | Testing the View Book Invoice button on the View Menu | This button opens up a new window displaying details about the selected book's book invoices | Click View Book Invoices | Normal | The program should open a new window containing details about the selected book's book invoices | | |
| 1.16 | Testing the Delete book button on the View Menu | This button opens up a new window asking for verification on deleting a selected book record | Click Delete book | Normal | The program should open a Verification window, asking the user for confirmation and verification on removing the selected customer record | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2.1 | Verify that some criteria has been entered when using the search | At least one set of the input boxes and drop-down lists must have been filled in or selected from, else the program will prompt the user about the error | Selected from list with names and values entered<br>Selected from list, values enteredand no names entered<br>Just values entered<br>Nothing | Normal<br>Normal<br>Erroneous<br>Erroneous | Accept<br>Accept<br>Error<br>Error | | |
| 2.2 | Verify that a valid email has been entered on the log in screen | The program will prompt the user telling them they have inputted an error | test@testmail.com<br>helloworld<br>test @test-mail.com<br>test.com@testmail | Normal<br>Erroneous<br>Erroneous<br>Erroneous | Accept<br>Error<br>Error<br>Error | | |
| 2.3 | Verify that a valid First-name has been entered when adding an entry | The user will be prompted with an error | John<br>Jo hn<br>Jo?hn<br>Jo2n | Normal<br>Erroneous<br>Erroneous<br>Erroneous | Accept<br>Error<br>Error<br>Error | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2.4 | Verify that a valid Lastname has been entered when adding an entry | The user will be prompted with an error | Smith<br>Sm ith<br>Smi?th<br>Smi1th | Normal<br>Erroneous<br>Erroneous<br>Erroneous | Accept<br>Error<br>Error<br>Error | | |
| 2.5 | Verify that a valid Phonenumber has been entered when adding an entry | The user will be prompted with an error | 07123456789<br>07123123.3<br>071CO2 | Normal<br>Erroneous<br>Erroneous | Accept<br>Error<br>Error | | |
| 2.7 | Verify that a valid Address has been entered when adding an entry | The user will be prompted with an error | 1 Example road<br>@@ £!<br>1231231 | Normal<br>Erroneous<br>Erroneous | Accept<br>Error<br>Error | | |
| 2.8 | Verify that a valid ISBN has been entered when adding a book | The user will be prompted with an error | 1234567890123<br>@@ £!<br>123<br>HelloWorld | Normal<br>Erroneous<br>Erroneous<br>Erroneous | Accept<br>Error<br>Error<br>Error | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2.9 | Verify that a valid NoOf-Pages has been entered when adding a book | The user will be prompted with an error | 123<br>123456789<br>2.3<br>@!<br>HelloWorld | Normal<br>Erroneous<br>Erroneous<br>Erroneous | Accept<br>Error<br>Error | | |
| 3.1 | Verify that all fields required are entered when adding a book | If all fields are filled in correctly, a book will be successfully added to the database | Fill all Fields Correctly and then click Add to Database<br>Leave Fields Blank and then click Add to Database | Normal<br>Erroneous | Verification screen should open<br>User will be prompted with an error | | |
| 3.2 | Verify that all fields required are entered and calculations are complete when adding an Invoice Item | If all fields are filled in correctly and the calculations are complete, then the invoice item will be added to the database | Fill all fields correctly and click calculate and then click Add to Database<br>Leave Fields Blank and click Calculate<br>Fill all fields correctly and click add to database | Normal<br>Erroneuos<br>Erroneous | Verification screen should open<br>User will be prompted with an error<br>User will be prompted with an error | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3.3 | Verify that all fields required are entered and calculations are complete when adding a Royalty Item | If all fields are filled in correctly and the calculations are complete, then the royalty item will be added to the database | Fill all fields correctly and click calculate and then click Add to Database<br>Leave Fields Blank and click Calculate<br>Fill all fields correctly and click add to database | Normal<br>Erroneous<br>Erroneous | Verification screen should open<br>User will be prompted with an error<br>User will be prompted to click Calculate | | |
| 4.1 | Verify that all author data has been added to the author database | All the information should be added to the correct fields in the author table | Author Information | Normal | Added to the Author Table | | |
| 4.2 | Verify that all book data has been added to the book database | All the information should be added to the correct fields in the book table | Book Information | Normal | Added to the Book Table | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 4.3 | Verify that all royalty item data has been added to the book database | All the information should be added to the correct fields in the royalty item table | Royalty Items Information | Normal | Added to the Royalty Items Table | | |
| 4.4 | Verify that all invoice items data has been added to the Invoice Items database | All the information should be added to the correct fields in the Book Invoice Items table | Invoice Items Information | Normal | Added to the Invoice Items Table | | |
| 4.5 | Verify that all publishing invoice data has been added to the publishing invoice database | All the information should be added to the correct fields in the Publishing Invoice table | Publishing Invoice Information | Normal | Added to the Publishing Invoice Table | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 4.6 | Verify that all Royalties data has been added to the Royalties database | All the information should be added to the correct fields in the Royalties table | Royalties Information | Normal | Added to the Royalties Table | | |
| 4.7 | Verify that all Book Invoice data has been added to the Book Invoice database | All the information should be added to the correct fields in the Book Invoice table | Book Invoice Information | Normal | Added to the Book Invoice Table | | |
| 5 | Verify that the program meets the requirements given | Run the program testing all parts to make sure they meet all of the requirements | Add entries for all possible inputs in order to test them all, update all entries, remove the entries, conduct a search, view all windows, change password and log out. | Normal | Program is up to the required standards | | |

### 3.1.4 Changes to Detailed Plan

The tests that have been removed are highlighted in gray. These were removed because they were not included in the sytem due to changes made during the deveopment of the system, as it was not possible to keep some features in the system. The tests that were changed/added are highlighted in light gray. These were changed because the functionality of the system had also been changed to make sure the system was still possible to develop, as it previously was not. The tests that were added were added because the system needed features to be tested because they were newly added to the system, or they were features that were required to be tested but I had not forseen this beforehand.

| **Test Series** | **Purpose of Test** | **Test Description** | **Test Data** | **Test Data Type (Normal/ Erroneous/ Boundary)** | **Expected Result** | **Actual Result** | **Evidence** |
|---|---|---|---|---|---|---|---|
| 1.1 | Test the Log in button on the log in screen | This should check whether the password and email match and exist in a record | Click the log in button | Normal | If the email and password match, the main menu should open, else the program should prompt the user with an error | Works as expected | Figure 3.1 on page 146, Figure 3.2 on page 147 and Figure 3.3 on page 148 . |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1.2 | Test the View button on the main menu | This button links the main menu to the view menu. | Click the View Button after selecting an author. | Normal | The program should open the View Menu in a new window, displaying the selected Customer's Books. | Works as expected | Figure 3.4 on page 149 |
| 1.3 | Testing the Log Out button on the Main Menu | This button links to the Login screen, where the user is required to log in again | Click the Log out button | Normal | The screen should switch to the log out screen | Works as expected | |
| 1.4 | Testing the Search Database button on the Main Menu | This button should prompt a seperate interface to open, and show details which can be used to search for specific items in the database | Click the Search Database button | Normal | The program should open a new window consisting of the Search Database screen | Works as expected | Figure 3.5 on page 150 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1.5 | Testing the Add Entry button on the Main Menu | This button should prompt a seperate interface to open and show the Add Entry screen | Click the Add Entry button | Normal | The program should open the Add Entry screen in a new window | Works as expected | Figure 3.6 on page 151 |
| 1.6 | Testing the Edit Entry button from the Main Menu | This button links to the Editing screen | Click the Edit Entry button | Normal | The program should switch to the Editing screen from the Main Menu | | |
| 1.7 | Testing the Update Entry button after an entry has been selected beforehand | These conditions should open the Update Entry screen upon clicking Update Entry, with data on the selected entry already filled in on the grid | Click on an entry, and then click Update Entry | Normal | The program should open the Update Entry screen in a new window | Works as expected | Figure 3.7 on page 152 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1.8 | Testing the Remove Entry Button after an entry has been selected beforehand | These conditions should prompt the user for verification on deleting a selected customer record | Click on an entry and then click Remove Entry | Normal | The program should open a Verification window, asking the user for confirmation and verification on removing the selected customer record | Works as expected | Figure 3.8 on page 153 |
| 1.9 | Testing the Change Password/Username Button on the Main Menu | This button prompts a window to open, with a set of buttons asking what the user wishes to changes | Click the Change Password/Username button | Normal | The program should open the Change Password/Username window, with buttons giving options of what is needed to be changed. | Works as expected | Figure 3.9 on page 154 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1.10 | Testing the Quick Search button on the Main Menu | This button returns the customer that matches the entered Name | Type in a Firstname, Lastname or both and click QuickSearch | Normal | The program should return any customers that match the entered name(s) and show it in the grid | Works as expected | Figure 3.10 on page 155 |
| 1.11 | Testing the back button on the View Menu | This button returns back to the main menu | Click Back | Normal | The program should return back to the main menu | Works as expected | |
| 1.12 | Testing the Add Book Button on the View Menu | This button opens up a new window for adding a book | Click Add Book | Normal | The program should open a new window for adding a book | Works as expected | Figure 3.11 on page 156 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1.13 | Testing the View Publishing Invoice button on the View Menu | This button opens up a new window displaying details about the selected book's publishing invoice | Click View Publishing invoice | Normal | The program should open a new window containing details about the selected book's publishing invoice | Works as expected | Figure 3.12 on page 157 |
| 1.14 | Testing the View Royalties button on the View Menu | This button opens up a new window displaying details about the selected book's royalties | Click View Royalties | Normal | The program should open a new window containing details about the selected book's royalties | Works as expected | Figure 3.13 on page 158 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1.15 | Testing the View Book Invoice button on the View Menu | This button opens up a new window displaying details about the selected book's book invoices | Click View Book Invoices | Normal | The program should open a new window containing details about the selected book's book invoices | Works as expected | Figure 3.14 on page 159 |
| 1.16 | Testing the Delete book button on the View Menu | This button opens up a new window asking for verification on deleting a selected book record | Click Delete book | Normal | The program should open a Verification window, asking the user for confirmation and verification on removing the selected customer record | Works as expected | Figure 3.15 on page 160 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1.17 | Testing the 'Forgotten Password' label on the log in screen | This opens a window, giving the user the default username and password if it's the first time of usage, or it opens an input box where an email is asked for, if the default credentials have been changed. | Click 'Forgotten Password' | Normal | This should open a window, giving the user the default username and password if it's the first time of usage, or it will open a input box where an email is asked for, if the default credentials have been changed | Works as expected | Figure 3.16 on page 161 |
| 1.18 | Testing the Change Password Button | This button opens a new window, where the user is presented with fields required for changing their password | Click Change Password | Normal | The program should open the Change Password window and present the necessary fields to the user | Works as expected | Figure 3.17 on page 162 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1.19 | Testing the Confirm button for changing passwords | This button should accept the entries the user has entered if the old password matches, and the new and retyped passwords matches otherwise the user will be prompted with an error. | Fill the necessary fields and click Confirm | Normal | This should accept the user's input, successfully changing the password if the criteria is met, else the user is prompted with an error | Works as expected | Figure 3.18 on page 163 |
| 1.20 | Testing the View Royalty Items button on the Royalties Menu | This button opens up a new window displaying details about the selected Royalty's items | Select a Royalty and click View Royalty Items | Normal | The program should open a new window containing details about the selected Royalty's Items | Works as expected | Figure 3.19 on page 164 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1.21 | Testing the View Book Invoice Items button on the Book Invoice Menu | This button opens up a new window displaying details about the selected Book Invoice's items | Select a Book Invoice and click View Book Invoice Items | Normal | The program should open a new window containing details about the selected Book Invoice's Items | Works as expected | Figure 3.20 on page 164 |
| 1.22 | Testing the Log Out button on the Menubar | This button links to the Login screen, where the user is required to log in again | Click the Log out button on the Menubar | Normal | The screen should switch to the log out screen | Works as expected | |
| 1.23 | Testing the Add Entry button on the Menubar | This button should prompt a seperate interface to open and show the Add Entry screen | Click the Add Entry button on the Menubar | Normal | The program should open the Add Entry screen in a new window | Works as expected | Figure 3.21 on page 165 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1.24 | Testing the Confirm button on the Search window | This button checks whether the entries are valid then conducts the search if they are, presenting results in the main window | Click the Confirm button on the search window | Normal | The program should validate the entries then conduct the search and present search results in the main menu, or prompt the user with an error. | Works as expected | Figure 3.22 on page 165 |
| 1.25 | Testing the Cancel button on the Add Entry window | This button should reject and close the window | Click the cancel button on the add entry window | Normal | The program should reject the window and close it | Works as expected | |
| 1.26 | Testing the Cancel button on the Search window | This button should reject and close the window | Click the cancel button on the Search window | Normal | The program should reject the window and close it | Works as expected | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1.27 | Testing the Update book button on the View Menu | This button opens up a new window with entries already filled in with the selected book's data. | Select an entry then click Update book | Normal | The program should open a window for updating the entry, with the selected data already filled in the entry boxes | Works as expected | Figure 3.23 on page 166 |
| 2.1 | Verify that some criteria has been entered when using the search | Firstname and surname must be filled in, and if a different category has been selected, the final input box must also be filled in. | Author selected, Firstname and Surname entered.<br>Author Selected, boxes left blank<br>Publishing Invoice selected, Service selected, boxes filled with appropriate data<br>Publishing Invoice selected, No boxes filled. | Normal<br>Erroneous<br>Normal<br>Erroneous | Accept<br>Error<br>Accept<br>Error | Works as expected | Figure 3.24 on page 166 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2.2 | Verify that a valid email has been entered on the log in screen | The program will prompt the user telling them they have inputted an error | test@testmail.com<br>helloworld<br>test<br>@testmail.com<br>test.com@testmai | Normal<br>Erroneous<br>Erroneous<br>Erroneous | Accept<br>Error<br>Error<br>Error | | |
| 2.3 | Verify that a valid First-name has been entered when adding an entry | The user will be prompted with an error | John<br>Jo hn<br>Jo?hn<br>Jo2n | Normal<br>Erroneous<br>Erroneous<br>Erroneous | Accept<br>Error<br>Error<br>Error | Works as expected - User is unable to enter 'Jo?hn', 'Jo hn' or 'Jo2n' | Figure 3.25 on page 167 |
| 2.4 | Verify that a valid Last-name has been entered when adding an entry | The user will be prompted with an error | Smith<br>Sm ith<br>Smi?th<br>Smi1th | Normal<br>Erroneous<br>Erroneous<br>Erroneous | Accept<br>Error<br>Error<br>Error | Works as expected - User is unable to enter 'Sm ith', 'Smi?th', or 'Smi1th'. | Figure 3.25 on page 167 |
| 2.5 | Verify that a valid Phonenumber has been entered when adding an entry | The user will be prompted with an error | 07123456789<br>07123123.3<br>071CO2 | Normal<br>Erroneous<br>Erroneous | Accept<br>Error<br>Error | Works as expected - User is unable to enter '07123123.3' or '071CO2' | Figure 3.25 on page 167 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2.6 | Verify that a valid Address has been entered when adding an entry | The user will be prompted with an error | 1 Example road<br>@@ £!<br>1231231 | Normal<br>Erroneous<br>Erroneous | Accept<br>Error<br>Error | Works as expected - User is unable to enter '@@ £!', but is prompted with an error when entering '1231231'. | Figure 3.25 on page 167 and Figure 3.26 on page 168 |
| 2.7 | Verify that a valid ISBN has been entered when adding a book | The user will be prompted with an error | 1234567890123<br>@@ £!<br>1234567890123456<br>0123 | Normal<br>Erroneous<br>Erroneous<br>Erroneous | Accept<br>Error<br>Error | Works as expected - unable to enter '@@ £!'. '123456789012' and '0123' are rejected and the user is prompted with an error. | Figure 3.27 on page 169 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2.8 | Verify that a valid NoOf-Pages has been entered when adding a book | The user will be prompted with an error | 123<br>123456789<br>2.3<br>@!<br>HelloWorld | Normal<br>Erroneous<br>Erroneous<br>Erroneous | Accept<br>Error<br>Error<br>Error | Works as expected - User is unable to type '2.3', '@!' and 'HelloWorld'. User is prompted with an error when entering '123456789'. | Figure 3.28 on page 170 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2.9 | Verify that a valid Discount has been entered when adding Book Invoice items | The user will be prompted with an error | 50<br>5.5<br>0<br>-1<br>100<br>101<br>abc | Normal<br>Normal<br>Boundary<br>Erroneous<br>Boundary<br>Erroneous<br>Erroneous | Accept<br>Error<br>Accept<br>Error<br>Accept<br>Error<br>Error | Works mostly - User is unable to type 'abc'. User is prompted with an error if '-1', or '101' is entered, but when calculate has been clicked and user clicks confirm, window is unexpectedly closed straight after the error prompt. | Figure 3.29 on page 171 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2.10 | Verify that a valid Discount has been entered when adding Royalty items | The user will be prompted with an error | 50<br>5.5<br>0<br>-1<br>100<br>101<br>abc | Normal<br>Normal<br>Boundary<br>Erroneous<br>Boundary<br>Erroneous<br>Erroneous | Accept<br>Error<br>Accept<br>Error<br>Accept<br>Error<br>Error | Works mostly - User is unable to type 'abc'. User is prompted with an error if '-1', or '101' is entered, but when calculate has been clicked and user clicks confirm, window is unexpectedly closed straight after the error prompt. | Figure 3.30 on page 172 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3.1 | Verify that all fields required are entered when adding a book | If all fields are filled in correctly, a book will be successfully added to the database | Fill all Fields Correctly and then click Add to Database<br>Leave Fields Blank and then click Add to Database | Normal<br>Erroneous | Data is added to the database.<br>User will be prompted with an error | Works as expected | Figure 3.31 on page 173 |
| 3.2 | Verify that all fields required are entered and calculations are complete when adding an Invoice Item | If all fields are filled in correctly and the calculations are complete, then the invoice item will be added to the database | Fill all fields correctly and click calculate and then click Add to Database<br>Leave Fields Blank and click Calculate<br>Fill all fields correctly and click add to database | Normal<br>Erroneous<br>Erroneous | Data should be successfully added to the database.<br>User will be prompted with an error<br>User will be prompted with an error | Works mostly as expected - when calculate has been clicked with the invalid entry, and user clicks confirm, window is unexpectedly closed straight after the error prompt. | Figure 3.32 on page 174 and Figure 3.29 on page 171 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3.3 | Verify that all fields required are entered and calculations are complete when adding a Royalty Item | If all fields are filled in correctly and the calculations are complete, then the royalty item will be added to the database | Fill all fields correctly and click calculate and then click Add to Database<br>Leave Fields Blank and click Calculate<br>Fill all fields correctly and click add to database | Normal<br>Erroneous<br>Erroneous | Verification screen should open<br>User will be prompted with an error<br>User will be prompted to click Calculate | Works mostly as expected - when calculate has been clicked with the invalid entry, and user clicks confirm, window is unexpectedly closed straight after the error prompt. | Figure 3.33 on page 175 and Figure 3.30 on page 172 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3.4 | Verify that the calculations function for working out the print cost | The calculations should be conducted and the result should be displayed | Enter values for the Royalty Items<br>Leave fields blank<br>Enter negative numbers into the required fields. | Normal<br>Erroneous<br>Erroneous | Accept<br>Error<br>Error | Calculations are correct, but when a negative number is entered, the calculation is still conducted. User is prompted with the error when they wish to confirm entry, and then the window is unexpectedly closed afterwards. | Figure 3.34 on page 176 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3.5 | Verify that the calculations function for working out the total Royalty Payment for a set of payments. | The calculations should be conducted and the result should be displayed in the Royalties table | Enter values for the Royalty Items and add it to the database<br>Leave fields blank<br>Enter negative numbers into the required fields. | Normal<br>Erroneous<br>Erroneous | Accept<br>Error<br>Error | Works as expected - User is prompted when Fields are invalid/blank. | Figure 3.35 on page 177 |
| 3.6 | Verify that the calculations function for working out the total Book Invoice Payment | The calculations should be conducted and the result should be displayed in the Book Invoice table | Enter values for the Book Invoice Items and add them to the database<br>Leave fields blank<br>Enter negative numbers into the required fields. | Normal<br>Erroneous<br>Erroneous | Accept<br>Error<br>Error | Works as expected - User is prompted when Fields are invalid/blank. | Figure 3.36 on page 177 |
| 4.1 | Verify that all author data has been added to the author database | All the information should be added to the correct fields in the author table | Author Information | Normal | Added to the Author Table | Works as expected | Figure 3.37 on page 177 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 4.2 | Verify that all book data has been added to the book database | All the information should be added to the correct fields in the book table | Book Information | Normal | Added to the Book Table | Works as expected | Figure 3.38 on page 178 |
| 4.3 | Verify that all publishing invoice data has been added to the publishing invoice database | All the information should be added to the correct fields in the Publishing Invoice table | Publishing Invoice Information | Normal | Added to the Publishing Invoice Table | Works as expected | Figure 3.39 on page 178 |
| 4.4 | Verify that all Royalties data has been added to the Royalties database | All the information should be added to the correct fields in the Royalties table | Royalties Information | Normal | Added to the Royalties Table | Works as expected | Figure 3.42 on page 180 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 4.5 | Verify that all Book Invoice data has been added to the Book Invoice database | All the information should be added to the correct fields in the Book Invoice table | Book Invoice Information | Normal | Added to the Book Invoice Table | Works as expected | Figure 3.40 on page 179 |
| 4.6 | Verify that all royalty item data has been added to the book database | All the information should be added to the correct fields in the royalty item table | Royalty Items Information | Normal | Added to the Royalty Items Table | Works as expected | Figure 3.43 on page 180 |
| 4.7 | Verify that all invoice items data has been added to the Invoice Items database | All the information should be added to the correct fields in the Book Invoice Items table | Invoice Items Information | Normal | Added to the Invoice Items Table | Works as expected | Figure 3.41 on page 179 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 4.8 | Verify that all author data has been successfully edited afted editing and saving changes | All the data should be changed to what the user's inputs were, and should still be in the correct fields in the Author Table | Author Information | Normal | Changes made to the Author Table | Works as expected | Figure 3.44 on page 181 |
| 4.9 | Verify that all book data has been successfully edited afted editing and saving changes | All the data should be changed to what the user's inputs were, and should still be in the correct fields in the Book Table | Book Information | Normal | Changes made to the Book Table | Works as expected | Figure 3.45 on page 182 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 4.10 | Verify that all Book Invoice data has been success-fully edited afted editing and saving changes | All the data should be changed to what the user's inputs were, and should still be in the correct fields in the Book Invoice Table | Book Invoice Information | Normal | Changes made to the Book Invoice Table | Works as expected | Figure 3.46 on page 183 |
| 4.11 | Verify that all Book Invoice Items data has been success-fully edited afted editing and saving changes | All the data should be changed to what the user's inputs were, and should still be in the correct fields in the Book Invoice Items Table | Book Invoice Items Information | Normal | Changes made to the Book Invoice Items Table | Works as expected | Figure 3.47 on page 184 |
| 4.12 | Verify that a set of royalty items are deleted when a deletion is requested by the user | The royalty items are deleted, and should no longer be in the table | Deleting the data | Normal | The royalty items should no longer exist in the database | Works as expected | Figure 3.48 on page 185 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 4.13 | Verify that a royalty payment is deleted when a deletion is requested by the user | The royalty payment is deleted, and should no longer be in the table | Deleting the data | Normal | The royalty payment should no longer exist in the database | Works as expected | Figure 3.49 on page 186 |
| 4.14 | Verify that all data linked with an author is deleted when a deletion is requested by the user | All data linked with the user is deleted, and should no longer be in the table | Deleting the data | Normal | The data linked with that athor shoiuld no longer be in their respective tables | Works as expected | Figure 3.50 on page 187 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | Verify that the program meets the requirements given | Run the program testing all parts to make sure they meet all of the requirements | Add entries for all possible inputs in order to test them all, update all entries, remove the entries, conduct a search, view all windows, change password and log out. | Normal | Program is up to the required standards | Program is up to the required standards. Minor exception of Add Royalties/-BookInvoice Items window closing when an invalid entry is entered. | |

## 3.2 Test Data

### 3.2.1 Original Test Data

The data is shown in the previous table.

### 3.2.2 Changes to Test Data

Some changes were made to the plan, as the implementation of the program made me decide to change some aspects of the system. The tests removed are highlighted in the changed table in grey, and the tests that were modified are highlighted in light grey.

## 3.3 Annotated Samples

### 3.3.1 Actual Results

The data is shown in the previous table.

### 3.3.2 Evidence

Figure 3.1: Test 1.1

Figure 3.2: Test 1.1

Figure 3.3: Test 1.1

Figure 3.4: Test 1.2

Figure 3.5: Test 1.4

Figure 3.6: Test 1.5

Figure 3.7: Test 1.7

Figure 3.8: Test 1.8

Figure 3.9: Test 1.9

Figure 3.10: Test 1.10

Figure 3.11: Test 1.12

Figure 3.12: Test 1.13

Figure 3.13: Test 1.14

Figure 3.14: Test 1.15

Figure 3.15: Test 1.16

Figure 3.16: Test 1.17

Figure 3.17: Test 1.18

Figure 3.18: Test 1.19

Figure 3.19: Test 1.20

Figure 3.20: Test 1.21

Figure 3.21: Test 1.23

Figure 3.22: Test 1.24

Figure 3.23: Test 1.27

Figure 3.24: Test 2.1

Figure 3.25: Test 2.3, 2.4, 2.5, 2.6

Figure 3.26: Test 2.6

Figure 3.27: Test 2.7

Figure 3.28: Test 2.8

Figure 3.29: Test 2.9

Figure 3.30: Test 2.10

Figure 3.31: Test 3.1

Figure 3.32: Test 3.2

Figure 3.33: Test 3.3

Figure 3.34: Test 3.4

Figure 3.35: Test 3.5

Figure 3.36: Test 3.6

Figure 3.37: Test 4.1

Figure 3.38: Test 4.2

Figure 3.39: Test 4.3

Figure 3.40: Test 4.4

Figure 3.41: Test 4.5

Figure 3.42: Test 4.6

Figure 3.43: Test 4.7

Figure 3.44: Test 4.8

Figure 3.45: Test 4.9

Figure 3.46: Test 4.10

Figure 3.47: Test 4.11

Figure 3.48: Test 4.12

Figure 3.49: Test 4.13

Figure 3.50: Test 4.14

## 3.4 Evaluation

### 3.4.1 Approach to Testing

I have chosen to use a range of testing strategies. For instance, series 1 of of the tests is testing the flow of control, making sure that the user navigates to the correct areas of the interface. Series 2 of the tests ensures that the validation of user inputs is performed correctly, whilst series 3 certifies that the calculations and algorithms in the system function correctly. Series 4 makes sure that the added data is stored correctly, updated data is saved to the correct tables and fields, and the deletion of data is performed on the correct data.

### 3.4.2 Problems Encountered

I encountered a few minor problems whilst testing my program.

Tests 2.9 and 2.10:, Figures 3.29 and 3.30 on pages 171 and 172 respectively

These tests encountered a minor problem, where if the user were to enter an invalid value for an entry, proceed to calculate using their inputs and confirm that they wish to add it to the database, the user would be correctly prompted with an error. However, upon dismissing the error, the window that they were using to add data to the database would close, forcing them to reopen the window and start entering data again. I have a vague idea of where the problem is, but as it isn't a serious problem for the user, this will be considered as an area for future updates.

This is the only problem my tests encountered, which also affected a few of the tests in series 3, causing the same issue.

### 3.4.3 Strengths of Testing

The strengths of my testing program consisted mainly of the consistency of the flow of data, and the flow of control. This proved that there wouldn't be any problem with the navigation of the user interface, and also no problems would be encountered when interacting with the data. Also, the testing program concluded that the user was limited to certain inputs for certain fields. For example, the user was required to enter a quantity for the Book Invoice Item, and was limited to being able to enter integers alone. Also, dropdown lists were used to prevent the user from incorrectly spelling certain inputs. This showed that the testing program would not be limited to identifying specific errors as the user was prevented from entering incorrect data in some parts of the program.

### 3.4.4 Weaknesses of Testing

A weakness of my testing program was that it did not test every single component of the system. because most of the program used similar coding for its functionality. Therefore, after testing a few of the components, others were not tested because of the similar functionality. This means that I am unable to establish that the system is entirely robust.

### 3.4.5 Reliability of Application

All inputs, outputs and changes function as the should do, and as they were planned to. The system mostly prevents te user from entering inaccurate data, so that the inputs and outputs would not malfunction. Although, test 3.4 shows that a calculation would still be conducted if the user had entered a negative number, which is invalid. This happened because the calculation is conducted immediately after the user types. However, the program would not allow the user to confirm their entry with this invalid data, despite the calculations being conducted. Consequently, the system is generally reliable as it is moderately effective in preventing invalid entries.

### 3.4.6 Robustness of Application

The testing program used proves that the system is fairly robust. The program never encounters runtime errors or index errors, which were carefully prevented. Also, the system prevents invalid entries so that the program doesn't crash, and my test program did not cause my program to crash once.

# Chapter 4

# System Maintenance

## 4.1 Environment

### 4.1.1 Software

I have used the following software in the creation of my system:

- Python 3.4
- IDLE
- PyQt4
- SQLite 3

### 4.1.2 Usage Explanation

Python 3.4:

- Only Programming language I am familiar with.
- Easy to learn, meaning it's most appropriate and convenient for me to use.

IDLE:

- Software is included with python
- Designed for python scripting

PyQt4:

- Designed especially for implementing graphical interface

SQLite 3:

- Software is included with python
- Syntax allows interactions with a database

### 4.1.3 Features Used

Python 3.4:

- Allowed me to develop my system
- Could test my system using a CLI or GUI interface
- Intended to be used to run my system in a GUI

IDLE:

- Used to create and develop my python scripts
- indentations and colour coded text made development significantly easier

PyQt4:

- Used features to create the GUI for the system
- Userd modules to create the full interface, including windows and dialogs

SQLite 3:

- Allowed interactions with a database to be conducted rather straightforwardly
- Helped to impose referential integrity

## 4.2 System Overview

### 4.2.1 Main window

The main window consists of a table, including data about all the company's customers. There are a number of options along the bottom, such as View, Search Database, Add Entry, Update Entry and Delete Entry and Change Password. Along the top is menu bar, with the same options, a log out button, and a quick search feature. The quick search feature requires the user to just type a firstname, surname or both in, and click quick search. The table will show customers with the matching names. The system uses stacked layouts to bring certain widgets to the front on specific occasions. For example, when the user wishes to view a customer's books, the index for the current stacked layout is changed, and a new layout of widgets are presented to the user for interactions with book data.

### 4.2.2 Adding Entries

The user is able to add entries for each entity, dependent on where in the interface they had clicked "Add Entry". On the main menu, the user can add a customer. A dialog opens upon clicking the button, displaying a table with the details required as column headers. If "Add Entry" is clicked elsewhere in the interface, the user is presented with a dialog, where a number of line edits are displayed, variant on how many attributes of the entity are required to be entered. On all of these windows, are a confirm and cancel button, to validate and add the entries and to cancel and close the window respectively. When adding a Book, Publishing Invoice, Book Invoice or a Royalty Payment, one of the line edits will have a date button, where a calendar opens upon clicking it. Here, the user can select a date to enter into the line edit, to complete the entry to the database.

### 4.2.3 Updating Entries

The user is able to update or delete entries for each entity. If the user has chosen to update a customer entry, the user will be presented with a dialog with a table, similar to when adding a customer entry. The table will be filled in with the selected customer's details. The user then selects a field, and clicks edit, and a new dialog opens asking the user to type in their new entry for the selected field. Once finished, the user can click confirm in order to verify their username and password to confirm changes. If the user is updating a different entity, the user is presented with a dialog with a number of line edits, similar to when adding an entry elsewhere in the interface. However, the line edits will be filled with the data which was selected by the user, and the user can freely edit data where desired. Upon clicking confirm, the user is required to type in their username and password to confirm the update. If the user wishes to update the ISBN of a book, which is the only primary key which can be edited, system will automatically update the ISBN in all parts of the system where it is used as a foreign key.

### 4.2.4 Deleting Entries

The user can select an entry and click "Delete Entry", when they wish to delete a certain entry. When deleting a customer entry, all data about the customer is deleted with it, but a username and password is required from the user to verify the action. If the user wishes to delete a different entity, the user must navigate to where the data is, select it and click delete. The user will be asked for their username and password. However, if the data cannot be deleted due to the enforced referential integrity, the user will be prompted with an error, and will be told what to delete first in order to delete the entry.

### 4.2.5 Calculations

Upon addition of Royalty/Book Invoice Items, calculations are conducted to find the print cost, and full payments. The print cost is calculated by using the company's standard printing prices, which depends on the attributes of the book, such as whether it is hard back or paper back. The print cost is used with the entered quantity and wholesale price to calculate the payment for that set of items. The items are totalled for the full invoice/royalty payment. If a book is updated, any of it's attributes which determine the print cost could have changed, therefore the calculations are reconducted.

### 4.2.6 Detailed Searches

The user has the option to click "Search database" on the main menu to open up a window for filling in and selecting criteria for a search. The user enters the firstname and surname of the author, and then selects which entity to search for and which category to search. The user then enters the values where required, and they click confirm to validate their entries and conduct the search. The results will show in the main window table, where the user can freely see them. If there aren't any matching searches, the user is prompted with a dialog box.

## 4.3 Code Structure

### 4.3.1 Function - Adding an Item

Lines 103-150 of Module 2, subsection 4.10.2. Generating an Add Window based on the current entity.

```
1       def AddItem(self): #initialising an add window for getting inputs for other entries
2           self.AddWindow = dbAddItemWindow()
3           self.AddWindow.setFixedSize(360,200)
4           self.AddWindow.AddType = self.CurrentTable
5           self.AddWindow.AnswerButtons()
6           self.AddWindow.Editing = False
7
8           if self.CurrentTable == "Book":
9               self.AddWindow.sql = "select * from Book"
10
11          elif self.CurrentTable == "PubInvoice":
12              self.AddWindow.setFixedSize(400,150)
13              self.AddWindow.sql = "select ISBN, AuthorID, PubInvoiceDate, PubInvoiceService,
                    PubInvoicePayment from PubInvoice"
14              self.AddWindow.selectedISBN = self.SelectedISBN
15
16          elif self.CurrentTable == "BookInvoice":
17              self.AddWindow.sql = "select AuthorID, BookInvoiceDate from BookInvoice"
18              self.AddWindow.setFixedSize(350,100)
19
20          elif self.CurrentTable == "Royalties":
21              self.AddWindow.sql = "select AuthorID, RoyaltiesDate from Royalties"
22              self.AddWindow.setFixedSize(350,100)
23
24          elif self.CurrentTable == "BookInvoiceItems":
25              self.AddWindow.sql = "select BookInvoiceID, ISBN, BookInvoiceQuantity, BookInvoiceDiscount,
                    ShippingType, ShippingPrice from BookInvoiceItems"
26              self.AddWindow.setFixedSize(450, 150)
27              self.AddWindow.selectedISBN = self.SelectedISBN
28              self.Editing = False
29              self.AddWindow.btnCalculate.clicked.connect(self.BookInvoiceItemCalculation)
```

```
30
31          elif self.CurrentTable == "RoyaltyItems":
32              self.AddWindow.sql = "select RoyaltiesID, ISBN, Currency, RoyaltyDiscount, WholesalePrice,
                    RoyaltyQuantity, PrintCost, ExcRateFromGBP from RoyaltyItems"
33              self.AddWindow.setFixedSize(450, 250)
34              self.AddWindow.selectedISBN = self.SelectedISBN
35              self.Editing = False
36              self.AddWindow.btnCalculate.clicked.connect(self.RoyaltyItemCalculation)
37
38          self.AddWindow.selectedID = self.SelectedID
39          self.AddWindow.CalendarWidget = dbCalendarWidget()
40          self.AddWindow.CalendarWidget.Calendar()
41
42          self.AddWindow.initAddItemWindow()
43          try:
44              if self.AddWindow.Valid == True: #inputs must pass validation before being added
45                  self.AddToDB()
46          except AttributeError:
47              pass #exception of window being closed
48          self.RefreshTables()
```

This is needed to be a function because it can be repeatedly called, whenever the user wishes to add a non-customer entry, as it instantiates a new window for adding entries. It is different to adding a customer entry because the customer entries are dealt with in a different manner. The SQL statement is dependent on whatever the table that is being added to is, as it is used to generate a certain number of line edits, used for user inputs. In line 44 of the function, the program checks whether the data passed the validation, in order for it to be clear for adding to the database.

### 4.3.2 Function - Removing an Item

Lines 262-344 of Module 2, subsection 4.10.2. Removing a selected Item.

```
1                       cursor.execute(sql)
2                       sql = "delete from Book where AuthorID = {}".format(self.ConfirmDialog.SelectedAuthorID)
3                       cursor.execute(sql)
4                       sql = "delete from Customer where AuthorID =
                            {}".format(self.ConfirmDialog.SelectedAuthorID)
5                       cursor.execute(sql)
6                       db.commit()
7                       self.RefreshTables()
8
9
10
11      def RemoveFromDB(self): #removing other entries
12          #getting primary key of the row
13          self.ConfirmDialog = dbConfirmationDialog()
14          self.ConfirmDialog.Username = self.Username
15          self.ConfirmDialog.DeleteMsg = self.CurrentTable
16          self.SelectedAuthorID = self.SelectedID
17
18          if self.CurrentTable == "Book":
19              self.ForeignKeyMsg = "Royalties and Invoices"
20              self.SelectedRow = self.ViewWindow.table.currentRow()
21              self.SelectedID = QTableWidgetItem(self.ViewWindow.table.item(self.SelectedRow, 0)).text()
22              self.SelectedIDName = "ISBN"
23              self.ConfirmDialog.Msg = "Are you sure you want to delete this book?"
24              self.ConfirmDialog.ConfirmedMsg = "Book was successfully deleted"
25
26          elif self.CurrentTable == "PubInvoice":
27              self.SelectedRow = self.PubInvoiceWindow.table.currentRow()
28              self.OldID = self.SelectedID
29              self.SelectedID = QTableWidgetItem(self.PubInvoiceWindow.table.item(self.SelectedRow, 0)).text()
30              self.SelectedIDName = "PubInvoiceID"
31              self.ConfirmDialog.Msg = "Are you sure you want to delete this Invoice?"
32              self.ConfirmDialog.ConfirmedMsg = "Invoice was successfully deleted"
33
34          elif self.CurrentTable == "BookInvoice":
35              self.ForeignKeyMsg = "Book Invoice Items"
36              self.SelectedRow = self.BookInvoiceWindow.table.currentRow()
37              self.OldID = self.SelectedID
38              self.SelectedID = QTableWidgetItem(self.BookInvoiceWindow.table.item(self.SelectedRow,
                    0)).text()
39              self.SelectedIDName = "BookInvoiceID"
40              self.ConfirmDialog.Msg = "Are you sure you want to delete this Invoice?"
```

```
41              self.ConfirmDialog.ConfirmedMsg = "Invoice was successfully deleted"
42
43          elif self.CurrentTable == "Royalties":
44              self.ForeignKeyMsg = "Royalty Items"
45              self.SelectedRow = self.RoyaltiesWindow.table.currentRow()
46              self.OldID = self.SelectedID
47              self.SelectedID = QTableWidgetItem(self.RoyaltiesWindow.table.item(self.SelectedRow, 0)).text()
48              self.SelectedIDName = "RoyaltiesID"
49              self.ConfirmDialog.Msg = "Are you sure you want to delete this Entry?"
50              self.ConfirmDialog.ConfirmedMsg = "Entry was successfully deleted"
51
52          elif self.CurrentTable == "BookInvoiceItems":
53              self.SelectedRow = self.BookInvoiceItemsWindow.table.currentRow()
54              self.OldID = self.SelectedID
55              self.SelectedID = QTableWidgetItem(self.BookInvoiceItemsWindow.table.item(self.SelectedRow,
                      0)).text()
56              self.SelectedIDName = "BookInvoiceItemsID"
57              self.ConfirmDialog.Msg = "Are you sure you want to delete this Item?"
58              self.ConfirmDialog.ConfirmedMsg = "Item was successfully deleted"
59
60          elif self.CurrentTable == "RoyaltyItems":
61              self.SelectedRow = self.RoyaltyItemsWindow.table.currentRow()
62              self.OldID = self.SelectedID
63              self.SelectedID = QTableWidgetItem(self.RoyaltyItemsWindow.table.item(self.SelectedRow,
                      0)).text()
64              self.SelectedIDName = "RoyaltyItemsID"
65              self.ConfirmDialog.Msg = "Are you sure you want to delete this Item?"
66              self.ConfirmDialog.ConfirmedMsg = "Item was successfully deleted"
67          self.ConfirmDialog.Prevention = True #deletion may be prevented if integrity error occurs
68          if self.SelectedRow != -1:
69              self.ConfirmDialog.VerifyDlg() #verification
70              try:
71                  if self.ConfirmDialog.ConfirmedDialog.Accepted == True:
72                      with sqlite3.connect("PP.db") as db:
73                          cursor = db.cursor()
74                          cursor.execute("PRAGMA foreign_keys = ON")
75                          sql = "delete from {} where {} = '{}'".format(self.CurrentTable,
                                  self.SelectedIDName, self.SelectedID)
76                          cursor.execute(sql)
77                          db.commit()
78                          self.ConfirmDialog.ConfirmedDialog.Confirmed()
79                          self.SelectedID = self.SelectedAuthorID
80                          self.RefreshTables()
81              except sqlite3.IntegrityError:
82                  self.Msg = QMessageBox()
83                  self.Msg.setWindowTitle("Error")
84                  self.Msg.setText("You must delete all the {} first.".format(self.ForeignKeyMsg))
85                  self.Msg.exec_() #informs user that selected entry cannot be deleted and what must be done
                        for it to be deleted
86
87              if self.CurrentTable in ["Royalties", "BookInvoice", "PubInvoice", "BookInvoiceItems",
                      "RoyaltyItems"]:
88                  self.SelectedID = self.OldID
89
90              try:
91                  self.RecalculateItems()
92              except:
93                  pass
```

This is a function because it needs to be called whenever the user wishes to delete in entry. Dependent on the table that has been selected from, the appropriate ID is retrieved, and is used to delete that entry. The function keeps referential integrity, and the function informs the user what is needed to be deleted in order to delete what they had desired to delete. This can be seen in line 81 of the function.

### 4.3.3 Function - Generating an update window

Lines 571-675 of Module 2, subsection 4.10.2. Generating and update window after having selected an entry.

```
1       def UpdateEntry(self): #getting the update input
2           self.Selection = False
3           self.EditWindow = dbAddItemWindow() #uses the add window to init same interface but fill boxes with
                  data
4           self.EditWindow.setFixedSize(360, 200)
5           self.EditWindow.AddType = self.CurrentTable
```

```
        self.EditWindow.AnswerButtons()
        self.EditWindow.ReadyToVerify = False
        self.EditWindow.originalItemList = []

        if self.CurrentTable == "Book":
            self.EditWindow.sql = "select * from Book"
            self.ViewWindow.SelectedRow = self.ViewWindow.table.currentRow()
            if self.ViewWindow.SelectedRow != -1:
                self.Selection = True
                with sqlite3.connect("PP.db") as db:
                    cursor = db.cursor() #fetching data from database for the edit window
                    sql = "select * from Book where ISBN =
                          {}".format(self.ViewWindow.table.item(self.ViewWindow.SelectedRow, 0).text())
                    cursor.execute(sql)
                    self.EditWindow.originalItemList = list(cursor.fetchone())

        elif self.CurrentTable == "PubInvoice":
            self.EditWindow.setFixedSize(400,150)
            self.EditWindow.selectedISBN = self.SelectedISBN
            self.EditWindow.sql = "select ISBN, AuthorID, PubInvoiceDate, PubInvoiceService,
                  PubInvoicePayment from PubInvoice"
            self.PubInvoiceWindow.SelectedRow = self.PubInvoiceWindow.table.currentRow()
            if self.PubInvoiceWindow.SelectedRow != -1:
                self.Selection = True
                with sqlite3.connect("PP.db") as db:
                    cursor = db.cursor() #fetching data from database for the edit window
                    sql = "select ISBN, AuthorID, PubInvoiceDate, PubInvoiceService, PubInvoicePayment from
                          PubInvoice where PubInvoiceID =
                          {}".format(self.PubInvoiceWindow.table.item(self.PubInvoiceWindow.SelectedRow,
                          0).text())
                    cursor.execute(sql)
                    self.EditWindow.originalItemList = list(cursor.fetchone())

        elif self.CurrentTable == "BookInvoice":
            self.EditWindow.setFixedSize(350, 100)
            self.EditWindow.selectedID = self.SelectedID
            self.EditWindow.sql = "select AuthorID, BookInvoiceDate from BookInvoice"
            self.BookInvoiceWindow.SelectedRow = self.BookInvoiceWindow.table.currentRow()
            if self.BookInvoiceWindow.SelectedRow != -1:
                self.Selection = True
                with sqlite3.connect("PP.db") as db:
                    cursor = db.cursor() #fetching data from database for the edit window
                    sql = "select AuthorID, BookInvoiceDate from BookInvoice where BookInvoiceID =
                          {}".format(self.BookInvoiceWindow.table.item(self.BookInvoiceWindow.SelectedRow,
                          0).text())
                    cursor.execute(sql)
                    self.EditWindow.originalItemList = list(cursor.fetchone())

        elif self.CurrentTable == "Royalties":
            self.EditWindow.setFixedSize(350, 100)
            self.EditWindow.selectedID = self.SelectedID
            self.EditWindow.sql = "select AuthorID, RoyaltiesDate from Royalties"
            self.RoyaltiesWindow.SelectedRow = self.RoyaltiesWindow.table.currentRow()
            if self.RoyaltiesWindow.SelectedRow != -1:
                self.Selection = True
                with sqlite3.connect("PP.db") as db:
                    cursor = db.cursor() #fetching data from database for the edit window
                    sql = "select AuthorID, RoyaltiesDate from Royalties where RoyaltiesID =
                          {}".format(self.RoyaltiesWindow.table.item(self.RoyaltiesWindow.SelectedRow,
                          0).text())
                    cursor.execute(sql)
                    self.EditWindow.originalItemList = list(cursor.fetchone())

        elif self.CurrentTable == "BookInvoiceItems":
            self.EditWindow.setFixedSize(450, 150)
            self.EditWindow.selectedID = self.SelectedID
            self.Editing = True
            self.EditWindow.btnCalculate.clicked.connect(self.BookInvoiceItemCalculation)
            self.EditWindow.selectedISBN = self.SelectedISBN
            self.EditWindow.sql = "select BookInvoiceID, ISBN, BookInvoiceQuantity, BookInvoiceDiscount,
                  ShippingType, ShippingPrice from BookInvoiceItems"
            self.BookInvoiceItemsWindow.SelectedRow = self.BookInvoiceItemsWindow.table.currentRow()
            if self.BookInvoiceItemsWindow.SelectedRow != -1:
                self.Selection = True
                with sqlite3.connect("PP.db") as db:
                    cursor = db.cursor() #fetching data from database for the edit window
                    sql = "select BookInvoiceID, ISBN, BookInvoiceQuantity, BookInvoiceDiscount,
                          ShippingType, ShippingPrice from BookInvoiceItems where BookInvoiceItemsID =
                          {}".format(self.BookInvoiceItemsWindow.table.item(self.BookInvoiceItemsWindow.SelectedRow,
                          0).text())
                    cursor.execute(sql)
                    self.EditWindow.originalItemList = list(cursor.fetchone())

        elif self.CurrentTable == "RoyaltyItems":
            self.EditWindow.setFixedSize(450, 250)
            self.EditWindow.selectedID = self.SelectedID
            self.Editing = True
            self.EditWindow.btnCalculate.clicked.connect(self.RoyaltyItemCalculation)
            self.EditWindow.selectedISBN = self.SelectedISBN
            self.EditWindow.sql = "select RoyaltiesID, ISBN, Currency, RoyaltyDiscount, WholesalePrice,
                  RoyaltyQuantity, PrintCost, ExcRateFromGBP from RoyaltyItems"
```

```
            self.RoyaltyItemsWindow.SelectedRow = self.RoyaltyItemsWindow.table.currentRow()
            if self.RoyaltyItemsWindow.SelectedRow != -1:
                self.Selection = True
                with sqlite3.connect("PP.db") as db:
                    cursor = db.cursor() #fetching data from database for the edit window
                    sql = "select RoyaltiesID, ISBN, Currency, RoyaltyDiscount, WholesalePrice,
                          RoyaltyQuantity, PrintCost, ExcRateFromGBP from RoyaltyItems where
                          RoyaltyItemsID =
                          {}".format(self.RoyaltyItemsWindow.table.item(self.RoyaltyItemsWindow.SelectedRow,
                          0).text())
                    cursor.execute(sql)
                    self.EditWindow.originalItemList = list(cursor.fetchone())

        self.EditWindow.Editing = True

        if self.Selection == True: #initialising the edit window
            self.EditWindow.btnConfirm.clicked.connect(self.EditWindow.Validate)
            if self.CurrentTable in ["RoyaltyItems", "BookInvoiceItems"]:
                self.EditWindow.btnConfirm.clicked.connect(self.EditWindow.CheckCalculated)
            else:
                self.EditWindow.btnConfirm.clicked.connect(self.VerifyUpdate)
            self.EditWindow.AddType = self.CurrentTable
            self.EditWindow.selectedID = self.SelectedID

            self.EditWindow.CalendarWidget = dbCalendarWidget()
            self.EditWindow.CalendarWidget.Calendar()
            self.EditWindow.initAddItemWindow()
```

This uses the basis of the add entry layout to create an update entry layout, as the required layout is using the feature of adding entries, but filling in the data in the appropriate line edits. Depending on the entity which the user is updating in, an SQL statement is defined to fetch the data that has been selected using the selected row to find the primary key. After fetching the data, it instantiates the class and sets the values as an attribute of the window, so that the window can display the values in the respective line edits.

## 4.4 Variable Listing

The data dictionary can be found in the Design Section 2.6.4.

| **Variable Name** | **Purpose/Usage** | **Locations in code** |
|---|---|---|
| self.Username | The current username in the database, used for login, verification if the user wishes to update/delete data, if the user has forgotten their password and needs to type their username for their password to be emailed to them, or for changing the username/-password. | Module 1: Line 108, 113, 114, 125, 148, 152, 160, 185 Module 2: Line 30, 225, 265, 562, 1063. |
| self.Password | The current password in the database, used for login, verification if the user wishes to update/delete data, if the user has forgotten their password and needs to type their username for their password to be emailed to them, or for changing the username/-password. | Module 1: Line 110, 115, 151 Module 11: Line 65, 68 69. |
| self.CurrentTable | Used to hold the name of the current table that is being displayed | Module 2: Line 44, 155, 157, 161, 163, 326, 356, 364, 379, 384, 400, 405, 426, 432, 447, 450, 471, 774, 909 |
| self.SelectedID | Used to hold the primary key of the current table. | Module 2: Line 272, 280, 289, 298, 306, 314, 326, 330, 350, 367, 410, 424, 455, 469, 558, 707, 715, 722, 729, 736, 743, 766, 768, 770, 774, 853, 859, 868, 881, 886, 895 |
| self.SelectedAuthorID | The Author ID of the current selected Author, used for Adding, Updating, Deleting and Searching for data, using it as the unique identifier. | Module 2: Line 267, 330, 355, 368, 387, 408, 435, 549, 551, 557, 797, 801, 818, 827, 831, 840, 844 |

| | | |
|---|---|---|
| self.input_data | Data that is ready to be added to the database | Module 2: Line 168, 201, 204, 206, 212 |
| self.Firstname | Used to hold firstnames of customers, mainly for displaying names instead of ID's. | Module 2: Line 228, 230, 351 |
| self.Lastname | Used to hold lastnames of customers, mainly for displaying names instead of ID's. | Module 2: Line 229, 230, 352 |
| self.ISBNDeletion | Used to delete certain Royalty Items and Invoice Items when deleting all of a customer's records. | Module 2: Line 241, 242, 243, 245 |
| self.Results | Holds relevant ID numbers of search results | Module 2: Line 946, 949, 951, 959, 974, 977, 979, 988, 991, 993, 1003, 1016, 1026<br>Module 14: Line 13 |
| self.SelectedRow | Used to find the row of the table that has been selected. | Module 2: Line 223, 226, 228, 229, 271, 272, 278, 280, 287, 289, 296, 298, 304, 306, 312, 314, 349, 350, 351, 352, 548, 549 |

## 4.5 System Evidence

### 4.5.1 User Interface

Figure 4.1: Login Screen

Figure 4.2: Forgot Password Screen

Figure 4.3: Main Menu

Figure 4.4: Add Customer Entry

Figure 4.5: Update Customer Entry

Figure 4.6: Editing Customer Field

Figure 4.7: Update Customer Entry Verification

Figure 4.8: Update Customer Entry Success

Figure 4.9: Delete Customer Entry

Figure 4.10: Delete Entry Success

Figure 4.11: View Menu

Figure 4.12: View Publishing Invoices

Figure 4.13: Add Publishing Invoice

Figure 4.14: Update Publishing Invoice

Figure 4.15: Update Verification

Figure 4.16: Update Success

Figure 4.17: Delete Verification

Figure 4.18: Delete Success

Figure 4.19: View Book Invoices

Figure 4.20: Add Book Invoice

Figure 4.21: Update Book Invoice

Figure 4.22: View Book Invoice Items

Figure 4.23: Add Book Invoice Items

Figure 4.24: Update Book Invoice Items

Figure 4.25: View Royalties

Figure 4.26: Add Royalties

Figure 4.27: Update Royalties

Figure 4.28: View Royalty Items

Figure 4.29: Add Royalty Items

Figure 4.30: Update Royalty Items

Figure 4.31: Add Book

Figure 4.32: Update Book

Figure 4.33: Quick Search

Figure 4.34: Selection For Changing User Credentials

Figure 4.35: Change Username

Figure 4.36: Change Username Success

Figure 4.37: Change Password

Figure 4.38: Change Password Success

Figure 4.39: Search

Figure 4.40: Search with a different selection to Author

Figure 4.41: No Match Found For Search

Figure 4.42: Search Results Example

### 4.5.2 ER Diagram

Figure 4.43: ER Diagram

### 4.5.3 Database Table Views

Figure 4.44: Customer

| | AuthorID | FirstName | LastName | Email | PhoneNumber | Address | Postcode |
|---|---|---|---|---|---|---|---|
| 1 | 1 | John | Smith | Example@mail.com | 07123456789 | 1 Example Road | AB1 2CD |
| 2 | 2 | Sarah | Taylor | Example2@mail.com | 07123789456 | 2 Example Road | AB1 2CD |
| 3 | 3 | Amanda | Clarke | Example3@mail.com | 07987654321 | 1 Example Way | EF3 4GH |
| 4 | 4 | Richard | Wilson | Example4@mail.com | 07789123456 | 1 Example Aven... | IJ5 6KL |
| 5 | 5 | Jack | Parker | Example5@mail.com | 07789456123 | 123 Example Str... | MN7 8OP |
| 6 | 7 | Test | Data | testdata@mail.com | 07111222333 | 1 Test Road | ZY0 9XW |

Figure 4.45: Book

| | ISBN | AuthorID | BookTitle | NoOfPages | Size | Back | Cover | Paper | Font | FontSize | DatePublished | Price |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1234567891234 | 1 | Autobiography of John Smith | 250 | Small | Soft | Gloss | White | Times New Ro... | 12 | 25-02-2015 | 8.99 |
| 2 | 1234567890123 | 1 | Example Book | 180 | Large | Hard | Matte | Creme | Arial | 11 | 26-02-2015 | 12.99 |
| 3 | 3210987654321 | 5 | Learning SQL | 150 | Large | Soft | Gloss | White | Calibri | 11 | 23-02-2015 | 7.99 |
| 4 | 6546546540654 | 5 | Another Example | 299 | Small | Soft | Matte | Creme | Microsoft Sans ... | 10 | 10-12-2014 | 9.99 |
| 5 | 9879879870987 | 4 | An Entry | 199 | Large | Soft | Matte | White | Calibri | 12 | 26-02-2015 | 7.99 |
| 6 | 9781234560123 | 3 | Testing | 300 | Large | Soft | Gloss | White | Calibri | 11 | 27-02-2015 | 8.99 |
| 7 | 9876222555999 | 7 | Tested | 101 | Small | Soft | Gloss | Creme | Arial Narro | 12 | 20-11-2014 | 11.99 |

Figure 4.46: Publishing Invoice

Figure 4.47: Book Invoice

Figure 4.48: Book Invoice Items

| | BookInvoiceItemsID | BookInvoiceID | ISBN | BookInvoiceQuantity | ookInvoiceDiscou | ShippingType | ShippingPrice |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 3210987654321 | 25 | 40 | Premium | 5.25 |
| 2 | 5 | 1 | 3210987654321 | 10 | 40 | Rush | 8.99 |
| 3 | 6 | 1 | 6546546540654 | 30 | 40 | Economy | 2.99 |
| 4 | 7 | 4 | 9876222555999 | 51 | 50 | Rush | 6.32 |

Figure 4.49: Royalties

| | RoyaltiesID | AuthorID | RoyaltiesDate | RoyaltyPayment |
|---|---|---|---|---|
| 1 | 1 | 5 | 25-02-2015 | £176.28 |
| 2 | 2 | 1 | 18-03-2015 | £23.00 |

Figure 4.50: Royalty Items

| | RoyaltyItemsID | RoyaltiesID | ISBN | Currency | RoyaltyDiscount | WholesalePrice | RoyaltyQuantity | NetSales | PrintCost | ExcRateFromGBP |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 3210987654321 | £ | 40 | 4.79 | 64 | 306.56 | 145 | N/A |
| 2 | 2 | 1 | 3210987654321 | $ | 40 | 9.59 | 19 | 182.21 | 43.75 | 1.55 |
| 3 | 3 | 2 | 1234567890123 | £ | 10 | 10 | 10 | 100 | 77 | N/A |

### 4.5.4 Database SQL

The following SQL statements can be found in Module 1, subsection 4.10.1

Creating Customer Table:

```
CREATE TABLE Customer
                  (AuthorID integer,
                  FirstName text,
                  LastName text,
                  Email text,
                  PhoneNumber text,
                  Address text,
                  Postcode text,
                  primary key(AuthorID))
```

Creating Book Table:

```
CREATE TABLE Book
                  (ISBN text,
                  AuthorID integer,
                  BookTitle text,
                  NoOfPages integer,
                  Size text,
                  Back text,
                  Cover text,
                  Paper text,
                  Font text,
                  FontSize real,
```

```
                    DatePublished date,
                    Price real,
                    primary key(ISBN),
                    foreign key(AuthorID) references
                        Customer(AuthorID))
```

Creating Publishing Invoice Table:

```
CREATE TABLE PubInvoice
                    (PubInvoiceID integer,
                    ISBN text,
                    AuthorID integer,
                    PubInvoiceDate date,
                    PubInvoiceService text,
                    PubInvoicePayment real,
                    primary key(PubInvoiceID),
                    foreign key(AuthorID) references
                        Customer(AuthorID),
                    foreign key(ISBN) references
                        Book(ISBN))
```

Creating Book Invoice Table:

```
CREATE TABLE BookInvoice
                    (BookInvoiceID integer,
                    AuthorID integer,
                    BookInvoiceDate date,
                    BookInvoicePayment real,
                    primary key(BookInvoiceID),
                    foreign key(AuthorID) references
                        Customer(AuthorID))
```

Creating Book Invoice Items Table:

```
CREATE TABLE BookInvoiceItems
                    (BookInvoiceItemsID integer,
                    BookInvoiceID integer,
                    ISBN text,
                    BookInvoiceQuantity integer,
                    BookInvoiceDiscount real,
                    ShippingType text,
                    ShippingPrice real,
                    primary key(BookInvoiceItemsID),
                    foreign key(BookInvoiceID)
                        references
                        BookInvoice(BookInvoiceID),
```

```
                    foreign key(ISBN) references
                        Book(ISBN))
```

Creating Royalties Table:

```
CREATE TABLE Royalties
                    (RoyaltiesID integer,
                    AuthorID integer,
                    RoyaltiesDate date,
                    RoyaltyPayment real,
                    primary key(RoyaltiesID),
                    foreign key(AuthorID) references
                        Customer(AuthorID))
```

Creating Royalty Items Table:

```
CREATE TABLE RoyaltyItems
                    (RoyaltyItemsID integer,
                    RoyaltiesID integer,
                    ISBN text,
                    Currency text,
                    RoyaltyDiscount real,
                    WholesalePrice real,
                    RoyaltyQuantity integer,
                    NetSales real,
                    PrintCost real,
                    ExcRateFromGBP real,
                    primary key(RoyaltyItemsID),
                    foreign key(RoyaltiesID) references
                        Royalties(RoyaltiesID),
                    foreign key(ISBN) references
                        Book(ISBN))
```

Creating a Login Details Table if it does not exist, for login purposes:

```
CREATE TABLE if not exists LoginDetails
                    (Username text
                    Password text)
```

### 4.5.5 SQL Queries

Checking whether the login table is existent:

```
select name
from sqlite_master
```

```
where type='table' and name='LoginDetails'
```

Reference: Module 1, subsection 4.10.1, line 35.

Creating the default Username and Password.

```
insert into LoginDetails (Username, Password)
values (?, ?)
```

Reference: Module 1, subsection 4.10.1, line 47.

Fetching the current username:

```
select Username
from LoginDetails
```

Reference: Module 1, subsection 4.10.1, line 108.

Fetching the current password:

```
select Password
from LoginDetails
```

Reference: Module 1, subsection 4.10.1, line 110.

Fetching all Customer data:

```
select *
from Customer
```

Reference: Module 2, subsection 4.10.2, line 38.

Fetching all Book data:

```
select *
from Book
```

Reference: Module 2, subsection 4.10.2, line 111.

Selecting data to be entered by user for PubInvoice:

```
select ISBN, AuthorID, PubInvoiceDate,
    PubInvoiceService, PubInvoicePayment
from PubInvoice
```

Reference: Module 2, subsection 4.10.2, line 115.

Selecting data to be entered by user for BookInvoice:

```
select AuthorID, BookInvoiceDate
from BookInvoice
```

Reference: Module 2, subsection 4.10.2, line 119.

Selecting data to be entered by user for Royalties:

```
select AuthorID, RoyaltiesDate
from Royalties
```

Reference: Module 2, subsection 4.10.2, line 123.

Selecting data to be entered by user for BookInvoiceItems:

```
select BookInvoiceID, ISBN, BookInvoiceQuantity,
    BookInvoiceDiscount, ShippingType, ShippingPrice
from BookInvoiceItems
```

Reference: Module 2, subsection 4.10.2, line 127.

Selecting data to be entered by user for RoyaltyItems:

```
select RoyaltiesID, ISBN, Currency, RoyaltyDiscount,
    WholesalePrice, RoyaltyQuantity, PrintCost,
    ExcRateFromGBP
from BookInvoiceItems
```

Reference: Module 2, subsection 4.10.2, line 134.

The following code compiles an SQL string for adding data, depending on which table is chosen:

```
        self.input_data = []
        if self.CurrentTable == "Book":
            self.TableValues = "Book (ISBN, AuthorID, BookTitle, NoOfPages, Size, Back, Cover, Paper, Font,
                  FontSize, DatePublished, Price)"
            self.Placeholders = "(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)"
            self.NoOfEntries = 12

        elif self.CurrentTable == "PubInvoice":
            self.TableValues = "PubInvoice (ISBN, AuthorID, PubInvoiceDate, PubInvoiceService,
                  PubInvoicePayment)"
            self.Placeholders = "(?, ?, ?, ?, ?)"
            self.NoOfEntries = 5

        elif self.CurrentTable == "BookInvoice":
            self.TableValues = "BookInvoice (AuthorID, BookInvoiceDate)"
            self.Placeholders = "(?, ?)"
            self.NoOfEntries = 2

        elif self.CurrentTable == "Royalties":
            self.TableValues = "Royalties (AuthorID, RoyaltiesDate)"
            self.Placeholders = "(?, ?)"
            self.NoOfEntries = 2

        elif self.CurrentTable == "BookInvoiceItems":
            self.TableValues = "BookInvoiceItems (BookInvoiceID, ISBN, BookInvoiceQuantity,
                  BookInvoiceDiscount, ShippingType, ShippingPrice)"
            self.Placeholders = "(?, ?, ?, ?, ?, ?)"
            self.NoOfEntries = 6

        elif self.CurrentTable == "RoyaltyItems":
            self.TableValues = "RoyaltyItems (RoyaltiesID, ISBN, Currency, RoyaltyDiscount, WholesalePrice,
                  RoyaltyQuantity, PrintCost, ExcRateFromGBP, NetSales)"
            self.Placeholders = "(?, ?, ?, ?, ?, ?, ?, ?, ?)"
            self.NoOfEntries = 9

        for count in range(0, self.NoOfEntries):
            try: #putting the input data in a list
                self.input_data.append(str(self.AddWindow.inputList[count].currentText()))
            except:
                if count == 8 and self.CurrentTable == "RoyaltyItems":
                    self.input_data.append(self.NetSales) #Net sales are calculated, as opposed to being
                          entered
                else:
                    self.input_data.append(self.AddWindow.inputList[count].text())

        with sqlite3.connect("PP.db") as db:
            cursor = db.cursor()
            cursor.execute("PRAGMA foreign_keys = ON")
            self.sql = "insert into {} values {}".format(self.TableValues, self.Placeholders)
            cursor.execute(self.sql, self.input_data)
            db.commit()
```

Reference: Module 2, subsection 4.10.2, line 168-214. Line 212 is the base sql string, with the Table and values, and placeholders inserted dependent on the current table.

Selecting all books' ISBNs belonging to an Author with the matching selected ID:

```
                    cursor.execute(sql)
```

Reference: Module 2, subsection 4.10.2, line 240.

Deleting all data linked with an author, in a certain order to maintain referential integrity:

```
                        cursor.execute(sql)
                        sql = "delete from RoyaltyItems where ISBN =
                              {}".format(list(list(self.ISBNDeletion)[count])[0])
                        cursor.execute(sql)
                    sql = "delete from PubInvoice where AuthorID =
                          {}".format(self.ConfirmDialog.SelectedAuthorID)
                    cursor.execute(sql)
                    sql = "delete from BookInvoice where AuthorID =
                          {}".format(self.ConfirmDialog.SelectedAuthorID)
                    cursor.execute(sql)
                    sql = "delete from Royalties where AuthorID =
                          {}".format(self.ConfirmDialog.SelectedAuthorID)
```

```
9                   cursor.execute(sql)
10                  sql = "delete from Book where AuthorID = {}".format(self.ConfirmDialog.SelectedAuthorID)
11                  cursor.execute(sql)
12                  sql = "delete from Customer where AuthorID =
                        {}".format(self.ConfirmDialog.SelectedAuthorID)
13                  cursor.execute(sql)
```

Reference: Module 2, subsection 4.10.2, lines 244-256.

Deleting a selected entry:

```
1                   cursor.execute(sql)
```

Reference: Module 2, subsection 4.10.2, line 327.

Fetching the price of a selected book:

```
1                   self.Price =
                        list(cursor.fetchone())[0]
```

Reference: Module 2, subsection 4.10.2, line 492.

Fetching the existing customer details of the selected customer so that they can be displayed and edited:

```
1               self.UpdateEntryWindow.table.sql =
                    "select {} from Customer where
                    AuthorID = {}".format(self.selectText,
                    self.SelectedAuthorID)
2               self.SelectedID = self.SelectedAuthorID
```

Reference: Module 2, subsection 4.10.2, lines 557-558.

Fetching the existing book details of the selected book so that they can be displayed and edited:

```
1                   cursor.execute(sql)
```

Reference: Module 2, subsection 4.10.2, line 588.

Fetching the existing publishing invoice details of the selected publishing invoice so that they can be displayed and edited:

```
1                   cursor.execute(sql)
```

Reference: Module 2, subsection 4.10.2, line 601.

Fetching the existing book invoice details of the selected book invoice so that they can be displayed and edited:

```
                cursor.execute(sql)
```

Reference: Module 2, subsection 4.10.2, line 614.

Fetching the existing royalty details of the selected royalty payment so that they can be displayed and edited:

```
            self.RoyaltiesWindow.SelectedRow =
                self.RoyaltiesWindow.table.currentRow()
```

Reference: Module 2, subsection 4.10.2, line 627.

Fetching the existing book invoice items details of the selected book invoice items so that they can be displayed and edited:

```
                cursor.execute(sql)
```

Reference: Module 2, subsection 4.10.2, line 643.

Fetching the existing royalty items details of the selected royalty items so that they can be displayed and edited:

```
                cursor.execute(sql)
```

Reference: Module 2, subsection 4.10.2, line 659.

Updating the ISBN in all other entities if it has been changed by the user.

```
                        cursor.execute(sql)
                        sql = "update BookInvoiceItems
                            set ISBN = {0} where ISBN =
                            {1}".format(self.UpdateList[0],
                            self.SelectedID)
                        cursor.execute(sql)
                        sql = "update RoyaltyItems set
                            ISBN = {0} where ISBN =
                            {1}".format(self.UpdateList[0],
                            self.SelectedID)
                        cursor.execute(sql)
```

Reference: Module 2, subsection 4.10.2, lines 767-771.

Updating the the selected table with the inputs the user has given. The string is partially made using the iteration in lines 758-761

```
1            self.Update += "{} = '{}'".format(list(self.Headers[count])[0], self.UpdateList[count])
2            if count != len(self.UpdateList) - 1:
3                self.Update += ", "
```

The update string:

```
1                cursor.execute(sql)
```

Reference: Module 2, subsection 4.10.2, lines 758-761, and 775.

Fetching the Book Title, Firstname and Lastname of the Customer and their book, using the Book invoice ID and Book invoice items ID to find the matching Author ID and ISBNs.

```
1                    self.Title = list(cursor.fetchone())
```

Reference: Module 2, subsection 4.10.2, line 869

Fetching the Book Title, Firstname and Lastname of the Customer and their book, using the Royalties ID and Royalty items ID to find the matching Author ID and ISBNs.

```
1                    self.Title = list(cursor.fetchone())
```

Reference: Module 2, subsection 4.10.2, line 896

Fetching all Customers that match the firstname or lastname entered.

```
1                elif len(self.QSText) == 0:
```

Reference: Module 2, subsection 4.10.2, line 923

Inserting data into the customer table:

```
1 insert into Customer
2 (FirstName, LastName, Email, PhoneNumber, Address,
    Postcode)
3 values (?, ?, ?, ?, ?, ?)
```

Reference: Module 6, subsection 4.10.6, line 83

Changing the Password.

```
cursor.execute("update LoginDetails set Password = '{}' where Username =
      '{}'".format(self.Password, self.Username))
```

Reference: Module 18, subsection 4.10.18, line 61

Changing the Username.

```
cursor.execute("update LoginDetails
    set Username = '{}' where
    Username =
    '{}'".format(self.NewUsername,
    self.Username))
```

Reference: Module 17, subsection 4.10.17, line 61

Selecting the Quantity, Discount, Shipping Price and Price from Book Invoice Items and Book, to conduct calculations.

```
Selection = "BookInvoiceItems.BookInvoiceQuantity,
      BookInvoiceItems.BookInvoiceDiscount, BookInvoiceItems.ShippingPrice,
      Book.Price"
Tables = "BookInvoiceItems, Book"
sql = "select {} from {} where BookInvoiceItems.BookInvoiceID = {} and
      BookInvoiceItems.BookInvoiceItemsID = {} and BookInvoiceItems.ISBN = {} and
      Book.ISBN = BookInvoiceItems.ISBN".format(Selection, Tables,
      self.selectedID, count+1, list(self.ISBNs[count2])[0])
```

Reference: Module 12, subsection 4.10.12, lines 90-92

Updating the Book Invoice Payment with the newly calculated payment total.

```
sql = "update BookInvoice set
    BookInvoicePayment = '{}' where
    BookInvoiceID =
    {}".format(self.BookInvoicePayment,
    self.selectedID)
```

Reference: Module 12, subsection 4.10.12, line 121

Selecting the Currency, Net sales, Exchange Rate from GBP, Quantity, No Of Pages, Size, Cover type and back type from Royalty Items and Book, to conduct calculations to find the print cost, in order to also find the Royalty Payment.

```
Selection = "RoyaltyItems.Currency, RoyaltyItems.NetSales,
      RoyaltyItems.ExcRateFromGBP, RoyaltyItems.RoyaltyQuantity, Book.NoOfPages,
      Book.Size, Book.Cover, Book.Back"
Tables = "RoyaltyItems, Book"
sql = "select {} from {} where RoyaltyItems.RoyaltiesID = {} and
      RoyaltyItems.RoyaltyItemsID = {} and RoyaltyItems.ISBN = {} and Book.ISBN =
      RoyaltyItems.ISBN".format(Selection, Tables, self.selectedID, count+1,
      list(self.ISBNs[count2])[0])
```

Reference: Module 12, subsection 4.10.12, lines 152-154

Updating the Royalty Payment with the newly calculated payment total.

```
1                   sql = "update Royalties set
                        RoyaltyPayment = '{}' where
                        RoyaltiesID =
                        {}".format(self.RoyaltyPayment,
                        self.selectedID)
```

Reference: Module 12, subsection 4.10.12, line 200

Finding the ID's of the matching results of a detailed search, in order to fetch the necessary data for display.

```
1                   self.sql = "select Customer.AuthorID,
                        Book.AuthorID, Book.ISBN,
                        {0}.ISBN, {0}ID from Customer,
                        Book, {0} where
                        (Customer.Firstname like '{1}%' or
                        Customer.Lastname like '{2}%') and
                        {0}.{3} like '{4}%' and
                        Customer.AuthorID = Book.AuthorID
                        and Book.ISBN =
                        {0}.ISBN".format(self.Table,
                        self.Firstname, self.Lastname,
                        self.Category, self.Search)
```

Reference: Module 14, subsection 4.10.14, line 147

Finding the AuthorID of the Customers which match the entered first and last names, in order to fetch the necessary data for display.

```
1               self.sql = "select AuthorID from Customer
                    where Firstname like '{0}%' or
                    Lastname like
                    '{1}%'".format(self.Firstname,
                    self.Lastname)
```

Reference: Module 14, subsection 4.10.14, line 158

Updating the Customer's Details with the inputs given. Line 91 contains the base string.

```
    def UpdateChanges(self):
        UL = [] #UL = Update List
        for count in range(0, 6):
            UL.append(self.table.item(0,
                count).text())
        self.Update = "Firstname = '{}', Lastname =
            '{}', Email = '{}', Phonenumber = '{}',
            Address = '{}', Postcode =
            '{}'".format(UL[0], UL[1], UL[2], UL[3],
            UL[4], UL[5])
        with sqlite3.connect("PP.db") as db:
            cursor = db.cursor()
            cursor.execute("PRAGMA foreign_keys = ON")
            sql = "update {} set {} where {} =
                {}".format(self.TableName,
                self.Update, self.ID, self.selectedID)
            cursor.execute(sql)
            db.commit()
```

Reference: Module 7, subsection 4.10.7, lines 83-93

## 4.6 Testing

### 4.6.1 Summary of Results

My testing proved that my program was robust, as it did not crash during the tests, which is the main strength of my tests. For example, when the user entered erroneous data, the system would prompt the user about the invalid entries and prevent a crash. This can be seen in test results 3.1 on page 173, 3.2 on page 174 and 3.3 on page 175 where the fields are left blank. Also, validators were used to prevent the user from being able to enter some erroneous data. This is seen in test results 2.3, 2.4, 2.5, and 2.6 on page 3.25, and 2.7 on page 169. However, where erroneous data still could be entered, the user was prevented with a prompt, which can be seen on test results 2.6 and 2.8 on pages 168 and 170 respectively. This shows that the program was seemingly good at handling invalid entries and preventing crashes, meaning it is rather robust.

The main weakness of my testing was that not every component of the interface was tested. This is because I tested some of the components, and gained the assumption that the rest work fine, because they have the same functionality. For example, I only tested two Cancel buttons in tests 1.25 and 1.26. As the rest of the Cancel buttons in the system have the same code and functionality,

only a few tests were necessary, but the reliabiltity of every cancel button in this example is based on the two in the tests.

### 4.6.2 Known Issues

A minor problem was encountered during the testing, where if the user entered an invalid value for an entry, proceeded to calculate using their inputs they have entered and confirm that they wish to add it to the database, the user would be correctly prompted with an error. However, upon dismissing the error, the window that they were using to add data to the database would close, forcing them to reopen the window and start entering data again. I have a vague idea of where the problems are (lines 500-518 for Book Invoice items, and lines 531-545 for Royalty Items of Module two, subsection 4.10.2, and I have somewhat of an idea how to fix it. I have the assumption that a connection is made with the Confirm button, so that whether the entry is invalid or not, the window will still close because of the connection. This could possibly be fixed by closing the connection if the entry is invalid, and hardcoding the acceptance of the window if all entries are fine. Despite this minor bug, invalid entries for the items are still prevented from being added to the database.

## 4.7 Code Explanations

### 4.7.1 Difficult Sections

The following function from Module 12, subsection 4.10.12, calculates the full payment for all the book invoice items in a single invoice.

```
1       def CalculateBookInvoiceItems(self):
2           with sqlite3.connect("PP.db") as db:
3               cursor = db.cursor()
4               cursor.execute("PRAGMA foreign_keys_ = ON")
5               self.Empty = False
6               self.BookInvoicePayment = 0
7
8               self.IDList = []
9               sql = "select BookInvoiceItemsID from BookInvoiceItems where BookInvoiceID =
                      {}".format(self.selectedID)
10              cursor.execute(sql)
11              self.IDListTuple = list(cursor.fetchall())
12              for count in range(0, len(self.IDListTuple)): #conversion from tuple
13                  self.IDList.append(list(self.IDListTuple[count])[0])
14
15              self.currentID = 0
16              for count in range(0, len(self.IDList)):
17                  if self.currentID < self.IDList[count]: #finding biggest ID no.
18                      self.currentID = self.IDList[count]
19
20              sql = "select ISBN from BookInvoiceItems where BookInvoiceID = {}".format(self.selectedID)
21              cursor.execute(sql)
22              self.ISBNs = list(cursor.fetchall())
23
24              for count2 in range(0, len(self.ISBNs)):
25                  if self.currentID != 0:
26                      for count in range(0, self.currentID +1):
27                          self.Empty = False
28                          Selection = "BookInvoiceItems.BookInvoiceQuantity,
                                  BookInvoiceItems.BookInvoiceDiscount, BookInvoiceItems.ShippingPrice,
                                  Book.Price"
29                          Tables = "BookInvoiceItems, Book"
```

```
30                          sql = "select {} from {} where BookInvoiceItems.BookInvoiceID = {} and
                                  BookInvoiceItems.BookInvoiceItemsID = {} and BookInvoiceItems.ISBN = {} and
                                  Book.ISBN = BookInvoiceItems.ISBN".format(Selection, Tables,
                                  self.selectedID, count+1, list(self.ISBNs[count2])[0])
31                          cursor.execute(sql)
32                          try:
33                              self.SelectionList = list(cursor.fetchone())
34                          except:
35                              self.Empty = True
36
37                          if self.Empty != True:
38                              self.Quantity = self.SelectionList[0]
39                              self.Discount = self.SelectionList[1] / 100
40                              self.ShippingPrice = self.SelectionList[2]
41                              self.Price = self.SelectionList[3]
42
43                              self.TempPayment = (self.Quantity * self.Price)
44                              self.Discount *= self.TempPayment
45                              self.TempPayment -= self.Discount
46                              self.TempPayment += self.ShippingPrice
47
48                              self.BookInvoicePayment += self.TempPayment
49
50
51                      elif self.currentID == 0:
52                          self.BookInvoicePayment = None
53                          sql = "update BookInvoice set BookInvoicePayment = '{}' where BookInvoiceID =
                                  {}".format(self.BookInvoicePayment, self.selectedID)
54                          cursor.execute(sql)
55                          db.commit()
56
57                  if self.currentID != 0:
58                      self.BookInvoicePayment = "£{0:.2f}".format(self.BookInvoicePayment)
59                      sql = "update BookInvoice set BookInvoicePayment = '{}' where BookInvoiceID =
                                {}".format(self.BookInvoicePayment, self.selectedID)
60                      cursor.execute(sql)
61                      db.commit()
```

Lines 74 to 80 find the biggest ID no., so that the program can loop through each item ID, accummalating the total payment for each ID of each item that is in the invoice. Line 92 defines the sql statement to fetch all the necessary data to conduct the calculations. Then, upto line 110, the program is checking whether the set of items is in the invoice, and then calculates the payment. The payment for the single set items is then added on to the current total, which starts at £0. Lines 119-123 format the full final payment into a string, and update the payment in the invoice entry with the newly calculated payment.

The following function from Module 12, subsection 4.10.12, calculates the full payment for all the royalty items in a single payment.

```
1       def CalculateRoyaltyItems(self):
2           with sqlite3.connect("PP.db") as db:
3               cursor = db.cursor()
4               cursor.execute("PRAGMA foreign_keys = ON")
5               self.Empty = False
6               self.RoyaltyPayment = 0
7               self.Currency = ""
8               self.IDList = []
9               sql = "select RoyaltyItemsID from RoyaltyItems where RoyaltiesID = {}".format(self.selectedID)
10              cursor.execute(sql)
11              self.IDListTuple = list(cursor.fetchall())
12              for count in range(0, len(self.IDListTuple)): #conversion from tuple
13                  self.IDList.append(list(self.IDListTuple[count])[0])
14
15              self.currentID = 0
16              for count in range(0, len(self.IDList)):
17                  if self.currentID < self.IDList[count]: #finding biggest ID no. for iteration limit
18                      self.currentID = self.IDList[count]
19
20              sql = "select ISBN from RoyaltyItems where RoyaltiesID = {}".format(self.selectedID)
21              cursor.execute(sql)
22              self.ISBNs = list(cursor.fetchall())
23
24              for count2 in range(0, len(self.ISBNs)):
25                  if self.currentID != 0:
26                      for count in range(0, self.currentID+1):
27                          self.Empty = False
28                          Selection = "RoyaltyItems.Currency, RoyaltyItems.NetSales,
                                  RoyaltyItems.ExcRateFromGBP, RoyaltyItems.RoyaltyQuantity, Book.NoOfPages,
                                  Book.Size, Book.Cover, Book.Back"
```

```
29                          Tables = "RoyaltyItems, Book"
30                          sql = "select {} from {} where RoyaltyItems.RoyaltiesID = {} and
                                  RoyaltyItems.RoyaltyItemsID = {} and RoyaltyItems.ISBN = {} and Book.ISBN =
                                  RoyaltyItems.ISBN".format(Selection, Tables, self.selectedID, count+1,
                                  list(self.ISBNs[count2])[0])
31                          cursor.execute(sql)
32
33                          try:
34                              self.SelectionList = list(cursor.fetchone())
35                          except:
36                             self.Empty = True
37
38                          if self.Empty != True:
39                              self.Currency = self.SelectionList[0]
40                              self.NetSales = float(self.SelectionList[1])
41                              self.Quantity = self.SelectionList[3]
42                              self.NoOfPages = int(self.SelectionList[4])
43                              self.Size = self.SelectionList[5]
44                              self.Cover = self.SelectionList[6]
45                              self.Back = self.SelectionList[7]
46
47                              if self.Size == "Large":
48                                  self.PagePrice = 0.015 * self.NoOfPages
49                                  if self.Back == "Hard":
50                                      self.CoverPrice = 5
51                                  elif self.Back == "Soft":
52                                      self.CoverPrice = 1
53                              elif self.Size == "Small":
54                                  self.PagePrice = 0.01 * self.NoOfPages
55                                  if self.Back == "Hard":
56                                      self.CoverPrice = 4
57                                  elif self.Back == "Soft":
58                                      self.CoverPrice = 0.7
59
60                              self.PrintCost = (self.PagePrice + self.CoverPrice) * self.Quantity
61
62                              self.TempPayment= self.NetSales - self.PrintCost
63                              if self.Currency != "£":
64                                  self.ExcRateFromGBP = float(self.SelectionList[2])
65                                  self.TempPayment /= self.ExcRateFromGBP
66                              self.RoyaltyPayment += self.TempPayment
67
68                  elif self.currentID == 0:
69                      self.BookInvoicePayment = None
70                      sql = "update Royalties set RoyaltyPayment = '{}' where RoyaltiesID =
                              {}".format(self.RoyaltyPayment, self.selectedID)
71                      cursor.execute(sql)
72                      db.commit()
73
74              if self.currentID != 0:
75                  self.RoyaltyPayment = "£{0:.2f}".format(self.RoyaltyPayment)
76                  sql = "update Royalties set RoyaltyPayment = '{}' where RoyaltiesID =
                          {}".format(self.RoyaltyPayment, self.selectedID)
77                  cursor.execute(sql)
78                  db.commit()
```

Lines 139 to 143 find the biggest ID no., so that the program can loop through each item ID, accummalating the total payment for each ID of each item that is in the royalty payment. Line 154 defines the sql statement to fetch all the necessary data to conduct the calculations. Then, upto line 110, the program is checking whether the set of items is in the invoice, and then calculates the payment, starting with the print cost in lines 171-184. The payment for the single set items is then added on to the current total, which starts at £0. Lines 198-202 format the full final payment into a string, and update the payment in the royalty entry with the newly calculated payment.

### 4.7.2 Self-created Algorithms

I have created the following algorithm to calculate the book invoice item payment when adding/updating a book invoice item. Lines 476-498 of Module 2, subsection 4.10.2.

```
1     def BookInvoiceItemCalculation(self): #calculating the bookinvoicepayment
```

```
        try:
            if self.Editing == False:
                self.Quantity = int(self.AddWindow.inputList[2].text())
                self.Discount = float(self.AddWindow.inputList[3].text()) / 100
                self.ShippingPrice = float(self.AddWindow.inputList[5].text())
                self.ISBN = self.AddWindow.inputList[1].text()
            elif self.Editing == True:
                self.Quantity = int(self.EditWindow.inputList[2].text())
                self.Discount = float(self.EditWindow.inputList[3].text()) / 100
                self.ShippingPrice = float(self.EditWindow.inputList[5].text())
                self.ISBN = self.EditWindow.inputList[1].text()

            with sqlite3.connect("PP.db") as db: #fetching data from db
                cursor = db.cursor()
                cursor.execute("select Price from Book where ISBN = {}".format(self.ISBN))
                self.Price = list(cursor.fetchone())[0]
                db.commit()

            self.BookInvoiceItemPayment = (self.Quantity * self.Price)
            self.Discount = self.BookInvoiceItemPayment * self.Discount
            self.BookInvoiceItemPayment -= self.Discount
            self.BookInvoiceItemPayment += self.ShippingPrice
```

This extract of Module 2 retrieves the entered Quantity, Discount, Shipping Price and ISBN from the window. The function goes on to fetch the price of the book from the database. Using the retrieved values, the calculations are conducted and the Invoice item payment is found.

I have created the following algorithm to calculate the royalty item payment when adding/updating a royalty item. Lines 514-529 of Module 2, subsection 4.10.2.

```
    def RoyaltyItemCalculation(self): #calculating the royalty payment
        try:
            if self.Editing == False:
                self.Currency = self.AddWindow.inputList[2].text()
                self.WholesalePrice = float(self.AddWindow.inputList[4].text())
                self.Quantity = int(self.AddWindow.inputList[5].text())
                self.PrintCost = float(self.AddWindow.inputList[6].text())
            elif self.Editing == True:
                self.Currency = self.EditWindow.inputList[2].text()
                self.WholesalePrice = float(self.EditWindow.inputList[4].text())
                self.Quantity = int(self.EditWindow.inputList[5].text())
                self.PrintCost = float(self.EditWindow.inputList[6].text())

            self.NetSales = self.WholesalePrice * self.Quantity
            self.RoyaltyItemPayment = self.NetSales - self.PrintCost
            self.RoyaltyItemPayment = "{0:.2f}".format(self.RoyaltyItemPayment)
```

This extract of Module 2 retrieves the entered Currency, Wholesale price, Quantity and Print cost from the window. Using the retrieved values, the calculations are conducted and the Royalty item payment is found.

## 4.8 Settings

In order for my program to function properly, Python 3.4, SQLite3 and PyQt4 must be installed on the machine being used.

## 4.9 Acknowledgements

The following pieces of code used regular expressions which were found on www.regexlib.com.

```
        self.inputList[2].setValidator(QRegExpValidator(QRegExp("^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$")))
```

(Used in line 31 of Module 6, subsection 4.10.6)

```
        self.regexp =
              "^(((\+44\s?\d{4}|\(?0\d{4}\)?)\s?\d{3}\s?\d{3})|((\+44\s?\d{3}|\(?0\d{3}\)?)\s?\d{3}\s?\d{4})|"
        self.regexp2 = "((\+44\s?\d{2}|\(?0\d{2}\)?)\s?\d{4}\s?\d{4}))(\s?\#(\d{4}|\d{3}))?$"
```

(Used in lines 32-33 of Module 6, subsection 4.10.6)

```
        self.inputList[5].setValidator(QRegExpValidator(QRegExp("^[a-zA-Z]{1,2}[0-9][0-9A-Za-z]{0,1}
              {0,1}[0-9][A-Za-z]{2}$")))
```

(Used in line 36 of Module 6, subsection 4.10.6)

```
        self.leNewUsername.setValidator(QRegExpValidator(QRegExp("^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$")))
        self.leOldUsername.setValidator(QRegExpValidator(QRegExp("^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$")))
```

(Used in lines 17-18 of Module 17, subsection 4.10.17)

The Calendar Widget, (Module 10, on page 4.10.10) was reimplemented from the source code I found on http://zetcode.com/gui/pyqt4/widgets.

```
import sys
from PyQt4.QtCore import *
from PyQt4.QtGui import *

class dbCalendarWidget(QDialog):

    def __init__(self):
        super().__init__()

    def Calendar(self):
        self.setFixedSize(265, 275)
        self.setWindowTitle('Calendar')
        calendar = QCalendarWidget(self)
        calendar.setGridVisible(True)
        calendar.clicked[QDate].connect(self.date)
        date = calendar.selectedDate()
        self.lblInstruction = QLabel(self)
        self.lblInstruction.setText("Please select a date")
        self.lblInstruction.setAlignment(Qt.AlignCenter)
        self.qle = QLineEdit(self)
        self.qle.setText(date.toString("dd-MM-yyyy"))
        self.qle.setFixedSize(85,20)
        self.qle.setAlignment(Qt.AlignCenter)
        self.btnSelect = QPushButton("Select", self)
        self.btnCancel = QPushButton("Cancel", self)

        self.horizontalTop = QHBoxLayout()
        self.horizontalTop.addStretch(1)
        self.horizontalTop.addWidget(self.lblInstruction)
        self.horizontalTop.addStretch(1)

        self.horizontalMid1 = QHBoxLayout()
        self.horizontalMid1.addStretch(1)
        self.horizontalMid1.addWidget(calendar)
        self.horizontalMid1.addStretch(1)

        self.horizontalMid2 = QHBoxLayout()
        self.horizontalMid2.addStretch(1)
        self.horizontalMid2.addWidget(self.qle)
        self.horizontalMid2.addStretch(1)

        self.horizontalBottom = QHBoxLayout()
        self.horizontalBottom.addWidget(self.btnCancel)
```

```
        self.horizontalBottom.addStretch(1)
        self.horizontalBottom.addWidget(self.btnSelect)

        self.vertical = QVBoxLayout()
        self.vertical.addLayout(self.horizontalTop)
        self.vertical.addLayout(self.horizontalMid1)
        self.vertical.addLayout(self.horizontalMid2)
        self.vertical.addLayout(self.horizontalBottom)
        self.setLayout(self.vertical)


    def DisplayCalendar(self):
        self.setModal(True)
        self.btnSelect.clicked.connect(self.accept)
        self.btnCancel.clicked.connect(self.reject)
        self.exec_()

    def date(self, date):
        self.qle.setText(date.toString("dd-MM-yyyy"))
```

## 4.10 Code Listing

### 4.10.1 Module 1 - Login Window and Database Creation

```
from PyQt4.QtCore import *
from PyQt4.QtGui import *
import sqlite3
import sys
import smtplib
import email
import time
from MainMenu import *

class dbLogin(QMainWindow):
    """db for login"""

    def __init__(self):
        self.initSplashScreen()
        self.initDetails()
        if self.details == ("Username", "Password"):
            self.customer_table()
            self.book_table()
            self.pub_invoice_table()
            self.book_invoice_table()
            self.book_invoice_items_table()
            self.royalties_table()
            self.royalty_items_table()
            self.MainProgram = MainWindow("Username") #runs main window if Username/Password haven't been changed before
            self.MainProgram.show()
        else:
            super().__init__() #runs login screen if they have been changed
            self.initLoginScreen()

    def initDetails(self):
        with sqlite3.connect("dbLogin.db") as db:
            cursor = db.cursor()

            self.sql = "select name from sqlite_master WHERE type='table' and name='LoginDetails'"
            cursor.execute(self.sql) #checking whether the login table exists
            try:
                self.Exists = list(cursor.fetchone())[0]
            except:
                self.Exists = False

```

```
            self.sql = "create table if not exists LoginDetails (Username text, Password text)"
            cursor.execute(self.sql) #creates login table if it doesn't exist

            if self.Exists == False:
                self.details = "Username", "Password"
                self.sql = "insert into LoginDetails (Username, Password) values (?, ?)"
                cursor.execute(self.sql, self.details) #adds default Username and Password
            else:
                self.details = True

    def initLoginScreen(self):
        self.setWindowTitle("Login")
        self.setWindowIcon(QIcon("LoginIcon.png"))
        self.setFixedSize(300, 190)
        self.lblLogin = QLabel("Please Log In", self)
        self.lblLogin.setFont(QFont("Calibri",14))
        self.lblLogin.setAlignment(Qt.AlignHCenter)
        self.btnLogin = QPushButton("Login", self)
        self.btnLogin.setFixedSize(self.btnLogin.sizeHint())
        self.lblUsername = QLabel("Username:", self)
        self.lblUsername.setFont(QFont("Calibri", 10))
        self.lblPassword = QLabel("Password:", self)
        self.lblPassword.setFont(QFont("Calibri", 10))
        self.leUsername = QLineEdit(self)
        self.leUsername.setPlaceholderText("Username")
        self.lePassword = QLineEdit(self)
        self.lePassword.setEchoMode(self.lePassword.Password)
        self.lePassword.setPlaceholderText("Password")
        self.lblForgot = QLabel("Help/Forgotten Password?", self)
        self.Underline = QFont("Calibri", 10)
        self.Underline.setUnderline(True)
        self.lblForgot.setFont(self.Underline)
        self.lblForgot.setAlignment(Qt.AlignHCenter)
        self.lblForgot.mousePressEvent = self.getEmail
        self.lblVertical = QVBoxLayout()
        self.leVertical = QVBoxLayout()
        self.horizontalLogin = QHBoxLayout()
        self.lblVertical.addWidget(self.lblUsername)
        self.leVertical.addWidget(self.leUsername)
        self.lblVertical.addWidget(self.lblPassword)
        self.leVertical.addWidget(self.lePassword)
        self.horizontalLogin.addStretch(1)
```

```
        self.horizontalLogin.addWidget(self.btnLogin)
        self.horizontalLogin.addStretch(1)
        self.vertical = QVBoxLayout()
        self.vertical.addWidget(self.lblLogin)
        self.horizontalEntry = QHBoxLayout()
        self.horizontalEntry.addLayout(self.lblVertical)
        self.horizontalEntry.addLayout(self.leVertical)
        self.vertical.addLayout(self.horizontalEntry)
        self.vertical.addWidget(self.lblForgot)
        self.vertical.addLayout(self.horizontalLogin)
        self.vertical.addStretch(1)
        self.horizontal = QHBoxLayout()
        self.horizontal.addStretch(1)
        self.horizontal.addLayout(self.vertical)
        self.horizontal.addStretch(1)
        self.CentralWidget = QWidget()
        self.CentralWidget.setLayout(self.horizontal)
        self.setCentralWidget(self.CentralWidget)
        self.btnLogin.clicked.connect(self.Login)
        self.lblInvalid = None

    def Login(self):
        with sqlite3.connect("dbLogin.db") as db:
            cursor = db.cursor()
            cursor.execute("select Username from LoginDetails")
            self.Username = list(cursor.fetchall()) #fetches original username and password
            cursor.execute("select Password from LoginDetails")
            self.Password = list(cursor.fetchall())
            self.Valid = False

            for count in range(0, len(self.Username)):
                if self.leUsername.text().lower() == list(self.Username[count])[0].lower():
                    if self.lePassword.text() == list(self.Password[count])[0]:
                        self.Valid = True
                        self.customer_table() #checking all tables to see if they're existent
                        self.book_table()
                        self.pub_invoice_table()
                        self.book_invoice_table()
                        self.book_invoice_items_table()
                        self.royalties_table()
                        self.royalty_items_table()
                        self.hide() #Username and password match, so the user is logged in
```

```
                        self.MainProgram = MainWindow(list(self.Username[count])[0])
                        self.MainProgram.show()
                        break
                else:
                    self.Valid = False

            if self.Valid == False:
                if self.lblInvalid == None: #Username and password do not match, user is rejected
                    self.lblInvalid = QLabel("Invalid Username or Password - Please try again.", self)
                    self.lblInvalid.setWordWrap(True)
                    self.lblInvalid.setAlignment(Qt.AlignHCenter)
                    self.horizontalInvalid = QHBoxLayout()
                    self.horizontalInvalid.addStretch(1)
                    self.horizontalInvalid.addWidget(self.lblInvalid)
                    self.horizontalInvalid.addStretch(1)
                    self.vertical.addLayout(self.horizontalInvalid)
                else:
                    self.lblInvalid.show()

    def getEmail(self, QMouseEvent):
        with sqlite3.connect("dbLogin.db") as db:
            cursor = db.cursor()
            cursor.execute("select Username from LoginDetails")
            self.Username = list(cursor.fetchone())[0]
            cursor.execute("select Password from LoginDetails")
            self.Password = list(cursor.fetchone())[0]

        if self.Username == "Username" and self.Password == "Password":
            self.Msg = QMessageBox()
            self.Msg.setWindowTitle("First Time")
            self.Msg.setText("This is your first time using this application.\n Your Username is 'Username' and your
                 Password is 'Password'.\n Please change these once logged in.")
            self.Msg.exec_() #default username and password is shown to the user
        else:
            self.Email, ok = QInputDialog.getText(self, 'Forgotten Password', 'Enter your Email:')
            self.Email = self.Email.lower() #email is received from the user
            if self.Email == self.Username:
                with sqlite3.connect("dbLogin.db") as db:
                    cursor = db.cursor()
                    cursor.execute("select Password from LoginDetails")
                    self.CurrentPassword = list(cursor.fetchone())[0] #gets password from database
                self.sender = "pp.loginhelp@gmail.com"
```

```
                self.recipient = [str(self.Email)]
                self.server = smtplib.SMTP('smtp.gmail.com', 587)
                self.server.ehlo()
                self.server.starttls()
                self.server.ehlo()
                self.server.login("pp.loginhelp@gmail.com", "DB1061NH31P")

                self.Subject = "Forgotten Login Details"
                self.message = "From: {}\r\nTo: {}\r\nSubject: {}\r\n\r\n".format(self.sender, ", ".join(self.recipient),
                    self.Subject)
                self.message += "Your Password is: {}\nPlease change this upon login for security
                    reasons.".format(self.CurrentPassword)

                self.server.sendmail(self.sender, self.recipient, self.message)
                self.server.close()

                self.Msg = QMessageBox()
                self.Msg.setWindowTitle("Email Sent") #email is sent if email is existent in the database, with password
                    details
                self.Msg.setText("You have been sent an email with the corresponding password details")
                self.Msg.exec_()

            elif self.Email != self.Username and ok == True:
                self.Msg = QMessageBox()
                self.Msg.setWindowTitle("No Match found") #email is not found, so email is not sent
                self.Msg.setText("No matching Email was found")
                self.Msg.exec_()

    def keyReleaseEvent(self, QKeyEvent):
        if self.lblInvalid != None:
            self.lblInvalid.hide()

    def create_table(self):
        with sqlite3.connect("PP.db") as db:
            cursor = db.cursor()
            cursor.execute("PRAGMA foreign_keys = ON")
            self.Exists = True
            self.FindTable = "select name from sqlite_master WHERE type='table' and name='{}'".format(self.TableName)
            cursor.execute(self.FindTable) #checks if the tables exist
            try:
                self.Exists = list(cursor.fetchone())
            except:
```

```
                self.Exists = False

            if self.Exists == False: #creates tables for the database if they don't already exist
                cursor.execute(self.sql)
                db.commit()

    def customer_table(self):
        self.sql = """create table Customer
                 (AuthorID integer,
                 FirstName text,
                 LastName text,
                 Email text,
                 PhoneNumber text,
                 Address text,
                 Postcode text,
                 primary key(AuthorID))"""
        self.TableName = "Customer"
        self.create_table()

    def book_table(self):
        self.sql = """create table Book
                 (ISBN text,
                 AuthorID integer,
                 BookTitle text,
                 NoOfPages integer,
                 Size text,
                 Back text,
                 Cover text,
                 Paper text,
                 Font text,
                 FontSize real,
                 DatePublished date,
                 Price real,
                 primary key(ISBN),
                 foreign key(AuthorID) references Customer(AuthorID))"""
        self.TableName = "Book"
        self.create_table()

    def pub_invoice_table(self):
        self.sql = """create table PubInvoice
                 (PubInvoiceID integer,
                 ISBN text,
```

```
                AuthorID integer,
                PubInvoiceDate date,
                PubInvoiceService text,
                PubInvoicePayment real,
                primary key(PubInvoiceID),
                foreign key(AuthorID) references Customer(AuthorID),
                foreign key(ISBN) references Book(ISBN))"""
        self.TableName = "PubInvoice"
        self.create_table()

    def book_invoice_table(self):
        self.sql = """create table BookInvoice
                (BookInvoiceID integer,
                AuthorID integer,
                BookInvoiceDate date,
                BookInvoicePayment real,
                primary key(BookInvoiceID),
                foreign key(AuthorID) references Customer(AuthorID))"""
        self.TableName = "BookInvoice"
        self.create_table()

    def book_invoice_items_table(self):
        self.sql = """create table BookInvoiceItems
                (BookInvoiceItemsID integer,
                BookInvoiceID integer,
                ISBN text,
                BookInvoiceQuantity integer,
                BookInvoiceDiscount real,
                ShippingType text,
                ShippingPrice real,
                primary key(BookInvoiceItemsID),
                foreign key(BookInvoiceID) references BookInvoice(BookInvoiceID),
                foreign key(ISBN) references Book(ISBN))"""
        self.TableName = "BookInvoiceItems"
        self.create_table()

    def royalties_table(self):
        self.sql = """create table Royalties
                (RoyaltiesID integer,
                AuthorID integer,
                RoyaltiesDate date,
                RoyaltyPayment real,
```

```
                    primary key(RoyaltiesID),
                    foreign key(AuthorID) references Customer(AuthorID))"""
        self.TableName = "Royalties"
        self.create_table()

    def royalty_items_table(self):
        self.sql = """create table RoyaltyItems
                    (RoyaltyItemsID integer,
                    RoyaltiesID integer,
                    ISBN text,
                    Currency text,
                    RoyaltyDiscount real,
                    WholesalePrice real,
                    RoyaltyQuantity integer,
                    NetSales real,
                    PrintCost real,
                    ExcRateFromGBP real,
                    primary key(RoyaltyItemsID),
                    foreign key(RoyaltiesID) references Royalties(RoyaltiesID),
                    foreign key(ISBN) references Book(ISBN))"""
        self.TableName = "RoyaltyItems"
        self.create_table()

    def initSplashScreen(self):
        self.pixmap = QPixmap("PerfectPublishersLtd.png")
        self.Splashscreen = QSplashScreen(self.pixmap, Qt.WindowStaysOnTopHint)
        self.Splashscreen.setMask(self.pixmap.mask())
        self.Splashscreen.show()
        time.sleep(2)
        self.Splashscreen.finish(self.Splashscreen)


def main():
    app = QApplication(sys.argv)
    launcher = dbLogin()
    try:
        launcher.raise_()
        launcher.show()
    except RuntimeError:
        pass
    app.exec_()

```

```python



if __name__ == "__main__":
    main()
```

### 4.10.2 Module 2 - Main Window and functionality for majority of system

```python
from PyQt4.QtCore import *
from PyQt4.QtGui import *
import sqlite3
import sys
import subprocess

from MenuBar import *
from initMainMenuButtons import *
from AddEntryWindow import *
from TableWidget import *
from ConfirmationDialog import *
from ViewWindow import *
from AddItemWindow import *
from ViewRoyaltiesAndInvoices import *
from UpdateEntryWindow import *
from CalendarWidget import *
from Items import *
from SearchDatabase import *
from LoginDB import *
from ChangePassword import *
from UsernameOrPassword import *
from ChangeUsername import *
from SearchResults import *

class MainWindow(QMainWindow):
    """main window"""

    def __init__(self, Username):
        super().__init__()
        self.Username = Username
        self.setWindowTitle("Main Menu")
```

```
        self.setWindowIcon(QIcon("PPIcon.png"))
        self.setFixedSize(735,400)
        self.MenuBar = dbMenuBar()
        self.setMenuBar(self.MenuBar)

        self.TableWidget = dbTableWidget()
        self.TableWidget.sql = "select * from Customer"
        self.TableWidget.initTable()
        self.MainMenuButtons = initMainMenuButtons()
        self.MainMenuButtons.vertical.addLayout(self.MainMenuButtons.horizontalTop)
        self.MainMenuButtons.vertical.addWidget(self.TableWidget)
        self.MainMenuButtons.vertical.addLayout(self.MainMenuButtons.horizontalBottom)
        self.CurrentTable = "Customer"
        self.MainMenuButtons.setLayout(self.MainMenuButtons.vertical)

        self.StackedLayout = QStackedLayout()

        self.centralWidget = QWidget()
        self.centralWidget.setLayout(self.StackedLayout)
        self.setCentralWidget(self.centralWidget)

        self.StackedLayout.addWidget(self.MainMenuButtons)

        self.ViewWindow = dbViewWindow() #instantiating view window
        self.ViewWindow.View()
        self.ViewWindow.vertical.addLayout(self.ViewWindow.horizontalTop)
        self.ViewWindow.table = dbTableWidget()
        self.ViewWindow.vertical.addWidget(self.ViewWindow.table)
        self.ViewWindow.vertical.addLayout(self.ViewWindow.horizontalBottom)
        self.ViewWindow.setLayout(self.ViewWindow.vertical)

        self.StackedLayout.addWidget(self.ViewWindow)

        self.SearchResults = initSearchResultsMenu()
        self.StackedLayout.addWidget(self.SearchResults)
        self.SearchTable = dbTableWidget()
        self.SearchResults.vertical.addWidget(self.SearchTable)

        #connections
        self.MainMenuButtons.btnAddEntry.clicked.connect(self.AddEntry)
        self.MainMenuButtons.btnRemoveEntry.clicked.connect(self.RemoveEntry)
        self.MainMenuButtons.btnView.clicked.connect(self.ViewCustomer)
```

```
        self.MainMenuButtons.btnUpdateEntry.clicked.connect(self.UpdateCustomerEntry)
        self.MainMenuButtons.btnQuickSearch.clicked.connect(self.QuickSearch)
        self.MainMenuButtons.btnSearchdb.clicked.connect(self.Search)
        self.MainMenuButtons.btnLogOut.clicked.connect(self.LogOut)
        self.MainMenuButtons.btnChangePassword.clicked.connect(self.ChangeUsernameOrPassword)
        self.MenuBar.search_database.triggered.connect(self.Search)
        self.MenuBar.add_entry.triggered.connect(self.AddEntry)
        self.MenuBar.remove_entry.triggered.connect(self.RemoveEntry)
        self.MenuBar.update_entry.triggered.connect(self.UpdateCustomerEntry)
        self.MenuBar.log_out.triggered.connect(self.LogOut)
        self.MenuBar.change_password.triggered.connect(self.ChangeUsernameOrPassword)
        self.ViewWindow.btnBack.clicked.connect(self.Back)
        self.ViewWindow.btnAddBook.clicked.connect(self.AddItem)
        self.ViewWindow.btnUpdateBook.clicked.connect(self.UpdateEntry)
        self.ViewWindow.btnDeleteBook.clicked.connect(self.RemoveFromDB)
        self.ViewWindow.btnViewPubInvoice.clicked.connect(self.ViewPubInvoice)
        self.ViewWindow.btnViewBookInvoices.clicked.connect(self.ViewBookInvoice)
        self.ViewWindow.btnViewRoyalties.clicked.connect(self.ViewRoyalties)
        self.SearchResults.btnBack.clicked.connect(self.Back)


    def AddEntry(self): #adding customer entry
        self.AddEntryWindow = dbAddEntryWindow()
        self.AddEntryWindow.Added = False
        self.AddEntryWindow.initAddEntryWindow()
        self.RefreshTables()



    def AddItem(self): #initialising an add window for getting inputs for other entries
        self.AddWindow = dbAddItemWindow()
        self.AddWindow.setFixedSize(360,200)
        self.AddWindow.AddType = self.CurrentTable
        self.AddWindow.AnswerButtons()
        self.AddWindow.Editing = False

        if self.CurrentTable == "Book":
            self.AddWindow.sql = "select * from Book"

        elif self.CurrentTable == "PubInvoice":
            self.AddWindow.setFixedSize(400,150)
```

```
            self.AddWindow.sql = "select ISBN, AuthorID, PubInvoiceDate, PubInvoiceService, PubInvoicePayment from PubInvoice"
            self.AddWindow.selectedISBN = self.SelectedISBN

        elif self.CurrentTable == "BookInvoice":
            self.AddWindow.sql = "select AuthorID, BookInvoiceDate from BookInvoice"
            self.AddWindow.setFixedSize(350,100)

        elif self.CurrentTable == "Royalties":
            self.AddWindow.sql = "select AuthorID, RoyaltiesDate from Royalties"
            self.AddWindow.setFixedSize(350,100)

        elif self.CurrentTable == "BookInvoiceItems":
            self.AddWindow.sql = "select BookInvoiceID, ISBN, BookInvoiceQuantity, BookInvoiceDiscount, ShippingType, ShippingPrice from BookInvoiceItems"
            self.AddWindow.setFixedSize(450, 150)
            self.AddWindow.selectedISBN = self.SelectedISBN
            self.Editing = False
            self.AddWindow.btnCalculate.clicked.connect(self.BookInvoiceItemCalculation)

        elif self.CurrentTable == "RoyaltyItems":
            self.AddWindow.sql = "select RoyaltiesID, ISBN, Currency, RoyaltyDiscount, WholesalePrice, RoyaltyQuantity, PrintCost, ExcRateFromGBP from RoyaltyItems"
            self.AddWindow.setFixedSize(450, 250)
            self.AddWindow.selectedISBN = self.SelectedISBN
            self.Editing = False
            self.AddWindow.btnCalculate.clicked.connect(self.RoyaltyItemCalculation)

        self.AddWindow.selectedID = self.SelectedID
        self.AddWindow.CalendarWidget = dbCalendarWidget()
        self.AddWindow.CalendarWidget.Calendar()

        self.AddWindow.initAddItemWindow()
        try:
            if self.AddWindow.Valid == True: #inputs must pass validation before being added
                self.AddToDB()
        except AttributeError:
            pass #exception of window being closed
        self.RefreshTables()

    def RecalculateItems(self): #recalculates after changes
        if self.CurrentTable == "BookInvoiceItems":
```

```
            self.BookInvoiceItemsWindow.CalculateBookInvoiceItems()
            self.CurrentTable = "BookInvoice"
            self.RefreshTables()
            self.CurrentTable = "BookInvoiceItems"
            self.RefreshTables()
        elif self.CurrentTable == "RoyaltyItems" or self.BookEdited == True:
            self.RoyaltyItemsWindow.CalculateRoyaltyItems()
            self.CurrentTable = "Royalties"
            self.RefreshTables()
            self.CurrentTable = "RoyaltyItems"
            self.RefreshTables()


    def AddToDB(self): #Adding Entries to database
        self.input_data = []
        if self.CurrentTable == "Book":
            self.TableValues = "Book (ISBN, AuthorID, BookTitle, NoOfPages, Size, Back, Cover, Paper, Font, FontSize,
                DatePublished, Price)"
            self.Placeholders = "(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)"
            self.NoOfEntries = 12

        elif self.CurrentTable == "PubInvoice":
            self.TableValues = "PubInvoice (ISBN, AuthorID, PubInvoiceDate, PubInvoiceService, PubInvoicePayment)"
            self.Placeholders = "(?, ?, ?, ?, ?)"
            self.NoOfEntries = 5

        elif self.CurrentTable == "BookInvoice":
            self.TableValues = "BookInvoice (AuthorID, BookInvoiceDate)"
            self.Placeholders = "(?, ?)"
            self.NoOfEntries = 2

        elif self.CurrentTable == "Royalties":
            self.TableValues = "Royalties (AuthorID, RoyaltiesDate)"
            self.Placeholders = "(?, ?)"
            self.NoOfEntries = 2

        elif self.CurrentTable == "BookInvoiceItems":
            self.TableValues = "BookInvoiceItems (BookInvoiceID, ISBN, BookInvoiceQuantity, BookInvoiceDiscount,
                ShippingType, ShippingPrice)"
            self.Placeholders = "(?, ?, ?, ?, ?, ?)"
            self.NoOfEntries = 6

```

```
        elif self.CurrentTable == "RoyaltyItems":
            self.TableValues = "RoyaltyItems (RoyaltiesID, ISBN, Currency, RoyaltyDiscount, WholesalePrice,
                RoyaltyQuantity, PrintCost, ExcRateFromGBP, NetSales)"
            self.Placeholders = "(?, ?, ?, ?, ?, ?, ?, ?, ?)"
            self.NoOfEntries = 9

        for count in range(0, self.NoOfEntries):
            try: #putting the input data in a list
                self.input_data.append(str(self.AddWindow.inputList[count].currentText()))
            except:
                if count == 8 and self.CurrentTable == "RoyaltyItems":
                    self.input_data.append(self.NetSales) #Net sales are calculated, as opposed to being entered
                else:
                    self.input_data.append(self.AddWindow.inputList[count].text())

        with sqlite3.connect("PP.db") as db:
            cursor = db.cursor()
            cursor.execute("PRAGMA foreign_keys = ON")
            self.sql = "insert into {} values {}".format(self.TableValues, self.Placeholders)
            cursor.execute(self.sql, self.input_data)
            db.commit()

        self.RefreshTables()

        try:
            self.RecalculateItems()
        except:
            pass

    def RemoveEntry(self): #removing a customer
        self.SelectedRow = self.TableWidget.currentRow()
        self.ConfirmDialog = dbConfirmationDialog()
        self.ConfirmDialog.Username = self.Username
        self.ConfirmDialog.SelectedAuthorID = QTableWidgetItem(self.TableWidget.item(self.SelectedRow, 0)).text() #getting
            AuthorID of a row
        if self.ConfirmDialog.SelectedAuthorID != "":
            self.Firstname = QTableWidgetItem(self.TableWidget.item(self.SelectedRow, 1)).text()
            self.Lastname = QTableWidgetItem(self.TableWidget.item(self.SelectedRow, 2)).text()
            self.ConfirmDialog.Name = "{} {}".format(self.Firstname, self.Lastname)
            self.ConfirmDialog.Msg = "Are you sure you want to delete {} and all records about
                them?".format(self.ConfirmDialog.Name)
            self.ConfirmDialog.ConfirmedMsg = "{}'s records have been successfully erased.".format(self.ConfirmDialog.Name)
```

```
            self.ConfirmDialog.VerifyDlg()

            if self.ConfirmDialog.ConfirmedDialog.Accepted == True:
                with sqlite3.connect("PP.db") as db:
                    cursor = db.cursor()
                    cursor.execute("PRAGMA foreign_keys = ON")
                    sql = "select ISBN from Book where AuthorID = {}".format(self.ConfirmDialog.SelectedAuthorID)
                    cursor.execute(sql)
                    self.ISBNDeletion = list(cursor.fetchall())
                    for count in range(0, len(self.ISBNDeletion)): #removes all customer details, starting with foreign
                        keys for referential integrity
                        sql = "delete from BookInvoiceItems where ISBN =
                            {}".format(list(list(self.ISBNDeletion)[count])[0])
                        cursor.execute(sql)
                        sql = "delete from RoyaltyItems where ISBN = {}".format(list(list(self.ISBNDeletion)[count])[0])
                        cursor.execute(sql)
                    sql = "delete from PubInvoice where AuthorID = {}".format(self.ConfirmDialog.SelectedAuthorID)
                    cursor.execute(sql)
                    sql = "delete from BookInvoice where AuthorID = {}".format(self.ConfirmDialog.SelectedAuthorID)
                    cursor.execute(sql)
                    sql = "delete from Royalties where AuthorID = {}".format(self.ConfirmDialog.SelectedAuthorID)
                    cursor.execute(sql)
                    sql = "delete from Book where AuthorID = {}".format(self.ConfirmDialog.SelectedAuthorID)
                    cursor.execute(sql)
                    sql = "delete from Customer where AuthorID = {}".format(self.ConfirmDialog.SelectedAuthorID)
                    cursor.execute(sql)
                    db.commit()
                    self.RefreshTables()



    def RemoveFromDB(self): #removing other entries
        #getting primary key of the row
        self.ConfirmDialog = dbConfirmationDialog()
        self.ConfirmDialog.Username = self.Username
        self.ConfirmDialog.DeleteMsg = self.CurrentTable
        self.SelectedAuthorID = self.SelectedID

        if self.CurrentTable == "Book":
            self.ForeignKeyMsg = "Royalties and Invoices"
            self.SelectedRow = self.ViewWindow.table.currentRow()
            self.SelectedID = QTableWidgetItem(self.ViewWindow.table.item(self.SelectedRow, 0)).text()
```

```
            self.SelectedIDName = "ISBN"
            self.ConfirmDialog.Msg = "Are you sure you want to delete this book?"
            self.ConfirmDialog.ConfirmedMsg = "Book was successfully deleted"

        elif self.CurrentTable == "PubInvoice":
            self.SelectedRow = self.PubInvoiceWindow.table.currentRow()
            self.OldID = self.SelectedID
            self.SelectedID = QTableWidgetItem(self.PubInvoiceWindow.table.item(self.SelectedRow, 0)).text()
            self.SelectedIDName = "PubInvoiceID"
            self.ConfirmDialog.Msg = "Are you sure you want to delete this Invoice?"
            self.ConfirmDialog.ConfirmedMsg = "Invoice was successfully deleted"

        elif self.CurrentTable == "BookInvoice":
            self.ForeignKeyMsg = "Book Invoice Items"
            self.SelectedRow = self.BookInvoiceWindow.table.currentRow()
            self.OldID = self.SelectedID
            self.SelectedID = QTableWidgetItem(self.BookInvoiceWindow.table.item(self.SelectedRow, 0)).text()
            self.SelectedIDName = "BookInvoiceID"
            self.ConfirmDialog.Msg = "Are you sure you want to delete this Invoice?"
            self.ConfirmDialog.ConfirmedMsg = "Invoice was successfully deleted"

        elif self.CurrentTable == "Royalties":
            self.ForeignKeyMsg = "Royalty Items"
            self.SelectedRow = self.RoyaltiesWindow.table.currentRow()
            self.OldID = self.SelectedID
            self.SelectedID = QTableWidgetItem(self.RoyaltiesWindow.table.item(self.SelectedRow, 0)).text()
            self.SelectedIDName = "RoyaltiesID"
            self.ConfirmDialog.Msg = "Are you sure you want to delete this Entry?"
            self.ConfirmDialog.ConfirmedMsg = "Entry was successfully deleted"

        elif self.CurrentTable == "BookInvoiceItems":
            self.SelectedRow = self.BookInvoiceItemsWindow.table.currentRow()
            self.OldID = self.SelectedID
            self.SelectedID = QTableWidgetItem(self.BookInvoiceItemsWindow.table.item(self.SelectedRow, 0)).text()
            self.SelectedIDName = "BookInvoiceItemsID"
            self.ConfirmDialog.Msg = "Are you sure you want to delete this Item?"
            self.ConfirmDialog.ConfirmedMsg = "Item was successfully deleted"

        elif self.CurrentTable == "RoyaltyItems":
            self.SelectedRow = self.RoyaltyItemsWindow.table.currentRow()
            self.OldID = self.SelectedID
            self.SelectedID = QTableWidgetItem(self.RoyaltyItemsWindow.table.item(self.SelectedRow, 0)).text()
```

```
            self.SelectedIDName = "RoyaltyItemsID"
            self.ConfirmDialog.Msg = "Are you sure you want to delete this Item?"
            self.ConfirmDialog.ConfirmedMsg = "Item was successfully deleted"
        self.ConfirmDialog.Prevention = True #deletion may be prevented if integrity error occurs
        if self.SelectedRow != -1:
            self.ConfirmDialog.VerifyDlg() #verification
            try:
                if self.ConfirmDialog.ConfirmedDialog.Accepted == True:
                    with sqlite3.connect("PP.db") as db:
                        cursor = db.cursor()
                        cursor.execute("PRAGMA foreign_keys = ON")
                        sql = "delete from {} where {} = '{}'".format(self.CurrentTable, self.SelectedIDName,
                            self.SelectedID)
                        cursor.execute(sql)
                        db.commit()
                        self.ConfirmDialog.ConfirmedDialog.Confirmed()
                        self.SelectedID = self.SelectedAuthorID
                        self.RefreshTables()
            except sqlite3.IntegrityError:
                self.Msg = QMessageBox()
                self.Msg.setWindowTitle("Error")
                self.Msg.setText("You must delete all the {} first.".format(self.ForeignKeyMsg))
                self.Msg.exec_() #informs user that selected entry cannot be deleted and what must be done for it to be
                    deleted

            if self.CurrentTable in ["Royalties", "BookInvoice", "PubInvoice", "BookInvoiceItems", "RoyaltyItems"]:
                self.SelectedID = self.OldID

            try:
                self.RecalculateItems()
            except:
                pass



    def ViewCustomer(self): #displaying customer data
        self.SelectedRow = self.TableWidget.currentRow()
        self.SelectedID = QTableWidgetItem(self.TableWidget.item(self.SelectedRow, 0)).text()
        self.Firstname = QTableWidgetItem(self.TableWidget.item(self.SelectedRow, 1)).text()
        self.Lastname = QTableWidgetItem(self.TableWidget.item(self.SelectedRow, 2)).text()

        if self.SelectedID != "":
```

```
            self.SelectedAuthorID = self.SelectedID
            self.CurrentTable = "Book"
            self.RefreshTables()
            self.StackedLayout.setCurrentIndex(1)
            self.MenuBar.setVisible(False)



    def ViewPubInvoice(self): #initialising the view publishing invoice window
        self.CurrentTable = "PubInvoice"
        self.ViewWindow.SelectedRow = self.ViewWindow.table.currentRow()
        self.SelectedISBN = QTableWidgetItem(self.ViewWindow.table.item(self.ViewWindow.SelectedRow, 0)).text()
        self.SelectedID = QTableWidgetItem(self.TableWidget.item(self.TableWidget.currentRow(), 0)).text()
        self.SelectedAuthorID = self.SelectedID
        if self.SelectedISBN != "":
            self.PubInvoiceWindow = dbRoyaltiesAndInvoices()
            self.PubInvoiceWindow.PubInvoiceButtons()
            self.PubInvoiceWindow.btnAddPubInvoice.clicked.connect(self.AddItem)
            self.PubInvoiceWindow.btnUpdatePubInvoice.clicked.connect(self.UpdateEntry)
            self.PubInvoiceWindow.btnDeleteEntry.clicked.connect(self.RemoveFromDB)
            self.PubInvoiceWindow.table = dbTableWidget()
            self.RefreshTables()
            self.PubInvoiceWindow.table.setFixedSize(620, 150)
            self.PubInvoiceWindow.PubInvoice()
        self.CurrentTable = "Book"



    def ViewBookInvoice(self): #initialising the view book invoice window
        self.CurrentTable = "BookInvoice"
        self.ViewWindow.SelectedRow = self.ViewWindow.table.currentRow()
        self.SelectedISBN = QTableWidgetItem(self.ViewWindow.table.item(self.ViewWindow.SelectedRow, 0)).text()
        self.SelectedAuthorID = QTableWidgetItem(self.TableWidget.item(self.TableWidget.currentRow(), 0)).text()

        if self.SelectedISBN != "":
            self.BookInvoiceWindow = dbRoyaltiesAndInvoices()
            self.BookInvoiceWindow.BookInvoiceButtons()
            self.BookInvoiceWindow.btnViewBookInvoiceItems.clicked.connect(self.ViewBookInvoiceItems)
            self.BookInvoiceWindow.btnAddBookInvoice.clicked.connect(self.AddItem)
            self.BookInvoiceWindow.btnUpdateBookInvoice.clicked.connect(self.UpdateEntry)
            self.BookInvoiceWindow.btnDeleteEntry.clicked.connect(self.RemoveFromDB)
            self.BookInvoiceWindow.table = dbTableWidget()
```

```
            self.RefreshTables()
            self.BookInvoiceWindow.table.setFixedSize(620, 150)
            self.BookInvoiceWindow.BookInvoice()
        self.CurrentTable = "Book"



    def ViewBookInvoiceItems(self): #initialising the view book invoice items window
        self.CurrentTable = "BookInvoiceItems"
        self.BookInvoiceWindow.SelectedRow = self.BookInvoiceWindow.table.currentRow()
        self.holder = self.SelectedAuthorID
        self.SelectedAuthorID = self.SelectedID
        self.BookInvoiceWindow.selectedISBN = self.SelectedISBN
        self.SelectedID = QTableWidgetItem(self.BookInvoiceWindow.table.item(self.BookInvoiceWindow.SelectedRow, 0)).text()

        if self.SelectedID != "":
            self.BookInvoiceItemsWindow = dbItems()
            self.BookInvoiceItemsWindow.selectedID = self.SelectedID
            self.BookInvoiceItemsWindow.selectedISBN = self.SelectedISBN
            self.BookInvoiceItemsWindow.BookInvoiceItemsButtons()
            self.BookInvoiceItemsWindow.btnCalculate.clicked.connect(self.AddItem)
            self.BookInvoiceItemsWindow.btnUpdateBookInvoiceItems.clicked.connect(self.UpdateEntry)
            self.BookInvoiceItemsWindow.btnDeleteEntry.clicked.connect(self.RemoveFromDB)
            self.BookInvoiceItemsWindow.table = dbTableWidget()
            self.RefreshTables()
            self.BookInvoiceItemsWindow.table.setFixedSize(620, 150)
            self.BookInvoiceItemsWindow.BookInvoiceItems()
        self.SelectedID = self.SelectedAuthorID
        self.SelectedAuthorID = self.holder
        self.CurrentTable = "BookInvoice"
        self.RefreshTables()



    def ViewRoyalties(self): #initialising the view royalties window
        self.CurrentTable = "Royalties"
        self.ViewWindow.SelectedRow = self.ViewWindow.table.currentRow()
        self.SelectedISBN = QTableWidgetItem(self.ViewWindow.table.item(self.ViewWindow.SelectedRow, 0)).text()
        self.SelectedAuthorID = QTableWidgetItem(self.TableWidget.item(self.TableWidget.currentRow(), 0)).text()
        if self.SelectedISBN != "":
            self.RoyaltiesWindow = dbRoyaltiesAndInvoices()
            self.RoyaltiesWindow.RoyaltiesButtons()
```

```
            self.RoyaltiesWindow.btnViewRoyaltyItems.clicked.connect(self.ViewRoyaltyItems)
            self.RoyaltiesWindow.btnAddRoyalties.clicked.connect(self.AddItem)
            self.RoyaltiesWindow.btnUpdateRoyalties.clicked.connect(self.UpdateEntry)
            self.RoyaltiesWindow.btnDeleteEntry.clicked.connect(self.RemoveFromDB)
            self.RoyaltiesWindow.table = dbTableWidget()
            self.RefreshTables()
            self.RoyaltiesWindow.table.setFixedSize(620, 150)
            self.RoyaltiesWindow.Royalties()
        self.CurrentTable = "Book"

    def ViewRoyaltyItems(self): #initialising the view royalty items window
        self.CurrentTable = "RoyaltyItems"
        self.RoyaltiesWindow.SelectedRow = self.RoyaltiesWindow.table.currentRow()
        self.holder = self.SelectedAuthorID
        self.SelectedAuthorID = self.SelectedID
        self.RoyaltiesWindow.selectedISBN = self.SelectedISBN
        self.SelectedID = QTableWidgetItem(self.RoyaltiesWindow.table.item(self.RoyaltiesWindow.SelectedRow, 0)).text()

        if self.SelectedID != "":
            self.RoyaltyItemsWindow = dbItems()
            self.RoyaltyItemsWindow.selectedID = self.SelectedID
            self.RoyaltyItemsWindow.selectedISBN = self.SelectedISBN
            self.RoyaltyItemsWindow.RoyaltyItemsButtons()
            self.RoyaltyItemsWindow.btnCalculate.clicked.connect(self.AddItem)
            self.RoyaltyItemsWindow.btnUpdateRoyaltyItems.clicked.connect(self.UpdateEntry)
            self.RoyaltyItemsWindow.btnDeleteEntry.clicked.connect(self.RemoveFromDB)
            self.RoyaltyItemsWindow.table = dbTableWidget()
            self.RefreshTables()
            self.RoyaltyItemsWindow.table.setFixedSize(620, 150)
            self.RoyaltyItemsWindow.RoyaltiesItems()
        self.SelectedID = self.SelectedAuthorID
        self.SelectedAuthorID = self.holder
        self.CurrentTable = "Royalties"
        self.RefreshTables()



    def BookInvoiceItemCalculation(self): #calculating the bookinvoicepayment
        try:
            if self.Editing == False:
                self.Quantity = int(self.AddWindow.inputList[2].text())
                self.Discount = float(self.AddWindow.inputList[3].text()) / 100
```

```
                self.ShippingPrice = float(self.AddWindow.inputList[5].text())
                self.ISBN = self.AddWindow.inputList[1].text()
            elif self.Editing == True:
                self.Quantity = int(self.EditWindow.inputList[2].text())
                self.Discount = float(self.EditWindow.inputList[3].text()) / 100
                self.ShippingPrice = float(self.EditWindow.inputList[5].text())
                self.ISBN = self.EditWindow.inputList[1].text()

            with sqlite3.connect("PP.db") as db: #fetching data from db
                cursor = db.cursor()
                cursor.execute("select Price from Book where ISBN = {}".format(self.ISBN))
                self.Price = list(cursor.fetchone())[0]
                db.commit()

            self.BookInvoiceItemPayment = (self.Quantity * self.Price)
            self.Discount = self.BookInvoiceItemPayment * self.Discount
            self.BookInvoiceItemPayment -= self.Discount
            self.BookInvoiceItemPayment += self.ShippingPrice
            self.BookInvoiceItemPayment = "£{0:.2f}".format(self.BookInvoiceItemPayment)
            if self.Editing == False:
                self.AddWindow.btnConfirm.clicked.connect(self.AddWindow.accept)
                self.AddWindow.qleCalculation.setText(self.BookInvoiceItemPayment)
            elif self.Editing == True:
                if self.EditWindow.Calculated == False:
                    self.EditWindow.Calculated = True #prevents duplicate connections
                    self.EditWindow.btnConfirm.clicked.connect(self.VerifyUpdate)
                self.EditWindow.qleCalculation.setText(self.BookInvoiceItemPayment)
        except:
            self.Msg = QMessageBox()
            self.Msg.setWindowTitle("Error")
            self.Msg.setText("You must fill all fields before clicking 'Calculate'.")
            self.Msg.exec_()

    def RoyaltyItemCalculation(self): #calculating the royalty payment
        try:
            if self.Editing == False:
                self.Currency = self.AddWindow.inputList[2].text()
                self.WholesalePrice = float(self.AddWindow.inputList[4].text())
                self.Quantity = int(self.AddWindow.inputList[5].text())
                self.PrintCost = float(self.AddWindow.inputList[6].text())
            elif self.Editing == True:
                self.Currency = self.EditWindow.inputList[2].text()
```

```
                self.WholesalePrice = float(self.EditWindow.inputList[4].text())
                self.Quantity = int(self.EditWindow.inputList[5].text())
                self.PrintCost = float(self.EditWindow.inputList[6].text())

            self.NetSales = self.WholesalePrice * self.Quantity
            self.RoyaltyItemPayment = self.NetSales - self.PrintCost
            self.RoyaltyItemPayment = "{0:.2f}".format(self.RoyaltyItemPayment)
            if self.Editing == False:
                self.AddWindow.btnConfirm.clicked.connect(self.AddWindow.accept)
                self.AddWindow.qleCalculation.setText("{}{}".format(self.Currency, self.RoyaltyItemPayment))
                self.AddWindow.NetSales = self.NetSales
            elif self.Editing == True:
                if self.EditWindow.Calculated == False:
                    self.EditWindow.Calculated = True #prevents duplicate connections
                    self.EditWindow.btnConfirm.clicked.connect(self.VerifyUpdate)
                self.EditWindow.qleCalculation.setText("{}{}".format(self.Currency, self.RoyaltyItemPayment))
                self.EditWindow.NetSales = self.NetSales
        except:
            self.Msg = QMessageBox()
            self.Msg.setWindowTitle("Error")
            self.Msg.setText("You must fill all fields before clicking 'Calculate'.")
            self.Msg.exec_()


    def UpdateCustomerEntry(self): #updating customer entries only
        self.SelectedRow = self.TableWidget.currentRow()
        self.SelectedAuthorID = QTableWidgetItem(self.TableWidget.item(self.SelectedRow, 0)).text()

        if self.SelectedAuthorID != "":
            self.UpdateEntryWindow = dbUpdateEntryWindow()
            self.UpdateEntryWindow.setFixedSize(640, 115)
            self.UpdateEntryWindow.selectedID = self.SelectedAuthorID
            self.UpdateEntryWindow.table = dbTableWidget()
            self.selectText = "Firstname, Lastname, Email, Phonenumber, Address, Postcode"
            self.UpdateEntryWindow.table.sql = "select {} from Customer where AuthorID = {}".format(self.selectText,
                self.SelectedAuthorID)
            self.SelectedID = self.SelectedAuthorID
            self.UpdateEntryWindow.table.initTable()
            self.UpdateEntryWindow.table.setFixedSize(617, 55)
            self.UpdateEntryWindow.Verify = dbConfirmationDialog()
            self.UpdateEntryWindow.Verify.Username = self.Username
            self.UpdateEntryWindow.initConfirmBtn()
```

```
            self.UpdateEntryWindow.btnConfirm.clicked.connect(self.VerifyCustomerUpdate)
            self.UpdateEntryWindow.initUpdateEntryWindowDlg()
            self.TableWidget.sql = "select * from Customer"
            self.RefreshTables()



    def UpdateEntry(self): #getting the update input
        self.Selection = False
        self.EditWindow = dbAddItemWindow() #uses the add window to init same interface but fill boxes with data
        self.EditWindow.setFixedSize(360, 200)
        self.EditWindow.AddType = self.CurrentTable
        self.EditWindow.AnswerButtons()
        self.EditWindow.ReadyToVerify = False
        self.EditWindow.originalItemList = []

        if self.CurrentTable == "Book":
            self.EditWindow.sql = "select * from Book"
            self.ViewWindow.SelectedRow = self.ViewWindow.table.currentRow()
            if self.ViewWindow.SelectedRow != -1:
                self.Selection = True
                with sqlite3.connect("PP.db") as db:
                    cursor = db.cursor() #fetching data from database for the edit window
                    sql = "select * from Book where ISBN =
                        {}".format(self.ViewWindow.table.item(self.ViewWindow.SelectedRow, 0).text())
                    cursor.execute(sql)
                    self.EditWindow.originalItemList = list(cursor.fetchone())

        elif self.CurrentTable == "PubInvoice":
            self.EditWindow.setFixedSize(400,150)
            self.EditWindow.selectedISBN = self.SelectedISBN
            self.EditWindow.sql = "select ISBN, AuthorID, PubInvoiceDate, PubInvoiceService, PubInvoicePayment from
                PubInvoice"
            self.PubInvoiceWindow.SelectedRow = self.PubInvoiceWindow.table.currentRow()
            if self.PubInvoiceWindow.SelectedRow != -1:
                self.Selection = True
                with sqlite3.connect("PP.db") as db:
                    cursor = db.cursor() #fetching data from database for the edit window
                    sql = "select ISBN, AuthorID, PubInvoiceDate, PubInvoiceService, PubInvoicePayment from PubInvoice
                        where PubInvoiceID =
                        {}".format(self.PubInvoiceWindow.table.item(self.PubInvoiceWindow.SelectedRow, 0).text())
                    cursor.execute(sql)
```

```
                self.EditWindow.originalItemList = list(cursor.fetchone())

        elif self.CurrentTable == "BookInvoice":
            self.EditWindow.setFixedSize(350, 100)
            self.EditWindow.selectedID = self.SelectedID
            self.EditWindow.sql = "select AuthorID, BookInvoiceDate from BookInvoice"
            self.BookInvoiceWindow.SelectedRow = self.BookInvoiceWindow.table.currentRow()
            if self.BookInvoiceWindow.SelectedRow != -1:
                self.Selection = True
                with sqlite3.connect("PP.db") as db:
                    cursor = db.cursor() #fetching data from database for the edit window
                    sql = "select AuthorID, BookInvoiceDate from BookInvoice where BookInvoiceID =
                        {}".format(self.BookInvoiceWindow.table.item(self.BookInvoiceWindow.SelectedRow, 0).text())
                    cursor.execute(sql)
                    self.EditWindow.originalItemList = list(cursor.fetchone())

        elif self.CurrentTable == "Royalties":
            self.EditWindow.setFixedSize(350, 100)
            self.EditWindow.selectedID = self.SelectedID
            self.EditWindow.sql = "select AuthorID, RoyaltiesDate from Royalties"
            self.RoyaltiesWindow.SelectedRow = self.RoyaltiesWindow.table.currentRow()
            if self.RoyaltiesWindow.SelectedRow != -1:
                self.Selection = True
                with sqlite3.connect("PP.db") as db:
                    cursor = db.cursor() #fetching data from database for the edit window
                    sql = "select AuthorID, RoyaltiesDate from Royalties where RoyaltiesID =
                        {}".format(self.RoyaltiesWindow.table.item(self.RoyaltiesWindow.SelectedRow, 0).text())
                    cursor.execute(sql)
                    self.EditWindow.originalItemList = list(cursor.fetchone())

        elif self.CurrentTable == "BookInvoiceItems":
            self.EditWindow.setFixedSize(450, 150)
            self.EditWindow.selectedID = self.SelectedID
            self.Editing = True
            self.EditWindow.btnCalculate.clicked.connect(self.BookInvoiceItemCalculation)
            self.EditWindow.selectedISBN = self.SelectedISBN
            self.EditWindow.sql = "select BookInvoiceID, ISBN, BookInvoiceQuantity, BookInvoiceDiscount, ShippingType,
                ShippingPrice from BookInvoiceItems"
            self.BookInvoiceItemsWindow.SelectedRow = self.BookInvoiceItemsWindow.table.currentRow()
            if self.BookInvoiceItemsWindow.SelectedRow != -1:
                self.Selection = True
                with sqlite3.connect("PP.db") as db:
```

```
                    cursor = db.cursor() #fetching data from database for the edit window
                    sql = "select BookInvoiceID, ISBN, BookInvoiceQuantity, BookInvoiceDiscount, ShippingType,
                        ShippingPrice from BookInvoiceItems where BookInvoiceItemsID =
                        {}".format(self.BookInvoiceItemsWindow.table.item(self.BookInvoiceItemsWindow.SelectedRow,
                        0).text())
                    cursor.execute(sql)
                    self.EditWindow.originalItemList = list(cursor.fetchone())

        elif self.CurrentTable == "RoyaltyItems":
            self.EditWindow.setFixedSize(450, 250)
            self.EditWindow.selectedID = self.SelectedID
            self.Editing = True
            self.EditWindow.btnCalculate.clicked.connect(self.RoyaltyItemCalculation)
            self.EditWindow.selectedISBN = self.SelectedISBN
            self.EditWindow.sql = "select RoyaltiesID, ISBN, Currency, RoyaltyDiscount, WholesalePrice, RoyaltyQuantity,
                PrintCost, ExcRateFromGBP from RoyaltyItems"
            self.RoyaltyItemsWindow.SelectedRow = self.RoyaltyItemsWindow.table.currentRow()
            if self.RoyaltyItemsWindow.SelectedRow != -1:
                self.Selection = True
                with sqlite3.connect("PP.db") as db:
                    cursor = db.cursor() #fetching data from database for the edit window
                    sql = "select RoyaltiesID, ISBN, Currency, RoyaltyDiscount, WholesalePrice, RoyaltyQuantity,
                        PrintCost, ExcRateFromGBP from RoyaltyItems where RoyaltyItemsID =
                        {}".format(self.RoyaltyItemsWindow.table.item(self.RoyaltyItemsWindow.SelectedRow, 0).text())
                    cursor.execute(sql)
                    self.EditWindow.originalItemList = list(cursor.fetchone())

        self.EditWindow.Editing = True

        if self.Selection == True: #initialising the edit window
            self.EditWindow.btnConfirm.clicked.connect(self.EditWindow.Validate)
            if self.CurrentTable in ["RoyaltyItems", "BookInvoiceItems"]:
                self.EditWindow.btnConfirm.clicked.connect(self.EditWindow.CheckCalculated)
            else:
                self.EditWindow.btnConfirm.clicked.connect(self.VerifyUpdate)
            self.EditWindow.AddType = self.CurrentTable
            self.EditWindow.selectedID = self.SelectedID

            self.EditWindow.CalendarWidget = dbCalendarWidget()
            self.EditWindow.CalendarWidget.Calendar()
            self.EditWindow.initAddItemWindow()

```

```python


    def VerifyCustomerUpdate(self):
        self.UpdateEntryWindow.Verify = dbConfirmationDialog()
        #new instance for customer verification



    def VerifyUpdate(self): #verification dialog
        if self.EditWindow.ReadyToVerify == True:
            self.EditWindow.Verify = dbConfirmationDialog()
            self.EditWindow.Verify.Msg = "Insert Password to confirm all changes"
            self.EditWindow.Verify.ConfirmedMsg = "Update successful"
            self.EditWindow.Verify.VerifyDlg()
            if self.EditWindow.Verify.ConfirmedDialog.Accepted == True:
                self.UpdateChanges()
                self.EditWindow.accept()



    def UpdateChanges(self): #committing changes
        self.UpdateList = []

        with sqlite3.connect("PP.db") as db:
            cursor = db.cursor()
            if self.CurrentTable == "Book":
                self.NoOfEntries = 12
                cursor.execute("select * from Book")
                self.ID = "ISBN"
                self.SelectedISBN = QTableWidgetItem(self.ViewWindow.table.item(self.ViewWindow.SelectedRow, 0)).text()
                self.SelectedID = self.SelectedISBN
                self.BookEdited = True

            elif self.CurrentTable == "PubInvoice":
                cursor.execute("select ISBN, AuthorID, PubInvoiceDate, PubInvoiceService, PubInvoicePayment from
                    PubInvoice")
                self.NoOfEntries = 5
                self.ID = "PubInvoiceID"
                self.OldID = self.SelectedID
                self.SelectedID = QTableWidgetItem(self.PubInvoiceWindow.table.item(self.PubInvoiceWindow.SelectedRow,
                    0)).text()

```

```
            elif self.CurrentTable == "BookInvoice":
                cursor.execute("select AuthorID, BookInvoiceDate from BookInvoice")
                self.NoOfEntries = 2
                self.ID = "BookInvoiceID"
                self.OldID = self.SelectedID
                self.SelectedID = QTableWidgetItem(self.BookInvoiceWindow.table.item(self.BookInvoiceWindow.SelectedRow,
                    0)).text()

            elif self.CurrentTable == "Royalties":
                cursor.execute("select AuthorID, RoyaltiesDate from Royalties")
                self.NoOfEntries = 2
                self.ID = "RoyaltiesID"
                self.OldID = self.SelectedID
                self.SelectedID = QTableWidgetItem(self.RoyaltiesWindow.table.item(self.RoyaltiesWindow.SelectedRow,
                    0)).text()

            elif self.CurrentTable == "BookInvoiceItems":
                cursor.execute("select BookInvoiceID, ISBN, BookInvoiceQuantity, BookInvoiceDiscount, ShippingType,
                    ShippingPrice from BookInvoiceItems")
                self.NoOfEntries = 6
                self.ID = "BookInvoiceItemsID"
                self.OldID = self.SelectedID
                self.SelectedID =
                    QTableWidgetItem(self.BookInvoiceItemsWindow.table.item(self.BookInvoiceItemsWindow.SelectedRow,
                    0)).text()

            elif self.CurrentTable == "RoyaltyItems":
                cursor.execute("select RoyaltiesID, ISBN, Currency, RoyaltyDiscount, WholesalePrice, RoyaltyQuantity,
                    PrintCost, ExcRateFromGBP from RoyaltyItems")
                self.NoOfEntries = 6
                self.ID = "RoyaltyItemsID"
                self.OldID = self.SelectedID
                self.SelectedID = QTableWidgetItem(self.RoyaltyItemsWindow.table.item(self.RoyaltyItemsWindow.SelectedRow,
                    0)).text()

        for count in range(0, self.NoOfEntries): #creating the update string for sql
            try:
                self.UpdateList.append(str(self.EditWindow.inputList[count].currentText()))
            except:
                if count == 8 and self.CurrentTable == "RoyaltyItems":
                    self.UpdateList.append(str(self.NetSales))
                else:
```

```
                    self.UpdateList.append(self.EditWindow.inputList[count].text())

        self.Update = ""

        self.Headers = list(cursor.description)
        for count in range(0, len(self.UpdateList)):
            self.Update += "{} = '{}'".format(list(self.Headers[count])[0], self.UpdateList[count])
            if count != len(self.UpdateList) - 1:
                self.Update += ", "

        with sqlite3.connect("PP.db") as db: #update
            cursor = db.cursor()
            if self.CurrentTable == "Book": #ISBN is primary key which may be changed, so all foreign keys will change
                 first for it
                try:
                    sql = "update PubInvoice set ISBN = {0} where ISBN = {1}".format(self.UpdateList[0], self.SelectedID)
                    cursor.execute(sql)
                    sql = "update BookInvoiceItems set ISBN = {0} where ISBN = {1}".format(self.UpdateList[0],
                         self.SelectedID)
                    cursor.execute(sql)
                    sql = "update RoyaltyItems set ISBN = {0} where ISBN = {1}".format(self.UpdateList[0], self.SelectedID)
                    cursor.execute(sql)
                except:
                    pass # error if no entry is there for pubinvoice/bookinvoiceitems/royaltyitems
            sql = "update {} set {} where {} = {}".format(self.CurrentTable, self.Update, self.ID, self.SelectedID)
            cursor.execute(sql)
            db.commit()

        if self.CurrentTable in ["Royalties", "BookInvoice", "PubInvoice", "BookInvoiceItems", "RoyaltyItems"]:
            self.SelectedID = self.OldID

        self.RefreshTables()

        try:
            self.RecalculateItems()
            self.BookEdited = False
        except:
            pass



    def RefreshTables(self): #refreshing tables
```

```
        if self.CurrentTable == "Customer":
            self.TableWidget.sql = "select * from Customer"
            self.TableWidget.initTable()

        if self.CurrentTable == "Book":
            self.ViewWindow.table.sql = "select * from Book where AuthorID = {}".format(self.SelectedAuthorID)
            self.ViewWindow.table.initTable()
            with sqlite3.connect("PP.db") as db:
                cursor = db.cursor()
                cursor.execute("select Firstname, Lastname from Customer where AuthorID =
                    {}".format(self.SelectedAuthorID))
                self.Name = list(cursor.fetchone())
                self.Name = "{}, {}".format(self.Name[1], self.Name[0])
                for count in range(0, self.TableWidget.rowCount()+1):
                    self.ViewWindow.table.setItem(count, 1, QTableWidgetItem(self.Name))
                self.ViewWindow.table.setHorizontalHeaderItem(1, QTableWidgetItem("Author"))
                db.commit()
        elif self.CurrentTable == "PubInvoice":
            self.PubInvoiceWindow.table.sql = "select * from PubInvoice where ISBN = {}".format(self.SelectedISBN)
            self.PubInvoiceWindow.table.initTable()
            with sqlite3.connect("PP.db") as db:
                cursor = db.cursor()
                cursor.execute("select BookTitle from Book where ISBN = {}".format(self.SelectedISBN))
                self.Title = "{}".format(list(cursor.fetchone())[0])
                for count in range(0, self.TableWidget.rowCount()+1):
                    self.PubInvoiceWindow.table.setItem(count, 1, QTableWidgetItem(self.Title))
                self.PubInvoiceWindow.table.setHorizontalHeaderItem(1, QTableWidgetItem("BookTitle"))
                cursor.execute("select Firstname, Lastname from Customer where AuthorID =
                    {}".format(self.SelectedAuthorID))
                self.Name = list(cursor.fetchone())
                self.Name = "{}, {}".format(self.Name[1], self.Name[0])
                for count in range(0, self.TableWidget.rowCount()+1):
                    self.PubInvoiceWindow.table.setItem(count, 2, QTableWidgetItem(self.Name))
                self.PubInvoiceWindow.table.setHorizontalHeaderItem(2, QTableWidgetItem("Author"))
                db.commit()

        elif self.CurrentTable == "BookInvoice":
            self.BookInvoiceWindow.table.sql = "select * from BookInvoice where AuthorID =
                {}".format(self.SelectedAuthorID)
            self.BookInvoiceWindow.table.initTable()
            with sqlite3.connect("PP.db") as db:
                cursor = db.cursor()
```

```
                cursor.execute("select Firstname, Lastname from Customer where AuthorID =
                    {}".format(self.SelectedAuthorID))
                self.Name = list(cursor.fetchone())
                self.Name = "{}, {}".format(self.Name[1], self.Name[0])
                for count in range(0, self.TableWidget.rowCount()+1):
                    self.BookInvoiceWindow.table.setItem(count, 1, QTableWidgetItem(self.Name))
                self.BookInvoiceWindow.table.setHorizontalHeaderItem(1, QTableWidgetItem("Author"))
                db.commit()

        elif self.CurrentTable == "Royalties":
            self.RoyaltiesWindow.table.sql = "select * from Royalties where AuthorID = {}".format(self.SelectedAuthorID)
            self.RoyaltiesWindow.table.initTable()
            with sqlite3.connect("PP.db") as db:
                cursor = db.cursor()
                cursor.execute("select Firstname, Lastname from Customer where AuthorID =
                    {}".format(self.SelectedAuthorID))
                self.Name = list(cursor.fetchone())
                self.Name = "{}, {}".format(self.Name[1], self.Name[0])
                for count in range(0, self.TableWidget.rowCount()+1):
                    self.RoyaltiesWindow.table.setItem(count, 1, QTableWidgetItem(self.Name))
                self.RoyaltiesWindow.table.setHorizontalHeaderItem(1, QTableWidgetItem("Author"))
                db.commit()

        elif self.CurrentTable == "BookInvoiceItems":
            self.BookInvoiceItemsWindow.table.sql = "select * from BookInvoiceItems where BookInvoiceID =
                {}".format(self.SelectedID)

            self.BookInvoiceItemsWindow.table.initTable()
            with sqlite3.connect("PP.db") as db:
                cursor = db.cursor()
                self.ID = []
                sql = "select BookInvoiceItemsID from BookInvoiceItems where BookInvoiceID = {}".format(self.SelectedID)
                cursor.execute(sql)
                self.IDList = list(cursor.fetchall())
                for count in range(0, len(self.IDList)):
                    self.temp = list(self.IDList[count])[0]
                    self.ID.append(self.temp)

                for count in range(0, len(self.ID)):
                    try:
                        cursor.execute("select Book.BookTitle, BookInvoiceItems.ISBN, Firstname, Lastname,
                            BookInvoiceItems.BookInvoiceID from Book, BookInvoiceItems, Customer, BookInvoice where
```

```
                                BookInvoiceItems.BookInvoiceID = {} and BookInvoiceItems.BookInvoiceItemsID = {} and
                                Book.ISBN = BookInvoiceItems.ISBN and BookInvoice.BookInvoiceID =
                                BookInvoiceItems.BookInvoiceID and Customer.AuthorID =
                                BookInvoice.AuthorID".format(self.SelectedID, self.ID[count]))
                            self.Title = list(cursor.fetchone())
                            self.Name = "{}, {}".format(self.Title[3], self.Title[2])
                            self.Title = self.Title[0]
                            self.BookInvoiceItemsWindow.table.setItem(count, 2, QTableWidgetItem(str(self.Title)))
                            self.BookInvoiceItemsWindow.table.setItem(count, 1, QTableWidgetItem(str(self.Name)))
                        except:
                            pass
                    self.BookInvoiceItemsWindow.table.setHorizontalHeaderItem(2, QTableWidgetItem("BookTitle"))
                    self.BookInvoiceItemsWindow.table.setHorizontalHeaderItem(1, QTableWidgetItem("Author"))
                    db.commit()

        elif self.CurrentTable == "RoyaltyItems":
            self.RoyaltyItemsWindow.table.sql = "select * from RoyaltyItems where RoyaltiesID = {}".format(self.SelectedID)
            self.RoyaltyItemsWindow.table.initTable()
            with sqlite3.connect("PP.db") as db:
                cursor = db.cursor()
                self.ID = []
                sql = "select RoyaltyItemsID from RoyaltyItems where RoyaltiesID = {}".format(self.SelectedID)
                cursor.execute(sql)
                self.IDList = list(cursor.fetchall())
                for count in range(0, len(self.IDList)):
                    self.temp = list(self.IDList[count])[0]
                    self.ID.append(self.temp)

                for count in range(0, len(self.ID)):
                    try:
                        cursor.execute("select Book.BookTitle, RoyaltyItems.ISBN, Firstname, Lastname,
                                RoyaltyItems.RoyaltiesID from Book, RoyaltyItems, Customer, Royalties where
                                RoyaltyItems.RoyaltiesID = {} and RoyaltyItems.RoyaltyItemsID = {} and Book.ISBN =
                                RoyaltyItems.ISBN and Royalties.RoyaltiesID = RoyaltyItems.RoyaltiesID and Customer.AuthorID
                                = Royalties.AuthorID".format(self.SelectedID, self.ID[count]))
                        self.Title = list(cursor.fetchone())
                        self.Name = "{}, {}".format(self.Title[3], self.Title[2])
                        self.Title = self.Title[0]
                        self.RoyaltyItemsWindow.table.setItem(count, 2, QTableWidgetItem(str(self.Title)))
                        self.RoyaltyItemsWindow.table.setItem(count, 1, QTableWidgetItem(str(self.Name)))
                    except:
                        pass
```

```
                self.RoyaltyItemsWindow.table.setHorizontalHeaderItem(2, QTableWidgetItem("BookTitle"))
                self.RoyaltyItemsWindow.table.setHorizontalHeaderItem(1, QTableWidgetItem("Author"))
                db.commit()

    def Back(self): #going back from the view window to the main menu
        self.ViewWindow.table.selectedID = None
        self.CurrentTable = "Customer"
        self.StackedLayout.setCurrentIndex(0)
        self.MenuBar.setVisible(True)



    def QuickSearch(self): #quick search from main menu
        self.QSText = self.MainMenuButtons.leQuickSearch.text()
        self.QSText = self.QSText.split(' ')
        with sqlite3.connect("PP.db") as db:
            cursor = db.cursor()

            if len(self.QSText) == 1:
                self.TableWidget.sql = "select * from Customer where Firstname like '{0}%' or Lastname like
                    '{0}%'".format(self.QSText[0])
            elif len(self.QSText) == 0:
                self.TableWidget.sql = "select * from Customer"
            else:
                for count in range(1, len(self.QSText)):
                    if count != 1:
                        self.QSText[1] += " {}".format(self.QSText[count])
                self.TableWidget.sql = "select * from Customer where Firstname like '{0}%' and Lastname like
                    '{1}%'".format(self.QSText[0], self.QSText[1])
            self.TableWidget.initTable()



    def Search(self): #main search interface
        self.SearchDatabase = dbSearchDatabase()
        self.SearchDatabase.Table = None
        self.SearchDatabase.Valid = False
        self.SearchDatabase.CalendarWidget = dbCalendarWidget()
        self.SearchDatabase.CalendarWidget.Calendar()
        self.SearchDatabase.initLayout()
        self.errorIndex = False
        if self.SearchDatabase.Valid == True:
```

```
943             if self.SearchDatabase.Table != None:
944                 try:
945                     if self.SearchDatabase.Table == "Book":
946                         for count in range(0, len(self.SearchDatabase.Results) +1):
947                             try:#using the results to fetch the correct data
948                                 if count == 0:
949                                     self.SearchTable.sql = "select * from {} where ISBN =
                                            '{}'".format(self.SearchDatabase.Table,
                                            list(self.SearchDatabase.Results[count])[2])
950                                 else:
951                                     self.SearchTable.sql += " or ISBN =
                                            '{}'".format(list(self.SearchDatabase.Results[count])[2])
952
953                             except IndexError:
954
955                                 self.SearchTable.setHorizontalHeaderItem(1, QTableWidgetItem("Author"))
956                                 with sqlite3.connect("PP.db") as db:
957                                     cursor = db.cursor() #fetching firstname and lastname using foreign key
958                                     for count2 in range(0, self.SearchTable.rowCount()):
959                                         cursor.execute("select Firstname, Lastname, Book.AuthorID, Customer.AuthorID from
                                                Customer, Book where Book.ISBN = {} and Customer.AuthorID =
                                                Book.AuthorID".format(list(self.SearchDatabase.Results[count2])[2]))
960                                         self.Name = list(cursor.fetchone())
961                                         self.Name = "{}, {}".format(self.Name[1], self.Name[0])
962                                         self.SearchTable.setItem(count2, 1, QTableWidgetItem(self.Name))
963                                     db.commit()
964                                 self.SearchTable.initTable()
965
966
967
968                     if self.SearchDatabase.Table in ["RoyaltyItems", "BookInvoiceItems"]:
969                         index = 4
970                     else:
971                         index = 1
972
973                     if self.SearchDatabase.Table == "Customer":
974                         for count in range(0, len(self.SearchDatabase.Results)+1):
975                             try: #using the results to fetch the correct data
976                                 if count == 0:
977                                     self.SearchTable.sql = "select * from Customer where AuthorID =
                                            '{}'".format(list(self.SearchDatabase.Results[count])[0])
978                                 else:
```

```
                                self.SearchTable.sql += " or AuthorID =
                                    '{}'".format(list(self.SearchDatabase.Results[count])[0])
                            self.SearchTable.initTable()

                        except IndexError:
                            self.errorIndex = True
                            self.SearchTable.initTable()


                    elif self.SearchDatabase.Table != "Book" and self.SearchDatabase.Table != None:
                        for count in range(0, len(self.SearchDatabase.Results)+1):
                            try: #using the results to fetch the correct data
                                if count == 0:
                                    self.SearchTable.sql = "select * from {0} where {0}ID =
                                        '{1}'".format(self.SearchDatabase.Table,
                                        list(self.SearchDatabase.Results[count])[index])
                                else:
                                    self.SearchTable.sql += " or {0}ID = '{1}'".format(self.SearchDatabase.Table,
                                        list(self.SearchDatabase.Results[count])[index])
                                self.SearchTable.initTable()

                            except IndexError:
                                self.errorIndex = True
                                with sqlite3.connect("PP.db") as db:
                                    cursor = db.cursor() #fetching firstnames and lastnames and book titles using foreign
                                        keys
                                    if self.SearchDatabase.Table in ["Royalties", "BookInvoice"]:
                                        self.SearchTable.setHorizontalHeaderItem(1, QTableWidgetItem("Author"))
                                        for count2 in range(0, self.SearchTable.rowCount()):
                                            cursor.execute("select Firstname, Lastname, {0}.AuthorID, Customer.AuthorID
                                                from Customer, {0} where Customer.AuthorID = {1} and {0}.AuthorID =
                                                Customer.AuthorID".format(self.SearchDatabase.Table,
                                                list(self.SearchDatabase.Results[count2])[0]))
                                            self.Name = list(cursor.fetchone())
                                            self.Name = "{}, {}".format(self.Name[1], self.Name[0])
                                            self.SearchTable.setItem(count2, 1, QTableWidgetItem(self.Name))

                                    elif self.SearchDatabase.Table in ["RoyaltyItems", "BookInvoiceItems"]:
                                        self.SearchTable.setHorizontalHeaderItem(1, QTableWidgetItem("Author"))
                                        self.SearchTable.setHorizontalHeaderItem(2, QTableWidgetItem("Book Title"))
                                        if self.SearchDatabase.Table == "RoyaltyItems":
                                            self.IDType = "Royalties"
```

```
                                    else:
                                        self.IDType = "BookInvoice"
                                    for count2 in range(0, self.SearchTable.rowCount()):
                                        cursor.execute("select BookTitle, Book.ISBN, {0}.ISBN, {1}.AuthorID,
                                            Firstname, LastName from Book, {0}, {1}, Customer where Book.ISBN = {2}
                                            and {0}.ISBN = Book.ISBN and {1}.{1}ID = {0}.{1}ID and Customer.AuthorID
                                            = {1}.AuthorID".format(self.SearchDatabase.Table, self.IDType,
                                            list(self.SearchDatabase.Results[count2])[2]))
                                        self.Title = cursor.fetchone()
                                        self.Name = "{}, {}".format(self.Title[5], self.Title[4])
                                        self.Title = "{}".format(list(self.Title)[0])
                                        self.SearchTable.setItem(count2, 2, QTableWidgetItem(self.Title))
                                        self.SearchTable.setItem(count2, 1, QTableWidgetItem(self.Name))
                                elif self.SearchDatabase.Table == "PubInvoice":
                                    self.SearchTable.setHorizontalHeaderItem(1, QTableWidgetItem("Book Title"))
                                    self.SearchTable.setHorizontalHeaderItem(2, QTableWidgetItem("Author"))
                                    for count2 in range(0, self.SearchTable.rowCount()):
                                        cursor.execute("select Firstname, Lastname, BookTitle, {0}.AuthorID,
                                            Customer.AuthorID, Book.ISBN, {0}.ISBN from Customer, {0}, Book where
                                            Customer.AuthorID = {1} and {0}.AuthorID = Customer.AuthorID and
                                            Book.AuthorID = Customer.AuthorID and {0}.ISBN =
                                            Book.ISBN".format(self.SearchDatabase.Table,
                                            list(self.SearchDatabase.Results[count2])[0]))
                                        self.Name = list(cursor.fetchone())
                                        self.Title = "{}".format(self.Name[2])
                                        self.Name = "{}, {}".format(self.Name[1], self.Name[0])
                                        self.SearchTable.setItem(count2, 2, QTableWidgetItem(self.Name))
                                        self.SearchTable.setItem(count2, 1, QTableWidgetItem(self.Title))

                                db.commit()


                    self.StackedLayout.setCurrentIndex(2)
                    self.MenuBar.setVisible(False)

                except:
                    self.errorIndex = True

                if self.errorIndex == True:
                    self.StackedLayout.setCurrentIndex(0)
                    self.Msg = QMessageBox()
                    self.Msg.setWindowTitle("No Results Found")
```

```
                    self.Msg.setText("No data matches your search")
                    self.Msg.exec_()


    def keyReleaseEvent(self, QKeyEvent):
        if self.MainMenuButtons.leQuickSearch.text() == "":
            self.TableWidget.sql = "select * from Customer"
            self.TableWidget.initTable()


    def ChangeUsernameOrPassword(self):

        self.ChangeWhat = dbUsernameOrPassword()
        self.ChangeWhat.Selection = None
        self.ChangeWhat.ChangeSelection()
        if self.ChangeWhat.Selection == "Username":
            self.UsernameChange = dbChangeUsername()
            self.UsernameChange.Username = self.Username
            self.UsernameChange.initChangeUsernameScreen()
        elif self.ChangeWhat.Selection == "Password":
            self.PasswordChange = dbChangePassword()
            self.PasswordChange.Username = self.Username
            self.PasswordChange.initChangePasswordScreen()

    def LogOut(self):
        self.hide()
        subprocess.call("LoginDB.py", shell=True)
```

### 4.10.3 Module 3 - Initialising Main Menu Layout

```
from PyQt4.QtCore import *
from PyQt4.QtGui import *
import sqlite3
import sys


class initMainMenuButtons(QWidget):
    """initialising the main menu buttons"""

```

```
    def __init__(self):
        super().__init__()

        #creating the layout

        self.vertical = QVBoxLayout()

        self.btnLogOut = QPushButton("Log Out", self)
        self.btnLogOut.setFixedSize(100, 30)

        self.leQuickSearch = QLineEdit(self)
        self.leQuickSearch.setPlaceholderText("Author Name")
        self.leQuickSearch.setFixedSize(100, 25)

        self.btnQuickSearch = QPushButton("Quick Search", self)
        self.btnQuickSearch.setFixedSize(100, 30)

        self.horizontalTop = QHBoxLayout()
        self.horizontalTop.addWidget(self.btnLogOut)
        self.horizontalTop.addStretch(1)
        self.horizontalTop.addWidget(self.leQuickSearch)
        self.horizontalTop.addWidget(self.btnQuickSearch)

        self.btnView = QPushButton("View", self)
        self.btnView.setFixedSize(100, 40)

        self.btnSearchdb = QPushButton("Search Database", self)
        self.btnSearchdb.setFixedSize(100, 40)

        self.btnAddEntry = QPushButton("Add Entry", self)
        self.btnAddEntry.setFixedSize(100, 40)

        self.btnUpdateEntry = QPushButton("Update Entry", self)
        self.btnUpdateEntry.setFixedSize(100, 40)

        self.btnRemoveEntry = QPushButton("Remove Entry", self)
        self.btnRemoveEntry.setFixedSize(100, 40)

        self.btnChangePassword = QPushButton("Change Username/\nPassword", self)
        self.btnChangePassword.setFixedSize(100, 40)

        self.horizontalBottom = QHBoxLayout()
```

```python

        self.horizontalBottom.addWidget(self.btnView)
        self.horizontalBottom.addWidget(self.btnSearchdb)
        self.horizontalBottom.addWidget(self.btnAddEntry)
        self.horizontalBottom.addWidget(self.btnUpdateEntry)
        self.horizontalBottom.addWidget(self.btnRemoveEntry)
        self.horizontalBottom.addWidget(self.btnChangePassword)
```

### 4.10.4 Module 4 - Initialising Menu Bar

```python
from PyQt4.QtCore import *
from PyQt4.QtGui import *
import sys

class dbMenuBar(QMenuBar):
    """the menu bar"""

    def __init__(self):
        super().__init__()

        self.search_database = QAction("Search Database", self) #creating actions
        self.add_entry = QAction("Add Entry", self)
        self.update_entry = QAction("Update Entry", self)
        self.remove_entry = QAction("Remove Entry", self)
        self.change_password = QAction("Change Username/Password", self)
        self.log_out = QAction("Log Out", self)

        self.database_menu = self.addMenu("Database") #adding menus
        self.database_menu.addAction(self.search_database) #adding actions to menus

        self.actions_menu = self.addMenu("Actions")
        self.actions_menu.addAction(self.add_entry)
        self.actions_menu.addAction(self.update_entry)
        self.actions_menu.addAction(self.remove_entry)

        self.account_menu = self.addMenu("Account")
        self.account_menu.addAction(self.change_password)
        self.account_menu.addAction(self.log_out)
```

### 4.10.5 Module 5 - Initialising a Table Widget

```
from PyQt4.QtCore import *
from PyQt4.QtGui import *
import sys
import sqlite3

class dbTableWidget(QTableWidget):
    """main table widget"""

    def __init__(self):
        super().__init__()


    def initTable(self):
        self.clear()
        self.columns = []
        self.setSelectionBehavior(QAbstractItemView.SelectRows)
        self.setEditTriggers(QAbstractItemView.NoEditTriggers)
        self.setSortingEnabled(False)
        with sqlite3.connect("PP.db") as db: #fetching data from db
            cursor = db.cursor()
            cursor.execute(self.sql)
            self.ColumnNames = cursor.description
        for count in range(0, len(self.ColumnNames)):
            self.columns.append(list(list(self.ColumnNames)[count])[0])
        self.setRowCount(0)
        self.setColumnCount(len(self.columns))
        self.setHorizontalHeaderLabels(self.columns)
        for self.row, self.form in enumerate(cursor):
            self.insertRow(self.row)
            for self.column, self.unit in enumerate(self.form):
                self.setItem(self.row, self.column, QTableWidgetItem(str(self.unit)))
        self.setSortingEnabled(True)
```

### 4.10.6 Module 6 - Initialising Add Customer Entry Window

```
from PyQt4.QtCore import *
from PyQt4.QtGui import *
```

```
import sqlite3
import sys
import re

class dbAddEntryWindow(QDialog):
    """add entry window dialog"""

    def __init__(self):
        super().__init__()

    def initAddEntryWindow(self):
        self.setWindowTitle("Add Entry")
        self.setFixedSize(640,115)
        self.setModal(True) #modal window
        self.AddEntryTable = QTableWidget(self) #table for adding customer entry
        self.AddEntryTable.setRowCount(1)
        self.AddEntryTable.setColumnCount(6)
        self.AddEntryTable.setFixedSize(617, 55)
        self.inputList = []
        for count in range(0, 6):
            #0 firstname, 1 lastname, 2 email, 3 phoneno, 4 address, 5 postcode
            self.input = QLineEdit(self)
            self.input.setFrame(False)
            self.inputList.append(self.input)
            self.AddEntryTable.setCellWidget(0, count, self.input)

        self.inputList[0].setValidator(QRegExpValidator(QRegExp("[a-zA-Z\-\!]+")))
        self.inputList[1].setValidator(QRegExpValidator(QRegExp("[a-zA-Z\-\!]+")))
        self.inputList[2].setValidator(QRegExpValidator(QRegExp("^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$")))
        self.regexp = "^((((\+44\s?\d{4}|\(?0\d{4}\)?)\s?\d{3}\s?\d{3})|((\+44\s?\d{3}|\(?0\d{3}\)?)\s?\d{3}\s?\d{4})|"
        self.regexp2 = "((\+44\s?\d{2}|\(?0\d{2}\)?)\s?\d{4}\s?\d{4}))(\s?\#(\d{4}|\d{3}))?$"
        self.inputList[3].setValidator(QRegExpValidator(QRegExp("{}{}".format(self.regexp, self.regexp2))))
        self.inputList[4].setValidator(QRegExpValidator(QRegExp("[a-zA-Z \d\-\.]+")))
        self.inputList[5].setValidator(QRegExpValidator(QRegExp("^[a-zA-Z]{1,2}[0-9][0-9A-Za-z]{0,1}
            {0,1}[0-9][A-Za-z]{2}$")))
        #2, 3 & 5 from www.regexlib.com
        self.CustomerHeaders = ["Firstname", "Lastname", "Email", "Phonenumber", "Address", "Postcode"]
        self.AddEntryTable.setHorizontalHeaderLabels(self.CustomerHeaders)
        self.btnConfirm = QPushButton("Confirm", self) #buttons
        self.btnCancel = QPushButton("Cancel", self)
        self.horizontal = QHBoxLayout()
        self.horizontal.addStretch(1)
```

```
        self.horizontal.addWidget(self.btnCancel)
        self.horizontal.addWidget(self.btnConfirm)

        self.vertical = QVBoxLayout() #vbox layout
        self.vertical.addWidget(self.AddEntryTable)
        self.vertical.addStretch(1)
        self.vertical.addLayout(self.horizontal)
        self.setLayout(self.vertical)

        self.btnCancel.clicked.connect(self.reject) #reject on clicking cancel
        self.btnConfirm.clicked.connect(self.AddEntryTodb) #call function after clicking confirm
        self.exec_()

    def AddEntryTodb(self):
        #fetching inputs from table
        self.Valid = True
        self.Message = "All Fields must be filled."
        self.Firstname = self.inputList[0].text()
        self.Lastname = self.inputList[1].text()
        self.Email = self.inputList[2].text()
        self.Phonenumber = self.inputList[3].text()
        self.Address = self.inputList[4].text()
        self.Postcode = self.inputList[5].text()

        self.input_data = (self.Firstname, self.Lastname, self.Email, self.Phonenumber, self.Address, self.Postcode)
        for count in range(0, len(list(self.input_data))):
            if list(self.input_data)[count].replace(" ", "") == "":
                self.Valid = False
        try:
            float(self.input_data[4])
            self.Valid = False
            self.Message = "Invalid Address"
        except:
            pass

        if self.Valid == True:
            with sqlite3.connect("PP.db") as db:
                cursor = db.cursor()
                cursor.execute("PRAGMA foreign_keys = ON")
                sql = "insert into Customer (FirstName, LastName, Email, PhoneNumber, Address, Postcode) values (?, ?, ?,
                    ?, ?, ?)"
                cursor.execute(sql, self.input_data)
```

```
                db.commit()
                self.accept()
        else:
            self.Msg = QMessageBox()
            self.Msg.setWindowTitle("Invalid Entry")
            self.Msg.setText(self.Message)
            self.Msg.exec_()
```

### 4.10.7 Module 7 - Initialising Update Customer Entry Window

```
from PyQt4.QtCore import *
from PyQt4.QtGui import *
import sqlite3
import sys
from ConfirmationDialog import *

class dbUpdateEntryWindow(QDialog):
    """update entry window dialog"""

    def __init__(self):
        super().__init__()

    def initUpdateEntryWindowDlg(self):
        self.table.setSelectionBehavior(QAbstractItemView.SelectItems)
        self.setWindowTitle("Update Entry")

        self.setModal(True)
        self.btnEdit = QPushButton("Edit", self)
        #self.btnConfirm = QPushButton("Confirm", self)
        self.horizontal = QHBoxLayout()
        self.horizontal.addWidget(self.btnEdit)
        self.horizontal.addStretch(1)
        self.horizontal.addWidget(self.btnConfirm)
        self.vertical = QVBoxLayout()
        self.vertical.addWidget(self.table)
        self.vertical.addLayout(self.horizontal)
        self.setLayout(self.vertical)
        self.btnEdit.clicked.connect(self.Edit)
        self.btnConfirm.clicked.connect(self.Verification)
```

```
        self.TableName = "Customer"
        self.ID = "AuthorID"
        self.exec_()

    def initConfirmBtn(self): #creating confirm btn for instantiation of verification window
        self.btnConfirm = QPushButton("Confirm", self)

    def Verification(self):

        self.Verify.Msg = "Insert Password to confirm all changes"
        self.Verify.ConfirmedMsg = "Update successful"
        self.Verify.VerifyDlg()
        if self.Verify.ConfirmedDialog.Accepted == True:
            self.UpdateChanges()
            self.accept()

    def Edit(self):
        self.SelectedItem = self.table.currentItem()
        self.SelectedRow = self.table.currentRow()
        self.SelectedColumn = self.table.currentColumn()
        if self.SelectedItem != None:
            self.EditDlg = dbUpdateEntryWindow()
            self.EditDlg.setModal(True)
            self.EditDlg.setWindowTitle("Input Text")
            self.EditDlg.setFixedSize(210, 120)
            self.EditDlg.lbl = QLabel("Insert text to save over current entry", self)
            self.EditDlg.qle = QLineEdit(self)
            self.EditDlg.qle.setPlaceholderText("Insert text here")
            self.EditDlg.btnConfirm = QPushButton("Confirm", self)
            self.EditDlg.btnConfirm.setFixedSize(60, 25)

            self.EditDlg.qlehorizontal = QHBoxLayout()
            self.EditDlg.btnhorizontal = QHBoxLayout()
            self.EditDlg.qlehorizontal.addStretch(1)
            self.EditDlg.qlehorizontal.addWidget(self.EditDlg.qle)
            self.EditDlg.qlehorizontal.addStretch(1)
            self.EditDlg.btnhorizontal.addStretch(1)
            self.EditDlg.btnhorizontal.addWidget(self.EditDlg.btnConfirm)
            self.EditDlg.btnhorizontal.addStretch(1)
            self.EditDlg.vertical = QVBoxLayout()
            self.EditDlg.vertical.addWidget(self.EditDlg.lbl)
            self.EditDlg.vertical.addLayout(self.EditDlg.qlehorizontal)
```

```
            self.EditDlg.vertical.addLayout(self.EditDlg.btnhorizontal)
            self.EditDlg.setLayout(self.EditDlg.vertical)
            self.EditDlg.btnConfirm.clicked.connect(self.EditDlg.accept)
            self.EditDlg.btnConfirm.clicked.connect(self.GetInput)
            self.EditDlg.exec_()

    def GetInput(self):
        self.EditInput = self.EditDlg.qle.text()
        self.table.setItem(self.SelectedRow, self.SelectedColumn, QTableWidgetItem(self.EditInput))


    def UpdateChanges(self):
        UL = [] #UL = Update List
        for count in range(0, 6):
            UL.append(self.table.item(0, count).text())
        self.Update = "Firstname = '{}', Lastname = '{}', Email = '{}', Phonenumber = '{}', Address = '{}', Postcode =
             '{}'".format(UL[0], UL[1], UL[2], UL[3], UL[4], UL[5])
        with sqlite3.connect("PP.db") as db:
            cursor = db.cursor()
            cursor.execute("PRAGMA foreign_keys = ON")
            sql = "update {} set {} where {} = {}".format(self.TableName, self.Update, self.ID, self.selectedID)
            cursor.execute(sql)
            db.commit()
```

### 4.10.8 Module 8 - Initialising View Menu Layout

```
from PyQt4.QtCore import *
from PyQt4.QtGui import *
import sqlite3
import sys

class dbViewWindow(QWidget):
    """generic view window"""

    def __init__(self):
        super().__init__()

    def View(self):
        self.setWindowTitle("View Menu")
```

```
        self.setFixedSize(735,400)

        self.btnBack = QPushButton("Back", self)
        self.btnBack.setFixedSize(100, 30)
        self.btnViewRoyalties = QPushButton("View Royalties", self)
        self.btnViewRoyalties.setFixedSize(100, 40)
        self.btnViewBookInvoices = QPushButton("View Book  \n Invoices", self)
        self.btnViewBookInvoices.setFixedSize(100, 40)
        self.btnViewPubInvoice = QPushButton("View Publishing \n Invoice", self)
        self.btnViewPubInvoice.setFixedSize(100, 40)
        self.btnAddBook = QPushButton("Add Book", self)
        self.btnAddBook.setFixedSize(100, 40)
        self.btnUpdateBook = QPushButton("Update Book", self)
        self.btnUpdateBook.setFixedSize(100,40)
        self.btnDeleteBook = QPushButton("Delete Book", self)
        self.btnDeleteBook.setFixedSize(100, 40)

        self.horizontalTop = QHBoxLayout()
        self.horizontalTop.addStretch(1)
        self.horizontalTop.addWidget(self.btnBack)

        self.horizontalBottom = QHBoxLayout()
        self.horizontalBottom.addWidget(self.btnViewPubInvoice)
        self.horizontalBottom.addWidget(self.btnViewBookInvoices)
        self.horizontalBottom.addWidget(self.btnViewRoyalties)
        self.horizontalBottom.addWidget(self.btnAddBook)
        self.horizontalBottom.addWidget(self.btnUpdateBook)
        self.horizontalBottom.addWidget(self.btnDeleteBook)

        self.vertical = QVBoxLayout()
```

### 4.10.9 Module 9 - Initialising Add Entry Window for non-customer entries

```
from PyQt4.QtCore import *
from PyQt4.QtGui import *
import sqlite3
import sys
import re

```

```python

class dbAddItemWindow(QDialog):
    """add entry window dialog"""

    def __init__(self):
        super().__init__()

    def initAddItemWindow(self):
        self.setWindowTitle("Add {}".format(self.AddType))
        if self.Editing == True:
            self.setWindowTitle("Edit {}".format(self.AddType))
        self.ReadyToVerify = True
        self.setModal(True) #modal window
        self.Calculated = False
        with sqlite3.connect("PP.db") as db: #fetching data from db
            cursor = db.cursor()
            cursor.execute(self.sql)
            db.commit()

        self.Columns = []
        self.ColumnNames = cursor.description

        for count in range(0, len(self.ColumnNames)):
            self.Columns.append(list(list(self.ColumnNames)[count])[0])
        self.coordinates = []

        for count in range(int(round(len(self.Columns)/2, 1) + 1)):
            for count2 in range(4):
                    self.coordinates.append((count, count2))

        self.ColumnLength = len(self.Columns)
        for count in range(0, self.ColumnLength):
            self.Columns.insert((count * 2) + 1, "")

        db.close()
        self.inputList = []
        self.qlabelList = []
        self.gridLayout = QGridLayout()
        count = 0

        for self.coordinate, self.columnHeader in zip(self.coordinates, self.Columns):

```

```
            if self.columnHeader == "": #replacing spaces with line edits

                if self.AddType == "Book" and count in [0, 1, 3, 4, 5, 6, 7, 9, 10, 11]:
                    if count == 0:
                        self.input = QLineEdit(self)
                        self.input.setValidator(QRegExpValidator(QRegExp("[\w]+")))
                    elif count == 1: #exceptions for book
                        self.input = QLineEdit(self) #new line edit
                        self.input.setReadOnly(True)
                        self.input.setText(self.selectedID)
                    elif count in [3, 9, 11]:
                        self.input = QLineEdit(self)
                        if count == 3:
                            self.input.setValidator(QIntValidator())
                        else:
                            self.input.setValidator(QDoubleValidator())
                    elif count in [4, 5, 6, 7]: #exceptions for book where combobox is needed
                        self.input = QComboBox(self) #new combo box

                    elif count == 10:
                        self.input = QLineEdit(self)
                        self.input.setReadOnly(True)

                elif self.AddType == "PubInvoice" and count in [0, 1, 2, 3, 4]:

                    if count == 0: #setting  isbn ineditable
                        self.input = QLineEdit(self)
                        self.input.setReadOnly(True)
                        self.input.setText(self.selectedISBN)

                    elif count == 1: #setting authorID ineditable
                        self.input = QLineEdit(self)
                        self.input.setReadOnly(True)
                        self.input.setText(self.selectedID)

                    elif count == 2:
                        self.input = QLineEdit(self)
                        self.input.setReadOnly(True)

                    elif count == 3:
                        self.input = QComboBox(self)

```

```
            elif count == 4:
                self.input = QLineEdit(self)
                self.input.setValidator(QDoubleValidator())

        elif self.AddType in ["BookInvoice", "Royalties"] and count in [0, 1]:

            self.input = QLineEdit(self)
            self.input.setReadOnly(True)

            if count == 0:
                self.input.setText(self.selectedID)
        elif self.AddType == "BookInvoiceItems" and count in [0, 1, 2, 3, 4, 5]:

            if count == 0:
                self.input = QLineEdit(self)
                self.input.setReadOnly(True)
                self.input.setText(self.selectedID)
            elif count == 1:
                self.input = QLineEdit(self)
                self.input.setReadOnly(True)
                self.input.setText(self.selectedISBN)
            elif count in [2, 3, 5]:
                self.input = QLineEdit(self)
                if count == 2:
                    self.input.setValidator(QIntValidator())
                else:
                    self.input.setValidator(QDoubleValidator())
            elif count == 4:
                self.input = QComboBox(self)

        elif self.AddType == "RoyaltyItems" and count in [0, 1, 3, 4, 5, 6, 7]:
            with sqlite3.connect("PP.db") as db:
                cursor = db.cursor()
                Selection = "NoOfPages, Size, Back"
                sql = "select {} from Book where ISBN = {}".format(Selection, self.selectedISBN)
                cursor.execute(sql)
                self.SelectionList = list(cursor.fetchone())
                self.NoOfPages = int(self.SelectionList[0])
                self.Size = self.SelectionList[1]
                self.Back = self.SelectionList[2]

                if self.Size == "Large":
```

```
                            self.PagePrice = 0.015 * self.NoOfPages
                            if self.Back == "Hard":
                                self.CoverPrice = 5
                            elif self.Back == "Soft":
                                self.CoverPrice = 1

                        elif self.Size == "Small":
                            self.PagePrice = 0.01 * self.NoOfPages
                            if self.Back == "Hard":
                                self.CoverPrice = 4
                            elif self.Back == "Soft":
                                self.CoverPrice = 0.7

                    self.input = QLineEdit(self)

                    if count == 0:
                        self.input.setText(self.selectedID)
                        self.input.setReadOnly(True)
                    elif count == 1:
                        self.input.setText(self.selectedISBN)
                        self.input.setReadOnly(True)
                    elif count in [3, 4, 5, 6, 7]:
                        if count == 5:
                            self.input.setValidator(QIntValidator())
                        elif count == 6:
                            self.input.setReadOnly(True)
                            self.input.setValidator(QDoubleValidator())
                        else:
                            self.input.setValidator(QDoubleValidator())
                else:
                    self.input = QLineEdit(self) #new line edit #standard input method

                self.inputList.append(self.input)   #line edits/combo boxes appended to list for further reference
                self.gridLayout.addWidget(self.inputList[count], *self.coordinate)

                count += 1

            else: #adding qlabels with the line edits
                self.DateEntry = False
                if self.AddType == "Book" and count == 10: #adding date button instead of qlabel
                    self.DateEntry = True
                elif self.AddType == "PubInvoice" and count == 2: #data button instead of qlabel
```

```
                    self.DateEntry = True
                elif self.AddType == "BookInvoice" and count == 1:
                    self.DateEntry = True
                elif self.AddType == "Royalties" and count == 1:
                    self.DateEntry = True

                else:
                    if str(self.columnHeader) == "PubInvoiceService":
                        self.qlabel = QLabel("Service", self)
                    elif str(self.columnHeader) in ["PubInvoicePayment", "BookInvoicePayment", "RoyaltyPayment"]:
                        self.qlabel = QLabel("Payment", self)
                    elif str(self.columnHeader) in ["BookInvoiceDiscount", "RoyaltyDiscount"]:
                        self.qlabel = QLabel("Discount", self)
                    elif str(self.columnHeader) in ["BookInvoiceQuantity", "RoyaltyQuantity"]:
                        self.qlabel = QLabel("Quantity", self)
                    elif str(self.columnHeader) == "ExcRateFromGBP":
                        self.qlabel = QLabel("£1 = ", self)
                    elif str(self.columnHeader)[-2:] in ["ID", "BN"]:
                        self.qlabel = QLabel(str(self.columnHeader))

                    else:
                        self.Label = str(self.columnHeader)
                        self.LabelLength = len(self.Label)
                        self.CamelCase = False
                        for count2 in range(1, self.LabelLength):
                            if self.CamelCase == True:
                                pass
                            else:
                                self.CamelCase = False
                            if self.Label[count2].isupper() == True:
                                self.String1 = re.sub('(.)([A-Z][a-z]+)', r'\1 \2', self.Label[0:count2])
                                self.String2 = re.sub('(.)([A-Z][a-z]+)', r'\1 \2', self.Label[count2:self.LabelLength])
                                self.tempLabel = "{} {}".format(self.String1, self.String2)
                                self.CamelCase = True

                        if self.CamelCase == True:
                            self.Label = self.tempLabel

                        self.qlabel = QLabel(str(self.Label), self)
                    self.qlabelList.append(self.qlabel)
                    self.gridLayout.addWidget(self.qlabelList[count], *self.coordinate)

```

```
                if self.DateEntry == True:
                    self.btnDate = QPushButton("Date", self)
                    self.qlabelList.append(self.btnDate)
                    self.gridLayout.addWidget(self.qlabelList[count], *self.coordinate)
                    self.btnDate.clicked.connect(self.CalendarWidget.DisplayCalendar)
                    self.CalendarWidget.btnSelect.clicked.connect(self.getDate)

        if self.AddType == "Book":
            self.inputList[4].addItem("Large")
            self.inputList[4].addItem("Small")
            self.inputList[5].addItem("Hard")
            self.inputList[5].addItem("Soft")
            self.inputList[6].addItem("Matte")
            self.inputList[6].addItem("Gloss")
            self.inputList[7].addItem("White")
            self.inputList[7].addItem("Creme")

        elif self.AddType == "PubInvoice":
            self.inputList[3].addItem("Standard")
            self.inputList[3].addItem("Enhanced")
            self.inputList[3].addItem("Colour Publishing")
            self.inputList[3].addItem("Reprint")

        elif self.AddType == "BookInvoiceItems":
            self.inputList[4].addItem("Rush")
            self.inputList[4].addItem("Premium")
            self.inputList[4].addItem("Standard")
            self.inputList[4].addItem("Economy")
            self.inputList[4].addItem("International")

        if self.Editing == True:

            if self.AddType == "Book":

                for count in range(0, 12):
                    try: #for line edits
                        self.inputList[count].setText(str(self.originalItemList[count]))
                    except: #for comboboxes
                        self.originalIndex = self.inputList[count].findText(str(self.originalItemList[count]))
                        self.inputList[count].setCurrentIndex(self.originalIndex)

            elif self.AddType == "PubInvoice":
```

```
                for count in range(0, 5):
                    try:
                        self.inputList[count].setText(str(self.originalItemList[count]))
                    except:
                        self.originalIndex = self.inputList[count].findText(str(self.originalItemList[count]))
                        self.inputList[count].setCurrentIndex(self.originalIndex)

            elif self.AddType in ["BookInvoice", "Royalties"]:

                for count in range(0, 2):
                    self.inputList[count].setText(str(self.originalItemList[count]))
            elif self.AddType == "BookInvoiceItems":
                for count in range(0, 6):
                    try:
                        self.inputList[count].setText(str(self.originalItemList[count]))
                    except:
                        self.originalIndex = self.inputList[count].findText(str(self.originalItemList[count]))
                        self.inputList[count].setCurrentIndex(self.originalIndex)
            elif self.AddType == "RoyaltyItems":
                for count in range(0, 8):
                    self.inputList[count].setText(str(self.originalItemList[count]))

        self.horizontal = QHBoxLayout()
        if self.AddType in ["BookInvoiceItems", "RoyaltyItems"]:
            self.horizontal.addWidget(self.btnCalculate)
            self.horizontal.addWidget(self.qleCalculation)
        self.horizontal.addStretch(1)
        self.horizontal.addWidget(self.btnCancel)
        self.horizontal.addWidget(self.btnConfirm)
        if self.AddType in ["BookInvoiceItems", "RoyaltyItems"]:
            if self.Editing == False:
                self.btnConfirm.clicked.connect(self.CheckCalculated)
        if self.Editing == False:
            self.btnConfirm.clicked.connect(self.Validate)
        self.vertical = QVBoxLayout()
        self.vertical.addLayout(self.gridLayout)
        self.vertical.addLayout(self.horizontal)
        self.setLayout(self.vertical)


        self.btnCancel.clicked.connect(self.reject) #reject on clicking cancel
```

```
        self.exec_()

    def CheckCalculated(self):
        self.ReadyToVerify = False
        if len(self.qleCalculation.text()) == 0:
            self.Msg = QMessageBox()
            self.Msg.setWindowTitle("Calculation")
            self.Msg.setText("You must fill all fields and click 'Calculate' before attempting to add to the database.")
            self.Msg.exec_()
        else:
            self.ReadyToVerify = True

    def AnswerButtons(self): #so connections can be made outside of this class
        self.btnConfirm = QPushButton("Confirm", self)
        self.btnCancel = QPushButton("Cancel", self)
        if self.AddType in ["BookInvoiceItems", "RoyaltyItems"]:
            self.btnCalculate = QPushButton("Calculate", self)
            self.qleCalculation = QLineEdit(self)
            self.qleCalculation.setReadOnly(True)

    def getDate(self):
        self.CalendarWidget.date = self.CalendarWidget.qle.text()
        if self.AddType == "Book":
            self.inputList[10].setText(self.CalendarWidget.date)

        elif self.AddType == "PubInvoice":
            self.inputList[2].setText(self.CalendarWidget.date)

        elif self.AddType in ["BookInvoice", "Royalties"]:
            self.inputList[1].setText(self.CalendarWidget.date)

    def keyReleaseEvent(self, QKeyEvent):
        if self.AddType == "RoyaltyItems":
            if self.inputList[2].text() == "£":
                self.inputList[7].setText("N/A")
                self.inputList[7].setReadOnly(True)
            elif self.inputList[7].text() == "N/A":
                self.inputList[7].setText("")
                self.inputList[7].setReadOnly(False)
            if self.inputList[5].text() != "":
                self.PrintCost = (self.PagePrice + self.CoverPrice) * int(self.inputList[5].text())
                self.inputList[6].setText("{0:.2f}".format(self.PrintCost))
```

```
            elif self.inputList[5].text() == "":
                self.inputList[6].clear()



    def Validate(self):
        if self.ReadyToVerify == True:
            self.input_data = []
            self.Valid = True
            for count in range(0, len(self.inputList)):
                try: #gathering the input data
                    self.input_data.append(str(self.inputList[count].currentText()))
                except:
                    if count == 8 and self.AddType == "RoyaltyItems":
                        self.input_data.append(self.NetSales)
                    else:
                        self.input_data.append(self.inputList[count].text())

            for count in range(0, len(self.input_data)):

                if str(self.input_data[count]).replace(" ", "") == "": #presence check
                    self.Valid = False
                    self.ErrorMessage = "All Fields must be filled."
                    break
                try:
                    if float(self.input_data[count]) < 0: #range check
                        self.Valid = False
                        self.ErrorMessage = "Invalid Entry - Please check the fields."
                        break
                except:
                    pass

                if self.qlabelList[count].text() == "ISBN":
                    if len(self.input_data[count]) != 13 and len(self.input_data[count]) != 10: #isbn must be 10 or 13
                         digits
                        self.Valid = False
                        self.ErrorMessage = "Invalid ISBN - Must be 10 or 13 digits."
                        break
                elif self.qlabelList[count].text() == "No Of Pages":
                    if int(self.input_data[count]) > 2000:
                        self.Valid = False
                        self.ErrorMessage = "Invalid Entry - Please check the fields."
```

```
                        break
                elif self.qlabelList[count].text() == "Discount": #%'s must be between 0 and 100
                    if float(self.input_data[count]) > 100 or float(self.input_data[count]) < 0:
                        self.Valid = False
                        self.ErrorMessage = "Invalid Entry - Please check the fields."
                        break

            if self.Valid == False:
                self.Msg = QMessageBox()
                self.Msg.setWindowTitle("Invalid Entry")
                self.Msg.setText(self.ErrorMessage)
                self.Msg.exec_()
            else:
                if self.AddType not in ["BookInvoiceItems", "RoyaltyItems"]:
                    if self.Editing == False:
                        self.accept()
                    else:
                        self.ReadyToVerify = True
```



### 4.10.10 Module 10 - Initialising Calendar Widget

```
import sys
from PyQt4.QtCore import *
from PyQt4.QtGui import *

class dbCalendarWidget(QDialog):

    def __init__(self):
        super().__init__()

    def Calendar(self):
        self.setFixedSize(265, 275)
        self.setWindowTitle('Calendar')
        calendar = QCalendarWidget(self)
        calendar.setGridVisible(True)
        calendar.clicked[QDate].connect(self.date)
        date = calendar.selectedDate()
        self.lblInstruction = QLabel(self)
        self.lblInstruction.setText("Please select a date")
```

```
        self.lblInstruction.setAlignment(Qt.AlignCenter)
        self.qle = QLineEdit(self)
        self.qle.setText(date.toString("dd-MM-yyyy"))
        self.qle.setFixedSize(85,20)
        self.qle.setAlignment(Qt.AlignCenter)
        self.btnSelect = QPushButton("Select", self)
        self.btnCancel = QPushButton("Cancel", self)

        self.horizontalTop = QHBoxLayout()
        self.horizontalTop.addStretch(1)
        self.horizontalTop.addWidget(self.lblInstruction)
        self.horizontalTop.addStretch(1)

        self.horizontalMid1 = QHBoxLayout()
        self.horizontalMid1.addStretch(1)
        self.horizontalMid1.addWidget(calendar)
        self.horizontalMid1.addStretch(1)

        self.horizontalMid2 = QHBoxLayout()
        self.horizontalMid2.addStretch(1)
        self.horizontalMid2.addWidget(self.qle)
        self.horizontalMid2.addStretch(1)

        self.horizontalBottom = QHBoxLayout()
        self.horizontalBottom.addWidget(self.btnCancel)
        self.horizontalBottom.addStretch(1)
        self.horizontalBottom.addWidget(self.btnSelect)

        self.vertical = QVBoxLayout()
        self.vertical.addLayout(self.horizontalTop)
        self.vertical.addLayout(self.horizontalMid1)
        self.vertical.addLayout(self.horizontalMid2)
        self.vertical.addLayout(self.horizontalBottom)
        self.setLayout(self.vertical)


    def DisplayCalendar(self):
        self.setModal(True)
        self.btnSelect.clicked.connect(self.accept)
        self.btnCancel.clicked.connect(self.reject)
        self.exec_()

```

```
    def date(self, date):
        self.qle.setText(date.toString("dd-MM-yyyy"))
```

### 4.10.11 Module 11 - Initialising a Confirmation Dialog

```
from PyQt4.QtCore import *
from PyQt4.QtGui import *
import sqlite3
import sys

class dbConfirmationDialog(QDialog):
    """creating confirmation modal dialogs"""

    def __init__(self):
        super().__init__()
        self.Prevention = False

    def VerifyDlg(self):
        self.setWindowTitle("Verification")
        self.setFixedSize(275, 150)
        self.setModal(True)
        self.lblWarningMsg = QLabel(self.Msg, self)
        self.horizontal = QHBoxLayout()
        self.horizontal.addStretch(1)
        self.horizontal.addWidget(self.lblWarningMsg)
        self.horizontal.addStretch(1)
        self.lblWarningMsg.setFixedSize(250,30)
        self.lblWarningMsg.setWordWrap(True)
        self.lblWarningMsg.setAlignment(Qt.AlignHCenter)

        self.qlePasswordBox = QLineEdit(self)
        self.qlePasswordBox.setFixedSize(100, 25)
        self.qlePasswordBox.setEchoMode(self.qlePasswordBox.Password)
        self.lblPassword = QLabel("Password: ", self)

        self.horizontal1 = QHBoxLayout()
        self.horizontal1.addStretch(1)
        self.horizontal1.addWidget(self.lblPassword)
        self.horizontal1.addWidget(self.qlePasswordBox)
```

```
        self.horizontal1.addStretch(1)

        self.btnConfirm = QPushButton("Confirm", self)
        self.btnCancel = QPushButton("Cancel", self)
        self.horizontal2 = QHBoxLayout()
        self.horizontal2.addWidget(self.btnCancel)
        self.horizontal2.addWidget(self.btnConfirm)

        self.vertical = QVBoxLayout()
        self.vertical.addLayout(self.horizontal)
        self.vertical.addLayout(self.horizontal1)
        self.vertical.addLayout(self.horizontal2)
        self.vertical.addStretch(1)
        self.setLayout(self.vertical)


        self.btnCancel.clicked.connect(self.reject)
        self.ConfirmedDialog = dbConfirmationDialog()
        self.ConfirmedDialog.ConfirmedMsg = self.ConfirmedMsg
        self.lblInvalid = None
        self.btnConfirm.clicked.connect(self.PasswordCheck)

        self.ConfirmedDialog.Accepted = False
        self.exec_()


    def PasswordCheck(self):
        with sqlite3.connect("dbLogin.db") as db:
            cursor = db.cursor()
            cursor.execute("select Password from LoginDetails")
            self.Password = list(cursor.fetchall())
            self.Valid = False

            for count in range(0, len(self.Password)):
                if self.qlePasswordBox.text() == list(self.Password[count])[0]:
                    self.accept()
                    self.ConfirmedDialog.Accepted = True
                    if self.Prevention == False:
                        self.ConfirmedDialog.Confirmed()
                    break
                else:
                    self.Valid = False
```

```
            if self.Valid == False:
                if self.lblInvalid == None:
                    self.lblInvalid = QLabel("Invalid Username or Password - Please try again.", self)
                    self.lblInvalid.setWordWrap(True)
                    self.lblInvalid.setAlignment(Qt.AlignHCenter)
                    self.horizontalInvalid = QHBoxLayout()
                    self.horizontalInvalid.addStretch(1)
                    self.horizontalInvalid.addWidget(self.lblInvalid)
                    self.horizontalInvalid.addStretch(1)
                    self.vertical.addLayout(self.horizontalInvalid)
                else:
                    self.lblInvalid.show()


    def Confirmed(self):
        self.setWindowTitle("Confirmation")
        self.setFixedSize(275, 100)
        self.setModal(True)

        self.lblConfirmed = QLabel(self.ConfirmedMsg, self)
        self.lblConfirmed.setFixedSize(250, 50)
        self.lblConfirmed.setWordWrap(True)
        self.lblConfirmed.setAlignment(Qt.AlignHCenter)

        self.btnOk = QPushButton("OK", self)
        self.btnOk.setFixedSize(75, 30)
        self.btnOk.clicked.connect(self.accept)

        self.horizontal = QHBoxLayout()
        self.horizontal.addStretch(1)
        self.horizontal.addWidget(self.btnOk)
        self.horizontal.addStretch(1)

        self.vertical = QVBoxLayout()
        self.vertical.addWidget(self.lblConfirmed)
        self.vertical.addStretch(1)
        self.vertical.addLayout(self.horizontal)
        self.setLayout(self.vertical)
        self.exec_()

    def keyReleaseEvent(self, QKeyEvent):
```

```
        if self.lblInvalid != None:
            self.lblInvalid.hide()
```

### 4.10.12 Module 12 - Initialising a dialog for viewing the Items, and creating calculations for the payments.

```
from PyQt4.QtCore import *
from PyQt4.QtGui import *
import sqlite3
import sys

class dbItems(QDialog):
    """viewing royalty and invoice items"""

    def __init__(self):
        super().__init__()
        self.BookEdited = False
    def BookInvoiceItems(self):
        self.setWindowTitle("View Book Invoice Items")
        self.setModal(True)
        self.setFixedSize(640, 220)
        self.vertical = QVBoxLayout(self)
        self.vertical.addWidget(self.table)
        self.btnCalculate.setFixedSize(100, 40)
        self.btnUpdateBookInvoiceItems.setFixedSize(100, 40)
        self.btnDeleteEntry.setFixedSize(100, 40)
        self.horizontal = QHBoxLayout()
        self.horizontal.addWidget(self.btnCalculate)
        self.horizontal.addStretch(1)
        self.horizontal.addWidget(self.btnUpdateBookInvoiceItems)
        self.horizontal.addStretch(1)
        self.horizontal.addWidget(self.btnDeleteEntry)
        self.vertical.addLayout(self.horizontal)
        self.setLayout(self.vertical)

        self.exec_()

    def BookInvoiceItemsButtons(self):
        self.btnCalculate = QPushButton("Calculate", self)
```

```
        self.btnUpdateBookInvoiceItems = QPushButton("Update Book \n Invoice Items", self)
        self.btnDeleteEntry = QPushButton("Delete \n Entry", self)

    def RoyaltiesItems(self):
        self.setWindowTitle("View Royalty Items")
        self.setModal(True)
        self.setFixedSize(640, 220)
        self.vertical = QVBoxLayout(self)
        self.vertical.addWidget(self.table)
        self.btnCalculate.setFixedSize(100, 40)
        self.btnUpdateRoyaltyItems.setFixedSize(100, 40)
        self.btnDeleteEntry.setFixedSize(100, 40)
        self.horizontal = QHBoxLayout()
        self.horizontal.addWidget(self.btnCalculate)
        self.horizontal.addStretch(1)
        self.horizontal.addWidget(self.btnUpdateRoyaltyItems)
        self.horizontal.addStretch(1)
        self.horizontal.addWidget(self.btnDeleteEntry)
        self.vertical.addLayout(self.horizontal)
        self.setLayout(self.vertical)

        self.exec_()

    def RoyaltyItemsButtons(self):
        self.btnCalculate = QPushButton("Calculate", self)
        self.btnUpdateRoyaltyItems = QPushButton("Update \n Royalty Items", self)
        self.btnDeleteEntry = QPushButton("Delete \n Entry", self)


    def CalculateBookInvoiceItems(self):
        with sqlite3.connect("PP.db") as db:
            cursor = db.cursor()
            cursor.execute("PRAGMA foreign_keys_ = ON")
            self.Empty = False
            self.BookInvoicePayment = 0

            self.IDList = []
            sql = "select BookInvoiceItemsID from BookInvoiceItems where BookInvoiceID = {}".format(self.selectedID)
            cursor.execute(sql)
            self.IDListTuple = list(cursor.fetchall())
            for count in range(0, len(self.IDListTuple)): #conversion from tuple
                self.IDList.append(list(self.IDListTuple[count])[0])
```

```
            self.currentID = 0
            for count in range(0, len(self.IDList)):
                if self.currentID < self.IDList[count]: #finding biggest ID no.
                    self.currentID = self.IDList[count]

            sql = "select ISBN from BookInvoiceItems where BookInvoiceID = {}".format(self.selectedID)
            cursor.execute(sql)
            self.ISBNs = list(cursor.fetchall())

            for count2 in range(0, len(self.ISBNs)):
                if self.currentID != 0:
                    for count in range(0, self.currentID +1):
                        self.Empty = False
                        Selection = "BookInvoiceItems.BookInvoiceQuantity, BookInvoiceItems.BookInvoiceDiscount,
                            BookInvoiceItems.ShippingPrice, Book.Price"
                        Tables = "BookInvoiceItems, Book"
                        sql = "select {} from {} where BookInvoiceItems.BookInvoiceID = {} and
                            BookInvoiceItems.BookInvoiceItemsID = {} and BookInvoiceItems.ISBN = {} and Book.ISBN =
                            BookInvoiceItems.ISBN".format(Selection, Tables, self.selectedID, count+1,
                            list(self.ISBNs[count2])[0])
                        cursor.execute(sql)
                        try:
                            self.SelectionList = list(cursor.fetchone())
                        except:
                            self.Empty = True

                        if self.Empty != True:
                            self.Quantity = self.SelectionList[0]
                            self.Discount = self.SelectionList[1] / 100
                            self.ShippingPrice = self.SelectionList[2]
                            self.Price = self.SelectionList[3]

                            self.TempPayment = (self.Quantity * self.Price)
                            self.Discount *= self.TempPayment
                            self.TempPayment -= self.Discount
                            self.TempPayment += self.ShippingPrice

                            self.BookInvoicePayment += self.TempPayment


                elif self.currentID == 0:
```

```
                    self.BookInvoicePayment = None
                    sql = "update BookInvoice set BookInvoicePayment = '{}' where BookInvoiceID =
                        {}".format(self.BookInvoicePayment, self.selectedID)
                    cursor.execute(sql)
                    db.commit()

            if self.currentID != 0:
                self.BookInvoicePayment = "£{0:.2f}".format(self.BookInvoicePayment)
                sql = "update BookInvoice set BookInvoicePayment = '{}' where BookInvoiceID =
                    {}".format(self.BookInvoicePayment, self.selectedID)
                cursor.execute(sql)
                db.commit()

    def CalculateRoyaltyItems(self):
        with sqlite3.connect("PP.db") as db:
            cursor = db.cursor()
            cursor.execute("PRAGMA foreign_keys = ON")
            self.Empty = False
            self.RoyaltyPayment = 0
            self.Currency = ""
            self.IDList = []
            sql = "select RoyaltyItemsID from RoyaltyItems where RoyaltiesID = {}".format(self.selectedID)
            cursor.execute(sql)
            self.IDListTuple = list(cursor.fetchall())
            for count in range(0, len(self.IDListTuple)): #conversion from tuple
                self.IDList.append(list(self.IDListTuple[count])[0])

            self.currentID = 0
            for count in range(0, len(self.IDList)):
                if self.currentID < self.IDList[count]: #finding biggest ID no. for iteration limit
                    self.currentID = self.IDList[count]

            sql = "select ISBN from RoyaltyItems where RoyaltiesID = {}".format(self.selectedID)
            cursor.execute(sql)
            self.ISBNs = list(cursor.fetchall())

            for count2 in range(0, len(self.ISBNs)):
                if self.currentID != 0:
                    for count in range(0, self.currentID+1):
                        self.Empty = False
                        Selection = "RoyaltyItems.Currency, RoyaltyItems.NetSales, RoyaltyItems.ExcRateFromGBP,
                            RoyaltyItems.RoyaltyQuantity, Book.NoOfPages, Book.Size, Book.Cover, Book.Back"
```

```
                    Tables = "RoyaltyItems, Book"
                    sql = "select {} from {} where RoyaltyItems.RoyaltiesID = {} and RoyaltyItems.RoyaltyItemsID = {}
                         and RoyaltyItems.ISBN = {} and Book.ISBN = RoyaltyItems.ISBN".format(Selection, Tables,
                         self.selectedID, count+1, list(self.ISBNs[count2])[0])
                    cursor.execute(sql)

                    try:
                        self.SelectionList = list(cursor.fetchone())
                    except:
                       self.Empty = True

                    if self.Empty != True:
                        self.Currency = self.SelectionList[0]
                        self.NetSales = float(self.SelectionList[1])
                        self.Quantity = self.SelectionList[3]
                        self.NoOfPages = int(self.SelectionList[4])
                        self.Size = self.SelectionList[5]
                        self.Cover = self.SelectionList[6]
                        self.Back = self.SelectionList[7]

                        if self.Size == "Large":
                            self.PagePrice = 0.015 * self.NoOfPages
                            if self.Back == "Hard":
                                self.CoverPrice = 5
                            elif self.Back == "Soft":
                                self.CoverPrice = 1
                        elif self.Size == "Small":
                            self.PagePrice = 0.01 * self.NoOfPages
                            if self.Back == "Hard":
                                self.CoverPrice = 4
                            elif self.Back == "Soft":
                                self.CoverPrice = 0.7

                        self.PrintCost = (self.PagePrice + self.CoverPrice) * self.Quantity

                        self.TempPayment= self.NetSales - self.PrintCost
                        if self.Currency != "£":
                            self.ExcRateFromGBP = float(self.SelectionList[2])
                            self.TempPayment /= self.ExcRateFromGBP
                        self.RoyaltyPayment += self.TempPayment

                elif self.currentID == 0:
```

```
                    self.BookInvoicePayment = None
                    sql = "update Royalties set RoyaltyPayment = '{}' where RoyaltiesID = {}".format(self.RoyaltyPayment,
                        self.selectedID)
                    cursor.execute(sql)
                    db.commit()

            if self.currentID != 0:
                self.RoyaltyPayment = "£{0:.2f}".format(self.RoyaltyPayment)
                sql = "update Royalties set RoyaltyPayment = '{}' where RoyaltiesID = {}".format(self.RoyaltyPayment,
                    self.selectedID)
                cursor.execute(sql)
                db.commit()
```

### 4.10.13 Module 13 - Initialising a dialog for viewing Royalties and Invoices

```
from PyQt4.QtCore import *
from PyQt4.QtGui import *
import sqlite3
import sys

class dbRoyaltiesAndInvoices(QDialog):
    """viewing royalties and invoices"""

    def __init__(self):
        super().__init__()

    def PubInvoice(self):
        self.setWindowTitle("View Publishing Invoices")
        self.setModal(True)
        self.setFixedSize(640, 220)
        self.vertical = QVBoxLayout(self)
        self.vertical.addWidget(self.table)
        self.btnAddPubInvoice.setFixedSize(100, 40)
        self.btnUpdatePubInvoice.setFixedSize(100, 40)
        self.btnDeleteEntry.setFixedSize(100, 40)
        self.horizontal = QHBoxLayout()
        self.horizontal.addWidget(self.btnAddPubInvoice)
        self.horizontal.addStretch(1)
        self.horizontal.addWidget(self.btnUpdatePubInvoice)
```

```
        self.horizontal.addStretch(1)
        self.horizontal.addWidget(self.btnDeleteEntry)
        self.vertical.addLayout(self.horizontal)
        self.setLayout(self.vertical)
        self.exec_()

    def PubInvoiceButtons(self):
        self.btnAddPubInvoice = QPushButton("Add Publishing \n Invoice", self)
        self.btnUpdatePubInvoice = QPushButton("Update Publishing \n Invoice", self)
        self.btnDeleteEntry = QPushButton("Delete \n Entry", self)

    def BookInvoice(self):
        self.setWindowTitle("View Book Invoices")
        self.setModal(True)
        self.setFixedSize(640, 220)
        self.vertical = QVBoxLayout(self)
        self.vertical.addWidget(self.table)
        self.btnViewBookInvoiceItems.setFixedSize(100, 40)
        self.btnAddBookInvoice.setFixedSize(100, 40)
        self.btnUpdateBookInvoice.setFixedSize(100, 40)
        self.btnDeleteEntry.setFixedSize(100, 40)
        self.horizontal = QHBoxLayout()
        self.horizontal.addWidget(self.btnViewBookInvoiceItems)
        self.horizontal.addStretch(1)
        self.horizontal.addWidget(self.btnAddBookInvoice)
        self.horizontal.addStretch(1)
        self.horizontal.addWidget(self.btnUpdateBookInvoice)
        self.horizontal.addStretch(1)
        self.horizontal.addWidget(self.btnDeleteEntry)
        self.vertical.addLayout(self.horizontal)
        self.setLayout(self.vertical)
        self.exec_()

    def BookInvoiceButtons(self):
        self.btnViewBookInvoiceItems = QPushButton("View Book \n Invoice Items", self)
        self.btnAddBookInvoice = QPushButton("Add Book \n Invoice", self)
        self.btnUpdateBookInvoice = QPushButton("Update Book \n Invoice", self)
        self.btnDeleteEntry = QPushButton("Delete \n Entry", self)

    def Royalties(self):
        self.setWindowTitle("View Royalties")
        self.setModal(True)
```

```
        self.setFixedSize(640, 220)
        self.vertical = QVBoxLayout(self)
        self.vertical.addWidget(self.table)
        self.btnViewRoyaltyItems.setFixedSize(100, 40)
        self.btnAddRoyalties.setFixedSize(100, 40)
        self.btnUpdateRoyalties.setFixedSize(100, 40)
        self.btnDeleteEntry.setFixedSize(100, 40)
        self.horizontal = QHBoxLayout()
        self.horizontal.addWidget(self.btnViewRoyaltyItems)
        self.horizontal.addStretch(1)
        self.horizontal.addWidget(self.btnAddRoyalties)
        self.horizontal.addStretch(1)
        self.horizontal.addWidget(self.btnUpdateRoyalties)
        self.horizontal.addStretch(1)
        self.horizontal.addWidget(self.btnDeleteEntry)
        self.vertical.addLayout(self.horizontal)
        self.setLayout(self.vertical)
        self.exec_()

    def RoyaltiesButtons(self):
        self.btnViewRoyaltyItems = QPushButton("View Royalty \n Items", self)
        self.btnAddRoyalties = QPushButton("Add \n Royalties", self)
        self.btnUpdateRoyalties = QPushButton("Update \n Royalties", self)
        self.btnDeleteEntry = QPushButton("Delete \n Entry", self)
```

### 4.10.14 Module 14 - Initialising a Search Window

```
from PyQt4.QtCore import *
from PyQt4.QtGui import *
import sqlite3
import sys


class dbSearchDatabase(QDialog):
    """initialising the detailed search window"""

    def __init__(self):
        super().__init__()

```

```
    def initLayout(self):
        self.setWindowTitle("Search")
        self.gridLayout = QGridLayout()
        self.gridLayout.setVerticalSpacing(10)
        self.gridLayout.setHorizontalSpacing(10)
        self.setFixedSize(400, 200)
        self.leFirstname = QLineEdit(self)
        self.leLastname = QLineEdit(self)
        self.cbTable = QComboBox(self)
        self.cbCategory = QComboBox(self)
        self.btnDate = QPushButton("Add Date", self)
        self.leSearch = QLineEdit(self)
        self.btnCancel = QPushButton("Cancel", self)
        self.btnSearch = QPushButton("Search", self)
        self.btnDate.setFixedSize(75, 20)
        self.lblFirstname = QLabel("Author Firstname:")
        self.lblLastname = QLabel("Author Lastname:")
        self.gridLayout.addWidget(self.leFirstname, 0, 3)
        self.gridLayout.addWidget(self.leLastname, 1, 3)
        self.gridLayout.addWidget(self.lblFirstname, 0, 2, Qt.AlignRight)
        self.gridLayout.addWidget(self.lblLastname, 1, 2, Qt.AlignRight)
        self.gridLayout.addWidget(self.cbTable, 0, 0)
        self.gridLayout.addWidget(self.cbCategory, 2, 0)
        self.gridLayout.addWidget(self.btnDate, 2, 2, Qt.AlignHCenter)
        self.gridLayout.addWidget(self.leSearch, 2, 3)
        self.gridLayout.addWidget(self.btnCancel, 3, 2)
        self.gridLayout.addWidget(self.btnSearch, 3, 3)
        self.setLayout(self.gridLayout)
        self.cbTable.addItem("Author")
        self.cbTable.addItem("Book")
        self.cbTable.addItem("Publishing Invoice")
        self.cbTable.addItem("Book Invoice")
        self.cbTable.addItem("Book Invoice Items")
        self.cbTable.addItem("Royalties")
        self.cbTable.addItem("Royalty Items")
        self.cbTable.activated[str].connect(self.ChangeCategories)
        self.cbCategory.activated[str].connect(self.DateButton)
        self.btnDate.hide()
        self.leSearch.hide()
        self.CalendarWidget.btnSelect.clicked.connect(self.getDate)
        self.btnDate.clicked.connect(self.CalendarWidget.DisplayCalendar)
        self.leSearch.setFixedSize(self.leSearch.sizeHint())
```

```
        self.leFirstname.setFixedSize(self.leFirstname.sizeHint())
        self.leLastname.setFixedSize(self.leLastname.sizeHint())
        self.btnSearch.clicked.connect(self.getSearchData)
        self.btnCancel.clicked.connect(self.reject)
        self.exec_()

    def ChangeCategories(self):
        self.btnDate.hide()
        self.leSearch.show()
        self.cbCategory.clear()
        if self.cbTable.currentText() == "Author":
            self.leSearch.hide()
        elif self.cbTable.currentText() == "Book":
            self.cbCategory.addItem("Book Title")
            self.cbCategory.addItem("Price")
            self.cbCategory.addItem("Date Published")

        elif self.cbTable.currentText() == "Publishing Invoice":
            self.cbCategory.addItem("Service")
            self.cbCategory.addItem("Date")

        elif self.cbTable.currentText() == "Book Invoice":
            self.cbCategory.addItem("Date")
            self.btnDate.show()

        elif self.cbTable.currentText() == "Book Invoice Items":
            self.cbCategory.addItem("Quantity")
            self.cbCategory.addItem("Discount")
            self.cbCategory.addItem("Shipping Type")

        elif self.cbTable.currentText() == "Royalties":
            self.cbCategory.addItem("Date")
            self.btnDate.show()

        elif self.cbTable.currentText() == "Royalty Items":
            self.cbCategory.addItem("Quantity")
            self.cbCategory.addItem("Discount")
            self.cbCategory.addItem("Currency")

    def DateButton(self):
        if self.cbCategory.currentText()[:4] == "Date":
            self.btnDate.show()
```

```
            self.leSearch.setReadOnly(True)
        else:
            self.btnDate.hide()
            self.leSearch.setReadOnly(False)

    def getDate(self):
        self.CalendarWidget.date = self.CalendarWidget.qle.text()
        self.leSearch.setText(self.CalendarWidget.date)

    def getSearchData(self):

        self.Valid = True
        self.Firstname = self.leFirstname.text()
        self.Lastname = self.leLastname.text()
        self.Table = self.cbTable.currentText().replace(" ", "")
        if self.Table == "PublishingInvoice":
            self.Table = "PubInvoice"

        if self.Table != "Author":
            self.Category = self.cbCategory.currentText().replace(" ", "")
            self.Search = self.leSearch.text()

            if self.Firstname.replace(" ", "") == "" or self.Lastname.replace(" ", "") == "" or self.Category == "" or
                 self.Search.replace(" ", "") == "":
                self.Msg = QMessageBox()
                self.Msg.setWindowTitle("Invalid Entry")
                self.Msg.setText("You must fill in all fields.")
                self.Msg.exec_()
                self.Valid = False


            if self.Category in ["Discount", "Quantity"]:

                if self.Table == "RoyaltyItems":
                    self.Category = "Royalty{}".format(self.Category)

                elif self.Table == "BookInvoiceItems":
                    self.Category = "BookInvoice{}".format(self.Category)

            elif self.Category == "Date":
                self.Category = "{}Date".format(self.Table)

```

```
            elif self.Category == "Service":
                self.Category = "PubInvoiceService"

            if self.Table not in ["BookInvoiceItems", "RoyaltyItems"]:
                if self.Table in ["PubInvoice", "Royalties", "BookInvoice"]:
                    self.sql = "select Customer.AuthorID, {0}ID from Customer, {0} where (Customer.Firstname like '{1}%'
                        or Customer.Lastname like '{2}%') and {0}.{3} like '{4}%' and {0}.AuthorID =
                        Customer.AuthorID".format(self.Table, self.Firstname, self.Lastname, self.Category, self.Search)
                elif self.Table == "Book":
                    self.sql = "select Customer.AuthorID, Book.AuthorID, Book.ISBN from Customer, {0} where
                        (Customer.Firstname like '{1}%' or Customer.Lastname like '{2}%') and {0}.{3} like '{4}%' and
                        {0}.AuthorID = Customer.AuthorID".format(self.Table, self.Firstname, self.Lastname,
                        self.Category, self.Search)
            else:
                self.sql = "select Customer.AuthorID, Book.AuthorID, Book.ISBN, {0}.ISBN, {0}ID from Customer, Book, {0}
                    where (Customer.Firstname like '{1}%' or Customer.Lastname like '{2}%') and {0}.{3} like '{4}%' and
                    Customer.AuthorID = Book.AuthorID and Book.ISBN = {0}.ISBN".format(self.Table, self.Firstname,
                    self.Lastname, self.Category, self.Search)

        else:
            if self.Firstname.replace(" ", "") == "" or self.Lastname.replace(" ", "") == "":
                self.Msg = QMessageBox()
                self.Msg.setWindowTitle("Invalid Entry")
                self.Msg.setText("You must fill enter the Firstname and Lastname")
                self.Msg.exec_()
                self.Valid = False

            self.Table = "Customer" #Author table is referred to as 'Customer'
            self.sql = "select AuthorID from Customer where Firstname like '{0}%' or Lastname like
                '{1}%'".format(self.Firstname, self.Lastname)
        if self.Valid == True:
            with sqlite3.connect("PP.db") as db:
                cursor = db.cursor()
                cursor.execute(self.sql)
                self.Results = list(cursor.fetchall())
            self.accept()
```

### 4.10.15 Module 15 - Creating a layout for displaying Search Results in the Main Window

```python
from PyQt4.QtCore import *
from PyQt4.QtGui import *
import sys


class initSearchResultsMenu(QWidget):
    """main window"""

    def __init__(self):
        super().__init__()
        self.btnBack = QPushButton("Back", self)
        self.btnBack.setFixedSize(100, 30)
        self.horizontalTop = QHBoxLayout()
        self.horizontalTop.addStretch(1)
        self.horizontalTop.addWidget(self.btnBack)

        self.vertical = QVBoxLayout()
        self.vertical.addLayout(self.horizontalTop)
        self.setLayout(self.vertical)
```



### 4.10.16 Module 16 - Creating a dialog for the User to choose to change Username or Password

```python
from PyQt4.QtCore import *
from PyQt4.QtGui import *
import sqlite3
import sys

class dbUsernameOrPassword(QDialog):
    """main window"""

    def __init__(self):
        super().__init__()

    def ChangeSelection(self):
        self.setModal(True)
        self.setFixedSize(400, 50)
        self.setWindowTitle("Selection")
        self.btnUsername = QPushButton("Change Username", self)
        self.btnPassword = QPushButton("Change Password", self)
```

```
        self.btnCancel = QPushButton("Cancel", self)
        self.btnCancel.clicked.connect(self.reject)
        self.btnUsername.clicked.connect(self.UsernameSelected)
        self.btnPassword.clicked.connect(self.PasswordSelected)
        self.gridLayout = QGridLayout()
        self.gridLayout.addWidget(self.btnUsername, 0, 0)
        self.gridLayout.addWidget(self.btnPassword, 0, 1)
        self.gridLayout.addWidget(self.btnCancel, 0, 2)
        self.setLayout(self.gridLayout)
        self.exec_()

    def UsernameSelected(self):
        self.Selection = "Username"
        self.accept()

    def PasswordSelected(self):
        self.Selection = "Password"
        self.accept()
```

### 4.10.17 Module 17 - Initialising a dialog for changing the Username

```
from PyQt4.QtCore import *
from PyQt4.QtGui import *
import sqlite3
import sys

class dbChangeUsername(QDialog):
    """Change Username confirmation"""

    def __init__(self):
        super().__init__()

    def initChangeUsernameScreen(self):
        self.setWindowTitle("Change Username")
        self.setModal(True)
        self.leOldUsername = QLineEdit(self)
        self.leNewUsername = QLineEdit(self)
        self.leNewUsername.setValidator(QRegExpValidator(QRegExp("^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$")))
        self.leOldUsername.setValidator(QRegExpValidator(QRegExp("^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$")))
```

```
        #reg ex from www.regexlib.com
        self.leRetype = QLineEdit(self)
        self.lblOld = QLabel("Old Username:", self)
        self.lblNew = QLabel("New Username:", self)
        self.lblRetype = QLabel("Retype New Username:", self)
        self.leOldUsername.setPlaceholderText("Old Username")
        self.leNewUsername.setPlaceholderText("New Username")
        self.leRetype.setPlaceholderText("Retype New Username")
        self.btnConfirm = QPushButton("Confirm")
        self.btnCancel = QPushButton("Cancel")
        self.horizontal = QHBoxLayout()
        self.horizontal.addWidget(self.btnCancel)
        self.horizontal.addWidget(self.btnConfirm)
        self.gridLayout = QGridLayout()
        self.gridLayout.addWidget(self.lblOld, 0, 0)
        self.gridLayout.addWidget(self.lblNew, 1, 0)
        self.gridLayout.addWidget(self.lblRetype, 2, 0)
        self.gridLayout.addWidget(self.leOldUsername, 0, 1)
        self.gridLayout.addWidget(self.leNewUsername, 1, 1)
        self.gridLayout.addWidget(self.leRetype, 2, 1)
        self.gridLayout.addLayout(self.horizontal, 3, 2)
        self.setLayout(self.gridLayout)
        self.btnCancel.clicked.connect(self.reject)
        self.btnConfirm.clicked.connect(self.Check)
        self.exec_()

    def Check(self):
        if self.leNewUsername.text() == self.leRetype.text():

            if len(self.leNewUsername.text()) < 5:
                self.Msg = QMessageBox()
                self.Msg.setWindowTitle("Username")
                self.Msg.setText("New Username was too short")
                self.Msg.exec_()
            else:

                with sqlite3.connect("dbLogin.db") as db:
                    cursor = db.cursor()
                    cursor.execute("select Username from LoginDetails")
                    self.OldUsername = list(cursor.fetchone())[0]
                    if self.leOldUsername.text() == self.OldUsername:
                        self.NewUsername = self.leNewUsername.text()
```

```
                        cursor.execute("update LoginDetails set Username = '{}' where Username =
                            '{}'".format(self.NewUsername, self.Username))
                        self.Msg = QMessageBox()
                        self.Msg.setWindowTitle("Username")
                        self.Msg.setText("Username was successfully changed")
                        self.Msg.exec_()
                        self.accept()

                    else:
                        self.Msg = QMessageBox()
                        self.Msg.setWindowTitle("Username")
                        self.Msg.setText("Old Username was incorrect")
                        self.Msg.exec_()
        else:
            self.Msg = QMessageBox()
            self.Msg.setWindowTitle("Username")
            self.Msg.setText("New Usernames did not match")
            self.Msg.exec_()
```

### 4.10.18 Module 18 - Initialising a dialog for changing the Password

```
from PyQt4.QtCore import *
from PyQt4.QtGui import *
import sqlite3
import sys

class dbChangePassword(QDialog):
    """Change password confirmation"""

    def __init__(self):
        super().__init__()

    def initChangePasswordScreen(self):
        self.setWindowTitle("Change Password")
        self.setModal(True)
        self.leOldPassword = QLineEdit(self)
        self.leNewPassword = QLineEdit(self)
        self.leRetype = QLineEdit(self)
        self.leOldPassword.setEchoMode(self.leOldPassword.Password)
```

```
        self.leNewPassword.setEchoMode(self.leNewPassword.Password)
        self.leRetype.setEchoMode(self.leRetype.Password)
        self.lblOld = QLabel("Old Password:", self)
        self.lblNew = QLabel("New Password:", self)
        self.lblRetype = QLabel("Retype New Password:", self)
        self.leOldPassword.setPlaceholderText("Old Password")
        self.leNewPassword.setPlaceholderText("New Password")
        self.leRetype.setPlaceholderText("Retype New Password")
        self.btnConfirm = QPushButton("Confirm")
        self.btnCancel = QPushButton("Cancel")
        self.horizontal = QHBoxLayout()
        self.horizontal.addWidget(self.btnCancel)
        self.horizontal.addWidget(self.btnConfirm)
        self.gridLayout = QGridLayout()
        self.gridLayout.addWidget(self.lblOld, 0, 0)
        self.gridLayout.addWidget(self.lblNew, 1, 0)
        self.gridLayout.addWidget(self.lblRetype, 2, 0)
        self.gridLayout.addWidget(self.leOldPassword, 0, 1)
        self.gridLayout.addWidget(self.leNewPassword, 1, 1)
        self.gridLayout.addWidget(self.leRetype, 2, 1)
        self.gridLayout.addLayout(self.horizontal, 3, 2)
        self.setLayout(self.gridLayout)

        self.btnCancel.clicked.connect(self.reject)
        self.btnConfirm.clicked.connect(self.Check)
        self.exec_()

    def Check(self):
        if self.leNewPassword.text() == self.leRetype.text():

            if len(self.leNewPassword.text()) < 5:
                self.Msg = QMessageBox()
                self.Msg.setWindowTitle("Password")
                self.Msg.setText("New Password was too short")
                self.Msg.exec_()
            else:
                with sqlite3.connect("dbLogin.db") as db:
                    cursor = db.cursor()
                    cursor.execute("select Password from LoginDetails")
                    self.OldPassword = list(cursor.fetchone())[0]
                    if self.leOldPassword.text() == self.OldPassword:
                        self.Password = self.leNewPassword.text()
```

```
                        cursor.execute("update LoginDetails set Password = '{}' where Username =
                            '{}'".format(self.Password, self.Username))
                        self.Msg = QMessageBox()
                        self.Msg.setWindowTitle("Password")
                        self.Msg.setText("Password was successfully changed")
                        self.Msg.exec_()
                        self.accept()

                    else:
                        self.Msg = QMessageBox()
                        self.Msg.setWindowTitle("Password")
                        self.Msg.setText("Old Password was incorrect")
                        self.Msg.exec_()
        else:
            self.Msg = QMessageBox()
            self.Msg.setWindowTitle("Password")
            self.Msg.setText("New Passwords did not match")
            self.Msg.exec_()
```

# Chapter 5

# User Manual

## 5.1 Introduction

In this manual I will go through the correct way to install and utilise the program to its best abilities. The user will require a computer which meets certain specifications and a CD-ROM holding a zip file of the application. The intended audience for the system was the director of Perfect Publishers Ltd.

## 5.2 Installation

### 5.2.1 Prerequisite Installation

As the system has been compiled to a windows executable, the system does not require any programs to be installed on the computer which it is to be used on apart from the system itself. The program was intended to run on Windows 7/8, as these were the operating systems that it was designed on, and the operating system that the client owns.

The following is required for the system to reach its full capabilities:

- A Keyboard for user inputs
- A Mouse for user inputs
- A Hard Disk Drive for storage
- A Visual Display Unit for outputs generated by the system
- Connection the the internet if the user needs an email sent to their email when they've forgotten their password

- At least 512 megabytes of main memory to carry out processing

### 5.2.2 System Installation

The user requires the CD-R that has the necessary files on for installation. After inserting the CD-R, the user will need to open 'My Computer'.

The user will then need to right click the CD-R, and click open.

Then right click on the installation file, and click 'Install'.

The Windows installer will the commence installing. The user is required to select a destination to be installed at, with the default directory given.

Perfect Publishers Database System Setup
Select destination directory
Default Location is presented to user
Perfect Publishers Database System
Up
New
File path is shown here, User can edit this to change the desired file path.
C:\Program Files (x86)\Perfect Publishers Database System\
< Back
Next >
Cancel

The Installer initialises the screen for installation.

The user must grant the installer permission to complete installation. To do so, click 'Yes' when the prompt opens.

Figure 5.1: Prompt for permission

Once the user has granted the installer permission, the installation is completed.

Perfect Publishers Database System Setup
Install Perfect Publishers Database System
Please wait while the installer installs Perfect Publishers Database System.
Status: Copying new files
< Back
Next >
Cancel

### 5.2.3 Running the System

In order to run the system, the user could navigate to the path of installation, or the user could create a shortcut to run quickly.

To create a shortcut, the user must do the following:

The user will be presented with the interface to type the file path of the system file. The user should click 'Browse...', where the user should browse to the

correct file shown in Figure 5.2.3.

Then the user is required to name the shortcut. Afterwards, the user needs to click finish to finish creating a shortcut that the program can be run from.

## 5.3 Tutorial

### 5.3.1 Introduction

In this section I will go through the steps of using the system to its full capabilities, including adding, editing and deleting data in the database, conducting calculations to find costs and payment prices, and searching for data in the database. I will also cover the navigation of the program.

### 5.3.2 Assumptions

I expect that the user has at least a basic understanding of Windows 7/8. As my client is capable with using these operating systems, the tutorial should not be difficult to follow. Also, for functions in the system that have similar functionality, only one of these functions will be properly investigated in my tutorial. For example, I will cover how to search for a book, and in doing so, I can assume that the user will be able to conduct searches for Royalties/Invoices etc., because of the similar functionality.

### 5.3.3 Tutorial Questions

**Question 1**

**Question 2**

### 5.3.4 Saving

### 5.3.5 Limitations

## 5.4 Error Recovery

### 5.4.1 Error 1

### 5.4.2 Error 2

## 5.5 System Recovery

### 5.5.1 Backing-up Data

### 5.5.2 Restoring Data

# Chapter 6

# Evaluation

## 6.1 Customer Requirements

### 6.1.1 Objective Evaluation

## 6.2 Effectiveness

### 6.2.1 Objective Evaluation

## 6.3 Learnability

## 6.4 Usability

## 6.5 Maintainability

## 6.6 Suggestions for Improvement

## 6.7 End User Evidence

### 6.7.1 Questionnaires

### 6.7.2 Graphs

### 6.7.3 Written Statements