

# Computing A2 Coursework - Book Publishing Management System

Imran Rahman

March 26, 2015

# Contents

<b>1 Analysis</b>	<b>6</b>
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
<b>2</b>	<b>Design</b>	<b>36</b>
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
<b>3</b>	<b>Testing</b>	<b>103</b>
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
<b>4</b>	<b>System Maintenance</b>	<b>190</b>
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 . . . . .	192
4.2.1	Main window . . . . .	192
4.2.2	Adding Entries . . . . .	192
4.2.3	Updating Entries . . . . .	193
4.2.4	Deleting Entries . . . . .	193
4.2.5	Calculations . . . . .	193
4.2.6	Detailed Searches . . . . .	194
4.3	Code Structure . . . . .	194
4.3.1	Function - Adding an Item . . . . .	194
4.3.2	Function - Removing an Item . . . . .	195
4.3.3	Function - Generating an update window . . . . .	196
4.3.4	Function - Updating an Item . . . . .	198
4.3.5	Function - Creating a Table Widget . . . . .	199
4.3.6	Function - Quick Search . . . . .	200
4.3.7	Modules - View Window and Search Results Window. .	201
4.4	Variable Listing . . . . .	202
4.5	System Evidence . . . . .	206
4.5.1	User Interface . . . . .	206
4.5.2	ER Diagram . . . . .	228
4.5.3	Database Table Views . . . . .	229
4.5.4	Database SQL . . . . .	232
4.5.5	SQL Queries . . . . .	234
4.6	Testing . . . . .	244
4.6.1	Summary of Results . . . . .	244
4.6.2	Known Issues . . . . .	244
4.7	Code Explanations . . . . .	245
4.7.1	Difficult Sections . . . . .	245
4.7.2	Self-created Algorithms . . . . .	248
4.8	Settings . . . . .	249
4.9	Acknowledgements . . . . .	249
4.10	Code Listing . . . . .	253
4.10.1	Module 1 - Login Window and Database Creation . . .	253
4.10.2	Module 2 - Main Window and functionality for majority of system . . . . .	261

4.10.3 Module 3 - Initialising Main Menu Layout . . . . .	288
4.10.4 Module 4 - Initialising Menu Bar . . . . .	290
4.10.5 Module 5 - Initialising a Table Widget . . . . .	291
4.10.6 Module 6 - Initialising Add Customer Entry Window . . . . .	292
4.10.7 Module 7 - Initialising Update Customer Entry Window . . . . .	294
4.10.8 Module 8 - Initialising View Menu Layout . . . . .	296
4.10.9 Module 9 - Initialising Add Entry Window for non-customer entries . . . . .	297
4.10.10 Module 10 - Initialising Calendar Widget . . . . .	307
4.10.11 Module 11 - Initialising a Confirmation Dialog . . . . .	309
4.10.12 Module 12 - Initialising a dialog for viewing the Items, and creating calculations for the payments. . . . .	312
4.10.13 Module 13 - Initialising a dialog for viewing Royalties and Invoices . . . . .	317
4.10.14 Module 14 - Initialising a Search Window . . . . .	319
4.10.15 Module 15 - Creating a layout for displaying Search Re- sults in the Main Window . . . . .	324
4.10.16 Module 16 - Creating a dialog for the User to choose to change Username or Password . . . . .	324
4.10.17 Module 17 - Initialising a dialog for changing the Username	325
4.10.18 Module 18 - Initialising a dialog for changing the Password	327
<b>5 User Manual . . . . .</b>	<b>330</b>
5.1 Introduction . . . . .	332
5.1.1 Purpose of System . . . . .	332
5.1.2 Intended Audience . . . . .	332
5.2 Installation . . . . .	332
5.2.1 Prerequisite Installation . . . . .	332
5.2.2 System Installation . . . . .	333
5.2.3 Running the System . . . . .	342
5.3 Tutorial . . . . .	347
5.3.1 Introduction . . . . .	347
5.3.2 Assumptions . . . . .	347
5.3.3 Tutorial Questions . . . . .	348
5.3.4 Saving . . . . .	370
5.3.5 Limitations . . . . .	370
5.4 Error Recovery . . . . .	371
5.4.1 Invalid entries when calculating payments . . . . .	371
5.5 System Recovery . . . . .	372
5.5.1 Backing-up Data . . . . .	372
5.5.2 Restoring Data . . . . .	374
<b>6 Evaluation . . . . .</b>	<b>376</b>
6.1 Customer Requirements . . . . .	376
6.1.1 Objective Evaluation . . . . .	376
6.1.2 Summary . . . . .	381

6.2	Effectiveness . . . . .	384
6.2.1	Objective Evaluation . . . . .	384
6.3	Learnability . . . . .	384
6.4	Usability . . . . .	384
6.5	Maintainability . . . . .	384
6.6	Suggestions for Improvement . . . . .	384
6.7	End User Evidence . . . . .	384
6.7.1	Questionnaires . . . . .	384
6.7.2	Graphs . . . . .	386
6.7.3	Written Statements . . . . .	386

# **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 separately find the customer that the book was written by, as they are in separate 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 disorganized
3. What data do you record?  
Details about the authors - Name, email, phone number, Address  
Book details - Title, author, Pages, Hard or soft back, material, colour, size, 150N
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 organized 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:



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 separate spreadsheet, linked to the details of the book. These details are agreed separately 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.

<b>Source</b>	<b>Data</b>	<b>Example Data</b>	<b>Destination</b>
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 received 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

---

```

1: SET Payment TO false
2:
    Check Website for Price
    Create Invoice
    Send Invoice
3: WHILE Payment = false DO
    Check For Payment
4:     IF PaymentReceived THEN
        Payment = true
5: END IF
6: END WHILE

```

---

---

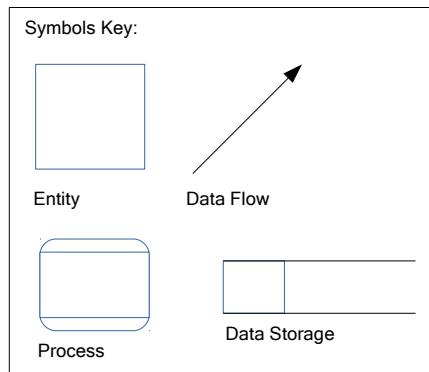
**Algorithm 2** Second Algorithm - Completing Work and Checking If Work is Completed

---

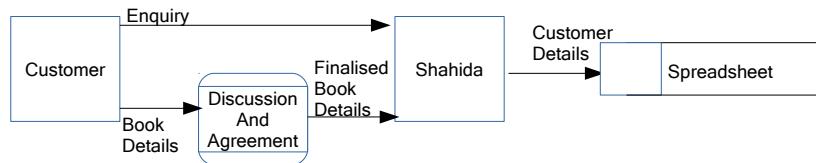
```
1: SET WorkComplete TO false
2: SET CoverComplete TO false
3: SET BookComplete TO false
4:
5: WHILE WorkComplete = false DO
   Get Completed Cover from Cover Designer
6:   SET CoverComplete TO true
   Get Completed Book from Editor
7:   SET BookComplete TO true
8:   IF BookComplete and CoverComplete THEN
9:     SET finished TO true
10:  END IF
11: END WHILE
```

---

### Data flow diagrams



Adding a new customer's details to the database:



Sending an Invoice, waiting for payment and completing the work:

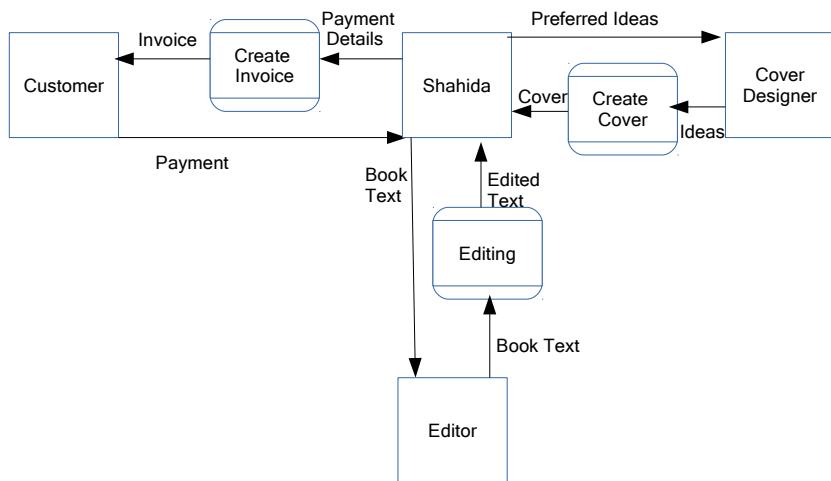


Figure 1.3: Data Flow Diagrams

**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

**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 Stoppapi  
via Al Marcadello 2  
CH-6988 Ponte Tresa  
Switzerland

Order Description	PRICE
12 Osho Therapy @ 50% discount. Retail price £25.99	£155.94
<b>Shipping</b>	£10.06
<b>TOTAL</b>	<b>£166.00</b>

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](http://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](mailto: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 £9.99	40% 40%	\$9.59 £5.99	19 69	\$182.21 £413.31	\$96.00 £233.10	\$91.01* £158.01
<b>TOTAL</b>						<b>£211.20</b>

---

 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](http://www.perfectpublishers.co.uk) | [www.facebook.com/ppublishers](https://www.facebook.com/ppublishers) | [www.twitter.com/ppublishers](https://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 separate 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

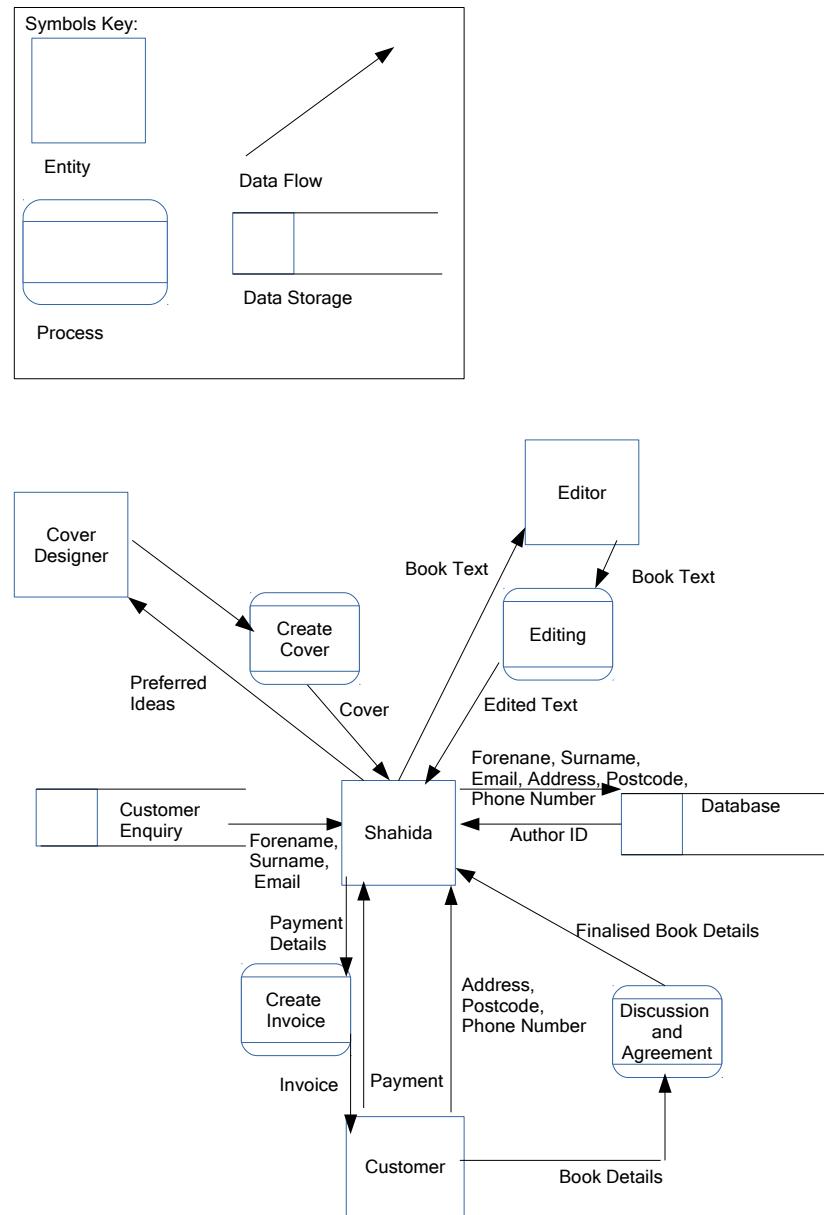
**Data flow diagram**

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 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.
- Ability to calculate royalties using given data

The System must be able to prevent unnecessary 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

Calculating royalties:

- Receives inputs from user (print cost, wholesale price, quantity sold etc)
- Uses inputs to calculate the royalties

### **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

Imran Rahman

Candidate No. 30928

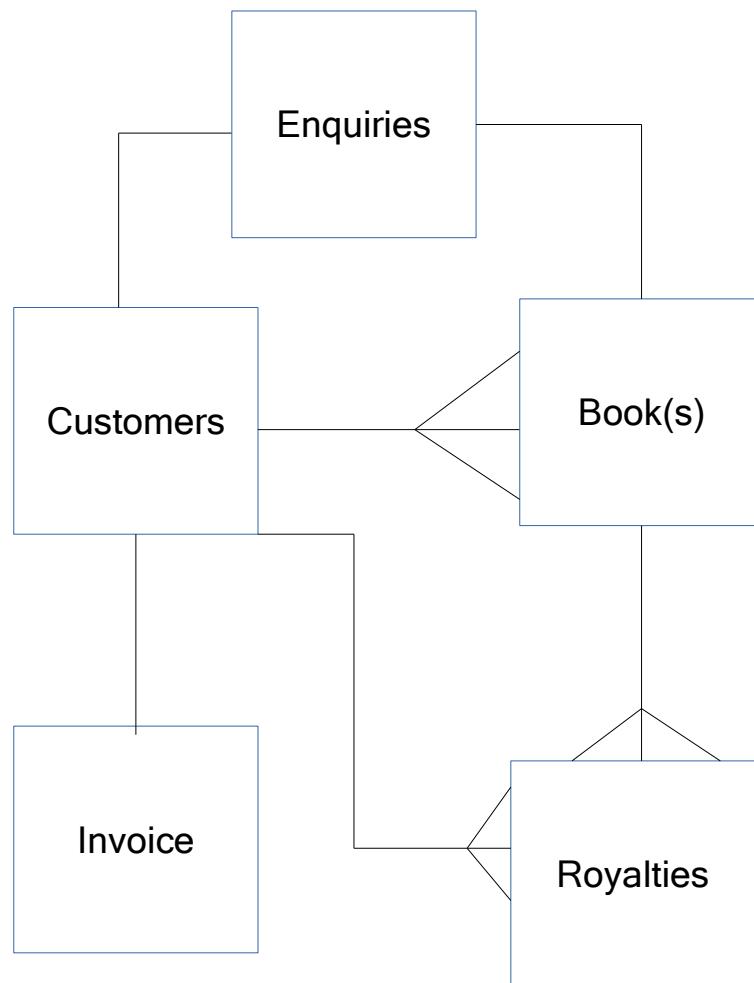
Centre No. 22151

---

## 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 separately from the database.

## 1.5 Object Analysis

### 1.5.1 Object Listing

- Shahida
- Customer
- Editor
- Cover Designer
- Spreadsheet

Imran Rahman

Candidate No. 30928

Centre No. 22151

---

### 1.5.2 Relationship diagrams

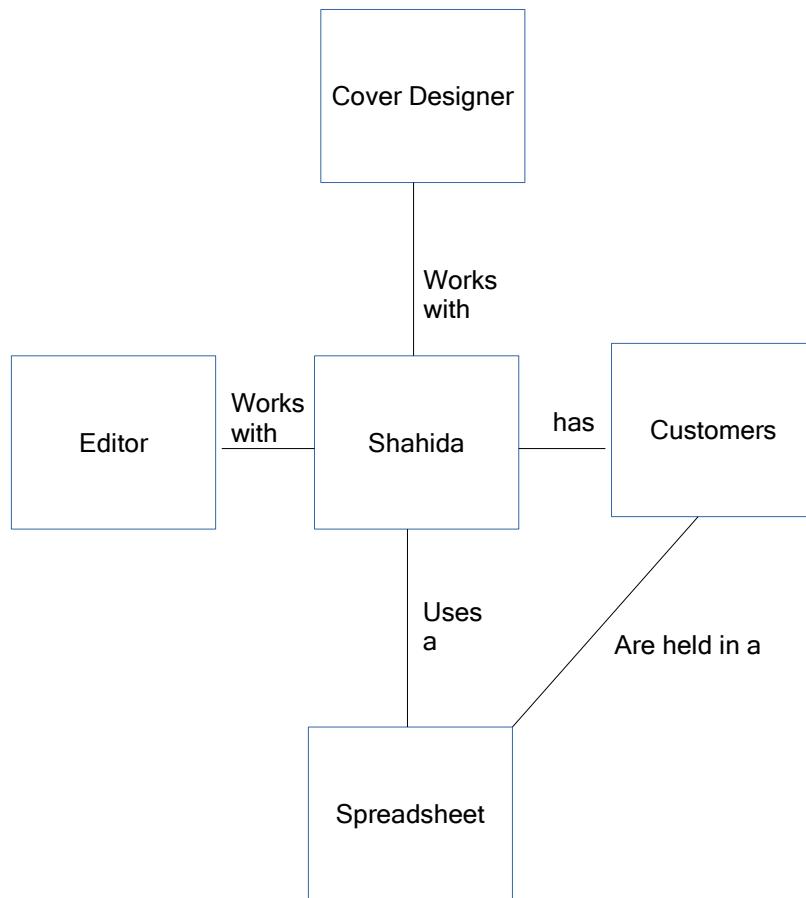


Figure 1.8: Relationship Diagram

### 1.5.3 Class definitions

Key:

Label
Attributes
Behaviours
Customer
Author ID
ForeName
Surname
Email
Address
Postcode
Phone Number
Add ForeName
Edit Forename
Add Surname
Edit Surname
Add Email
Edit Email
Add Address
Edit Address
Add Postcode
Edit Postcode
Add Phone Number
Edit Phone Number

<b>Book</b>
Title
ISBN
Pages
Size
Cover Type
Back Type
Paper
Font
Font Size
Price
Date Published
Add Title
Edit Title
Add ISBN
Edit ISBN
Add Pages
Edit Pages
Add Size
Edit Size
Add Cover Type
Edit      Cover
Type
Add Back Type
Edit Back Type
Add Paper
Edit Paper
Add Font
Edit Font
Add Font Size
Edit Font Size
Add Date Published
Edit Date Published
Add Price
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 themselves. 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 specifically 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 deterioration 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 values 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 will 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 separate 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 separate 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 separate 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 separate 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 beforehand, 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 then 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 separate 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

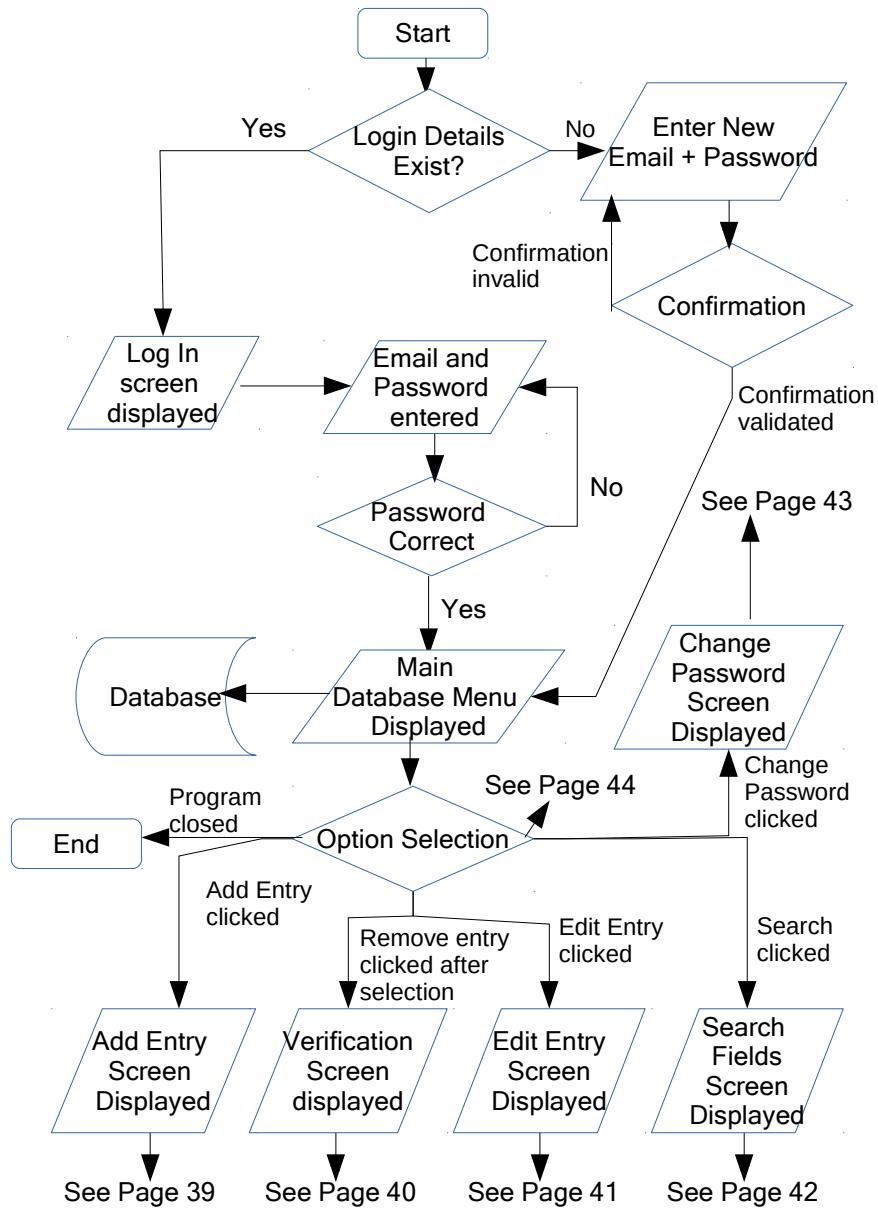


Figure 2.2: Flowchart 2

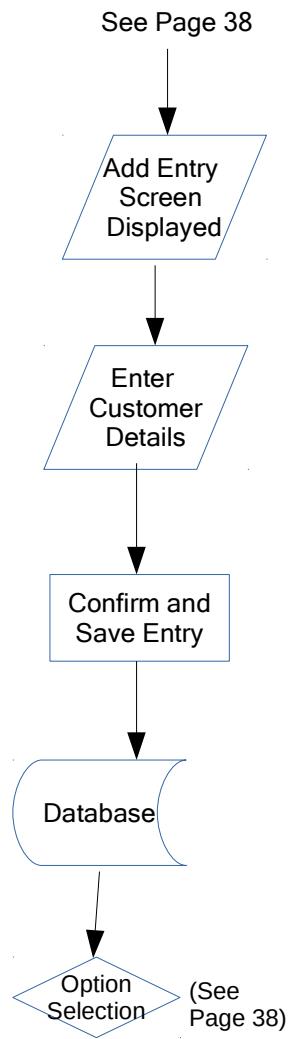


Figure 2.3: Flowchart 3

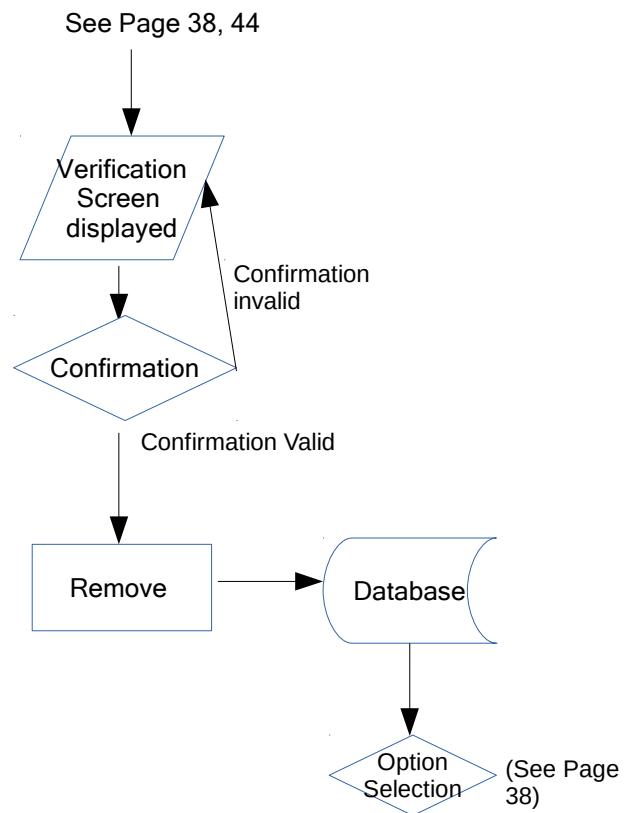


Figure 2.4: Flowchart 4

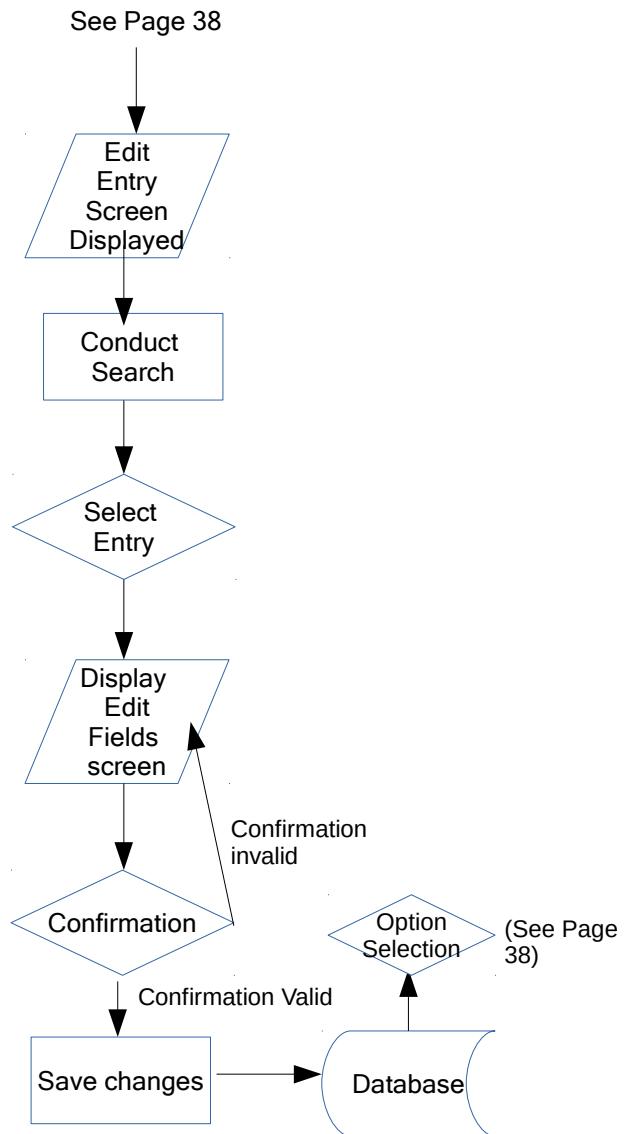


Figure 2.5: Flowchart 5

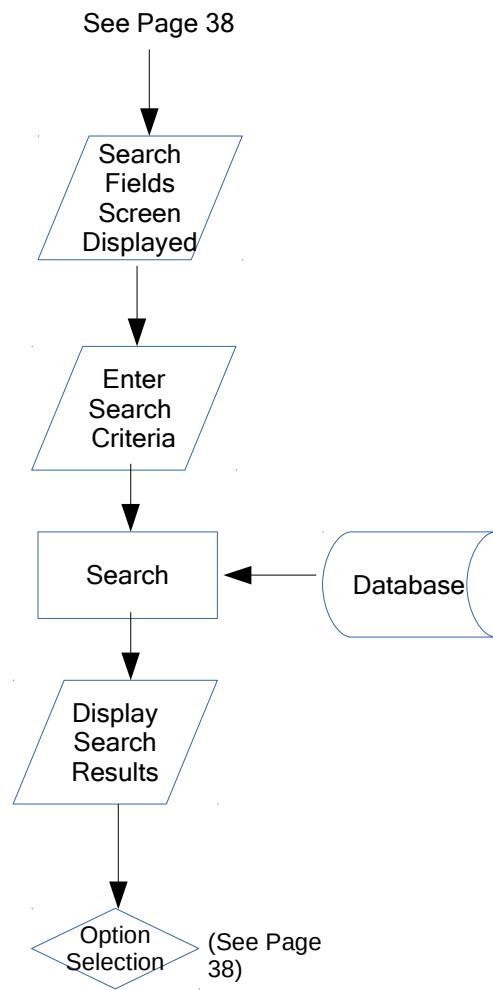


Figure 2.6: Flowchart 6

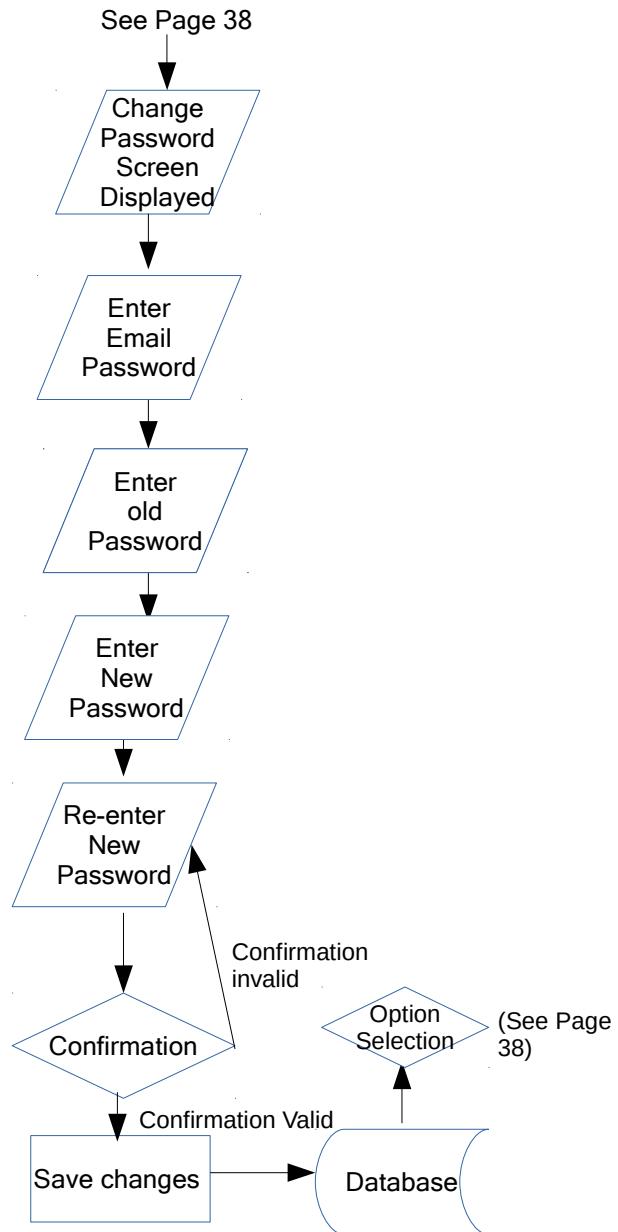
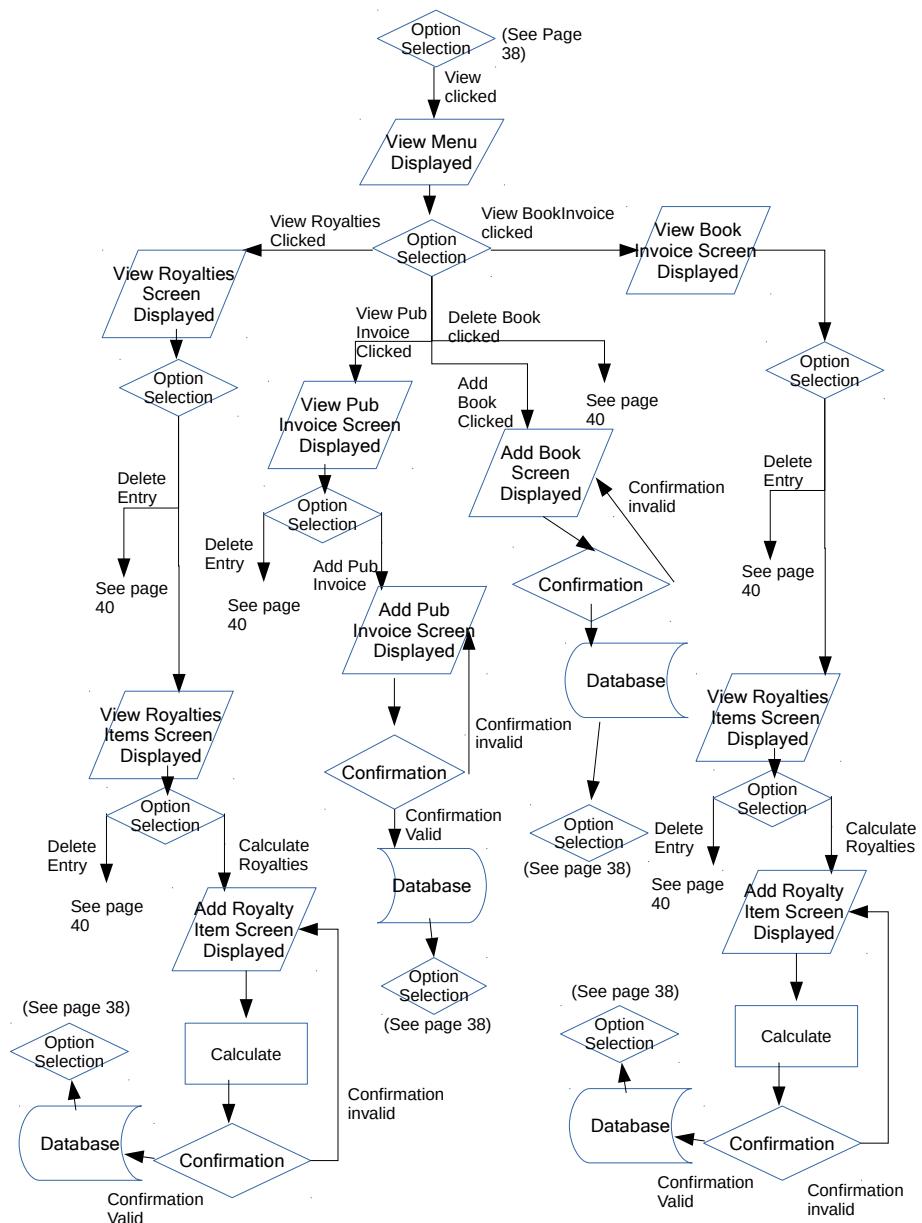


Figure 2.7: Flowchart 7



## 2.2 User Interface Designs

Figure 2.8: Login Screen and Main Menu

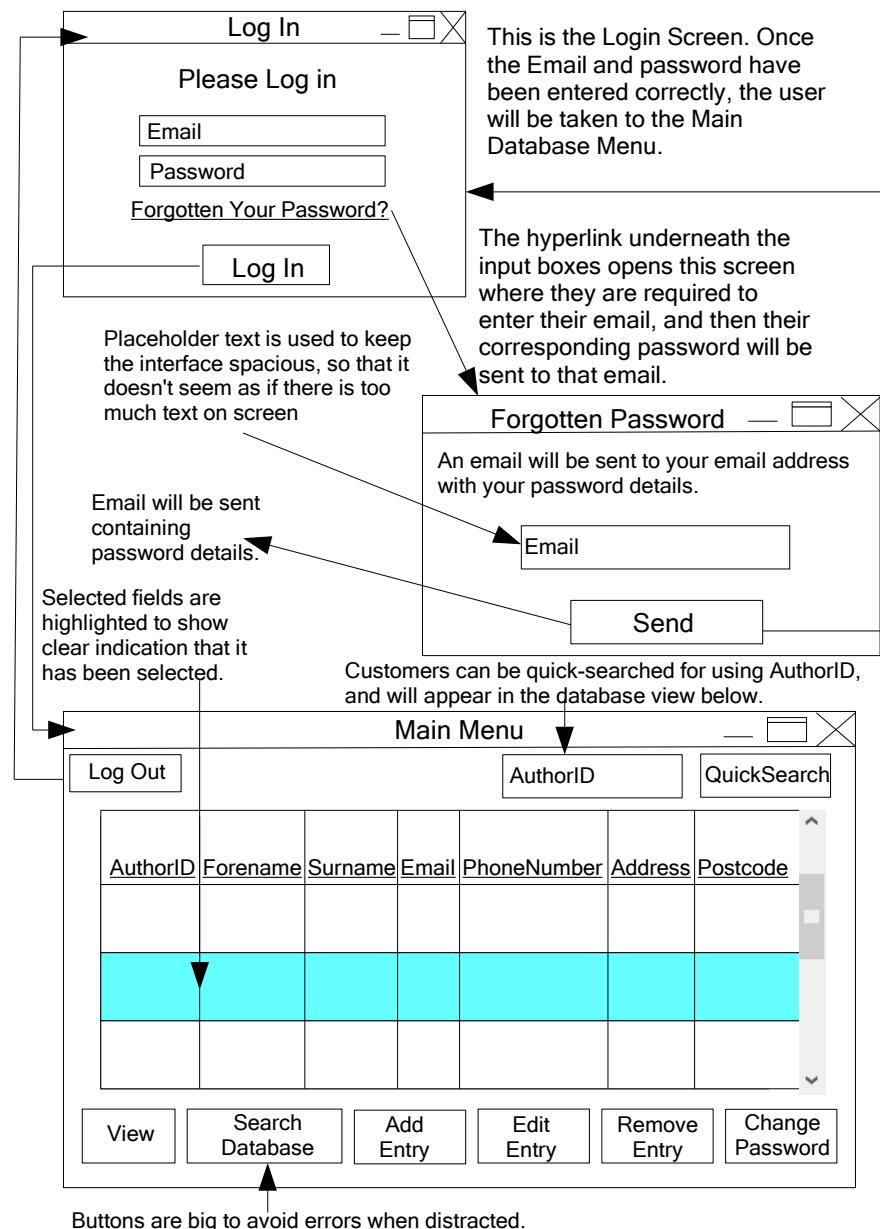


Figure 2.9: View Menu and Royalties

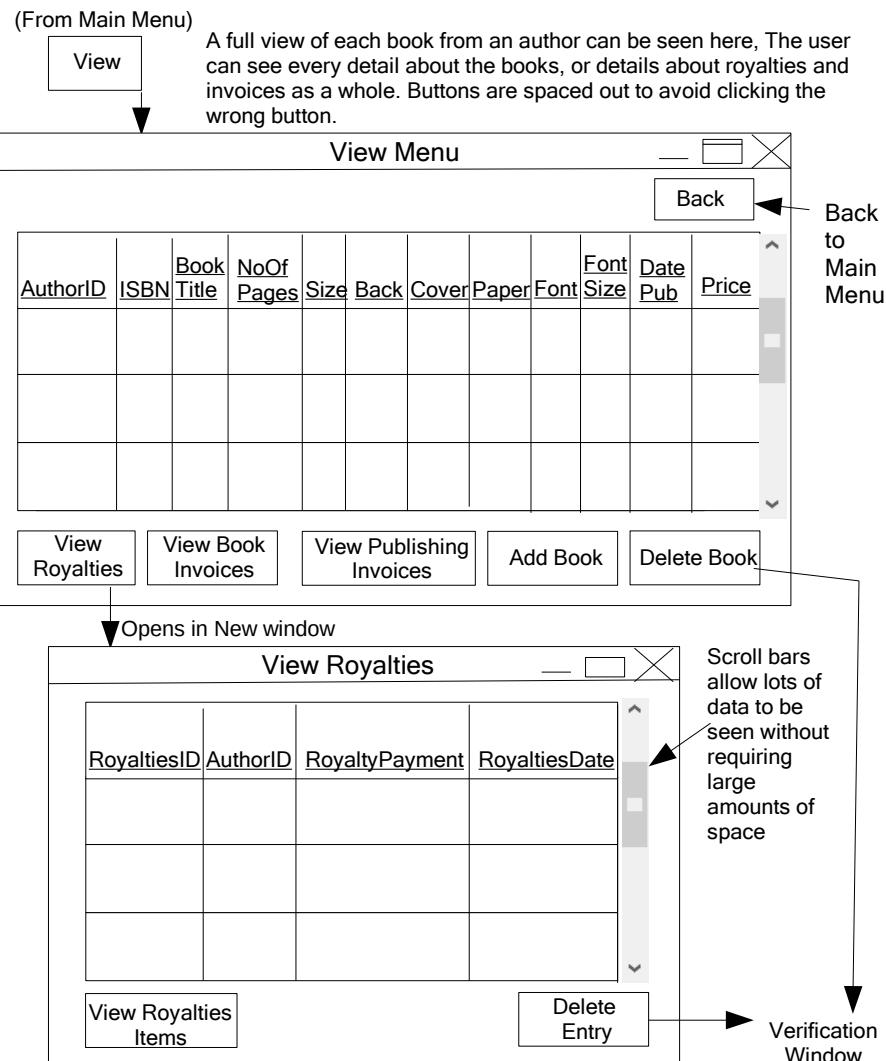


Figure 2.10: RoyaltiesItems and BookInvoice and Items

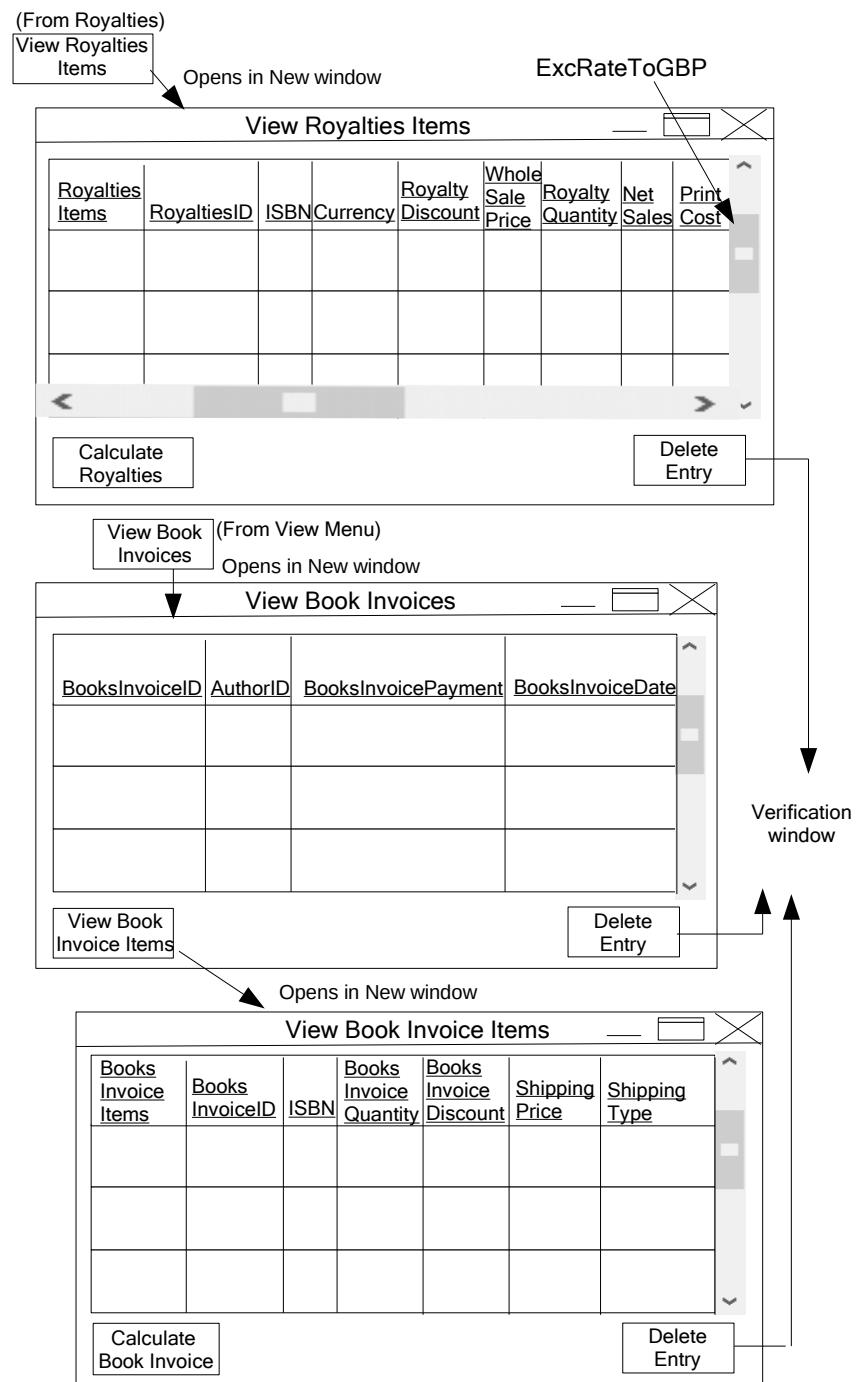


Figure 2.11: Publishing Invoice and Adding Entries

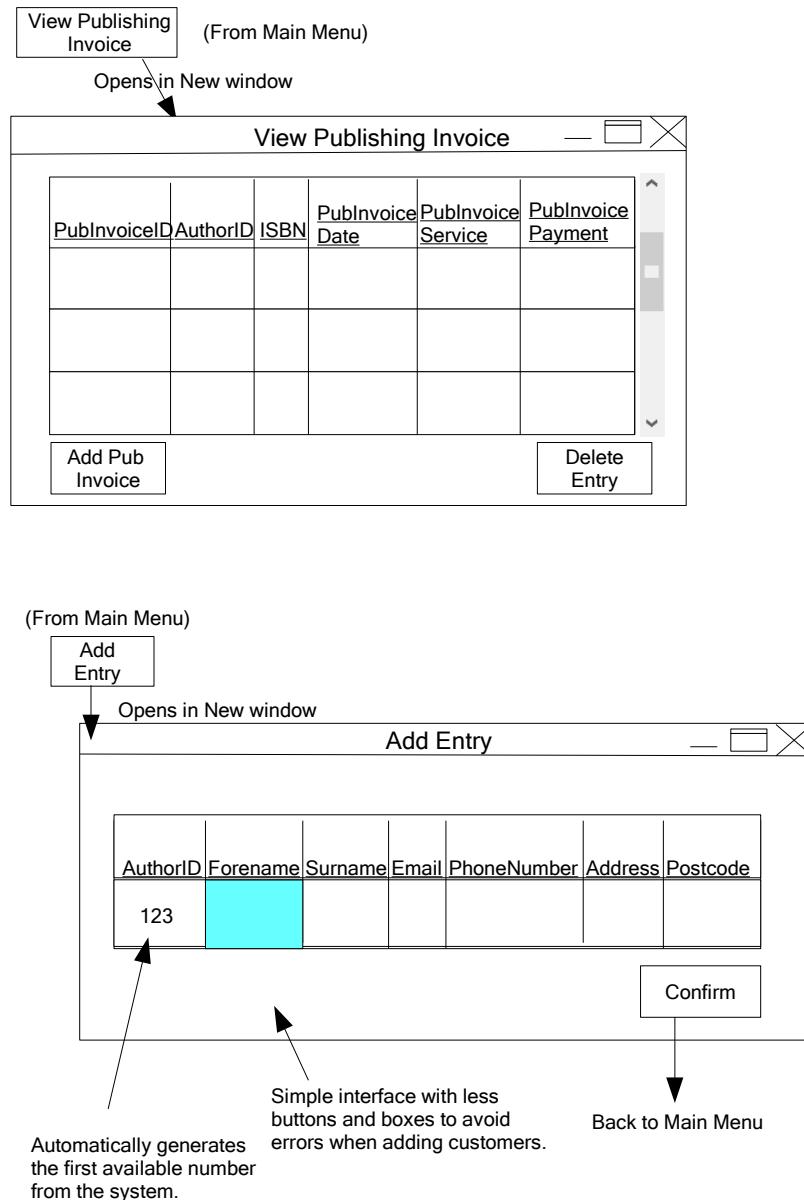


Figure 2.12: Search

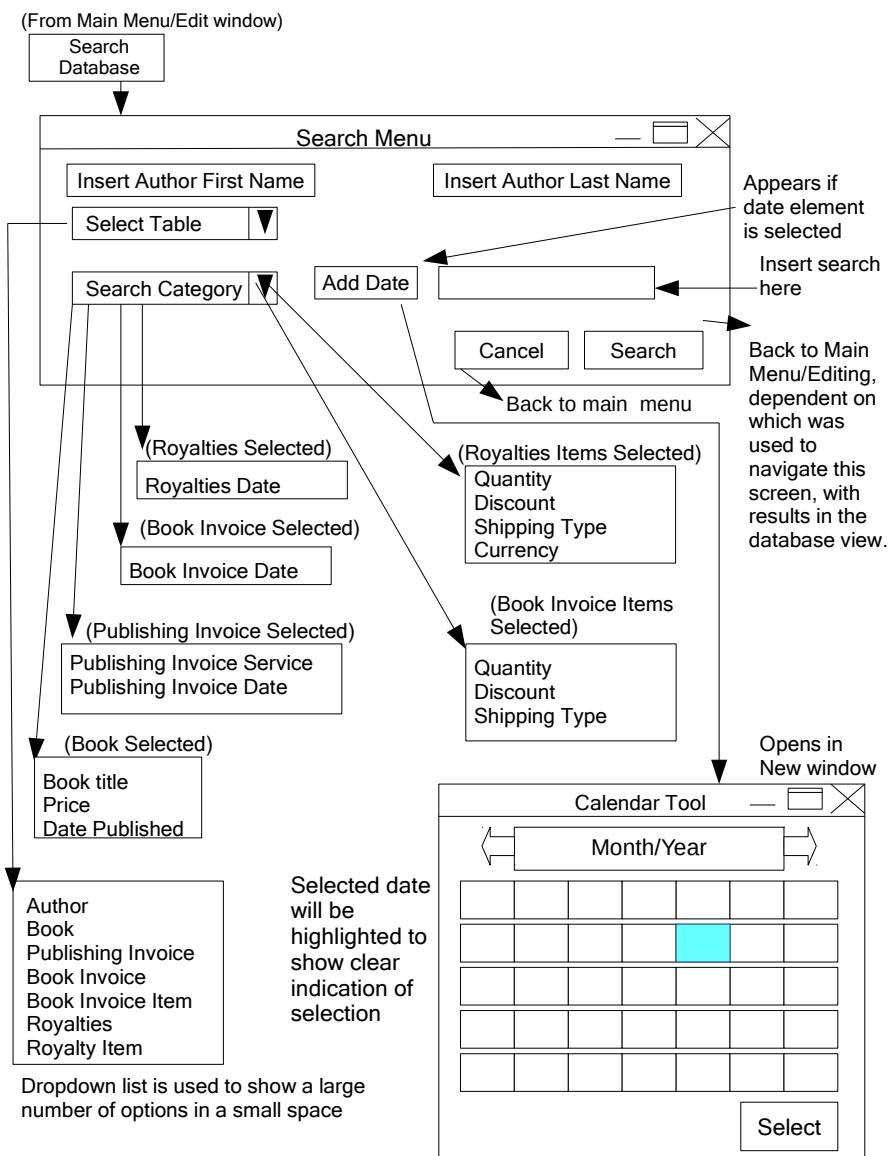


Figure 2.13: Editing Screens

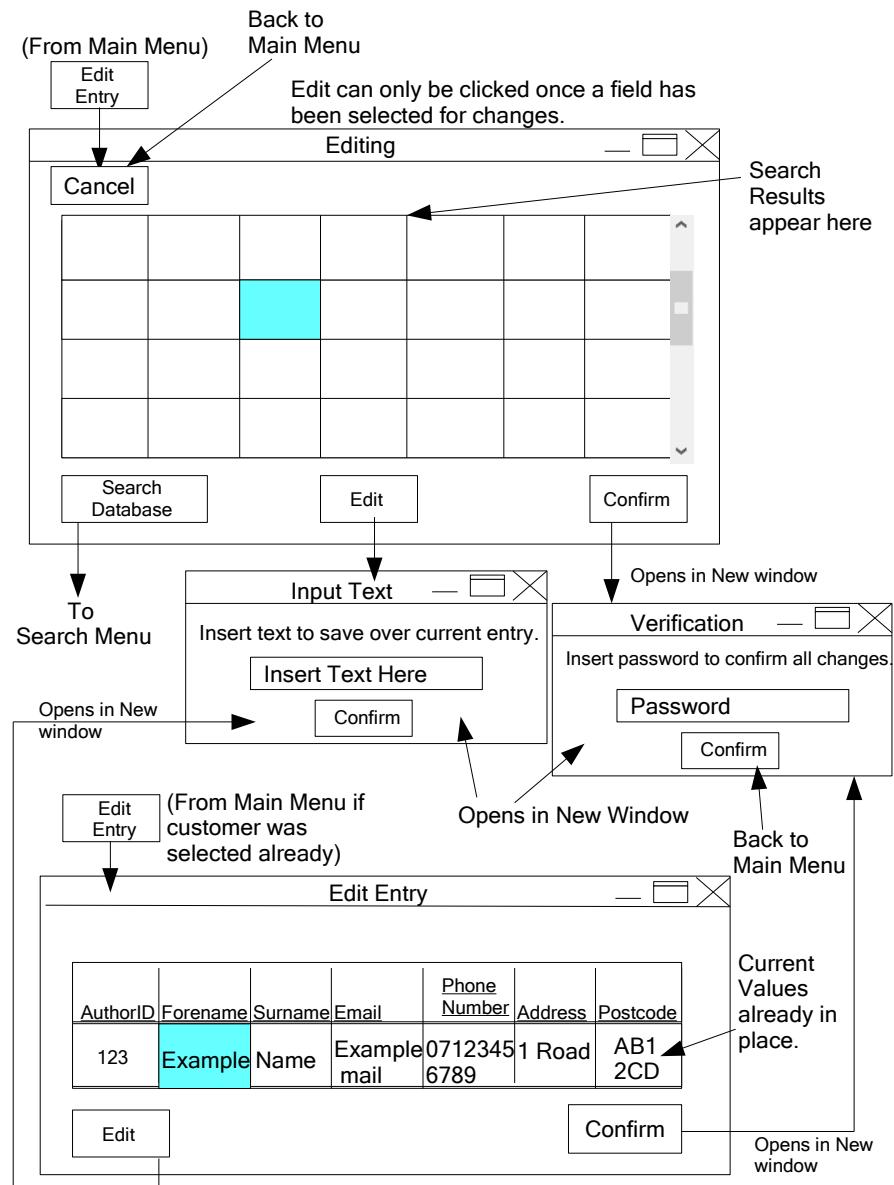


Figure 2.14: Remove Entry and Change Password

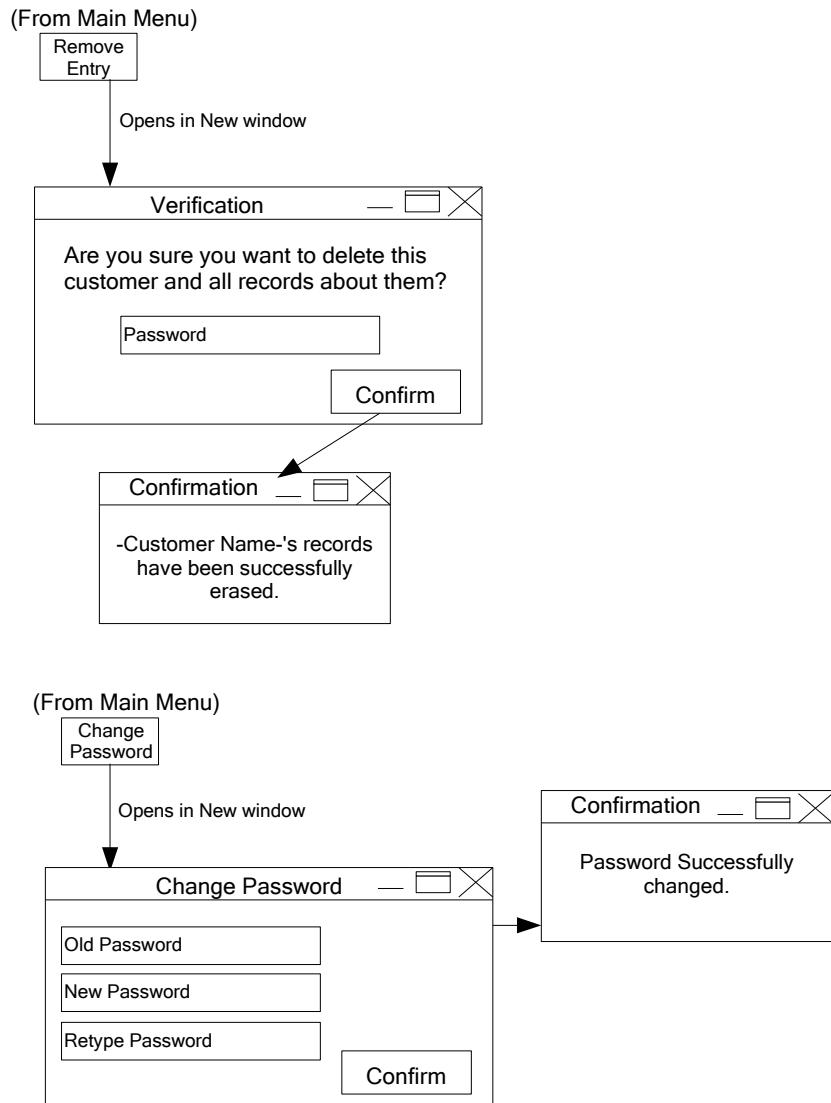


Figure 2.15: Calculations

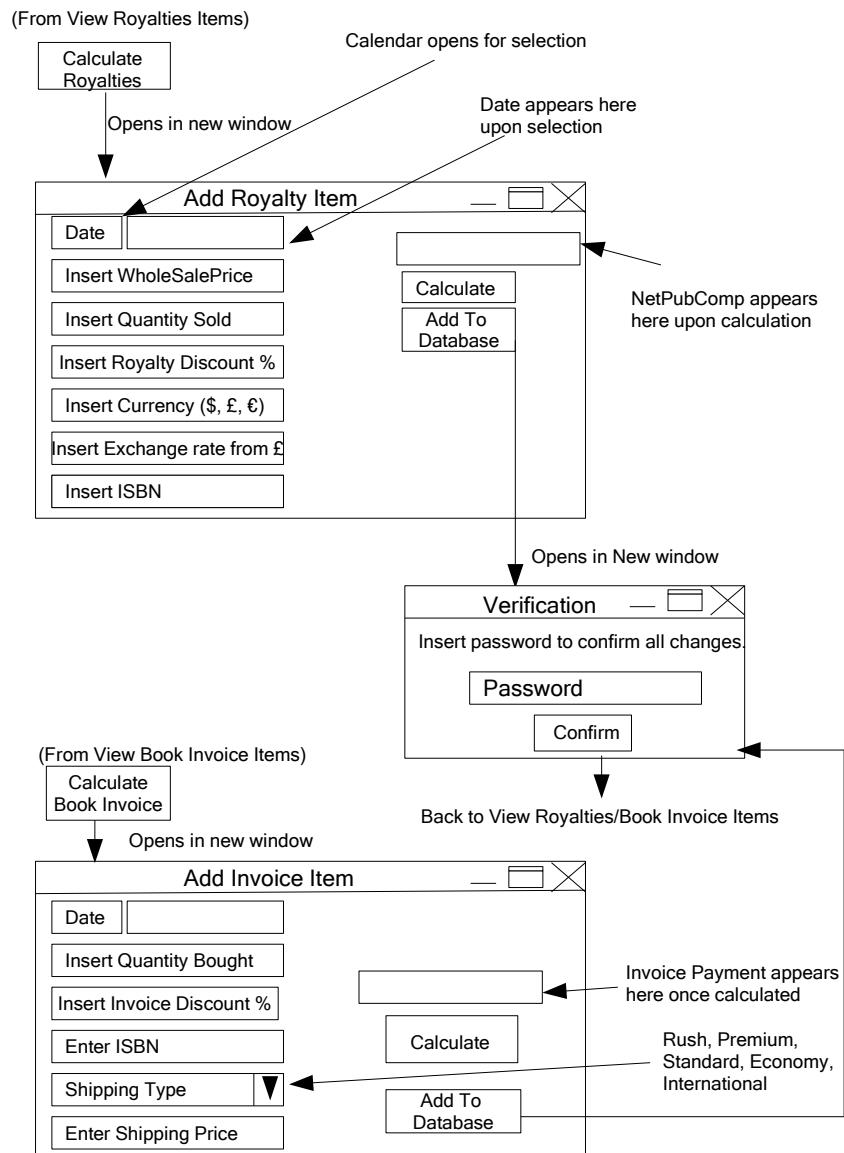


Figure 2.16: Adding Publishing Invoices

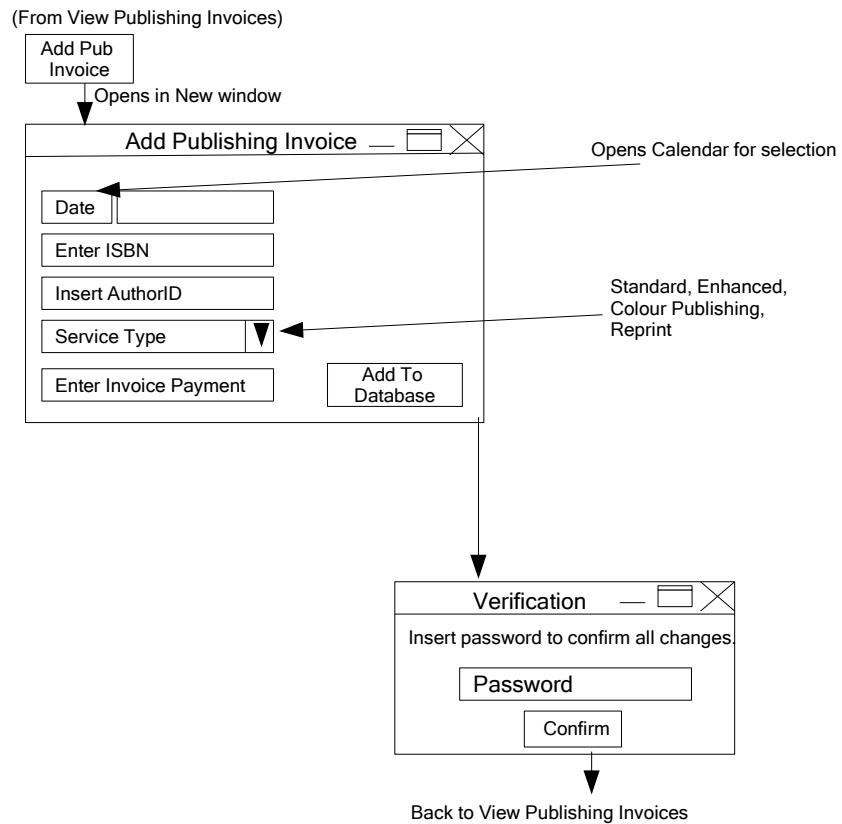
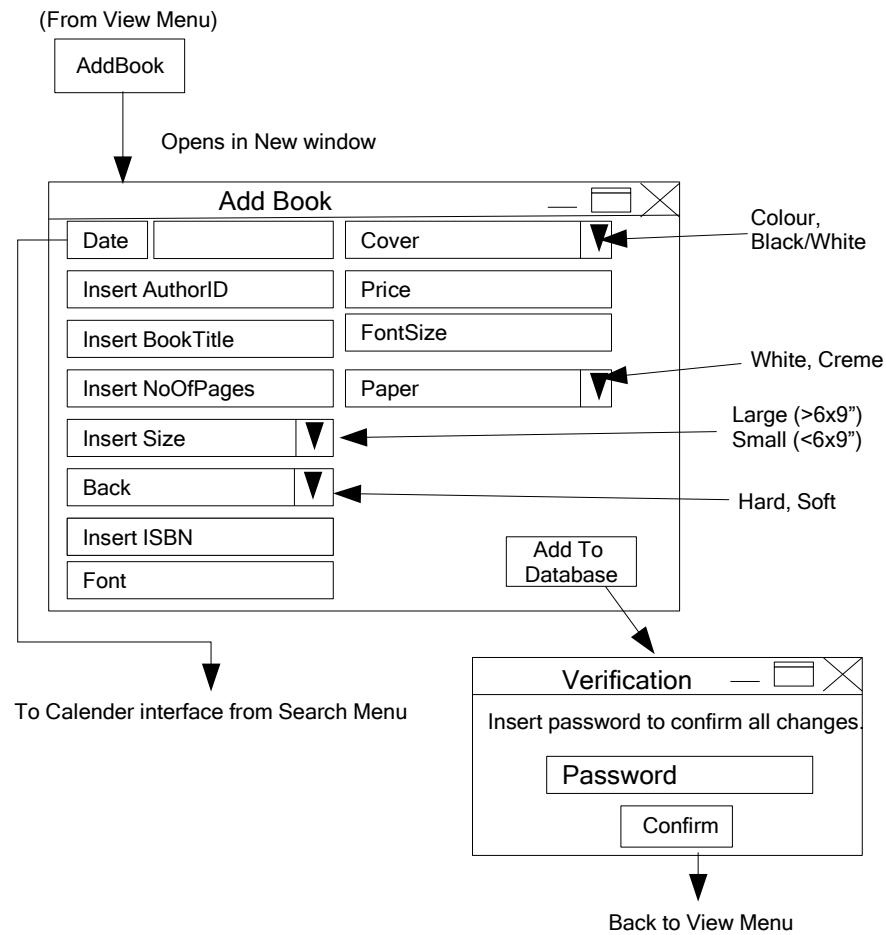


Figure 2.17: Add Book



## 2.3 Hardware Specification

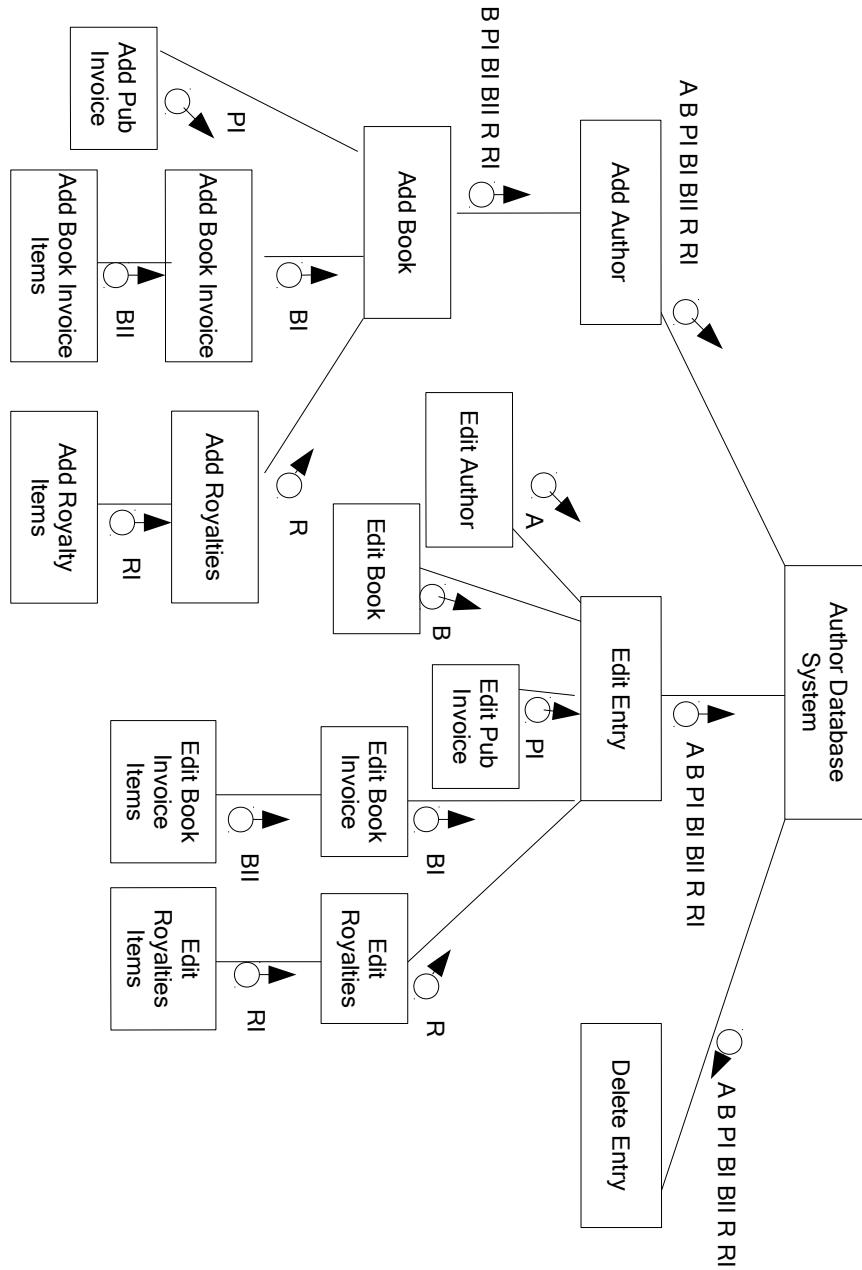
The system needs to be able to run on a laptop with a 1366 x 768, 16:9 aspect ratio screen which runs on Windows 8. This is imperative to the size of the application I will be creating because it must fit on the given screen size and can't be resizable. A mouse or touchpad will be used for navigational 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
 

---

```

1: function ADDENTRY(CustomerTable) Loop = 1
2:   SET GetNewAuthorID TO False
3:   SET ConfirmClicked TO False NewAuthorID = 0
4:   WHILE GetNewAuthorID = false DO
5:     NewAuthorID = loop Get AuthorID[loop] from CustomerTable
6:     IF NewAuthorID = AuthorID[loop] THEN
7:       loop = loop +1
8:     ELSE
9:       SET GetNewAuthorID TO True
10:      END IF
11:      END WHILE
12:      FirstName = null
13:      LastName = null
14:      Email = null
15:      PhoneNumber = null
16:      Address = null
17:      Postcode = null
18:      WHILE len(FirstName) <= 0 DO
19:        INPUT FirstName
20:      END WHILE
21:      WHILE len(LastName) <= 0 DO
22:        INPUT LastName
23:      END WHILE
24:      WHILE len>Email) <= 0 DO
25:        INPUT Email
26:      END WHILE
27:      WHILE len(PhoneNumber) <= 0 DO
28:        INPUT PhoneNumber
29:      END WHILE
30:      WHILE len(Address) <= 0 DO
31:        INPUT Address
32:      END WHILE
33:      WHILE len(Postcode) <= 0 DO
34:        INPUT Postcode
35:      END WHILE
36:      IF ConfirmClicked == True THEN
37:        CONNECT to Customer Database
38:      END IF
39:    END function

```

---

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

```

1: SET CalculateClicked TO False
2: SET AddToDatabaseClicked TO False
    Date = null
    WholeSalePrice = null
    QuantitySold = null
    Currency = null
    ExcRateFromGBP = null
    ISBN = null
    RoyaltyDiscount = null
3: WHILE len(Date) <= 0 DO
    INPUT Date
4: END WHILE
5: WHILE len(WholeSalePrice) <= 0 DO
    INPUT WholeSalePrice
6: END WHILE
7: WHILE len(QuantitySold) <= 0 DO
    INPUT LastName
8: END WHILE
9: WHILE len(Currency) <= 0 DO
    INPUT Currency
10: END WHILE
11: IF Currency <> $ THEN
12:     WHILE len(ExcRatefromGBP) <= 0 DO
            INPUT ExcRateFromGBP
13:     END WHILE
14: END IF
15: WHILE len(ISBN) <= 0 DO
    INPUT ISBN
16: END WHILE
17: WHILE len(RoyaltyDiscount) <= 0 DO
    INPUT RoyaltyDiscount
18:     IF RoyaltyDiscount > 100 THEN
        RoyaltyDiscount = null
19:     END IF
20:     IF RoyaltyDiscount < 0 THEN
        RoyaltyDiscount = null
21:     END IF
22: END WHILE

```

---

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

```

1: IF Cover == Colour THEN
    Size = Small
    PagePrice = 0.06 * NoOfPages
2:   IF Back == Hard THEN
    CoverPrice = 5
3:   END IF
4:   IF Back == Soft THEN
    CoverPrice = 3
5:   END IF
6: END IF
7: IF Cover == Black/White THEN
8:   IF Size == Large THEN
    PagePrice = 0.015 * NoOfPages
9:     IF Back == Hard THEN
    CoverPrice = 5
10:    END IF
11:    IF Back == Soft THEN
    CoverPrice = 1
12:    END IF
13:  END IF
14:  IF Size == Small THEN
    PagePrice = 0.01 * NoOfPages
15:    IF Back == Hard THEN
    CoverPrice = 4
16:    END IF
17:    IF Back == Soft THEN CoverPrice = 0.7
18:    END IF
19:  END IF
20: END IF
    PrintCost = (PagePrice + CoverPrice) * RoyaltyQuantity
21: IF CalculateClicked == True THEN
    NetSales = WholeSalePrice * RoyaltyQuantity
    RoyaltyPayment = NetSales - PrintCost
    OUTPUT RoyaltyPayment
22: END IF
23: IF AddToDatabaseClicked == True THEN
    CONNECT to RoyaltyItems Database
24: END IF

```

---

---

**Algorithm 6** Add and Calculate Invoice Items

---

```

1: function ADDINVOICEITEM(BookTable)
2:   SET CalculateClicked TO False
3:   SET AddToDatabaseClicked TO False
   Date = null
   QuantityBought = null
   InvoiceDiscount = null
   ShippingType= null
   ShippingPrice= null
   ISBN = null
4:   WHILE len(Date) <= 0 DO
   INPUT Date
5:   END WHILE
6:   WHILE len(QuantityBought) <= 0 DO
   INPUT QuantityBought
7:   END WHILE
8:   WHILE len(InvoiceDiscount) <= 0 DO
   INPUT InvoiceDiscount
9:     IF InvoiceDiscount > 100 THEN
   InvoiceDiscount = null
10:    END IF
11:    IF InvoiceDiscount < 0 THEN
   InvoiceDiscount = null
12:    END IF
13:   END WHILE
14:   WHILE len(ISBN) <= 0 DO
   INPUT ISBN
15:   END WHILE
16:   WHILE len(ShippingType) <= 0 DO
   INPUT ShippingType
17:   END WHILE
18:   WHILE len(ShippingPrice) <= 0 DO
   INPUT ShippingPrice
19:   END WHILE
20:   IF CalculateClicked == True THEN
   InvoicePayment = (QuantityBought * Price * InvoiceDiscount) + Ship-
   ping Price
   OUTPUT InvoicePayment
21:   END IF
22:   IF AddToDatabaseClicked == True THEN
   CONNECT to Invoice Database
23:   END IF
24: END function

```

---

**Algorithm 7** Add Book

---

```
1: function ADDINVOICEITEM(CustomerTable)
2:   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
3:   WHILE len(DatePublished) <= 0 DO
4:     INPUT DatePublished
5:   END WHILE
6:   WHILE len(ISBN) <= 0 DO
7:     INPUT ISBN
8:   END WHILE
9:   WHILE len(BookTitle) <= 0 DO
10:    INPUT BookTitle
11:   END WHILE
12:   WHILE len(Size) <= 0 DO
13:     INPUT Size
14:   END WHILE
15:   WHILE len(BackType) <= 0 DO
16:     INPUT BackType
17:   END WHILE
18:   WHILE len(NoOfPages) <= 0 DO
19:     INPUT NoOfPages
20:   END WHILE
21:   WHILE len(Paper) <= 0 DO
22:     INPUT Paper
23:   END WHILE
24:   WHILE len(Font) <= 0 DO
25:     INPUT Font
26:   END WHILE
27:   WHILE len(FontSize) <= 0 DO
28:     INPUT FontSize
29:   END WHILE
30:   WHILE len(Price) <= 0 DO
31:     INPUT Price
32:   END WHILE
33:   IF AddToDatabaseClicked == True THEN
34:     CONNECT to Book Database
35:   END IF
36: END function
```

---

**Algorithm 8** Add and Publishing Invoice

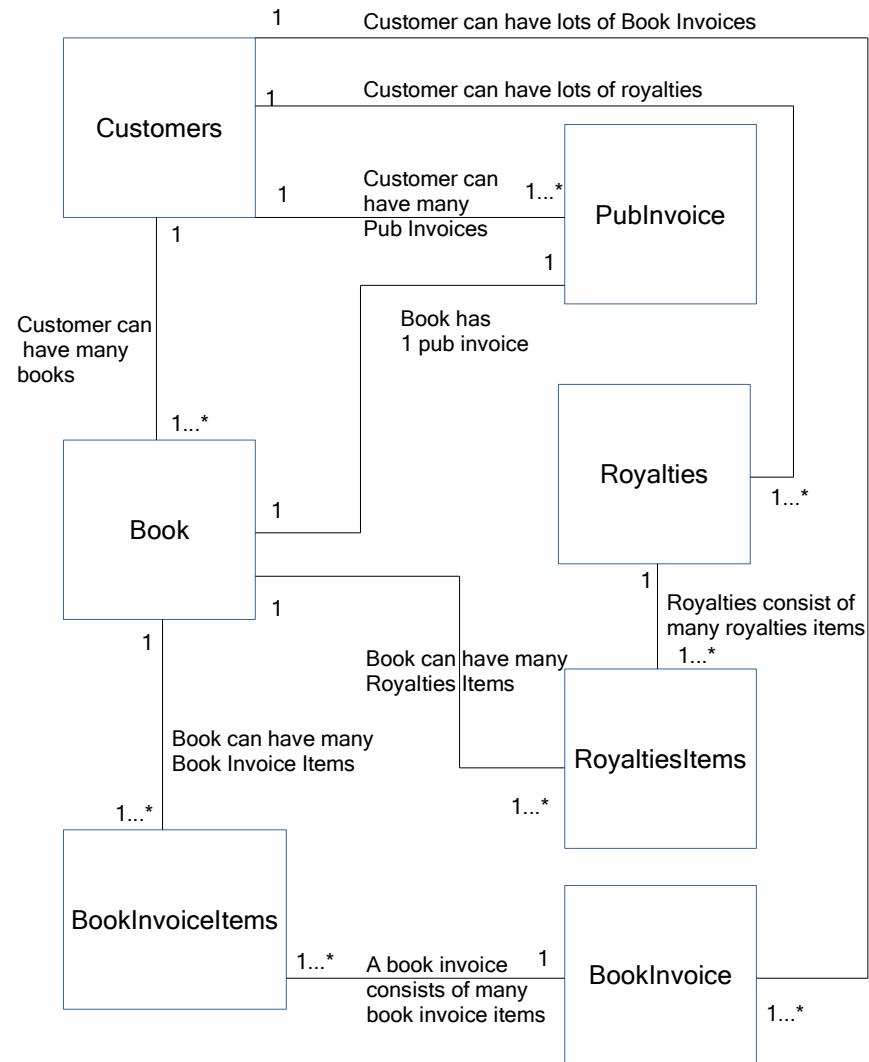
---

```
1: function ADDINVOICEITEM(BookTable, CustomerTable)
2:   SET AddToDatabaseClicked TO False
   Date = null
   AuthorID = null
   ServiceType = null
   InvoicePayment= null
   ISBN = null
3:   WHILE len(Date) <= 0 DO
   INPUT Date
4:   END WHILE
5:   WHILE len(AuthorID) <= 0 DO
   INPUT AuthorID
6:   END WHILE
7:   WHILE len(InvoicePayment) <= 0 DO
   INPUT InvoicePayment
8:   END WHILE
9:   WHILE len(ISBN) <= 0 DO
   INPUT ISBN
10:  END WHILE
11:  WHILE len(ServiceType) <= 0 DO
   INPUT ServiceType
12:  END WHILE
13:  IF AddToDatabaseClicked == True THEN
   CONNECT to PubInvoice Database
14:  END IF
15: END function
```

---

### 2.4.3 Object Diagrams

Figure 2.19: Object Diagram



#### 2.4.4 Class Definitions

Key:

Label
Attributes
Behaviours

Customer
Author ID
ForeName
Surname
Email
Address
Postcode
Phone Number
Add ForeName
Edit Forename
Add Surname
Edit Surname
Add Email
Edit Email
Add Address
Edit Address
Add Postcode
Edit Postcode
Add Phone Number
Edit Phone Number

<b>Book</b>
ISBN
AuthorID
BookTitle
NoOfPages
Size
Cover
Back
Paper
Font
FontSize
Price
DatePublished
Add ISBN
Edit ISBN
Add AuthorID
Edit AuthorID
Add BookTitle
Edit BookTitle
Add NoOfPages
Edit NoOfPages
Add Size
Edit Size
Add Cover
Edit Cover
Add Back
Edit Back
Add Paper
Edit Paper
Add Font
Edit Font
Add FontSize
Edit FontSize
Add DatePublished
Edit DatePublished
Add Price
Edit Price

<b>Royalties</b>
RoyaltiesID
AuthorID
RoyaltyPayment
RoyaltiesDate
Add AuthorID
Edit AuthorID
Add RoyaltiesDate
Edit Royalties Date

<b>Royalties Items</b>
RoyaltiesItems
RoyaltiesID
ISBN
Currency
RoyaltyDiscount
WholesalePrice
RoyaltyQuantity
NetSales
PrintCost
ExcRateFromGBP
Add RoyaltiesID
Edit RoyaltiesID
Add ISBN
Edit ISBN
Add Currency
Edit Currency
Add RoyaltyDiscount
Edit RoyaltyDiscount
Add WholesalePrice
Edit WholesalePrice
Add RoyaltyQuantity
Edit RoyaltyQuantity
Add ExcRateFromGBP
Edit ExcRateFromGBP

<b>BookInvoice</b>
BookInvoiceID
AuthorID
BookInvoicePayment
BookInvoiceDate
Add AuthorID
Edit AuthorID
Add BookInvoiceDate
Edit BookInvoiceDaate

<b>BookInvoiceItems</b>
BookInvoiceItems
BookInvoiceID
ISBN
BookInvoiceQuantity
BookInvoiceDiscount
ShippingType
ShippingPrice
Add BookInvoiceID
Edit BookInvoiceID
Add ISBN
Edit ISBN
Add BookInvoiceQuantity
Edit BookInvoiceQuantity
Add BookInvoiceDiscount
Edit BookInvoiceDiscount
Add ShippingType
Edit Shipping Type
Add ShippingPrice
Edit ShippingPrice

<b>PubInvoice</b>
PubInvoiceID
AuthorID
ISBN
PubInvoiceDate
PubInvoiceService
PubInvoicePayment
Add AuthorID
Edit AuthorID
Add ISBN
Edit ISBN
Add PubInvoiceDate
Edit PubInvoiceDate
Add PubInvoiceService
Edit PubInvoiceService
Add PubInvoicePayment
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

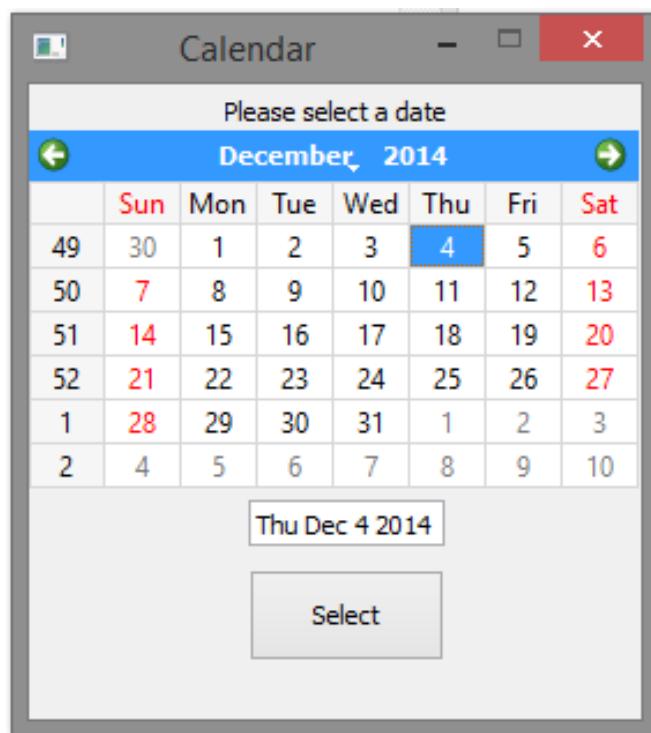
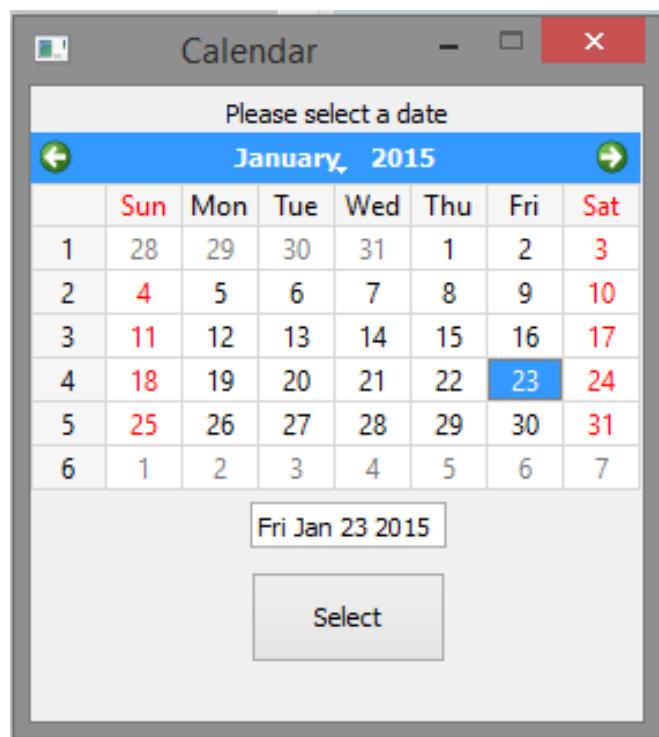


Figure 2.21: Calendar - Date Selection



## 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/yyyy	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

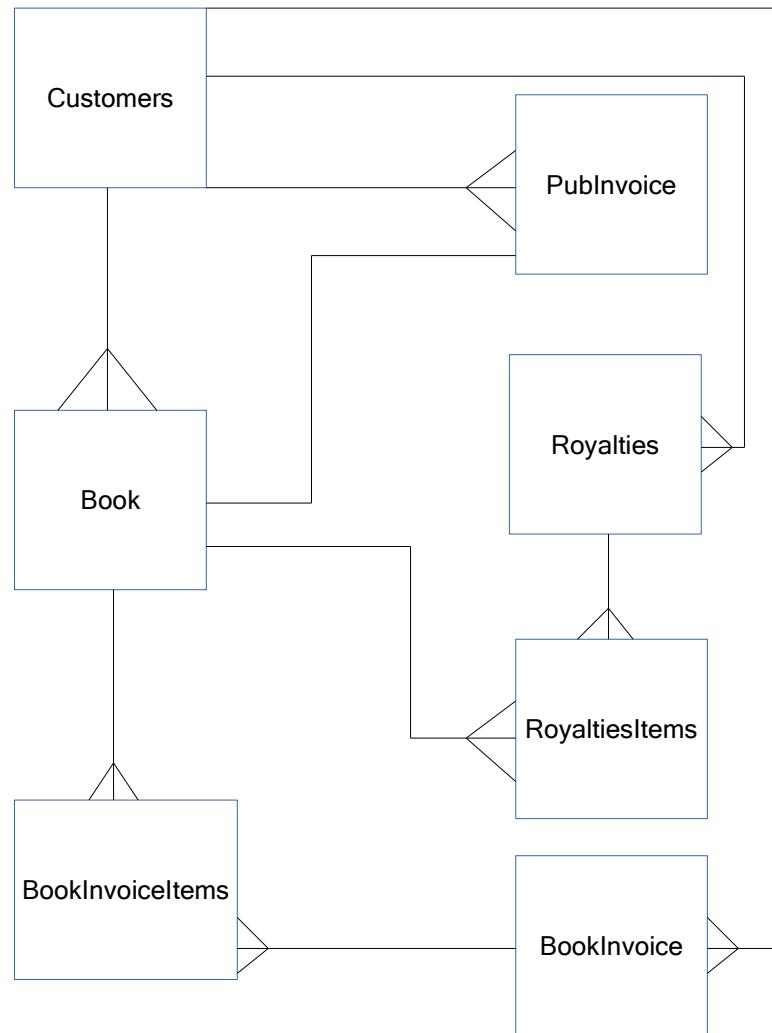
off site, as well as being able to hold all data files on it. The database will be needed to be kept for long term storage. This is the data will be required to be accessed a number of times.

## 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, separating the repeating attributes and the non-repeating attributes.

<b>AuthorID</b>	<b>ISBN</b>
FirstName	<b>AuthorID</b>
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.

<b>AuthorID</b>	<b>ISBN</b>	<b>ISBN</b>
FirstName	<b>AuthorID</b>	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	
	BookInvoiceItems	
	BookInvoicePayment	
	BookInvoiceDate	
	BookInvoiceDiscount	
	BookInvoiceQuantity	
	ShippingType	
	ShippingPrice	

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

<b>AuthorID</b>	<b>ISBN</b>	<b>RoyaltiesID</b>	<b>RoyaltiesItems</b>
FirstName	<i>AuthorID</i>	<i>AuthorID</i>	<i>RoyaltiesID</i>
LastName	BookTitle	RoyaltyPayment	<i>ISBN</i>
Email	NoOfPages	RoyaltiesDate	Currency
PhoneNumber	Size		RoyaltyDiscount
Address	Back		WholeSalePrice
PostCode	Cover		RoyaltyQuantity
	Paper		NetSales
	Font		PrintCost
	FontSize		ExcRateFromGBP
	DatePublished		
	Price		

<b>PubInvoiceID</b>	<b>BooksInvoiceID</b>	<b>BooksInvoiceItems</b>
<i>AuthorID</i>	<i>AuthorID</i>	<i>BooksInvoiceID</i>
<i>ISBN</i>	BookInvoicePayment	<i>ISBN</i>
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 first-name 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.

<b>Item</b>	<b>Example</b>	<b>Validation</b>	<b>Comments</b>
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.

<b>Item</b>	<b>Example</b>	<b>Validation</b>	<b>Comments</b>
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 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 separate 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 separate 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 dropdown 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 Selected from list, values entered and no names entered Just values entered Nothing	Normal Normal Erroneous Erroneous	Accept Accept Error 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 helloworld test @test-mail.com test.com@testmail	Normal Erroneous Erroneous Erroneous	Accept Error Error 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 Jo hn Jo?hn Jo2n	Normal Erroneous Erroneous Erroneous	Accept Error Error Error		

2.4	Verify that a valid Lastname has been entered when adding an entry	The user will be prompted with an error	Smith Sm ith Smilth Smi1th	Normal Erroneous Erroneous Erroneous	Accept Error Error Error		
2.5	Verify that a valid Phonenumber has been entered when adding an entry	The user will be prompted with an error	07123456789 07123123.3 071CO2	Normal Erroneous Erroneous	Accept Error 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 @@ £! 1231231	Normal Erroneous Erroneous	Accept Error Error		
2.8	Verify that a valid ISBN has been entered when adding a book	The user will be prompted with an error	1234567890123 @@ £! 123 HelloWorld	Normal Erroneous Erroneous Erroneous	Accept Error Error Error		
2.9	Verify that a valid NoOfPages has been entered when adding a book	The user will be prompted with an error	123 123456789 2.3 @! HelloWorld	Normal Erroneous Erroneous Erroneous	Accept Error Error		

2.10	Verify that a valid ISBN has been entered when adding a book	The user will be prompted with an error	1234567890123 @@ £! 123 HelloWorld	Normal Erroneous Erroneous Erroneous	Accept Error Error 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 Leave Fields Blank and then click Add to Database	Normal Erroneous	Verification screen should open 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 Leave Fields Blank and click Calculate Fill all fields correctly and click add to database	Normal Erroneous Erroneous	Verification screen should open User will be prompted with an error 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 Leave Fields Blank and click Calculate Fill all fields correctly and click add to database	Normal Erroneous Erroneous	Verification screen should open User will be prompted with an error 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

104

### 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

There were no changes to the purposes of the test series and strategies, however I have added all the strategy rationales.

Test Series	Purpose of Test Series	Testing Strategy	Strategy Rationale
1	Testing the flow of control between user interfaces	Top-down Testing	Tests the outputs alone, disregarding internal functionality
2	Testing the validation of input data	Bottom-up testing	Components will be tested after development
3	Testing the algorithms' functionality	White box testing	Components will be tested after development
4	Testing that the information has been successfully stored, and in the right places	Black box testing	Focuses on the process of adding/editing/deleting data
5	Testing the system and whether it meets the requirements	System testing	Tests the flow of control with all components briefly tested

### 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 separate 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 separate 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 Selected from list, values entered and no names entered Just values entered Nothing	Normal Normal Erroneous Erroneous	Accept Accept Error 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 helloworld test @test-mail.com test.com@testmail	Normal Erroneous Erroneous Erroneous	Accept Error Error 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 Jo hn Jo?hn Jo2n	Normal Erroneous Erroneous Erroneous	Accept Error Error Error		

2.4	Verify that a valid Last-name has been entered when adding an entry	The user will be prompted with an error	Smith Sm ith Smi?th Smi1th	Normal Erroneous Erroneous Erroneous	Accept Error Error Error		
2.5	Verify that a valid Phonenumber has been entered when adding an entry	The user will be prompted with an error	07123456789 07123123.3 071CO2	Normal Erroneous Erroneous	Accept Error 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 @@ £! 1231231	Normal Erroneous Erroneous	Accept Error Error		
2.8	Verify that a valid ISBN has been entered when adding a book	The user will be prompted with an error	1234567890123 @@ £! 123 HelloWorld	Normal Erroneous Erroneous Erroneous	Accept Error Error Error		

2.9	Verify that a valid NoOfPages has been entered when adding a book	The user will be prompted with an error	123 123456789 2.3 @! HelloWorld	Normal Erroneous Erroneous Erroneous	Accept Error 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 Leave Fields Blank and then click Add to Database	Normal Erroneous	Verification screen should open 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 Leave Fields Blank and click Calculate Fill all fields correctly and click add to database	Normal Erroneous Erroneous	Verification screen should open User will be prompted with an error 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 Leave Fields Blank and click Calculate Fill all fields correctly and click add to database	Normal Erroneous Erroneous	Verification screen should open User will be prompted with an error 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 system due to changes made during the development 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 foreseen 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 separate 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 Firsname, 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

Imran Rahman

Candidate No. 30928

Centre No. 22151

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 separate 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
128	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. Author Selected, boxes left blank Publishing Invoice selected, Service selected, boxes filled with appropriate data Publishing Invoice selected, No boxes filled.	Normal Erroneous Normal Erroneous	Accept Error Accept 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 helloworld test @testmail.com test.com@testmai	Normal Erroneous Erroneous Erroneous	Accept Error Error 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 Jo hn Jo?hn Jo2n	Normal Erroneous Erroneous Erroneous	Accept Error Error 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 Sm ith Smi?th Smi1th	Normal Erroneous Erroneous Erroneous	Accept Error Error 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 07123123.3 071CO2	Normal Erroneous Erroneous	Accept Error 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 @@ £! 1231231	Normal Erroneous Erroneous	Accept Error 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 @@ £! 1234567890123450 0123	Normal Erroneous Erroneous Erroneous	Accept Error 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 NoOfPages has been entered when adding a book	The user will be prompted with an error	123 123456789 2.3 @! HelloWorld	Normal Erroneous Erroneous Erroneous	Accept Error Error 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 5.5 0 -1 100 101 abc	Normal Normal Boundary Erroneous Boundary Erroneous Erroneous	Accept Error Accept Error Accept Error 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 5.5 0 -1 100 101 abc	Normal Normal Boundary Erroneous Boundary Erroneous Erroneous	Accept Error Accept Error Accept Error 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 Leave Fields Blank and then click Add to Database	Normal Erroneous	Data is added to the database. 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 Leave Fields Blank and click Calculate Fill all fields correctly and click add to database	Normal Erroneous Erroneous	Data should be successfully added to the database. User will be prompted with an error 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 Leave Fields Blank and click Calculate Fill all fields correctly and click add to database	Normal Erroneous Erroneous	Verification screen should open User will be prompted with an error 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 Leave fields blank Enter negative numbers into the required fields.	Normal Erroneous Erroneous	Accept Error 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 Leave fields blank Enter negative numbers into the required fields.	Normal Erroneous Erroneous	Accept Error 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 Leave fields blank Enter negative numbers into the required fields.	Normal Erroneous Erroneous	Accept Error 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 after 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 after 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 successfully edited after 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 successfully edited after 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 the user is deleted, and should no longer be in the table	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 author should 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

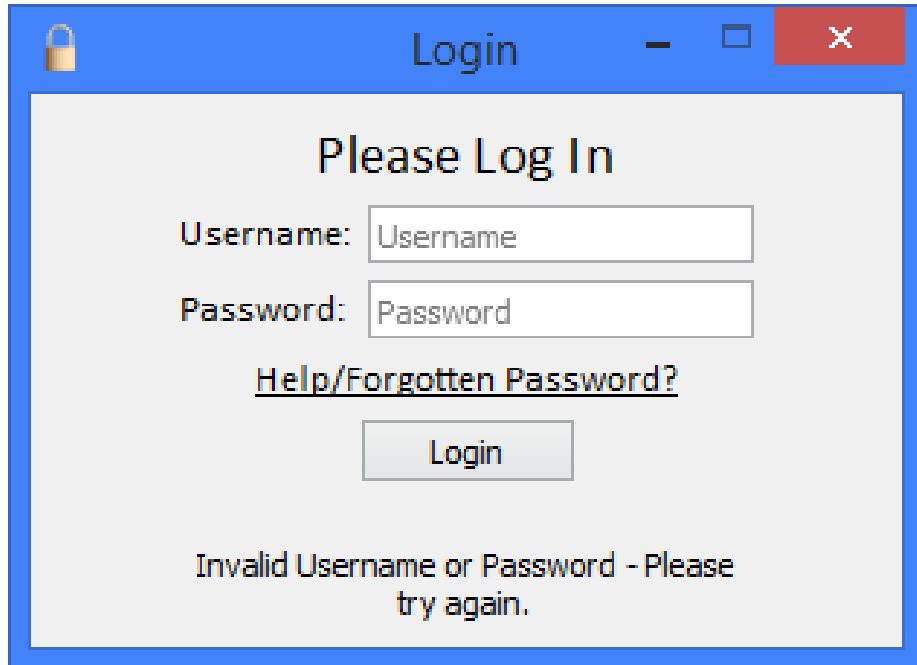


Figure 3.1: Test 1.1 - Invalid Login Credentials

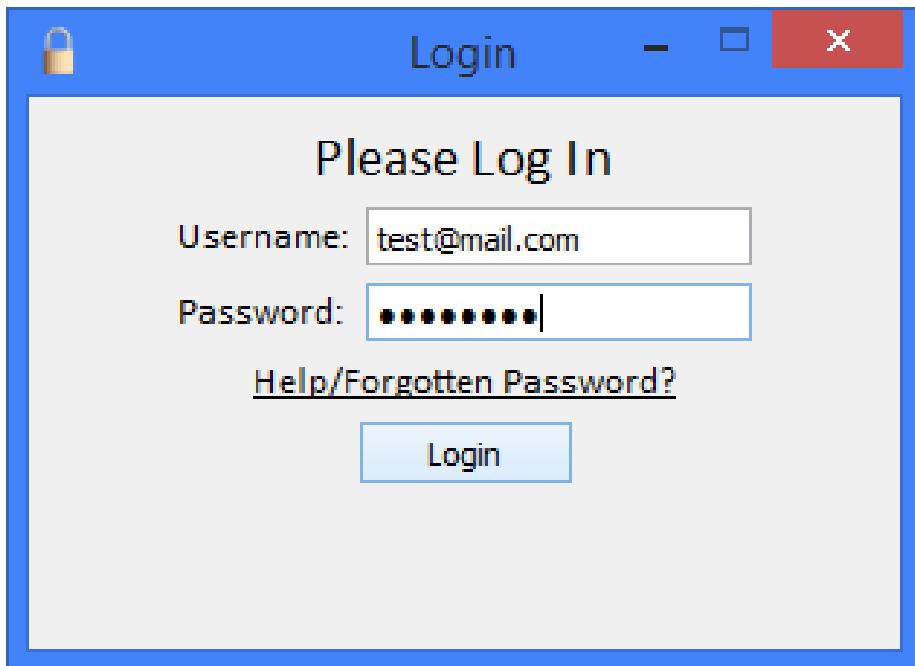


Figure 3.2: Test 1.1 - Correct Login Credentials

The screenshot shows a Windows application window titled "Main Menu". The menu bar includes "Database", "Actions", and "Account". A toolbar at the top right contains "Log Out", "Author Name", and "Quick Search". Below the toolbar is a table with columns: AuthorID, FirstName, LastName, Email, PhoneNumber, Address, and Postcode. The table contains five rows of sample data:

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

At the bottom of the window are buttons for "View", "Search Database", "Add Entry", "Update Entry", "Remove Entry", and "Change Username/Password".

Figure 3.3: Test 1.1 - User granted access

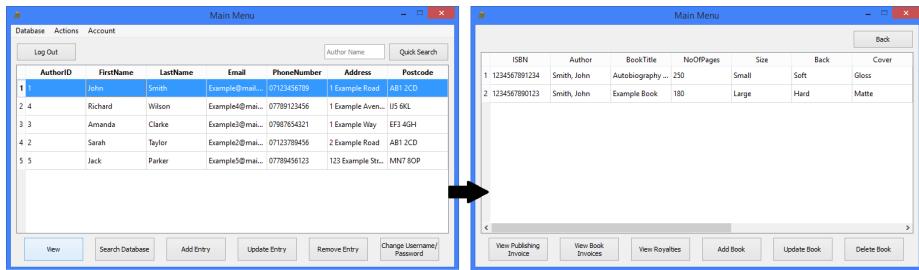


Figure 3.4: Test 1.2 - Testing the View Button after selecting a customer

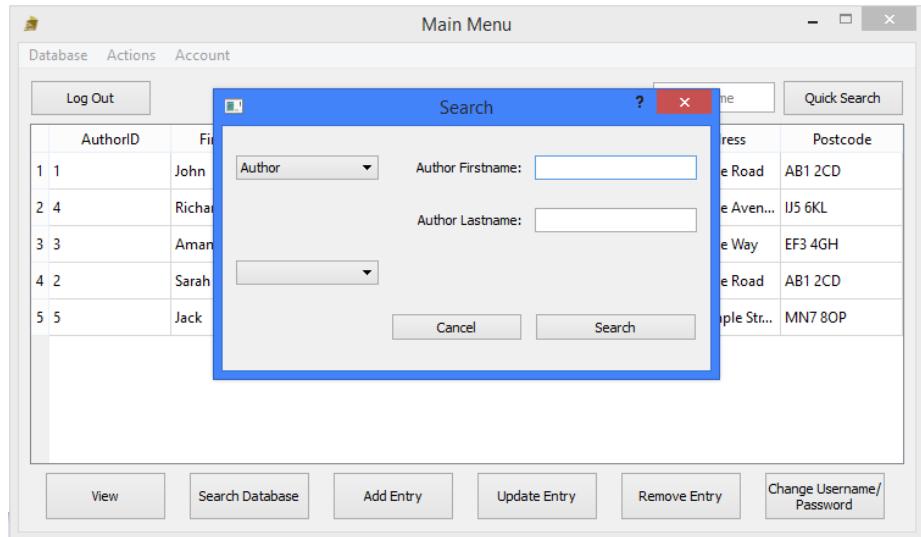


Figure 3.5: Test 1.4 - Testing the Search Database Button

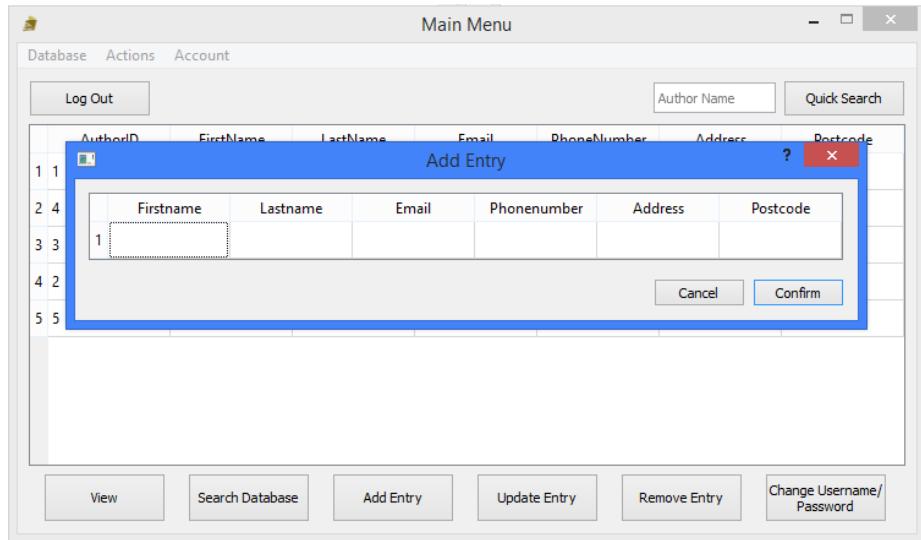


Figure 3.6: Test 1.5 - Testing the Add Entry button

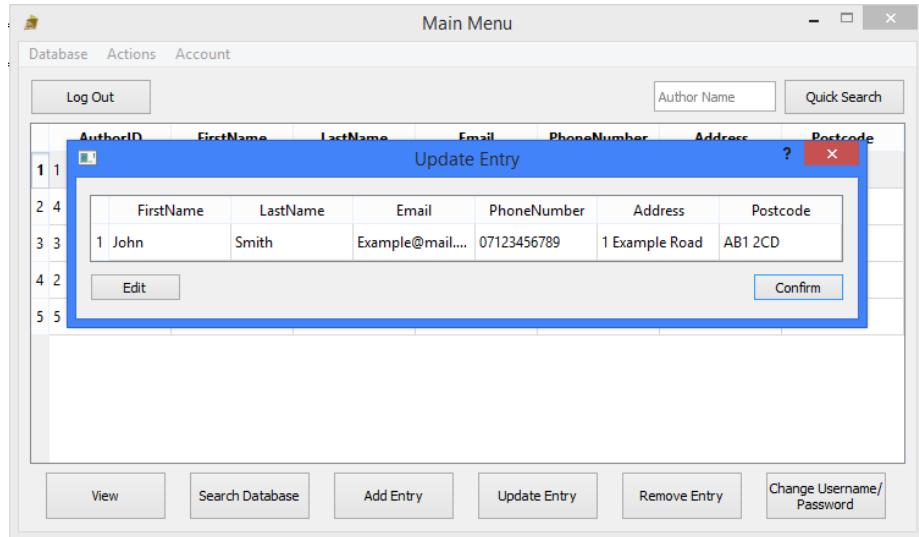


Figure 3.7: Test 1.7 - Testing the Update Entry Button after selecting customer

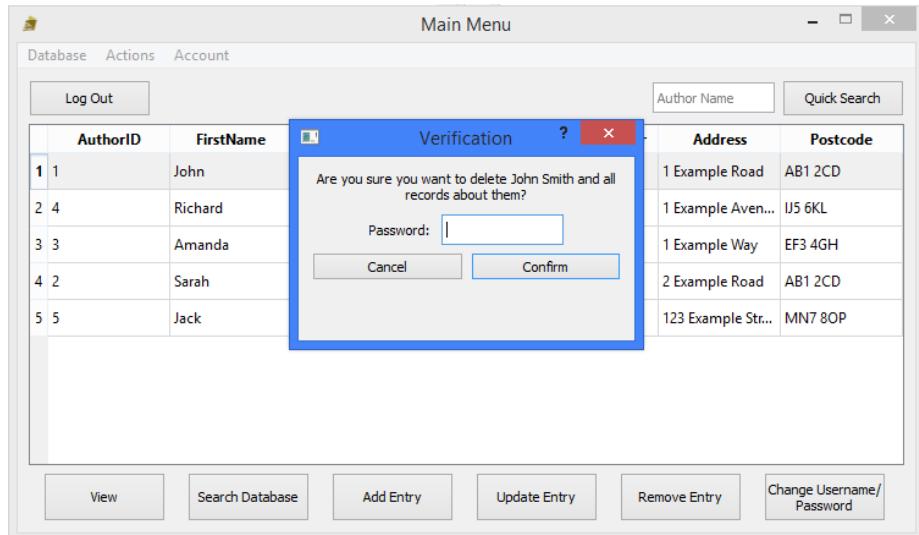


Figure 3.8: Test 1.8 - Testing the Remove Entry Button after selecting a customer

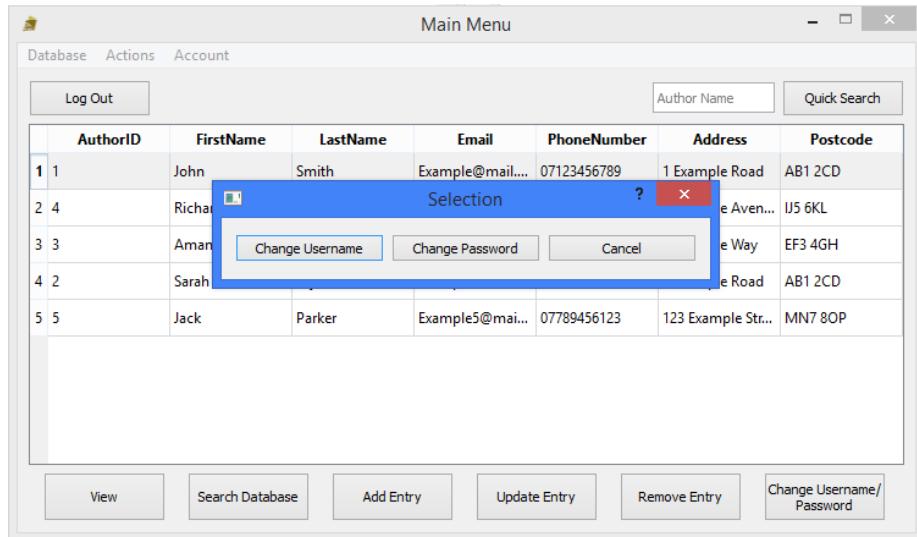


Figure 3.9: Test 1.9 - Testing the Change Username/Password Button

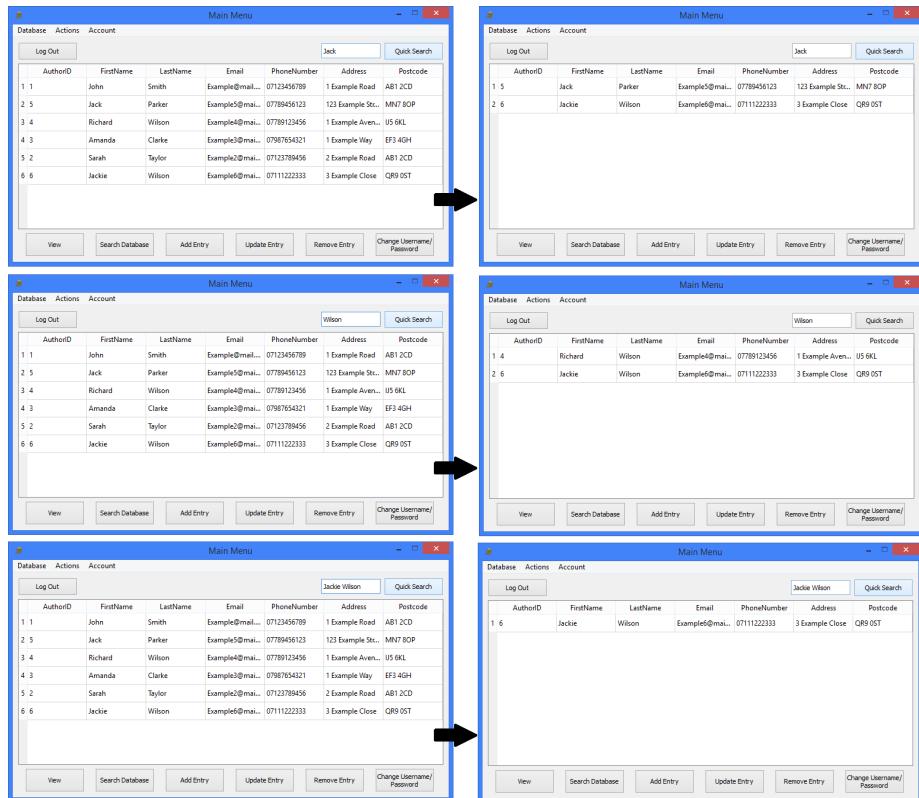


Figure 3.10: Test 1.10 - Testing the Quick Search Button after entering some data

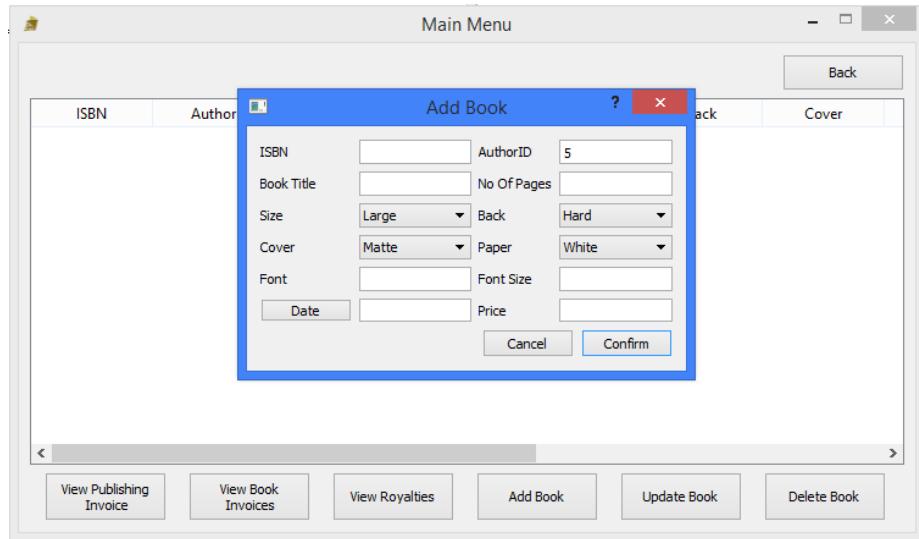


Figure 3.11: Test 1.12 - Testing the Add Book Button

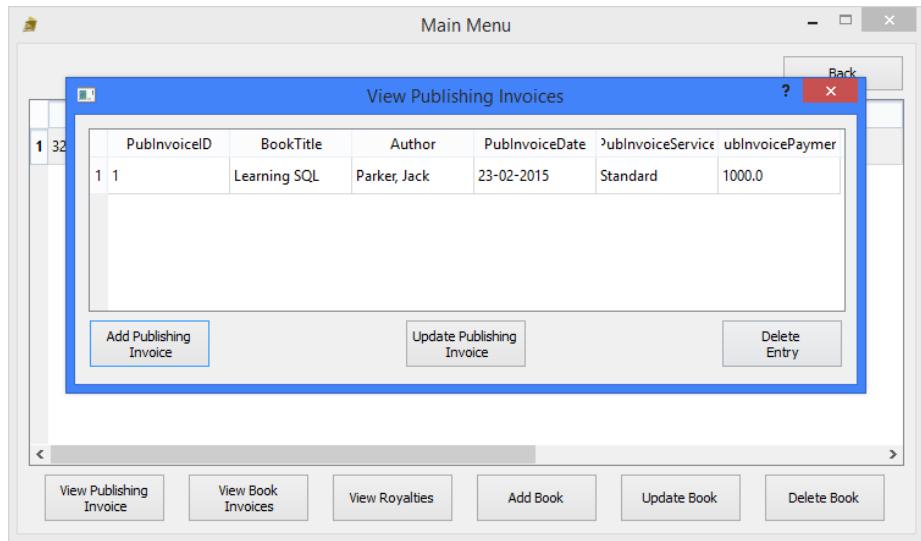


Figure 3.12: Test 1.13 - Testing the View Publishing Invoice Button after selecting a book

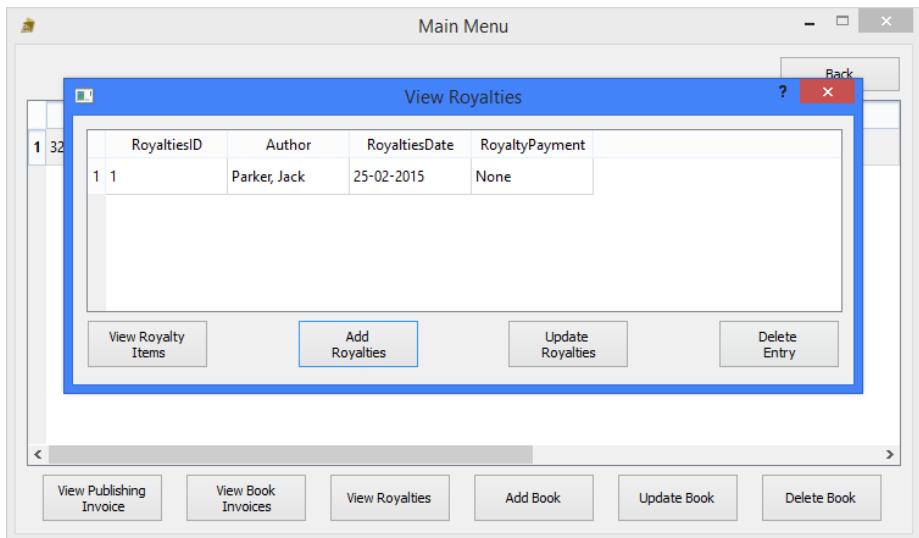


Figure 3.13: Test 1.14 - Testing the View Royalties Button after selecting a book

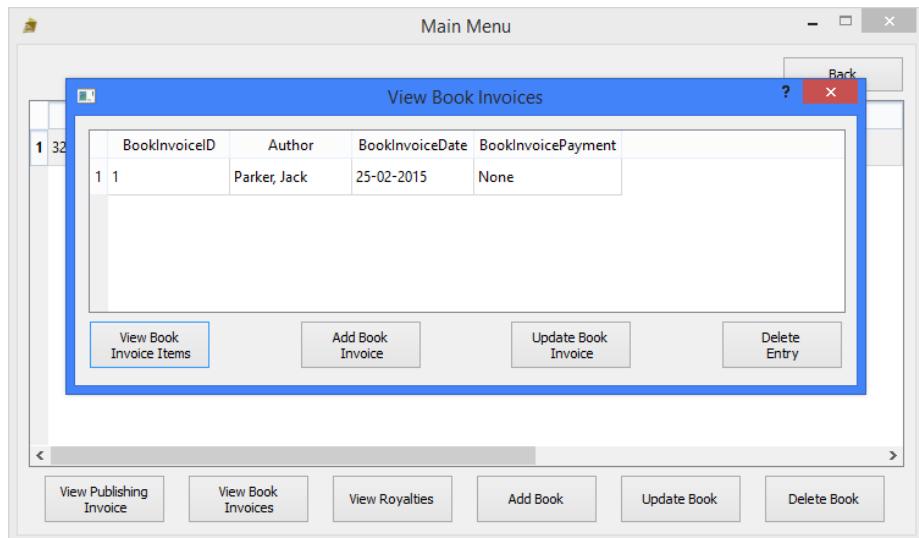


Figure 3.14: Test 1.15 - Testing the View Book Invoice Button after selecting a book

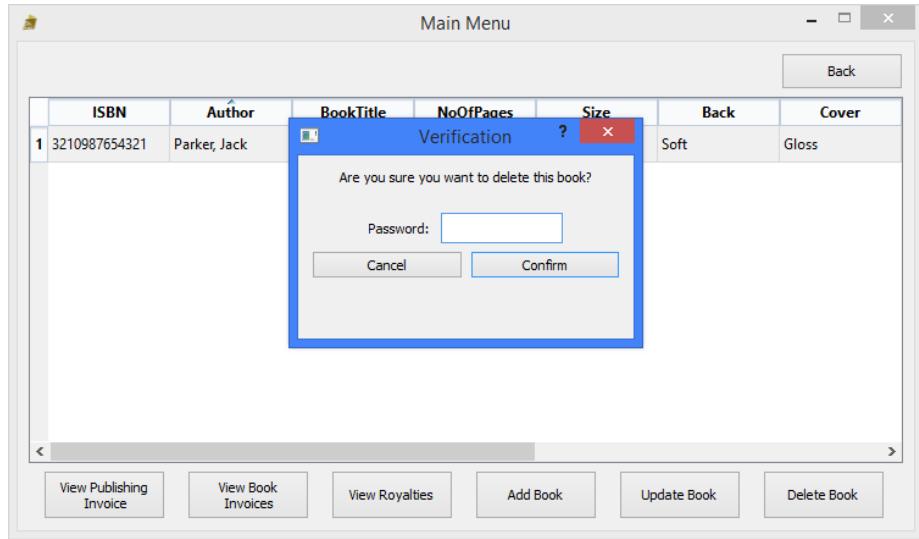


Figure 3.15: Test 1.16 - Testing the Delete Book Button after selecting a book

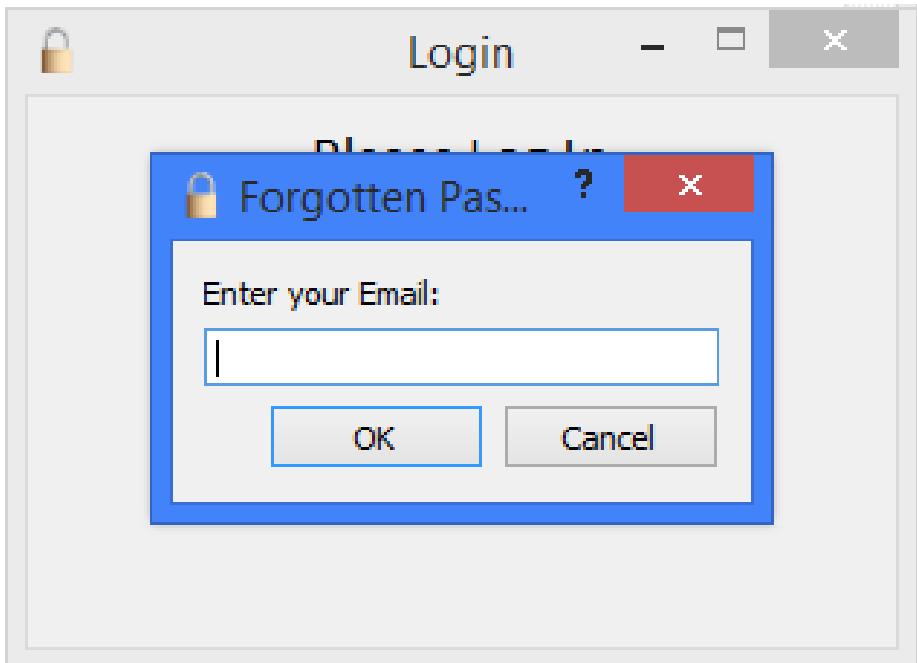


Figure 3.16: Test 1.17 - Testing the Forgot Password label on the login screen

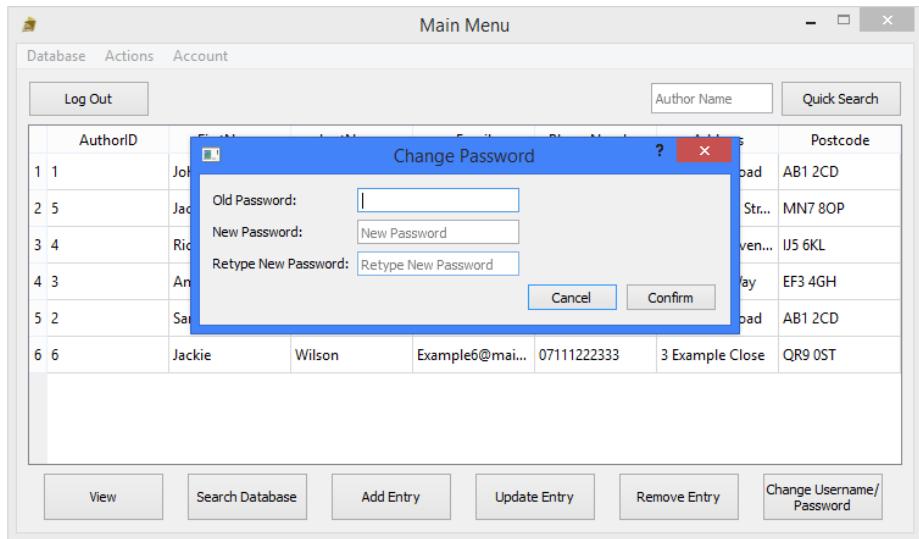


Figure 3.17: Test 1.18 - Testing the Change Password button



Figure 3.18: Test 1.19 - Testing the Confirm button after entering new password details

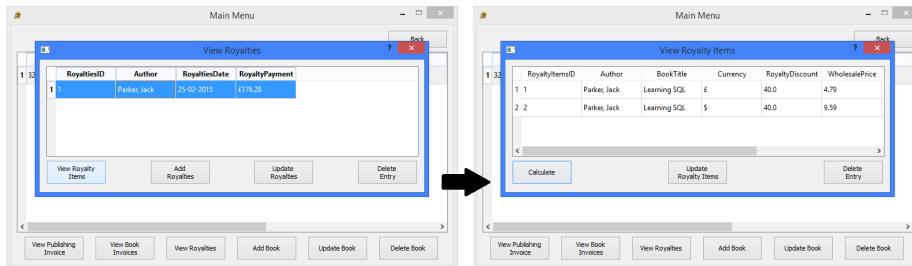


Figure 3.19: Test 1.20 - Testing the View Royalty Items button after selecting a Royalty payment

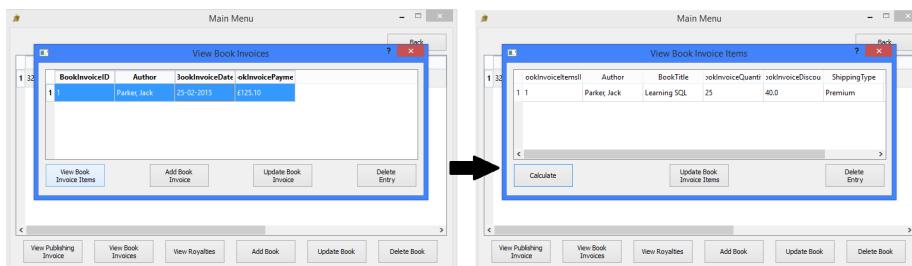


Figure 3.20: Test 1.21 - Testing the View Book Invoice Items button after selecting a Book Invoice

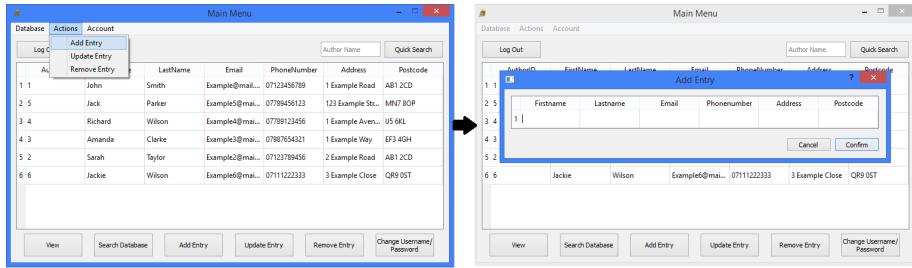


Figure 3.21: Test 1.23 - Testing the Add Entry option on the menu bar

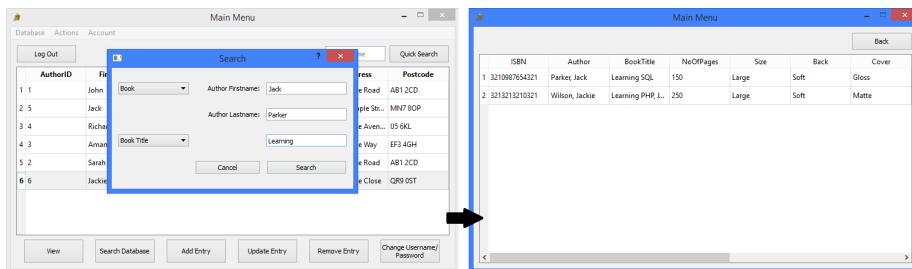


Figure 3.22: Test 1.24 - Testing the Confirm button on the search window after having entered some data

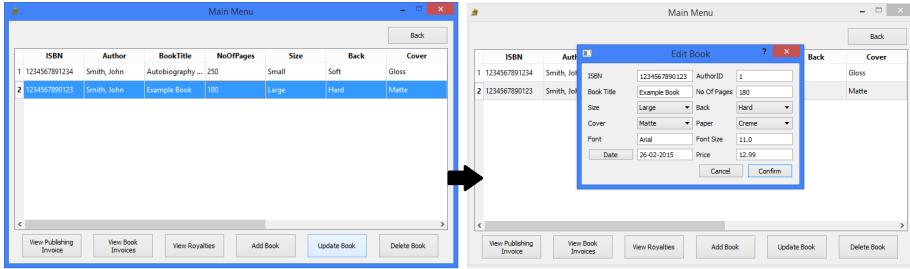


Figure 3.23: Test 1.27 - Testing the Update Book button after selecting a book

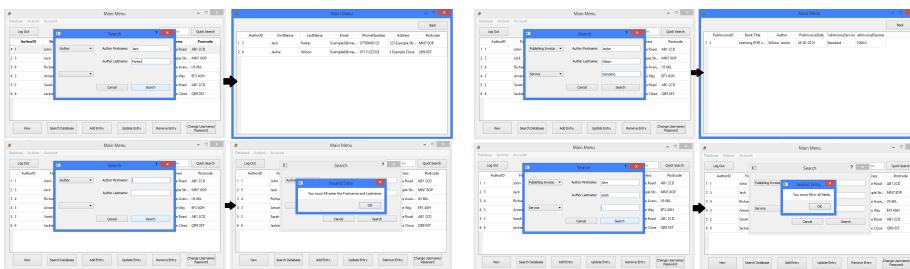


Figure 3.24: Test 2.1 - Validation for inputs in the search window

Here, the user is unable to type 'Sm ith', 'Smi?th', or 'Smi1th'

Each cell in the table has a validator, preventing certain entries for each entry type

Firstname	Lastname	Email	Phonenumber	Address	Postcode
1 John	Smith	jsmith@mail.com	07123456789	1 Example Road	AB1 2CD

Add Entry

Cancel Confirm

Figure 3.25: Test 2.3, 2.4, 2.5, 2.6 - Validation for adding customer details

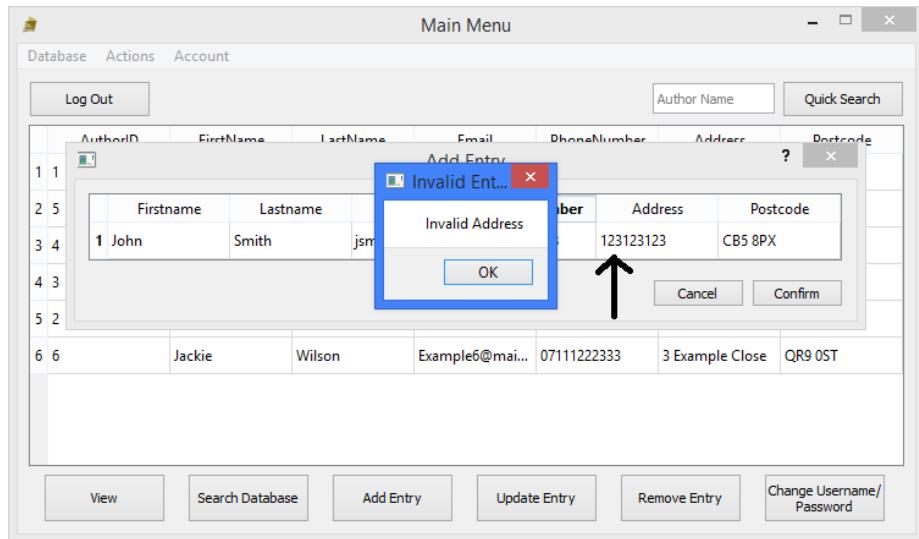


Figure 3.26: Test 2.6 - Prompting the user with an error when an invalid address has been entered

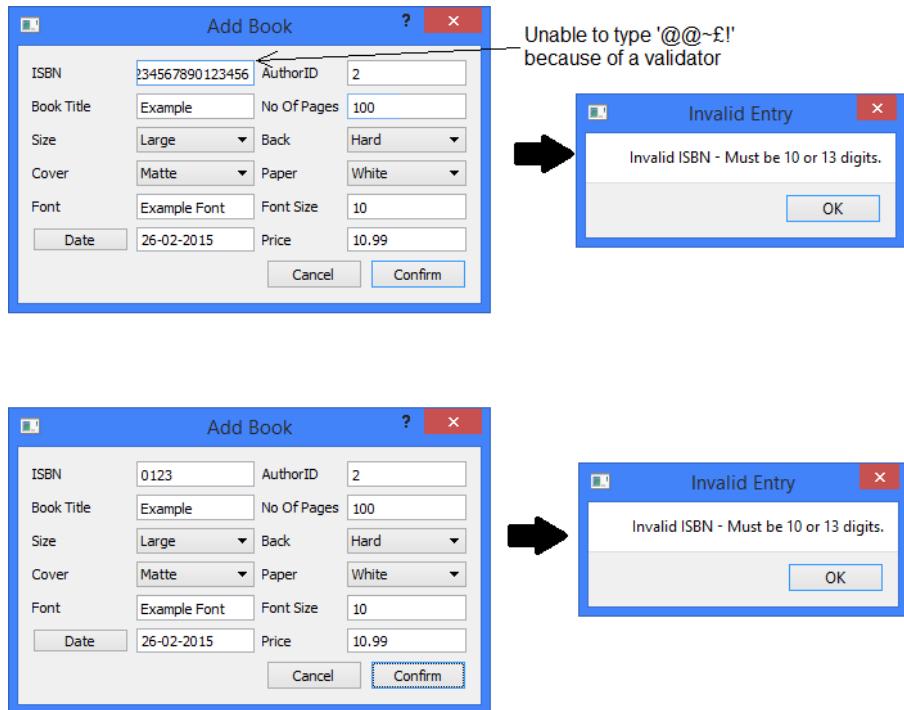


Figure 3.27: Test 2.7 - Validating the ISBN

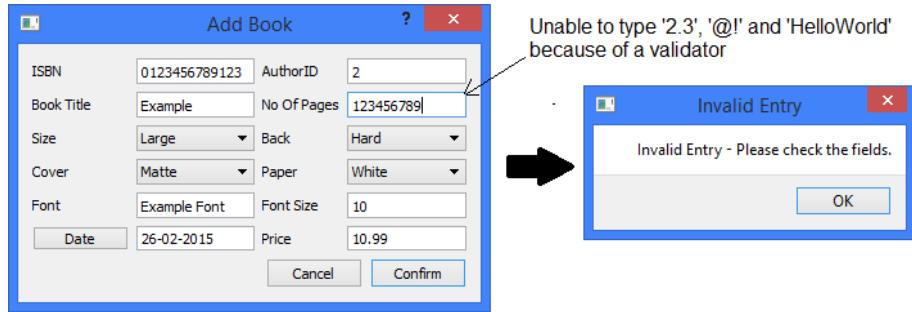


Figure 3.28: Test 2.8 - Validating the No of pages entered

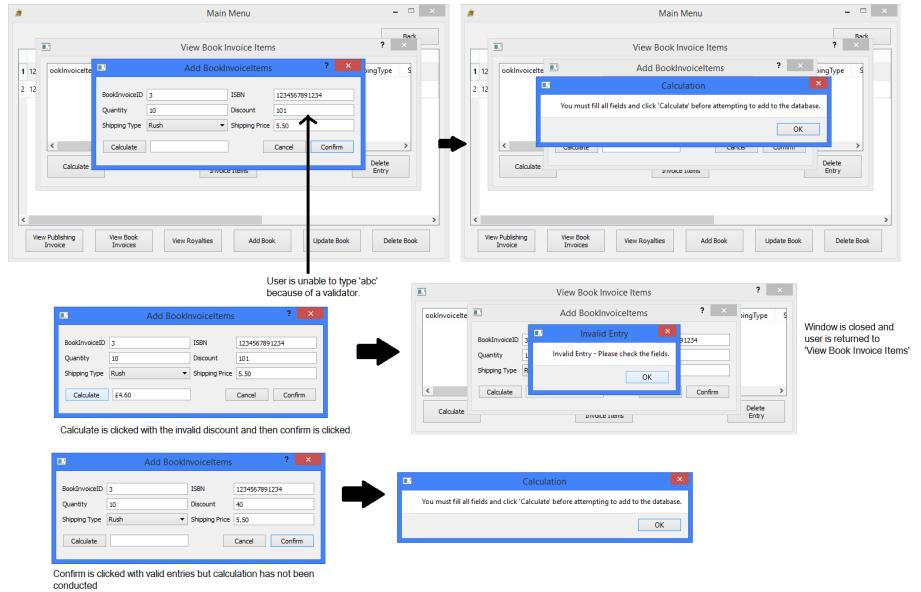


Figure 3.29: Test 2.9 - Validating the Discount when adding Book Invoice Items

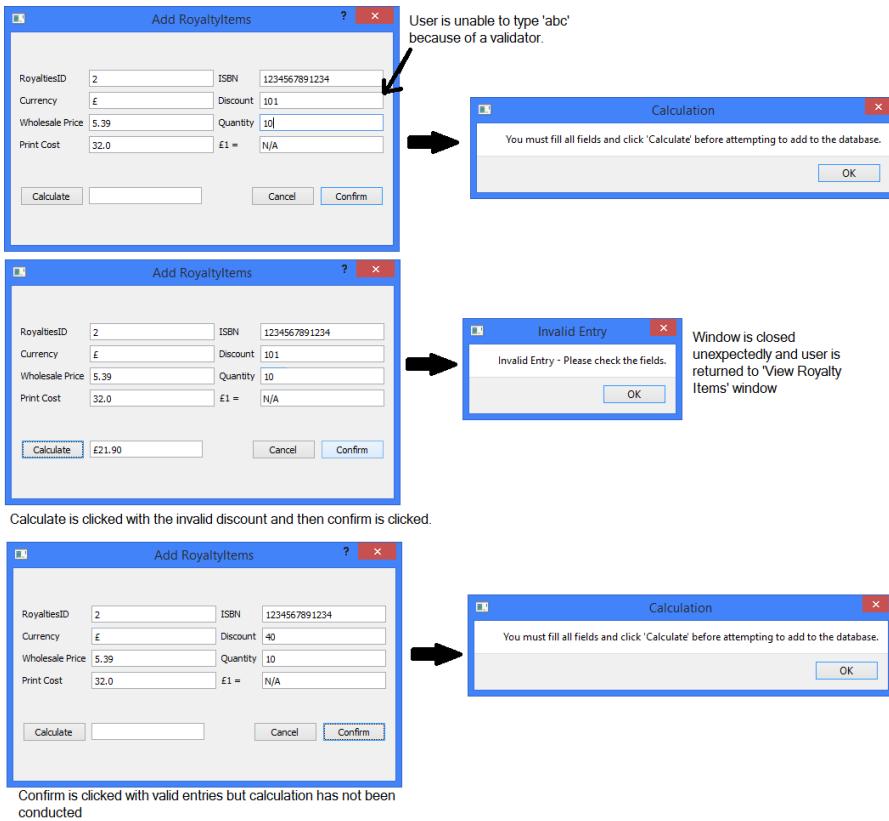


Figure 3.30: Test 2.10 - Validating the Discount when adding Royalty Items

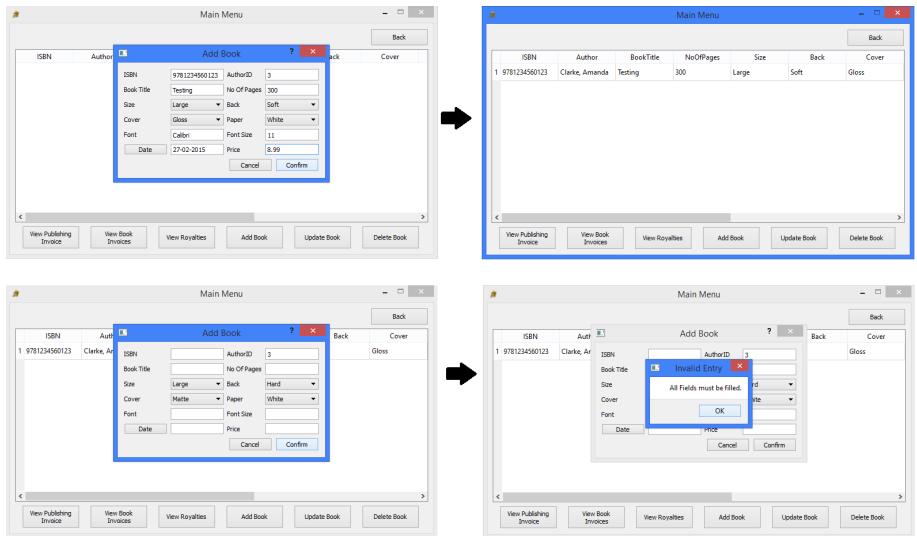


Figure 3.31: Test 3.1 - Adding a book

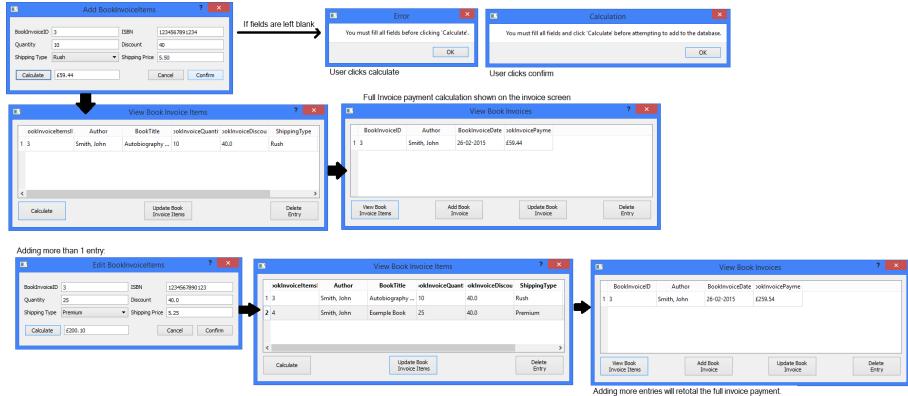


Figure 3.32: Test 3.2 - Calculations for adding Book Invoice items



Figure 3.33: Test 3.3 - Calculations for adding Royalty Items

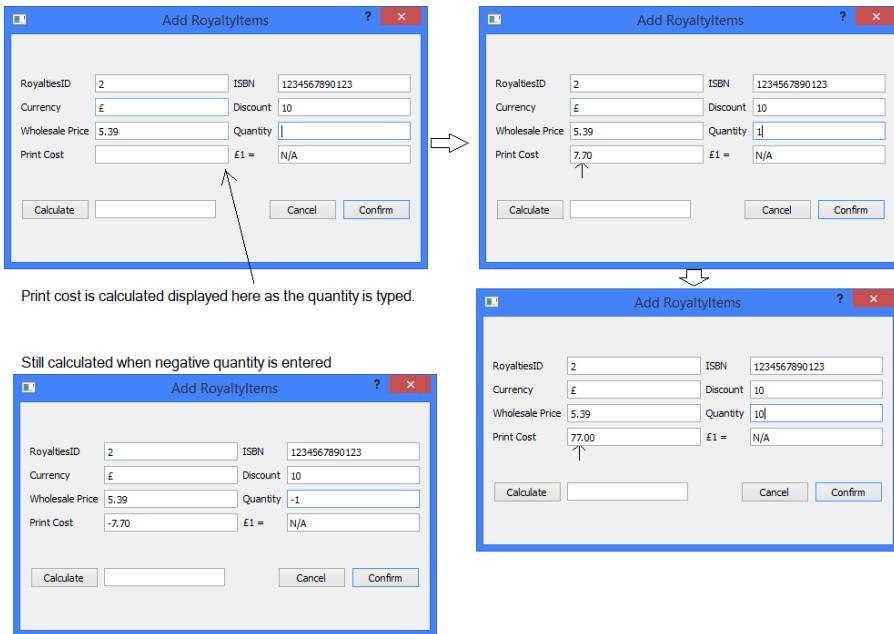
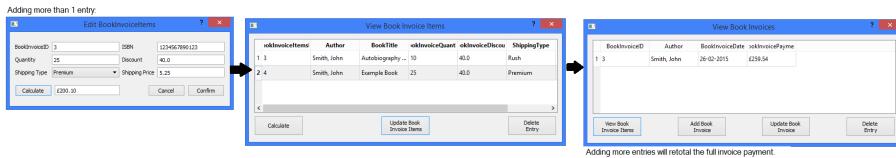


Figure 3.34: Test 3.4 - Testing values for Calculating Print cost



Figure 3.35: Test 3.5 - Calculating the Royalty Payment



177

Figure 3.36: Test 3.6 - Calculating the Book Invoice Payment

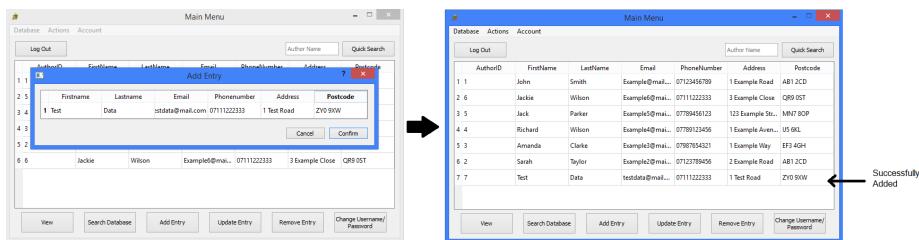


Figure 3.37: Test 4.1 - Testing the process of adding an author

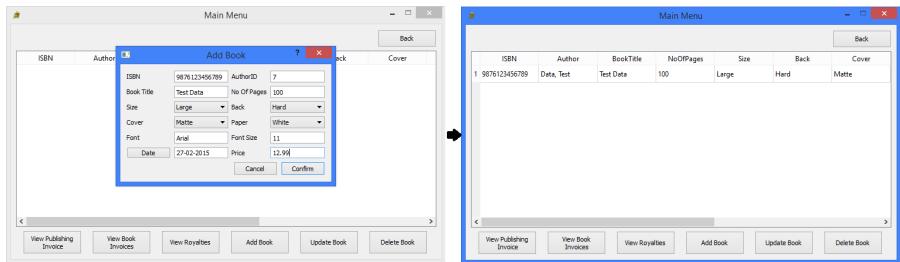


Figure 3.38: Test 4.2 - Testing the process of adding a book

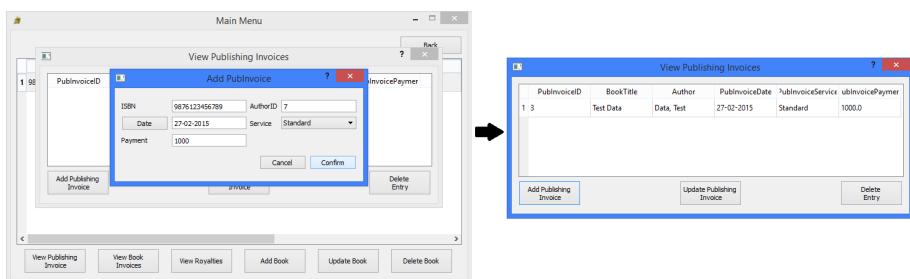


Figure 3.39: Test 4.3 - Testing the process of adding a Publishing Invoice

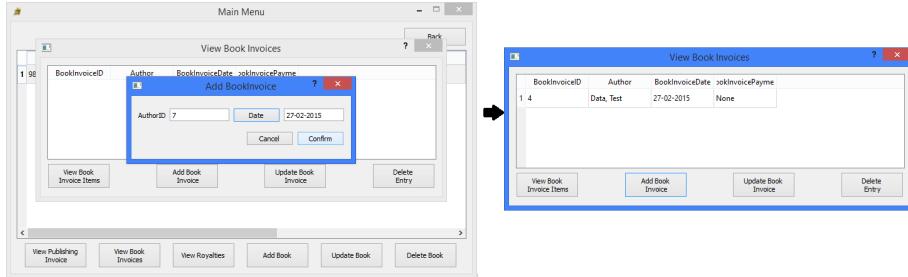


Figure 3.40: Test 4.4 - Testing the process of adding a Book Invoice

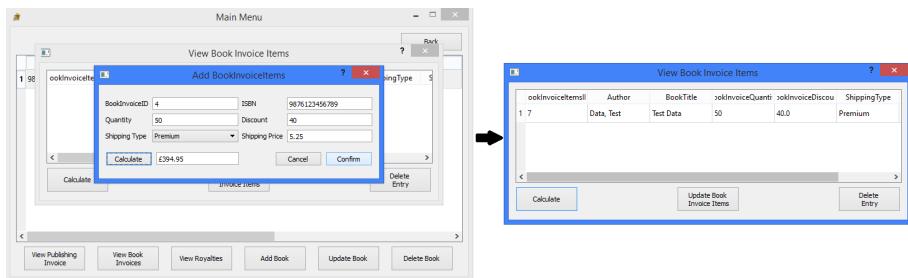


Figure 3.41: Test 4.5 - Testing the process of adding Book Invoice Items

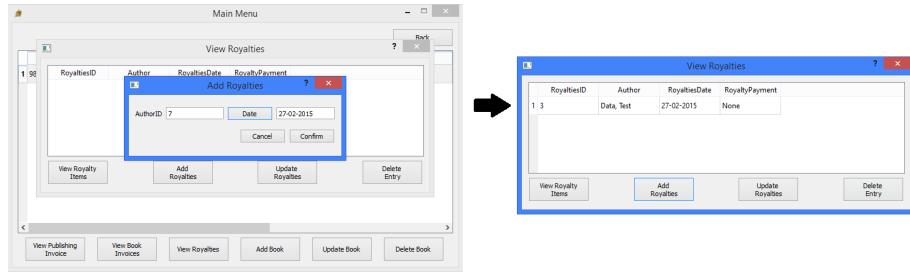


Figure 3.42: Test 4.6 - Testing the process of adding a Royalty Payment

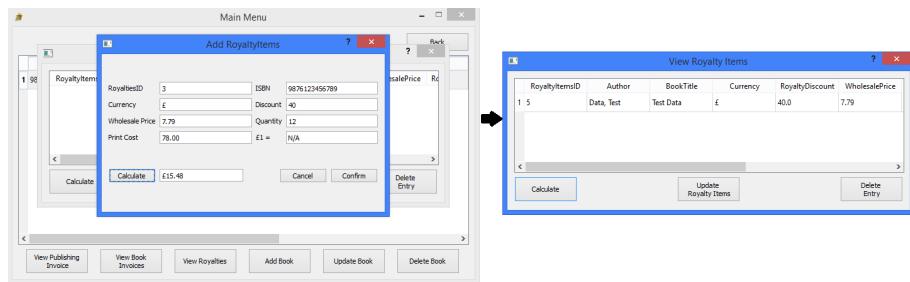


Figure 3.43: Test 4.7 - Testing the process of adding Royalty Items

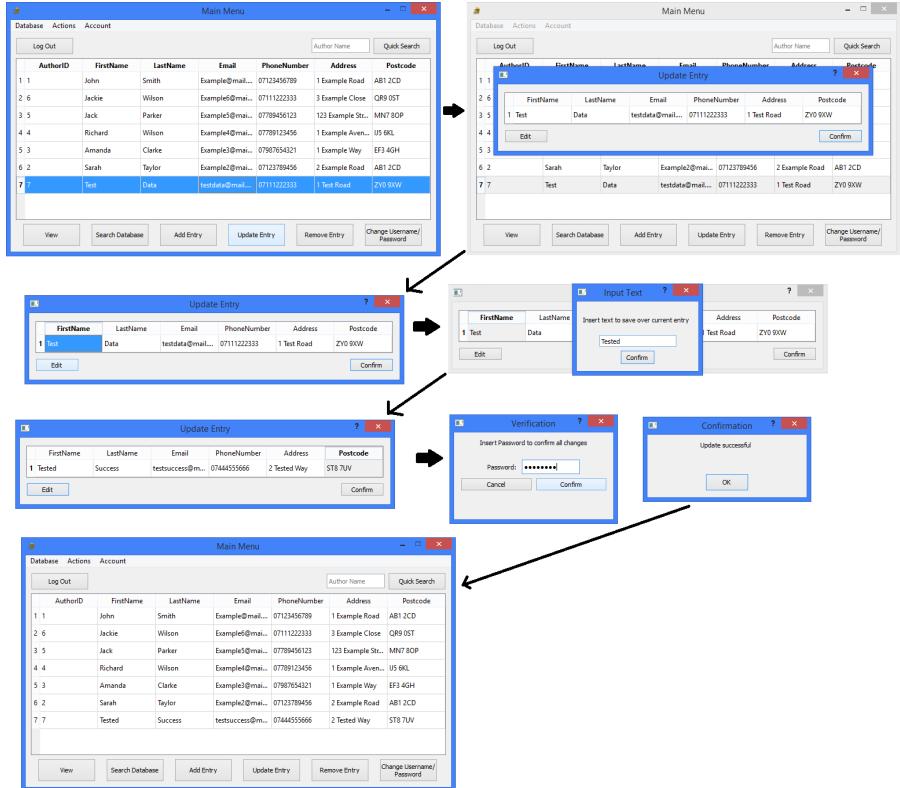


Figure 3.44: Test 4.8 - Testing the process of Updating an Author

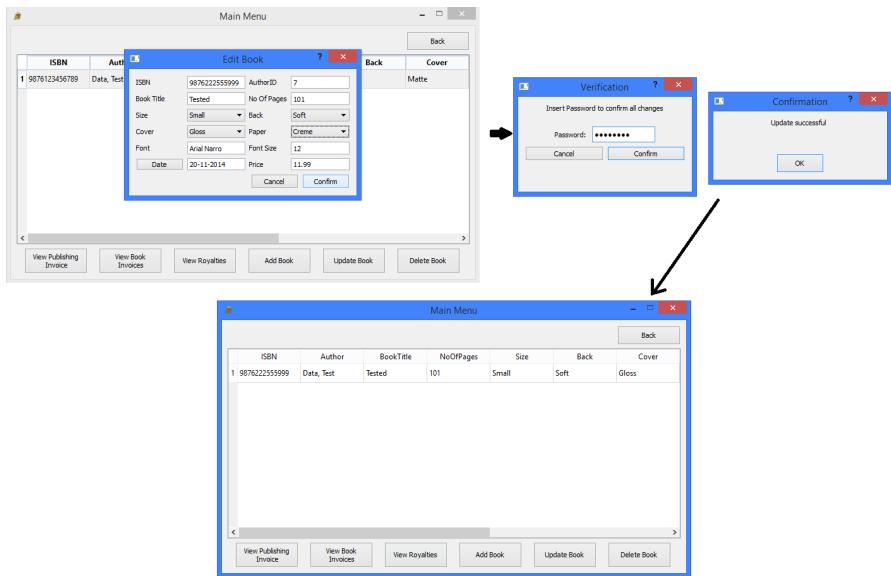


Figure 3.45: Test 4.9 - Testing the process of Updating a Book

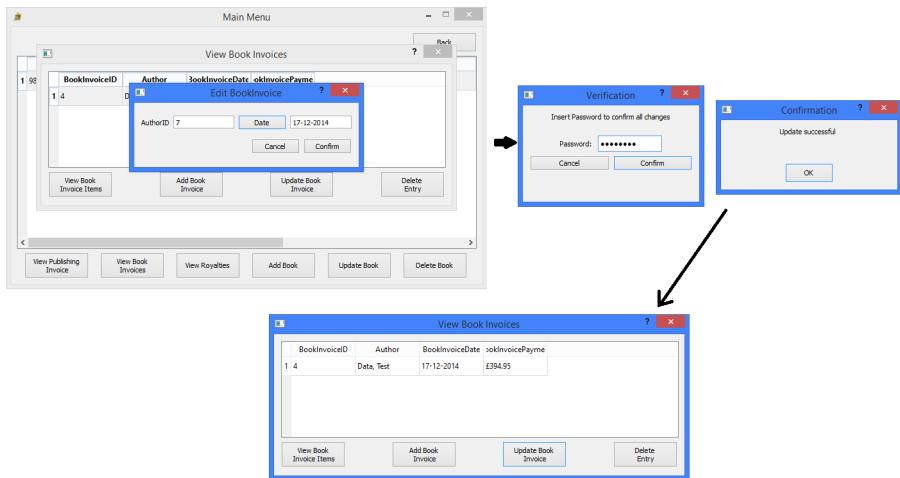
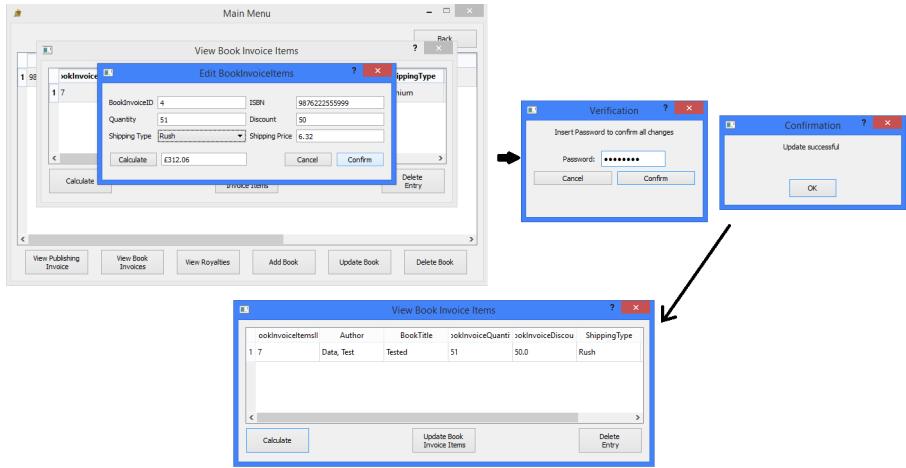


Figure 3.46: Test 4.10 - Testing the process of Updating a Book Invoice



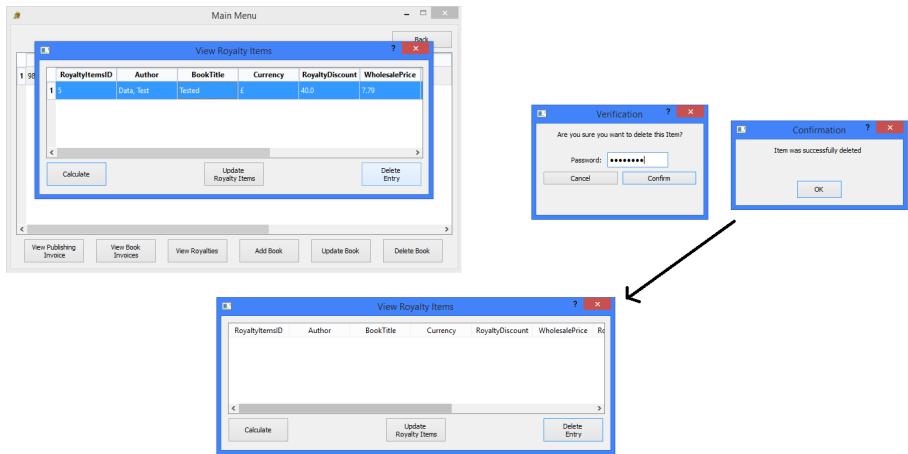


Figure 3.48: Test 4.12 - Testing the process of Deleting Royalty Items

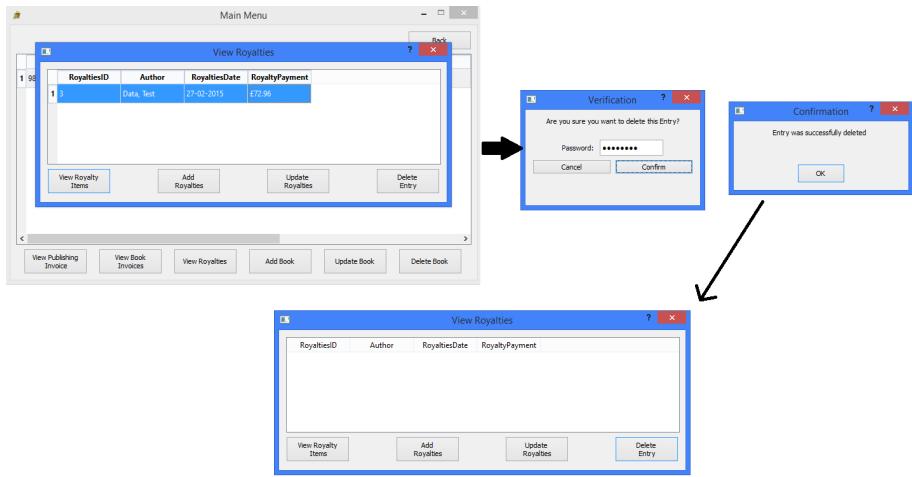


Figure 3.49: Test 4.13 - Testing the process of Deleting a Royalty Payment

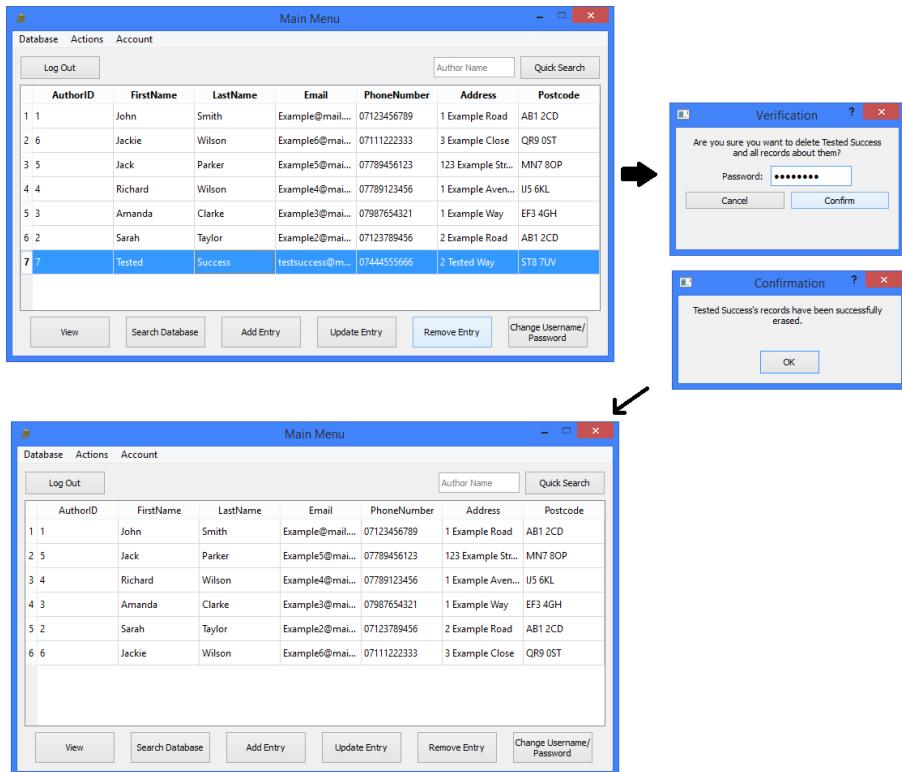


Figure 3.50: Test 4.14 - Testing the process of Deleting an Author and all data linked with it

## 3.4 Evaluation

### 3.4.1 Approach to Testing

I have chosen to use a range of testing strategies. For instance, series 1 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 discount for the Book Invoice Items, and was limited to being able to enter numbers alone. This is seen in the evidence for test 2.9, Figure 3.29 on page 171. Also, dropdown lists were used to prevent the user from incorrectly spelling certain inputs. This can be seen in a number of my examples of evidence, such as Figure 3.38 on page 178 for test 4.2 and Figure 3.47 on page 184 for test 4.11. 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. For example, a number of windows for adding, editing and deleting have a cancel button to close the window. I have only tested two of these, as I can gain the assumption that all the cancel buttons work to the required standard fromm testing to see whether these two work. This can be seen in tests 1.25 and 1.26, where I tested these cancel buttons. 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 they 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. This can be seen in the evidence for test 2.9, Figure 3.29 on page 171 where a calculation is conducted, but the user is prompted against the invalid entry. 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 during the development of my system. Also, the system prevents invalid entries so that the program doesn't crash, as seen in the evidence for test 2.6, Figure 3.26 on page 168, and my test program did not cause my system 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
- cx\_Freeze
- SQLite Inspector

#### **4.1.2 Usage Explanation**

Python 3.4:

- I have been learning this language for over 2 years.
- 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

cx\_Freeze:

- Makes an executable file of my system
- I'm now able to distribute my python scripts where needed

SQLite Inspector:

- Allowed me to browse the database that I was implementing into my system
- I used it to ensure that data had been successfully added/edited/deleted and to the correct places.

#### 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
- Used 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

cx\_Freeze:

- Allowed me to create an executable file for my system
- Creating an executable file meant that the client would not need to install all required packages

SQLite Inspector:

- Allowed execution of SQL statements
- Helped to test my SQL queries before implementing them into my system

## 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. This was used as it wouldn't require reinstatiations of any widgets for the main window.

### 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. I utilised the regular expression module in order to be able to set a validator on line edits used in the dialogs for adding entries, so that when the user enters data, they will be limited to only be able to type in certain characters that are accepted by the regular expression.

#### 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. Similar to Adding Entries, I utilised the regular expression module in order to also be able to set a validator on line edits used in the dialogs for editing entries, so that when the user enters data, they will be limited to only be able to type in certain characters that are accepted by the regular expression.

#### 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 from an 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 via a message box, and will be told what to delete first in order to delete the entry in order to maintain referential integrity.

#### 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. These values are fetched using SQL statements so that all data required for calculations is collected before commencing the calculations. 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 using arithmetic functions in

the system. If a book is updated, any of its attributes which determine the print cost could have been 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 dialog window for filling in and selecting criteria for a search. The user enters the firstname and surname of the author, and then selects from the first combo box which entity to search for and the second combo box which category to search. The user then enters the values where required, and they click confirm which is connected to a function which validates their entries and conduct the search. The results will show in the main window table widget, where the layout index has been changed with the use of a stacked layout. The user can then 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,
14                           PubInvoicePayment from PubInvoice"
15            self.AddWindow.SelectedISBN = self.SelectedISBN
16
17        elif self.CurrentTable == "BookInvoice":
18            self.AddWindow.sql = "select AuthorID, BookInvoiceDate from BookInvoice"
19            self.AddWindow.setFixedSize(350,100)
20
21        elif self.CurrentTable == "Royalties":
22            self.AddWindow.sql = "select AuthorID, RoyaltiesDate from Royalties"
23            self.AddWindow.setFixedSize(350,100)
24
25        elif self.CurrentTable == "BookInvoiceItems":
26            self.AddWindow.sql = "select BookInvoiceID, ISBN, BookInvoiceQuantity, BookInvoiceDiscount,
27                               ShippingType, ShippingPrice from BookInvoiceItems"
28            self.AddWindow.SelectedISBN = self.SelectedISBN
29            self.Editing = False
30            self.AddWindow.btnCalculate.clicked.connect(self.BookInvoiceItemCalculation)
31
32        elif self.CurrentTable == "RoyaltyItems":
33            self.AddWindow.sql = "select RoyaltiesID, ISBN, Currency, RoyaltyDiscount, WholesalePrice,
34                               RoyaltyQuantity, PrintCost, ExcRateFromGBP from RoyaltyItems"
35            self.AddWindow.setFixedSize(450, 250)
36            self.AddWindow.SelectedISBN = self.SelectedISBN
37            self.Editing = False
38            self.AddWindow.btnCalculate.clicked.connect(self.RoyaltyItemCalculation)
39
40        self.AddWindow.selectedID = self.SelectedID
41        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     def RemoveFromDB(self): #removing other entries
2         #getting primary key of the row
3         self.ConfirmDialog = dbConfirmationDialog()
4         self.ConfirmDialog.Username = self.Username
5         self.ConfirmDialog.DeleteMsg = self.CurrentTable
6         self.SelectedAuthorID = self.SelectedID
7
8         if self.CurrentTable == "Book":
9             self.ForeignKeyMsg = "Royalties and Invoices"
10            self.SelectedRow = self.Viewwindow.table.currentRow()
11            self.SelectedID = QTableWidgetItem(self.Viewwindow.table.item(self.SelectedRow, 0)).text()
12            self.SelectedIDName = "ISBN"
13            self.ConfirmDialog.Msg = "Are you sure you want to delete this book?"
14            self.ConfirmDialog.ConfirmedMsg = "Book was successfully deleted"
15
16        elif self.CurrentTable == "PubInvoice":
17            self.SelectedRow = self.PubInvoiceWindow.table.currentRow()
18            self.OldID = self.SelectedID
19            self.SelectedID = QTableWidgetItem(self.PubInvoiceWindow.table.item(self.SelectedRow, 0)).text()
20            self.SelectedIDName = "PubInvoiceID"
21            self.ConfirmDialog.Msg = "Are you sure you want to delete this Invoice?"
22            self.ConfirmDialog.ConfirmedMsg = "Invoice was successfully deleted"
23
24        elif self.CurrentTable == "BookInvoice":
25            self.ForeignKeyMsg = "Book Invoice Items"
26            self.SelectedRow = self.BookInvoiceWindow.table.currentRow()
27            self.OldID = self.SelectedID
28            self.SelectedID = QTableWidgetItem(self.BookInvoiceWindow.table.item(self.SelectedRow,
29                                         0)).text()
30            self.SelectedIDName = "BookInvoiceID"
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 == "Royalties":
35            self.ForeignKeyMsg = "Royalty Items"
36            self.SelectedRow = self.RoyaltiesWindow.table.currentRow()
37            self.OldID = self.SelectedID
38            self.SelectedID = QTableWidgetItem(self.RoyaltiesWindow.table.item(self.SelectedRow, 0)).text()
39            self.SelectedIDName = "RoyaltiesID"
40            self.ConfirmDialog.Msg = "Are you sure you want to delete this Entry?"
41            self.ConfirmDialog.ConfirmedMsg = "Entry was successfully deleted"
42
43        elif self.CurrentTable == "BookInvoiceItems":
44            self.SelectedRow = self.BookInvoiceItemsWindow.table.currentRow()
45            self.OldID = self.SelectedID
46            self.SelectedID = QTableWidgetItem(self.BookInvoiceItemsWindow.table.item(self.SelectedRow,
47                                         0)).text()
48            self.SelectedIDName = "BookInvoiceItemsID"
49            self.ConfirmDialog.Msg = "Are you sure you want to delete this Item?"
50            self.ConfirmDialog.ConfirmedMsg = "Item was successfully deleted"
51
52        elif self.CurrentTable == "RoyaltyItems":
53            self.SelectedRow = self.RoyaltyItemsWindow.table.currentRow()

```

---

```

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

```

This is a function because it needs to be called whenever the user wishes to delete an 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 71 of the function.

#### 4.3.3 Function - Generating an update window

Lines 571-675 of Module 2, subsection 4.10.2. Generating an 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
3             data
4         self.EditWindow.setFixedSize(360, 200)
5         self.EditWindow.AddType = self.CurrentTable
6         self.EditWindow.AnswerButtons()
7         self.EditWindow.ReadyToVerify = False
8         self.EditWindow.originalItemList = []
9
10        if self.CurrentTable == "Book":
11            self.EditWindow.sql = "select * from Book"
12            self.ViewWindow.SelectedRow = self.ViewWindow.table.currentRow()
13            if self.ViewWindow.SelectedRow != -1:
14                self.Selection = True
15                with sqlite3.connect("PP.db") as db:
16                    cursor = db.cursor() #fetching data from database for the edit window
17                    sql = "select * from Book where ISBN =
18                        {}".format(self.ViewWindow.table.item(self.ViewWindow.SelectedRow, 0).text())
19                    cursor.execute(sql)
20                    self.EditWindow.originalItemList = list(cursor.fetchone())
21
21        elif self.CurrentTable == "PubInvoice":
22            self.EditWindow.setFixedSize(400,150)
23            self.EditWindow.selectedISBN = self.SelectedISBN
24            self.EditWindow.sql = "select ISBN, AuthorID, PubInvoiceDate, PubInvoiceService,
25                                 PubInvoicePayment from PubInvoice"
25            self.PubInvoiceWindow.SelectedRow = self.PubInvoiceWindow.table.currentRow()

```

```

26         if self.PubInvoiceWindow.SelectedRow != -1:
27             self.Selection = True
28             with sqlite3.connect("PP.db") as db:
29                 cursor = db.cursor() #fetching data from database for the edit window
30                 sql = "select ISBN, AuthorID, PubInvoiceDate, PubInvoiceService, PubInvoicePayment from
31                     PubInvoice where PubInvoiceID =
32                         {}".format(self.PubInvoiceWindow.table.item(self.PubInvoiceWindow.SelectedRow,
33                                         0).text())
33                 cursor.execute(sql)
34                 self.EditWindow.originalItemList = list(cursor.fetchone())
35
36         elif self.CurrentTable == "BookInvoice":
37             self.EditWindow.setFixedSize(350, 100)
38             self.EditWindow.selectedID = self.SelectedID
39             self.EditWindow.sql = "select AuthorID, BookInvoiceDate from BookInvoice"
40             self.BookInvoiceWindow.SelectedRow = self.BookInvoiceWindow.table.currentRow()
41             if self.BookInvoiceWindow.SelectedRow != -1:
42                 self.Selection = True
43                 with sqlite3.connect("PP.db") as db:
44                     cursor = db.cursor() #fetching data from database for the edit window
45                     sql = "select AuthorID, BookInvoiceDate from BookInvoice where BookInvoiceID =
46                         {}".format(self.BookInvoiceWindow.table.item(self.BookInvoiceWindow.SelectedRow,
47                                         0).text())
47                     cursor.execute(sql)
48                     self.EditWindow.originalItemList = list(cursor.fetchone())
49
50         elif self.CurrentTable == "Royalties":
51             self.EditWindow.setFixedSize(350, 100)
52             self.EditWindow.selectedID = self.SelectedID
53             self.EditWindow.sql = "select AuthorID, RoyaltiesDate from Royalties"
54             self.RoyaltiesWindow.SelectedRow = self.RoyaltiesWindow.table.currentRow()
55             if self.RoyaltiesWindow.SelectedRow != -1:
56                 self.Selection = True
57                 with sqlite3.connect("PP.db") as db:
58                     cursor = db.cursor() #fetching data from database for the edit window
59                     sql = "select AuthorID, RoyaltiesDate from Royalties where RoyaltiesID =
60                         {}".format(self.RoyaltiesWindow.table.item(self.RoyaltiesWindow.SelectedRow,
61                                         0).text())
61                     cursor.execute(sql)
62                     self.EditWindow.originalItemList = list(cursor.fetchone())
63
64         elif self.CurrentTable == "BookInvoiceItems":
65             self.EditWindow.setFixedSize(450, 150)
66             self.EditWindow.selectedID = self.SelectedID
67             self.Editing = True
68             self.EditWindow.btnCalculate.clicked.connect(self.BookInvoiceItemCalculation)
69             self.EditWindow.selectedISBN = self.SelectedISBN
70             self.EditWindow.sql = "select BookInvoiceID, ISBN, BookInvoiceQuantity, BookInvoiceDiscount,
71                 ShippingType, ShippingPrice from BookInvoiceItems"
72             self.BookInvoiceItemsWindow.SelectedRow = self.BookInvoiceItemsWindow.table.currentRow()
73             if self.BookInvoiceItemsWindow.SelectedRow != -1:
74                 self.Selection = True
75                 with sqlite3.connect("PP.db") as db:
76                     cursor = db.cursor() #fetching data from database for the edit window
77                     sql = "select BookInvoiceID, ISBN, BookInvoiceQuantity, BookInvoiceDiscount,
78                         ShippingType, ShippingPrice from BookInvoiceItems where BookInvoiceItemsID =
79                         {}".format(self.BookInvoiceItemsWindow.table.item(self.BookInvoiceItemsWindow.SelectedRow,
80                                         0).text())
80                     cursor.execute(sql)
81                     self.EditWindow.originalItemList = list(cursor.fetchone())
82
83         elif self.CurrentTable == "RoyaltyItems":
84             self.EditWindow.setFixedSize(450, 250)
85             self.EditWindow.selectedID = self.SelectedID
86             self.Editing = True
87             self.EditWindow.btnCalculate.clicked.connect(self.RoyaltyItemCalculation)
88             self.EditWindow.selectedISBN = self.SelectedISBN
89             self.EditWindow.sql = "select RoyaltiesID, ISBN, Currency, RoyaltyDiscount, WholesalePrice,
90                 RoyaltyQuantity, PrintCost, ExRateFromGBP from RoyaltyItems"
91             self.RoyaltyItemsWindow.SelectedRow = self.RoyaltyItemsWindow.table.currentRow()
92             if self.RoyaltyItemsWindow.SelectedRow != -1:
93                 self.Selection = True
94                 with sqlite3.connect("PP.db") as db:
95                     cursor = db.cursor() #fetching data from database for the edit window
96                     sql = "select RoyaltiesID, ISBN, Currency, RoyaltyDiscount, WholesalePrice,
97                         RoyaltyQuantity, PrintCost, ExRateFromGBP from RoyaltyItems where
98                         RoyaltyItemsID =
99                         {}".format(self.RoyaltyItemsWindow.table.item(self.RoyaltyItemsWindow.SelectedRow,
100                                         0).text())
100                cursor.execute(sql)
101                self.EditWindow.originalItemList = list(cursor.fetchone())
102
103        self.EditWindow.Editing = True
104
105        if self.Selection == True: #initialising the edit window
106            self.EditWindow.btnConfirm.clicked.connect(self.EditWindow.Validate)
107            if self.CurrentTable in ["RoyaltyItems", "BookInvoiceItems"]:
108                self.EditWindow.btnConfirm.clicked.connect(self.EditWindow.CheckCalculated)
109            else:
110                self.EditWindow.btnConfirm.clicked.connect(self.VerifyUpdate)
111            self.EditWindow.AddType = self.CurrentTable

```

---

```

101         self.EditWindow.selectedID = self.SelectedID
102
103         self.EditWindow.CalendarWidget = dbCalendarWidget()
104         self.EditWindow.CalendarWidget.Calendar()
105         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.

#### 4.3.4 Function - Updating an Item

Lines 697-787 of Module 2, subsection 4.10.2. Updating a selected item with the entries received.

---

```

1     def UpdateChanges(self): #committing changes
2         self.UpdateList = []
3
4         with sqlite3.connect("PP.db") as db:
5             cursor = db.cursor()
6             if self.CurrentTable == "Book":
7                 self.NoOfEntries = 12
8                 cursor.execute("select * from Book")
9                 self.ID = "ISBN"
10                self.SelectedISBN =
11                    QTableWidgetItem(self.ViewWindow.table.item(self.ViewWindow.SelectedRow, 0).text())
12                self.SelectedID = self.SelectedISBN
13                self.BooKEdited = True
14
15            elif self.CurrentTable == "PubInvoice":
16                cursor.execute("select ISBN, AuthorID, PubInvoiceDate, PubInvoiceService, PubInvoicePayment
17                                from PubInvoice")
18                self.NoOfEntries = 5
19                self.ID = "PubInvoiceID"
20                self.OldID = self.SelectedID
21                self.SelectedID =
22                    QTableWidgetItem(self.PubInvoiceWindow.table.item(self.PubInvoiceWindow.SelectedRow,
23                                         0).text())
24
25            elif self.CurrentTable == "BookInvoice":
26                cursor.execute("select AuthorID, BookInvoiceDate from BookInvoice")
27                self.NoOfEntries = 2
28                self.ID = "BookInvoiceID"
29                self.OldID = self.SelectedID
30                self.SelectedID =
31                    QTableWidgetItem(self.BookInvoiceWindow.table.item(self.BookInvoiceWindow.SelectedRow,
32                                         0).text())
33
34            elif self.CurrentTable == "Royalties":
35                cursor.execute("select AuthorID, RoyaltiesDate from Royalties")
36                self.NoOfEntries = 2
37                self.ID = "RoyaltiesID"
38                self.OldID = self.SelectedID
39                self.SelectedID =
40                    QTableWidgetItem(self.RoyaltiesWindow.table.item(self.RoyaltiesWindow.SelectedRow,
41                                         0).text())
42
43            elif self.CurrentTable == "BookInvoiceItems":
44                cursor.execute("select BookInvoiceID, ISBN, BookInvoiceQuantity, BookInvoiceDiscount,
45                                ShippingType, ShippingPrice from BookInvoiceItems")
46                self.NoOfEntries = 6
47                self.ID = "BookInvoiceItemsID"
48                self.OldID = self.SelectedID
49                self.SelectedID =
50                    QTableWidgetItem(self.BookInvoiceItemsWindow.table.item(self.BookInvoiceItemsWindow.SelectedRow,
51                                         0).text())
52
53            elif self.CurrentTable == "RoyaltyItems":
54                cursor.execute("select RoyaltiesID, ISBN, Currency, RoyaltyDiscount, WholesalePrice,
55                                RoyaltyQuantity, PrintCost, ExcRateFromGBP from RoyaltyItems")
56                self.NoOfEntries = 6
57                self.ID = "RoyaltyItemsID"

```

---

---

```

46         self.OldID = self.SelectedID
47         self.SelectedID =
48             QTableWidgetItem(self.RoyaltyItemsWindow.table.item(self.RoyaltyItemsWindow.SelectedRow,
49             0).text())
50
51     for count in range(0, self.NoOfEntries): #creating the update string for sql
52     try:
53         self.UpdateList.append(str(self.EditWindow.inputList[count].currentText()))
54     except:
55         if count == 8 and self.CurrentTable == "RoyaltyItems":
56             self.UpdateList.append(str(self.NetSales))
57         else:
58             self.UpdateList.append(self.EditWindow.inputList[count].text())
59
60     self.Update = ""
61
62     self.Headers = list(cursor.description)
63     for count in range(0, len(self.UpdateList)):
64         self.Update += "{} = '{}'".format(list(self.Headers[count])[0], self.UpdateList[count])
65         if count != len(self.UpdateList) - 1:
66             self.Update += ","
67
68     with sqlite3.connect("PP.db") as db: #update
69         cursor = db.cursor()
70         if self.CurrentTable == "Book": #ISBN is primary key which may be changed, so all foreign keys
71             #will change first for it
72         try:
73             sql = "update PubInvoice set ISBN = {0} where ISBN = {1}".format(self.UpdateList[0],
74             self.SelectedID)
75             cursor.execute(sql)
76             sql = "update BookInvoiceItems set ISBN = {0} where ISBN =
77                 {1}".format(self.UpdateList[0], self.SelectedID)
78             cursor.execute(sql)
79             sql = "update RoyaltyItems set ISBN = {0} where ISBN = {1}".format(self.UpdateList[0],
80             self.SelectedID)
81             cursor.execute(sql)
82         except:
83             pass # error if no entry is there for pubinvoice/bookinvoiceitems/royaltyitems
84         sql = "update {} set {} where {} = {}".format(self.CurrentTable, self.Update, self.ID,
85             self.SelectedID)
86         cursor.execute(sql)
87         db.commit()
88
89     if self.CurrentTable in ["Royalties", "BookInvoice", "PubInvoice", "BookInvoiceItems",
90         "RoyaltyItems"]:
91         self.SelectedID = self.OldID
92
93     self.RefreshTables()
94
95     try:
96         self.RecalculateItems()
97         self.BookEdited = False
98     except:
99         pass

```

---

This Function is called whenever the user has entered the correct password to confirm that they wish to commit the given updates. The ID of the item used is fetched so that it can be referenced when committing the update. Lines 49-56 of the function accumulate a list, consisting of the validated updates that the user wishes to commit.

#### 4.3.5 Function - Creating a Table Widget

Lines 13-32 of Module 5, subsection 4.10.5. Creating a table widget and fetching data to display in it.

---

```

1     def initTable(self):
2         self.clear()
3         self.columns = []
4         self.setSelectionBehavior(QAbstractItemView.SelectRows)
5         self.setEditTriggers(QAbstractItemView.NoEditTriggers)
6         self.setSortingEnabled(False)
7         with sqlite3.connect("PP.db") as db: #fetching data from db
8             cursor = db.cursor()
9             cursor.execute(self.sql)
10            self.ColumnNames = cursor.description
11            for count in range(0, len(self.ColumnNames)):
12                self.columns.append(list(list(self.ColumnNames)[count])[0])

```

---

---

```

13     self.setRowCount(0)
14     self.setColumnCount(len(self.columns))
15     self.setHorizontalHeaderLabels(self.columns)
16     for self.row, self.form in enumerate(cursor):
17         self.insertRow(self.row)
18         for self.column, self.unit in enumerate(self.form):
19             self.setItem(self.row, self.column, QTableWidgetItem(str(self.unit)))
20     self.setSortingEnabled(True)

```

---

This has been made as a function so that it can be called whenever I need to generate a new table in my system. Using a given SQL statement which has been passed into this function, the table finds the column names in lines 9-10 of the function. The SQL statement is passed through this way so that this table can be used for all select statements which are fetching data from the database. The horizontal header labels of the table widget are then set in line 15 of the function. After this, the results are placed into the correct cells in the table in lines 16-19 of the function. This has also been made in a seperate module so that they can be instantiated when it is needed for another layout.

#### 4.3.6 Function - Quick Search

Lines 915-930 of Module 2, subsection 4.10.2. Conducting a customer search using user inputs.

---

```

1 def QuickSearch(self): #quick search from main menu
2     self.QSText = self.MainMenuButtons.lcQuickSearch.text()
3     self.QSText = self.QSText.split(' ')
4     with sqlite3.connect("PP.db") as db:
5         cursor = db.cursor()
6
7     if len(self.QSText) == 1:
8         self.TableWidget.sql = "select * from Customer where Firstname like '{0}%' or Lastname like
9         '{0}%'.format(self.QSText[0])
10    elif len(self.QSText) == 0:
11        self.TableWidget.sql = "select * from Customer"
12    else:
13        for count in range(1, len(self.QSText)):
14            if count != 1:
15                self.QSText[1] += " {}".format(self.QSText[count])
16        self.TableWidget.sql = "select * from Customer where Firstname like '{0}%' and Lastname
like '{1}%'.format(self.QSText[0], self.QSText[1])
17
18    self.TableWidget.initTable()

```

---

This function takes the user input and splits the string wherever a space is found. This is because if the user has entered a firstname and a lastname, the system will be able to identify this through splitting the string where a space is found, so that once the splitted values are put into a list, the system can check how long the list is to determine whether a firstname and a lastname were entered, or just a firstname was entered. Lines 7-11 of the function show a set of selection statements that are used to check the length of the newly created list to determine this. The iteration on line 12 make the first splitted section of the string into a firstname, and the rest of the string is combined into a lastname. After the string has been determined as a firstname.lastname, or both, an SQL statement is created and passed through the table widget function so that a table is initiated in the main window with the results of the search.

### 4.3.7 Modules - View Window and Search Results Window.

Modules 8, subsection 4.10.8. Creating the layout for displaying books of a customer.

---

```

1  from PyQt4.QtCore import *
2  from PyQt4.QtGui import *
3  import sqlite3
4  import sys
5
6  class dbViewWindow(QWidget):
7      """generic view window"""
8
9      def __init__(self):
10         super().__init__()
11
12     def View(self):
13         self.setWindowTitle("View Menu")
14         self.setFixedSize(735,400)
15
16         self.btnExit = QPushButton("Back", self)
17         self.btnExit.setFixedSize(100, 30)
18         self.btnViewRoyalties = QPushButton("View Royalties", self)
19         self.btnViewRoyalties.setFixedSize(100, 40)
20         self.btnViewBookInvoices = QPushButton("View Book \n Invoices", self)
21         self.btnViewBookInvoices.setFixedSize(100, 40)
22         self.btnViewPubInvoice = QPushButton("View Publishing \n Invoice", self)
23         self.btnViewPubInvoice.setFixedSize(100, 40)
24         self.btnAddBook = QPushButton("Add Book", self)
25         self.btnAddBook.setFixedSize(100, 40)
26         self.btnUpdateBook = QPushButton("Update Book", self)
27         self.btnUpdateBook.setFixedSize(100,40)
28         self.btnDeleteBook = QPushButton("Delete Book", self)
29         self.btnDeleteBook.setFixedSize(100, 40)
30
31         self.horizontalTop = QHBoxLayout()
32         self.horizontalTop.addStretch(1)
33         self.horizontalTop.addWidget(self.btnExit)
34
35         self.horizontalBottom = QHBoxLayout()
36         self.horizontalBottom.addWidget(self.btnViewPubInvoice)
37         self.horizontalBottom.addWidget(self.btnViewBookInvoices)
38         self.horizontalBottom.addWidget(self.btnViewRoyalties)
39         self.horizontalBottom.addWidget(self.btnAddBook)
40         self.horizontalBottom.addWidget(self.btnUpdateBook)
41         self.horizontalBottom.addWidget(self.btnDeleteBook)
42
43         self.vertical = QVBoxLayout()

```

---

Module 15, subsection 4.10.15. Creating the layout for displaying search results.

---

```

1  from PyQt4.QtCore import *
2  from PyQt4.QtGui import *
3  import sys
4
5
6  class initSearchResultsMenu(QWidget):
7      """main window"""
8
9      def __init__(self):
10         super().__init__()
11         self.btnExit = QPushButton("Back", self)
12         self.btnExit.setFixedSize(100, 30)
13         self.horizontalTop = QHBoxLayout()
14         self.horizontalTop.addStretch(1)
15         self.horizontalTop.addWidget(self.btnExit)
16
17         self.vertical = QVBoxLayout()
18         self.vertical.addLayout(self.horizontalTop)
19         self.setLayout(self.vertical)

```

---

These modules are designed to be instantiated by the main program, and then added to a stacked layout. These are made as separate modules to the main program so that the layouts can be clearly created. These are also placed in a stacked layout so that they can be indexed when they are needed, as opposed to having to reinstantiating them when necessary.

#### **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 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
self.Quantity	Used to hold the value for the book invoice/royalty quantity	Module 2: 479, 484, 495, 519, 524, 527 Module 12: Line 100, 105, 165, 184
self.Discount	Used to hold the value for the book invoice discount	Module 2: Line 480, 485, 496, 497 Module 12: Line 101, 106, 107
self.ShippingPrice	Used to hold the value for the shipping price	Module 2: Line 475, 486, 498 Module 12: Line 102, 108

self.Price	Used to hold the value for the book price	Module 2: Line 492, 495 Module 12: Line 103, 105
self.TempPayment	Used to hold the value for the payment which would accumulate during calculations	Module 12: Line 105, 106, 107, 108, 110, 186, 189, 190
self.BookInvoice-Payment	Used to hold the value for the full book invoice payment	Module 12: Line 68, 110, 114, 115, 120, 121, 193
self.RoyaltyPayment	Used to hold the value for the full royalty payment	Module 12: Line 130, 190, 194, 199, 200
self.Currency	Used to hold the value for the currency	Module 2: Line 517, 522, 532, 538 Module 12: Line 131, 163, 187
self.WholesalePrice	Used to hold the value for the wholesale price	Module 2: Line 518, 523, 527
self.NetSales	Used to hold the value for the net sales figure	Module 2: Line 204, 527, 528, 533, 539, 750 Module 12: Line 164, 186
self.RoyaltyItem-Payment	Used to hold the calculated payment value for the individual item payment	Module 2: Line 528, 529, 532, 538
self.Size	Used to hold the value for the size of the book, so that a price could be calculated dependent on the cost of the size.	Module 9: Line 129, 132, 139 Module 12: Line 167, 171, 177
self.Back	Used to hold the value for the back type of the book in order to determine a price using the cost of the back	Module 9: 130, 134, 136, 141, 143 Module 12: 169, 173, 175, 179, 181
self.CoverPrice	Used to hold the value for the cover price, used to calculate the print cost	Module 9: Line 135, 137, 142, 144, 341 Module 12: Line 174, 176, 180, 182, 184

self.PagePrice	Used to hold the value for the page price, used to calculate the print cost	Module 9: Line 133, 140, 341, Module 12: Line 174, 176, 180, 182, 184
self.NoOfPages	Used to hold the value for the number of pages in a book	Module 9: Line 128, 133, 140 Module 12: Line 166, 172, 178
self.PrintCost	Used to hold the value for the calculated print cost	Module 2: 521, 526, 529, Module 9: Line 341, 342 Module 12: Line 184, 186

## 4.5 System Evidence

### 4.5.1 User Interface

Figure 4.1: Login Screen

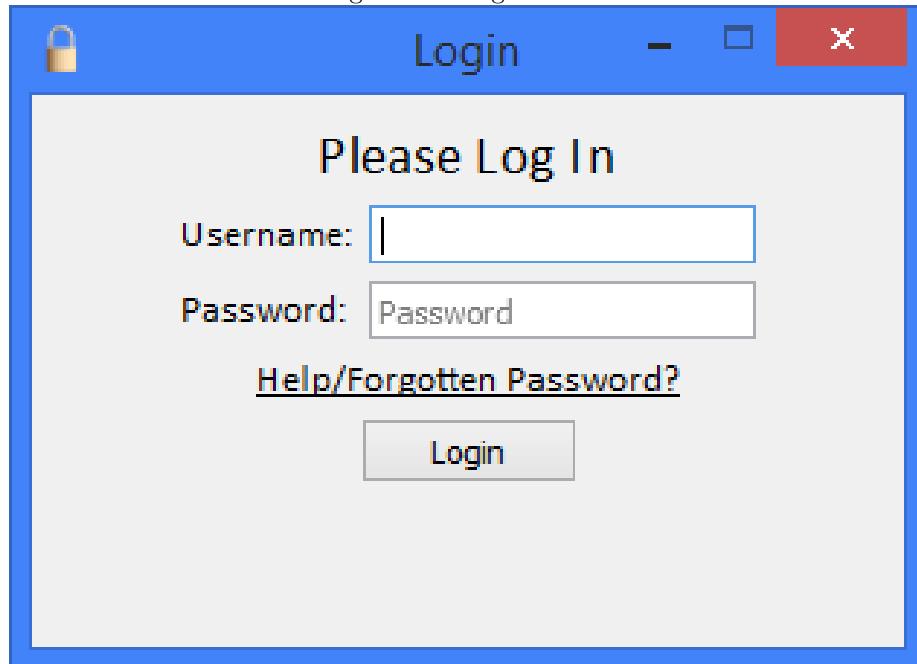


Figure 4.2: Forgot Password Screen - User Enters email to retrieve their password via email, if the email is correct.

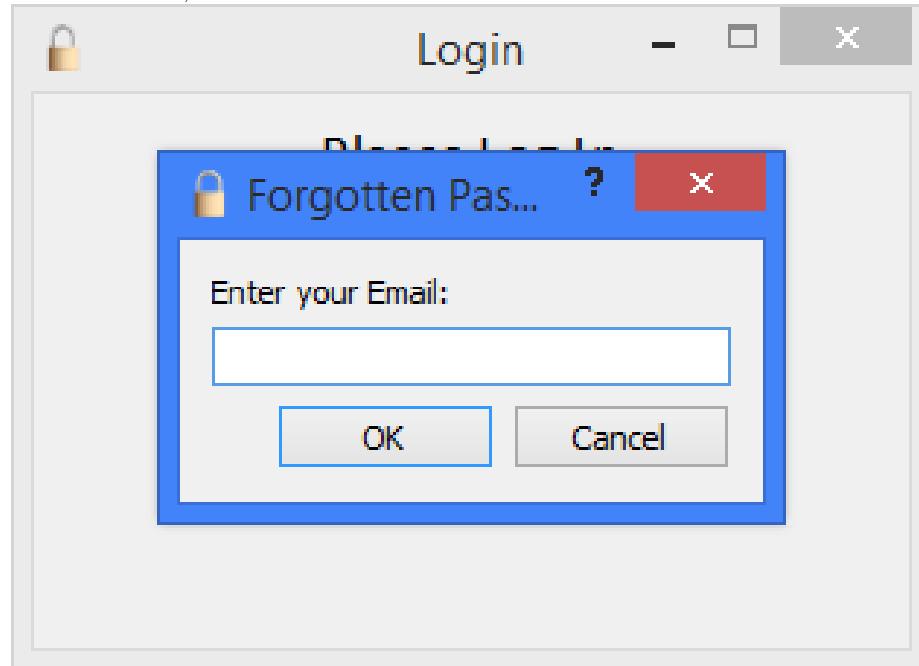


Figure 4.3: Main Menu

A screenshot of a Windows application window titled "Main Menu". The menu bar includes "Database", "Actions", and "Account". Below the menu is a toolbar with "Log Out" and search fields for "Author Name" and "Quick Search". The main area is a grid table with columns: AuthorID, FirstName, LastName, Email, PhoneNumber, Address, and Postcode. The table contains 7 rows of sample data. At the bottom are buttons for "View", "Search Database", "Add Entry", "Update Entry", "Remove Entry", and "Change Username/Password".

AuthorID	FirstName	LastName	Email	PhoneNumber	Address	Postcode
1 1	John	Smith	Example@mail...	07123456789	1 Example Road	AB1 2CD
2 6	Jackie	Wilson	Example6@mail...	07111222333	3 Example Close	QR9 0ST
3 5	Jack	Parker	Example5@mail...	07789456123	123 Example Str...	MN7 8OP
4 4	Richard	Wilson	Example4@mail...	07789123456	1 Example Aven...	IJ5 6KL
5 3	Amanda	Clarke	Example3@mail...	07987654321	1 Example Way	EF3 4GH
6 2	Sarah	Taylor	Example2@mail...	07123789456	2 Example Road	AB1 2CD
7 7	Test	Data	testdata@mail...	07111222333	1 Test Road	ZY0 9XW

Figure 4.4: Add Customer Entry

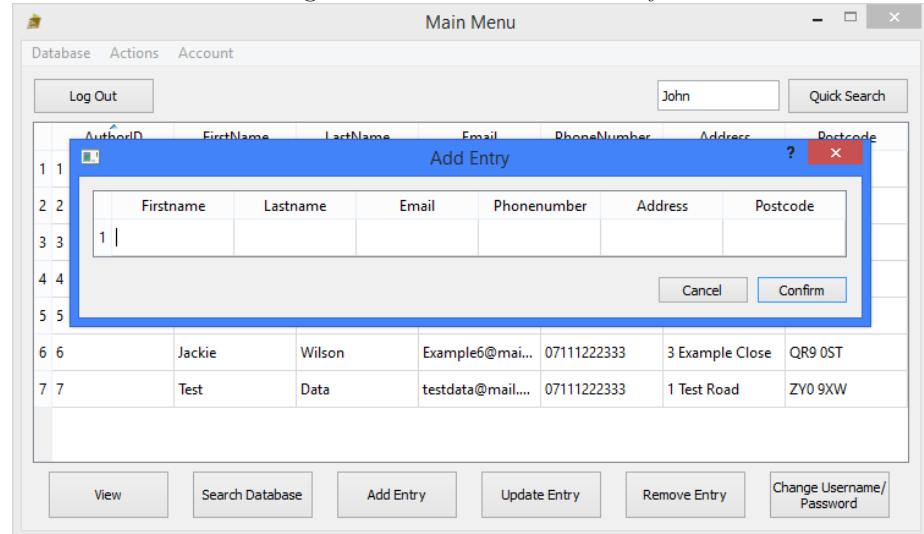


Figure 4.5: Update Customer Entry

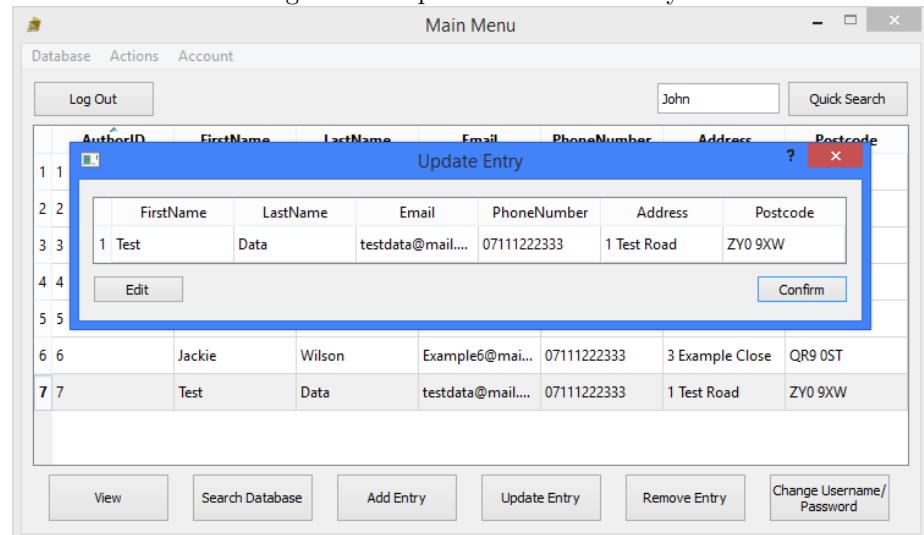


Figure 4.6: Editing Customer Field

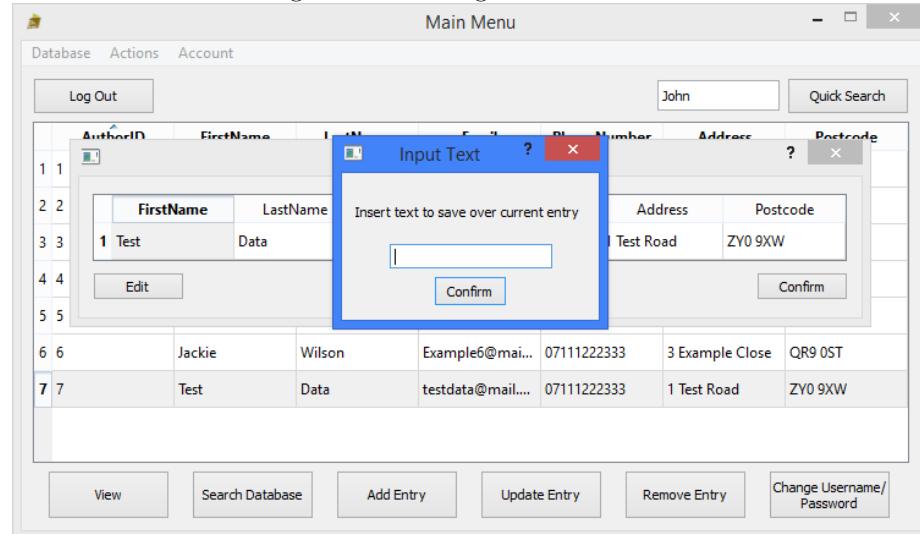


Figure 4.7: Update Customer Entry Verification

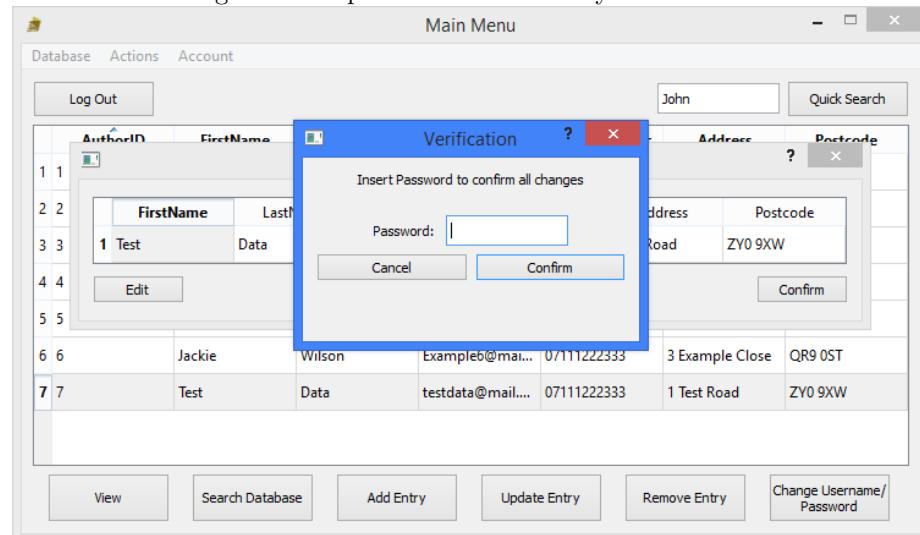


Figure 4.8: Update Customer Entry Success - User is given confirmation that their action has been successfully conducted

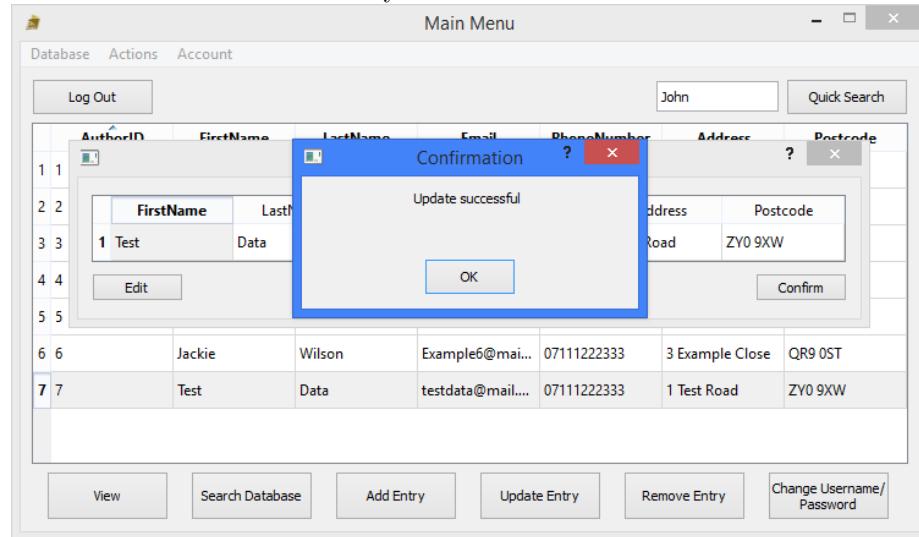


Figure 4.9: Delete Customer Entry - Verification, which warns the user of deleting all the data linked with the selected customer

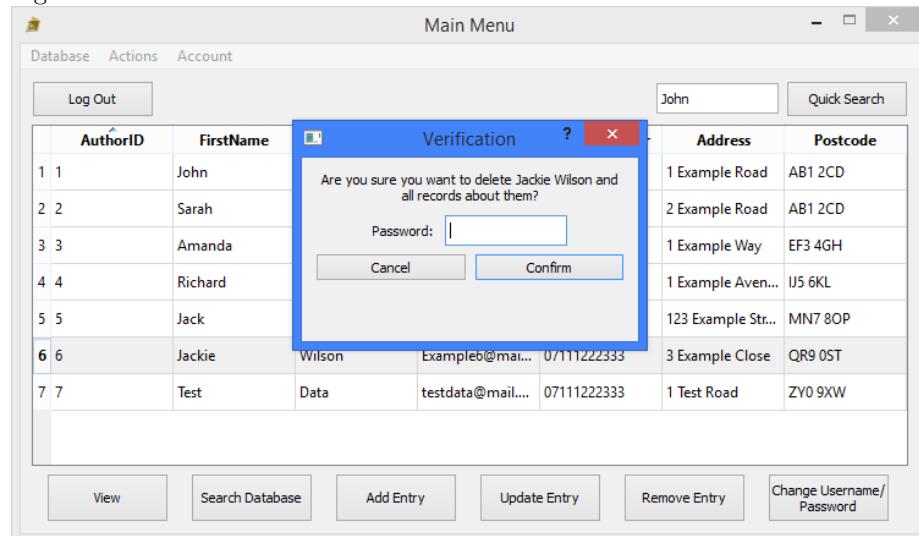


Figure 4.10: Delete Entry Success - User is given confirmation that their action has been successfully conducted

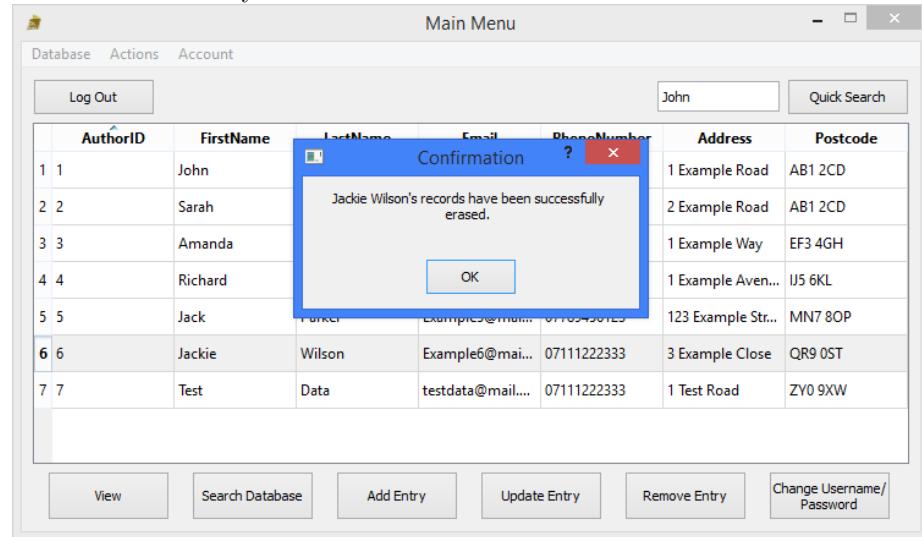


Figure 4.11: View Menu

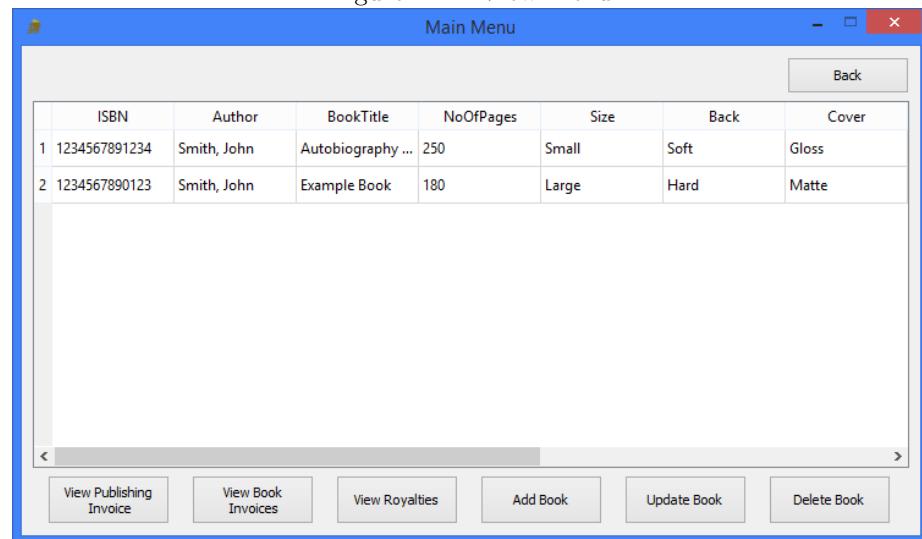


Figure 4.12: View Publishing Invoices

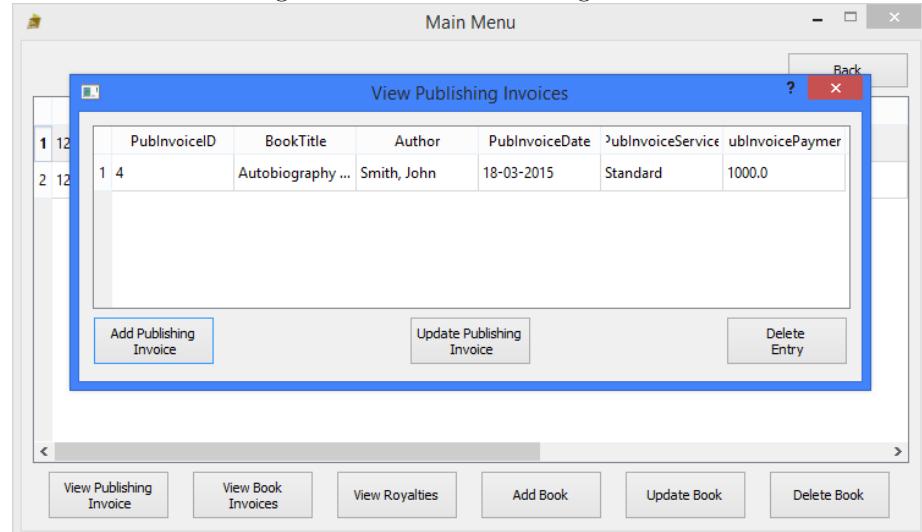


Figure 4.13: Add Publishing Invoice

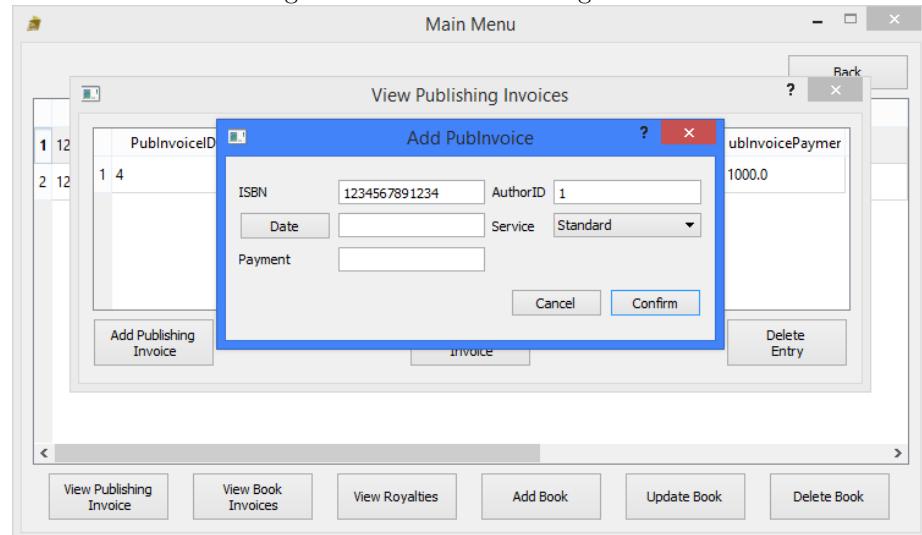


Figure 4.14: Update Publishing Invoice

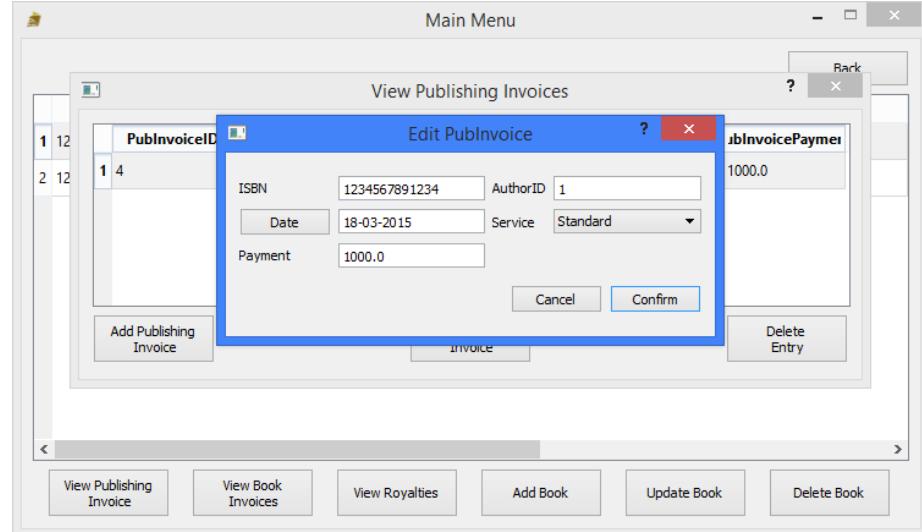


Figure 4.15: Update Verification

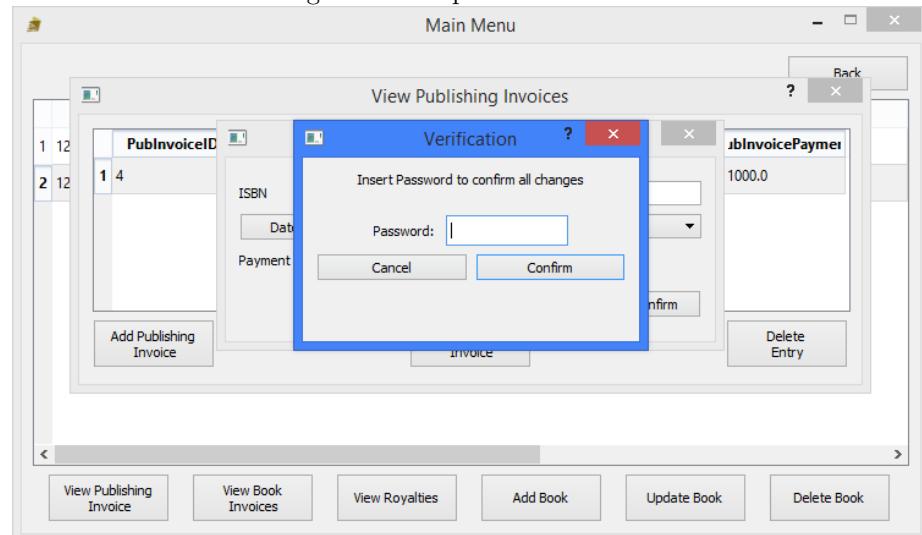


Figure 4.16: Update Success

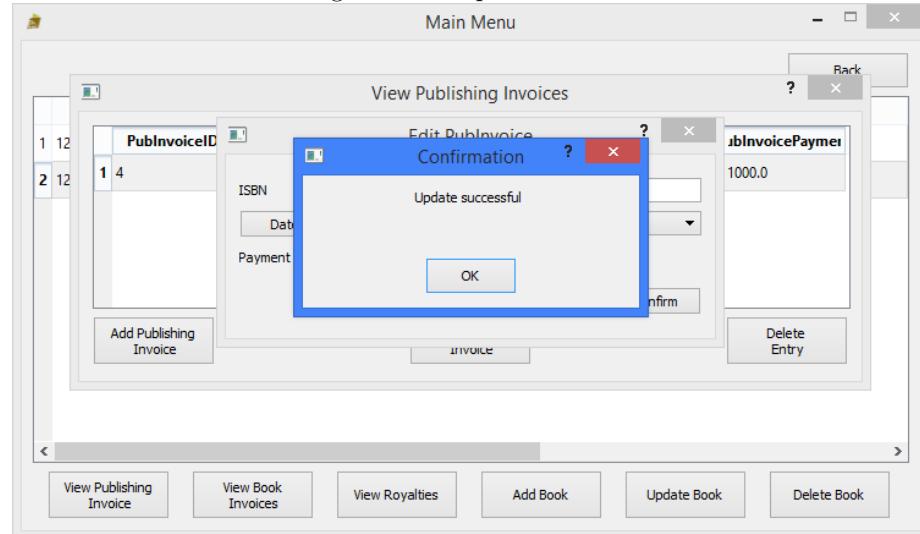


Figure 4.17: Delete Verification - Verification which grants the user access to delete the entry

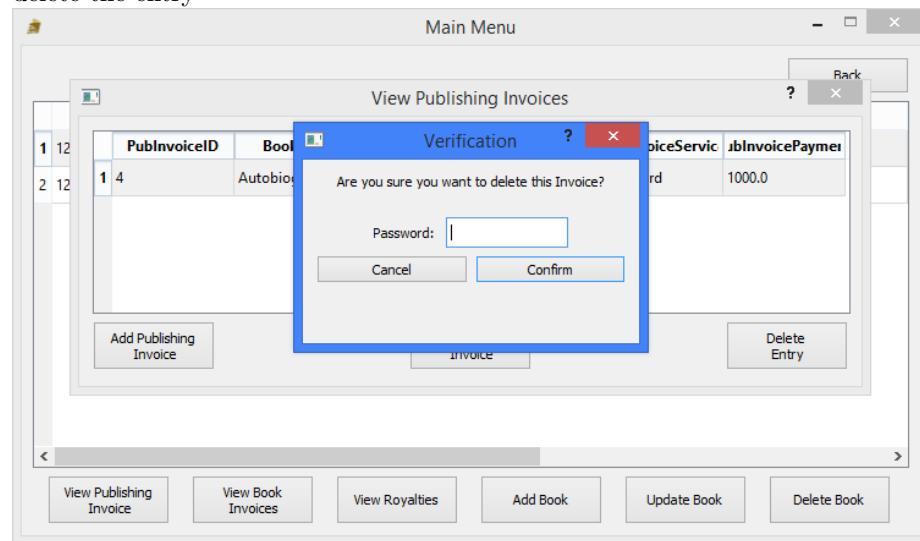


Figure 4.18: Delete Success - User is given confirmation that their action has been successfully conducted

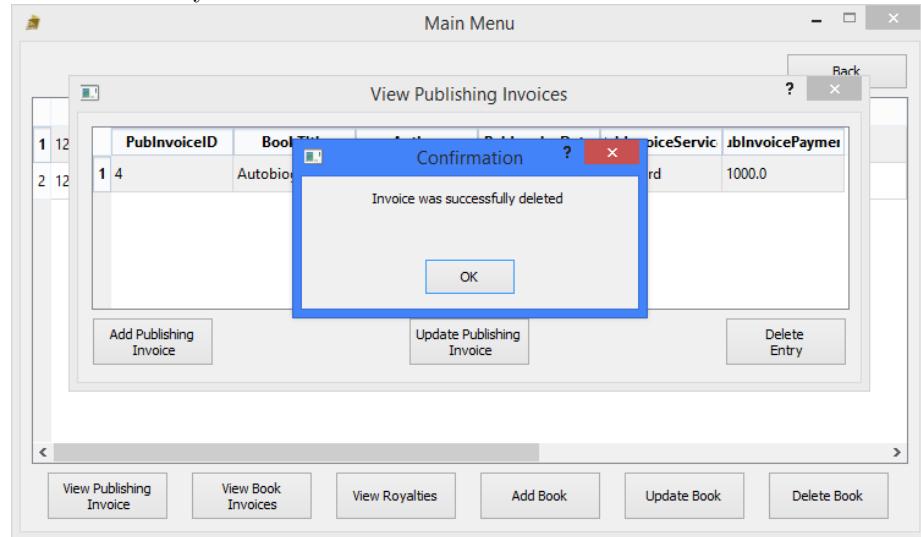


Figure 4.19: View Book Invoices

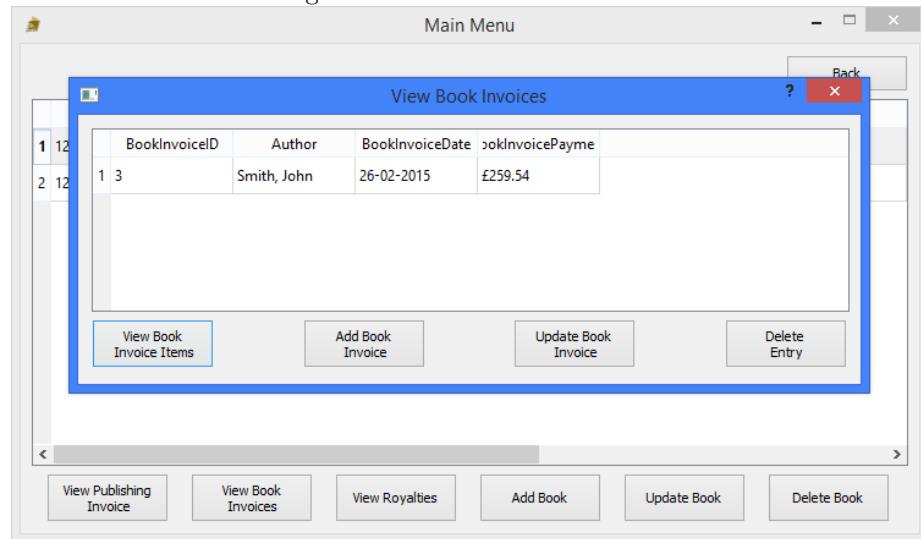


Figure 4.20: Add Book Invoice

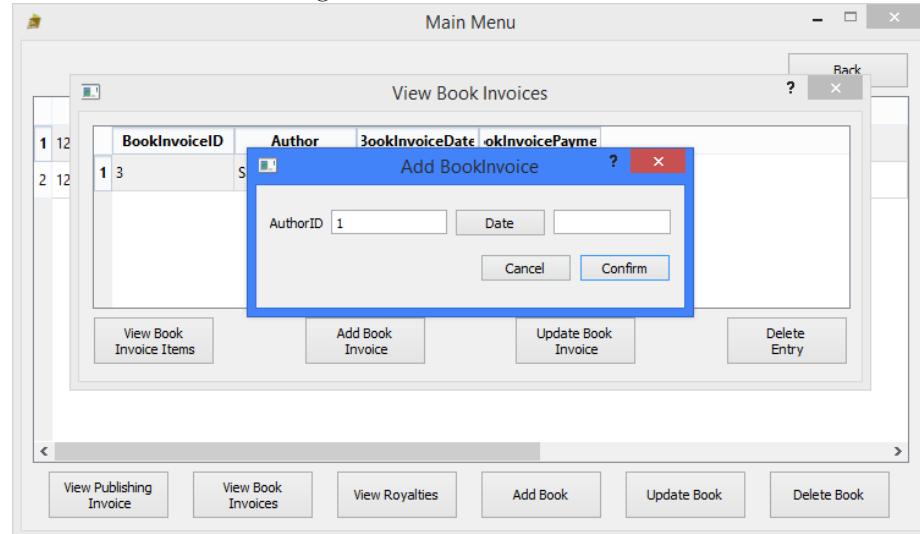


Figure 4.21: Update Book Invoice

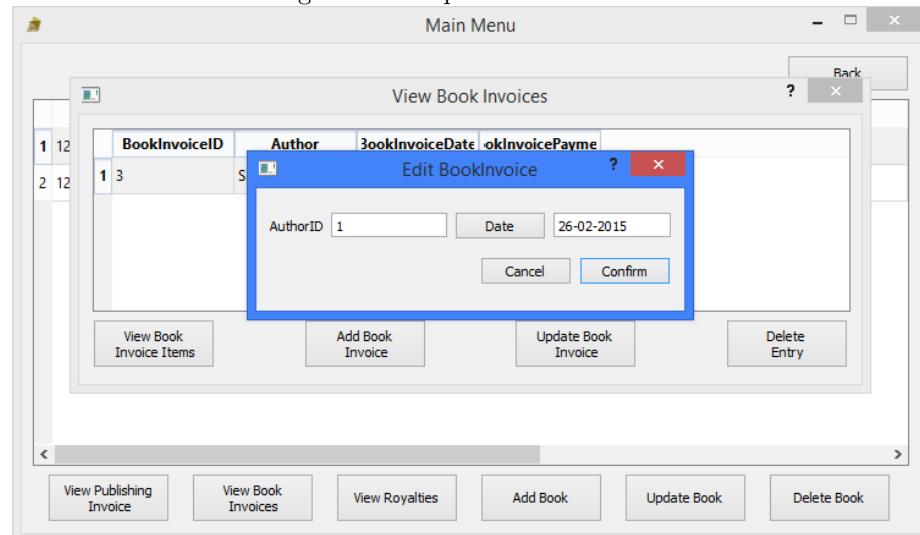


Figure 4.22: View Book Invoice Items

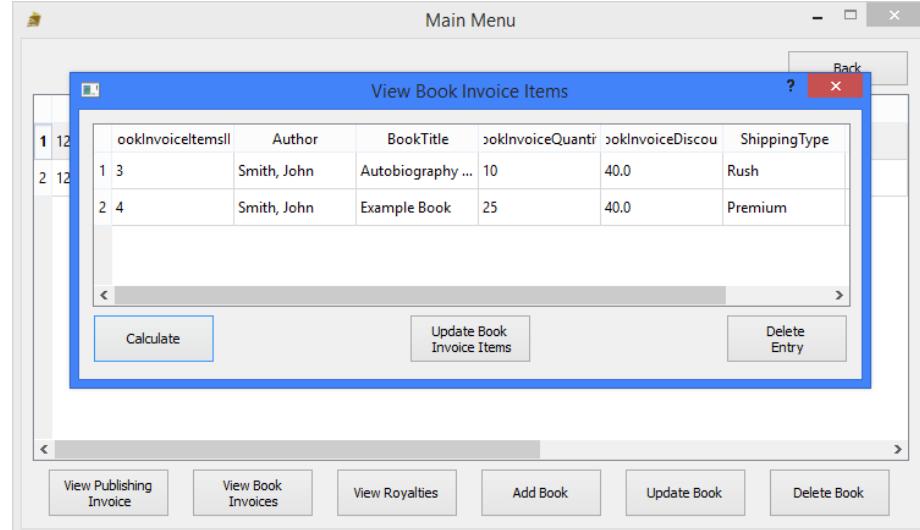


Figure 4.23: Add Book Invoice Items

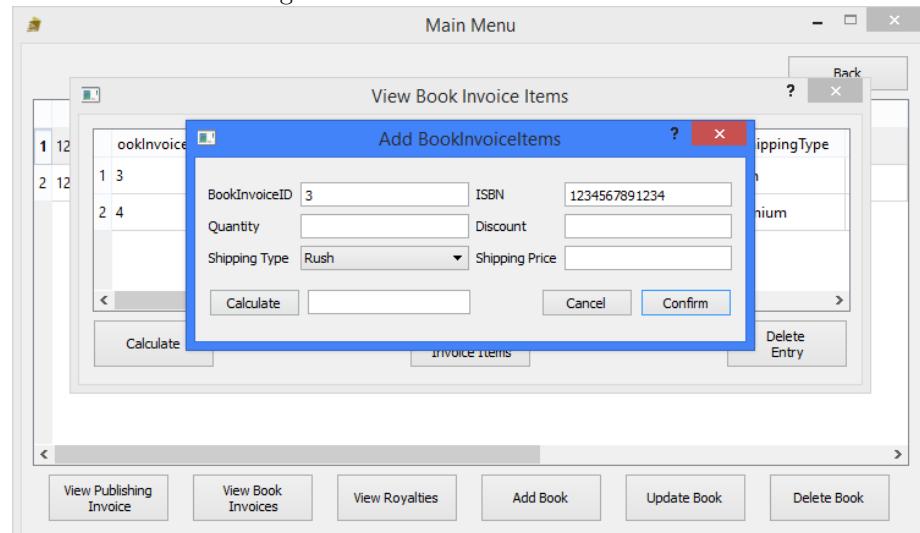


Figure 4.24: Update Book Invoice Items

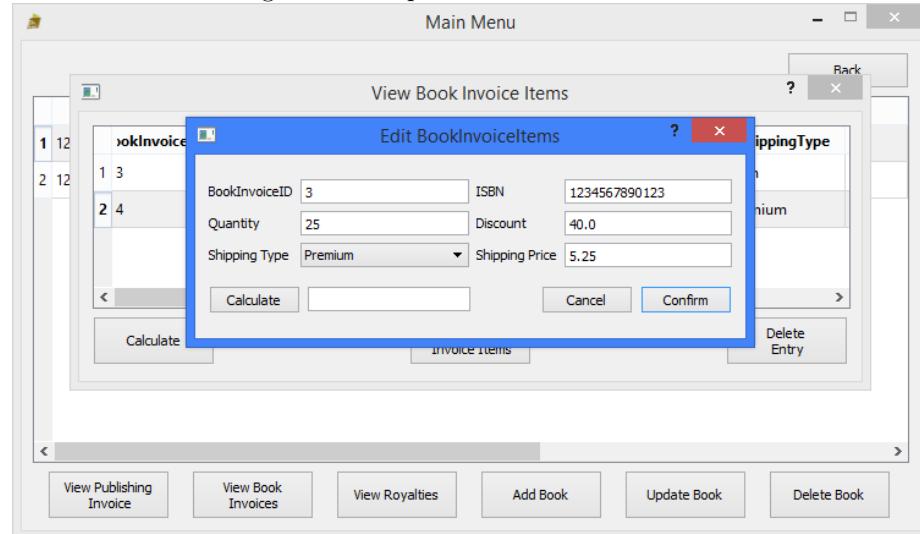


Figure 4.25: View Royalties

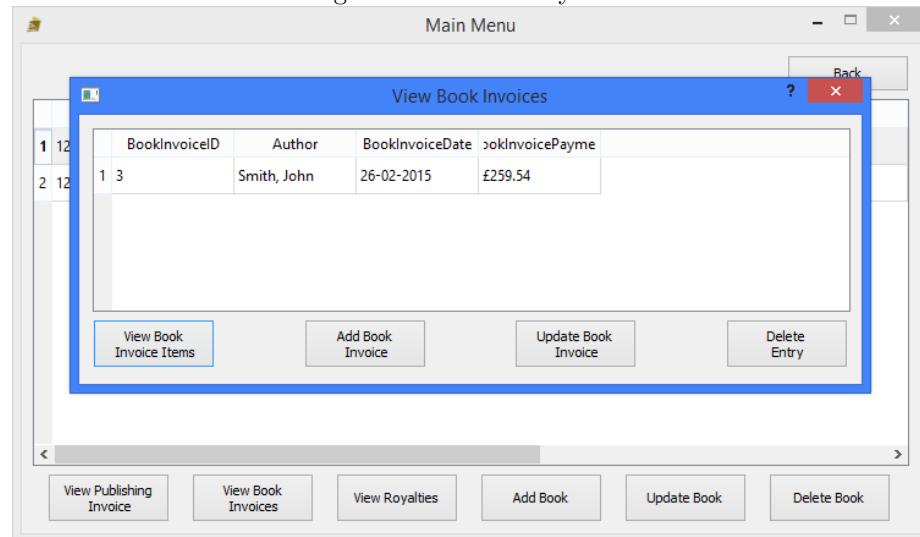


Figure 4.26: Add Royalties

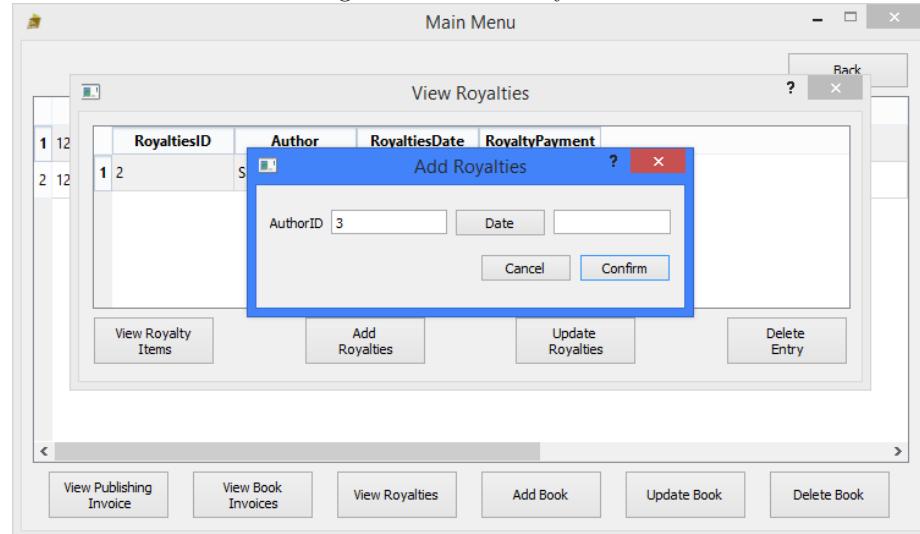


Figure 4.27: Update Royalties

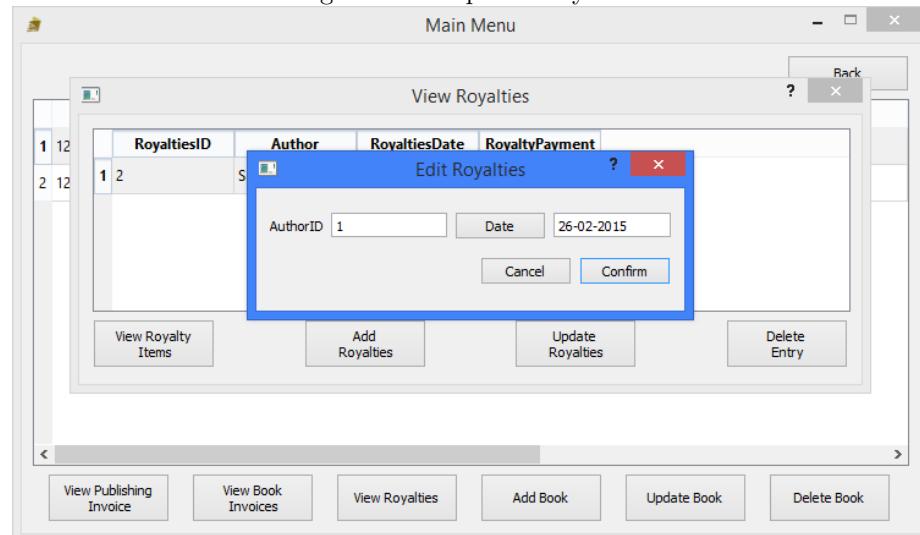


Figure 4.28: View Royalty Items

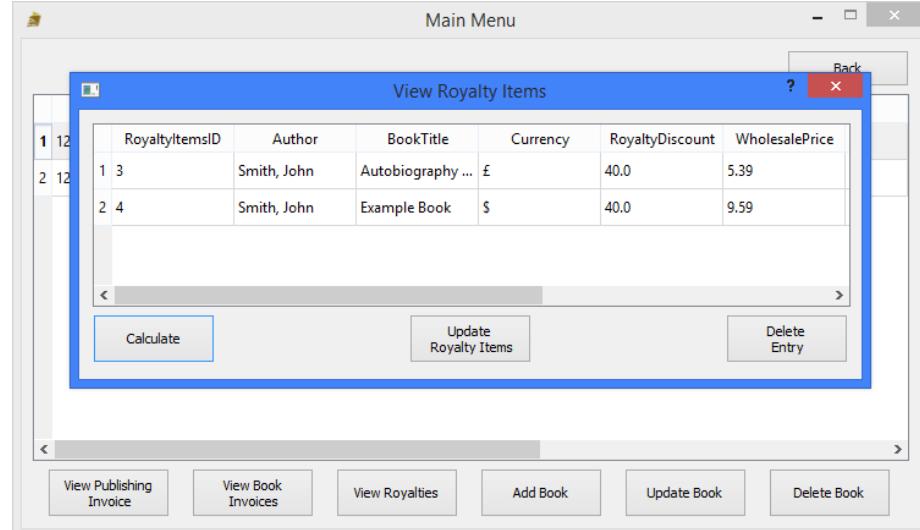


Figure 4.29: Add Royalty Items

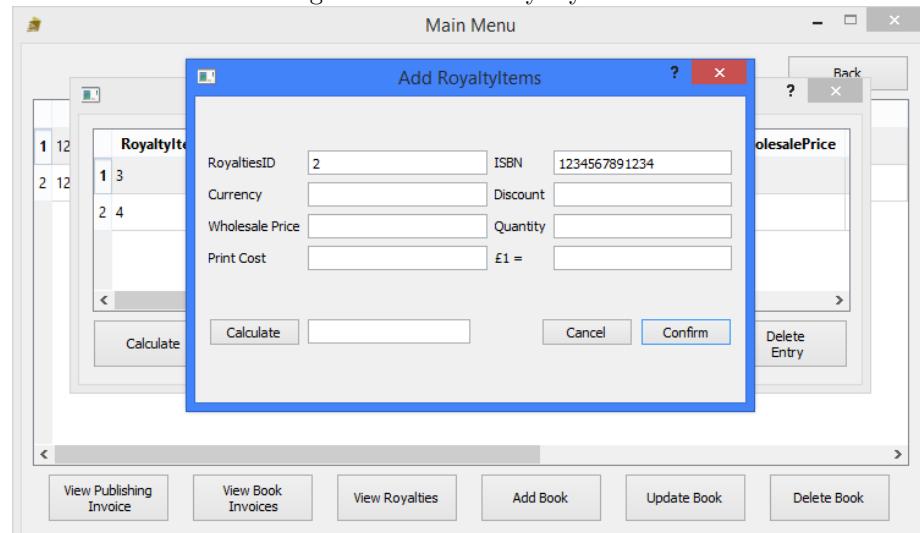


Figure 4.30: Update Royalty Items

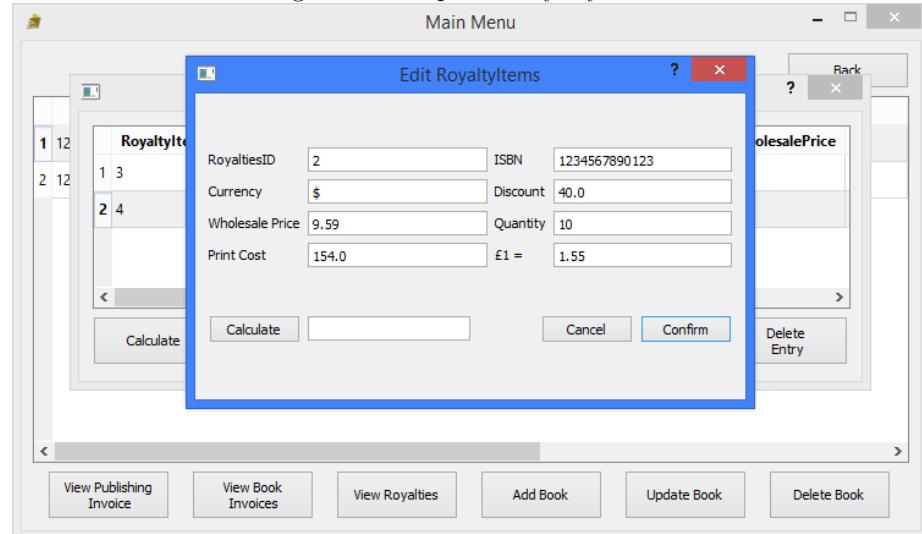


Figure 4.31: Add Book

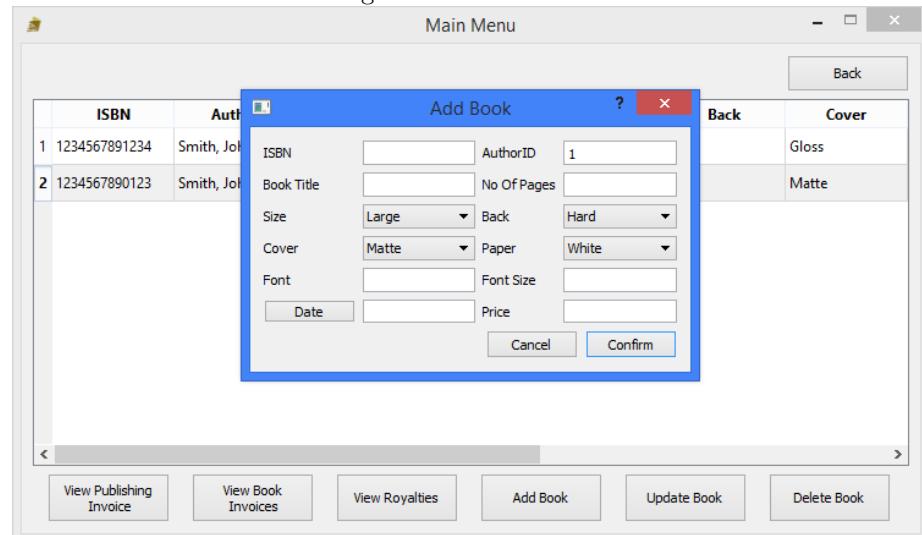


Figure 4.32: Update Book

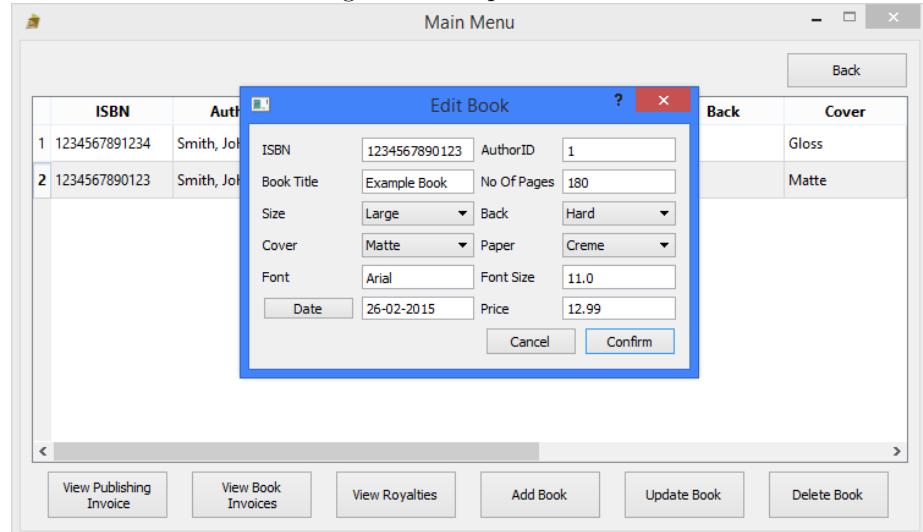


Figure 4.33: Quick Search

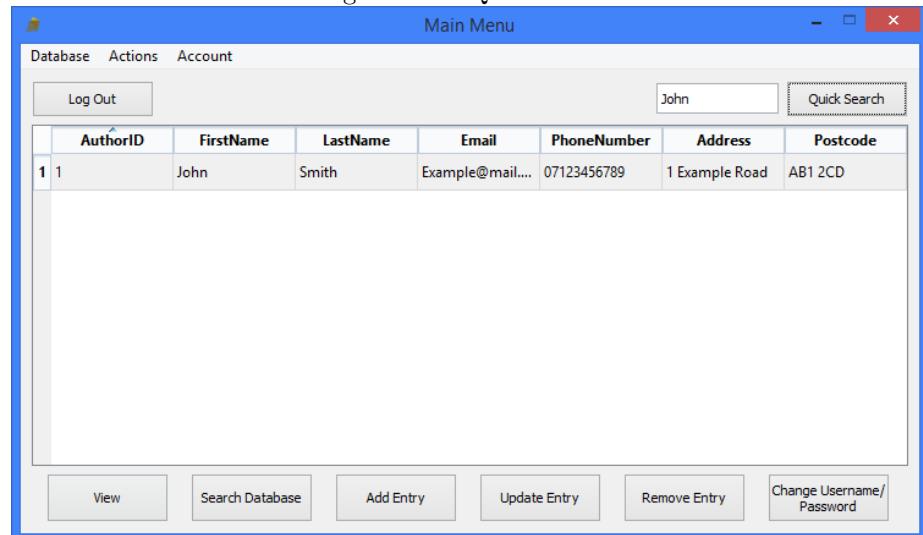


Figure 4.34: Selection For Changing User Credentials

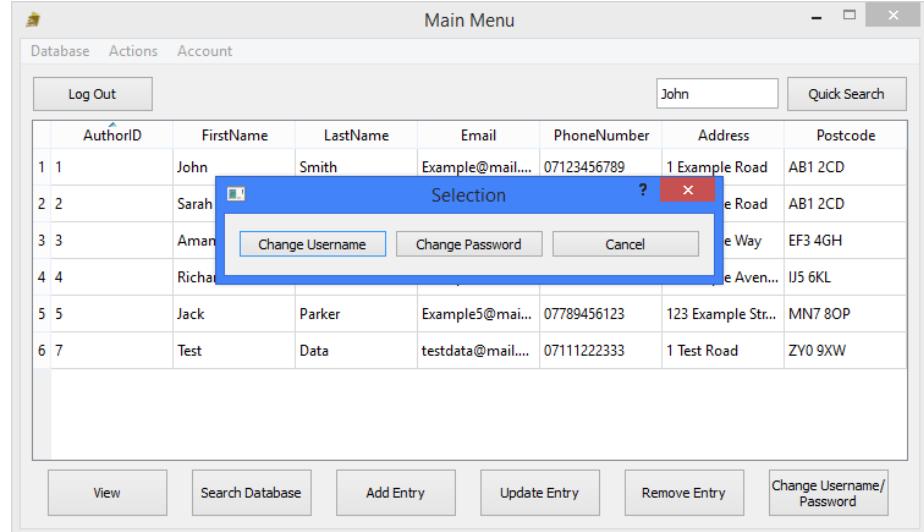


Figure 4.35: Change Username

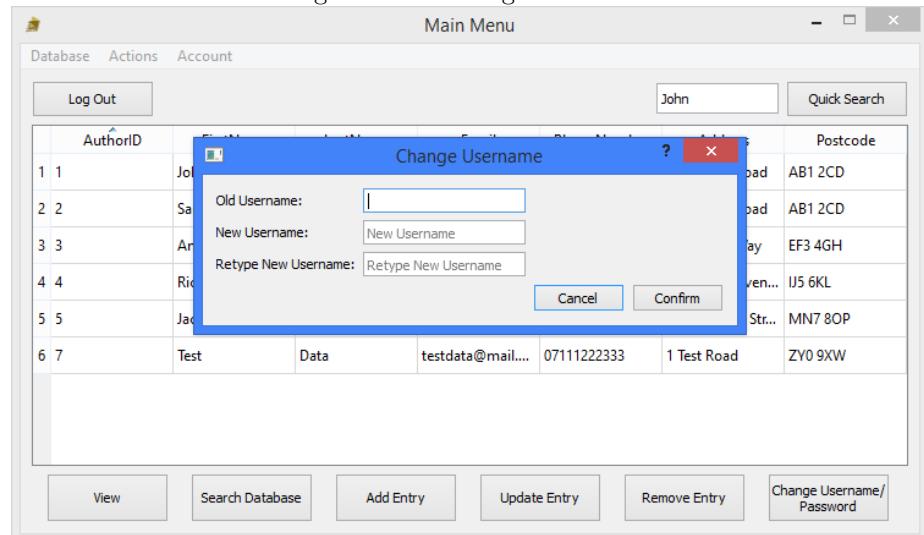


Figure 4.36: Change Username Success - User is given confirmation that their action has been successfully conducted

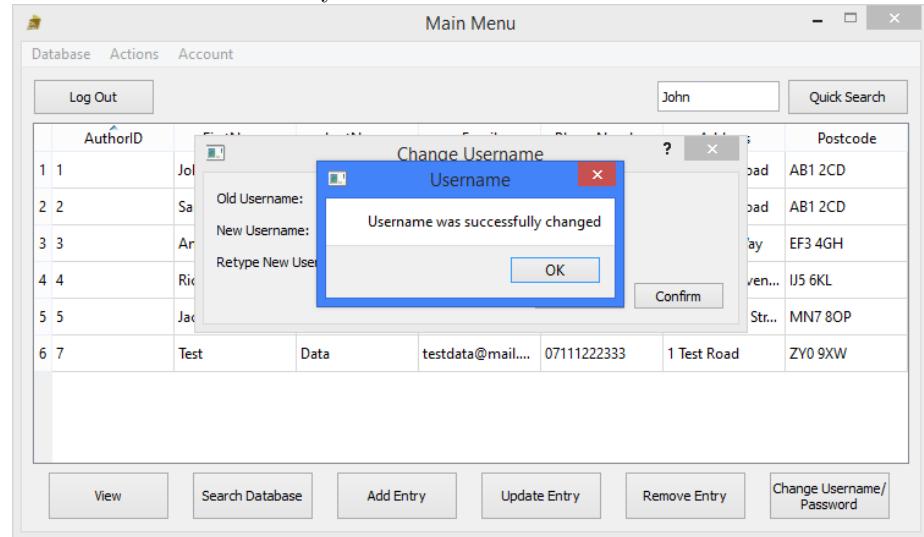


Figure 4.37: Change Password

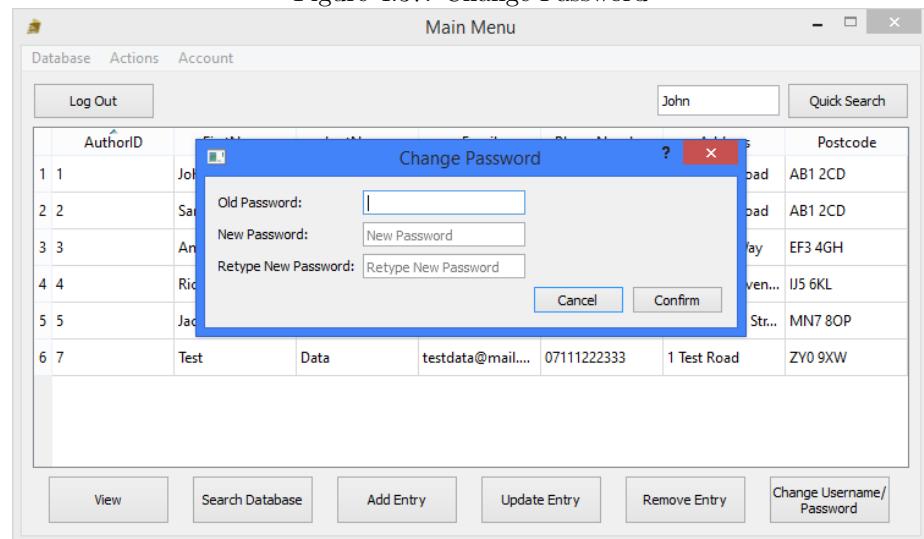


Figure 4.38: Change Password Success - User is given confirmation that their action has been successfully conducted

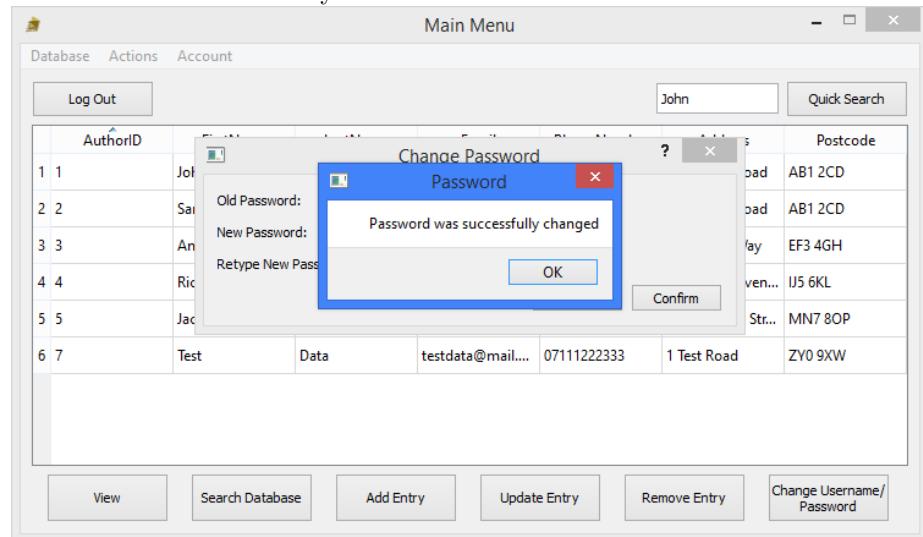


Figure 4.39: Search

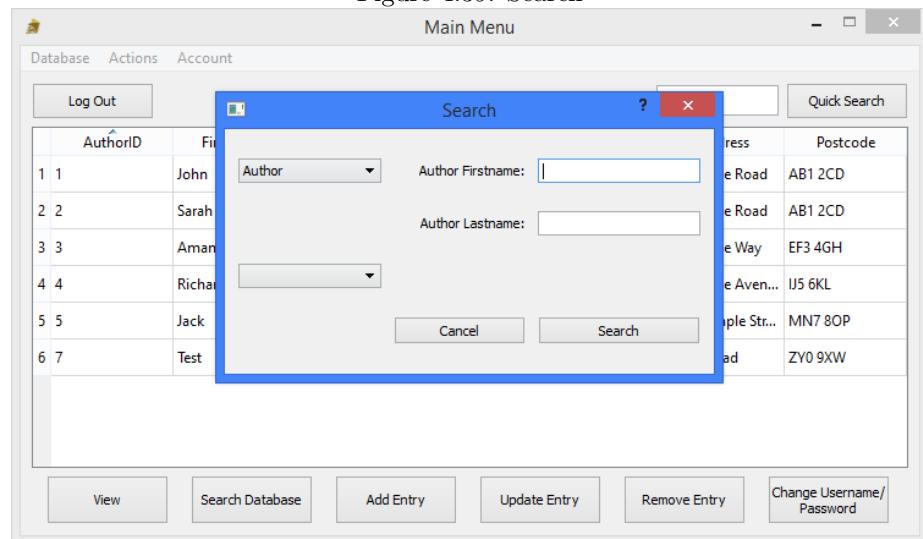


Figure 4.40: Search with a different selection to Author

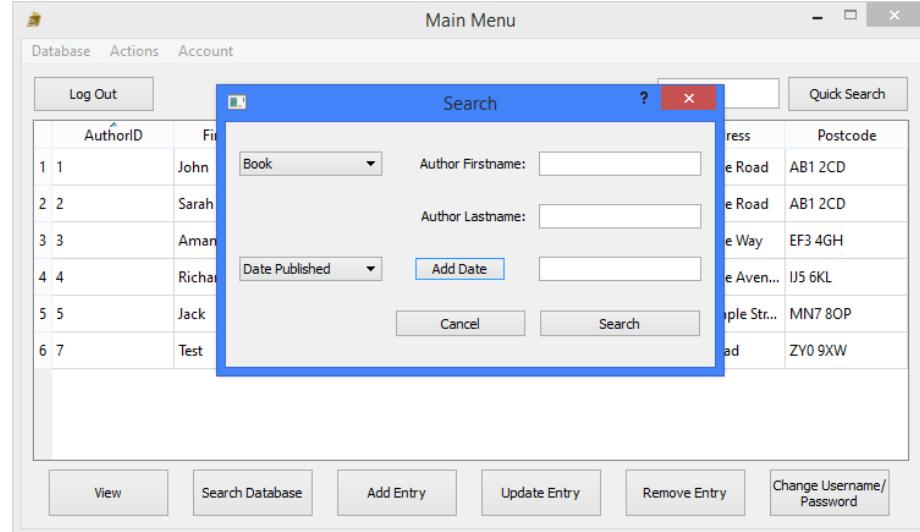


Figure 4.41: No Match Found For Search

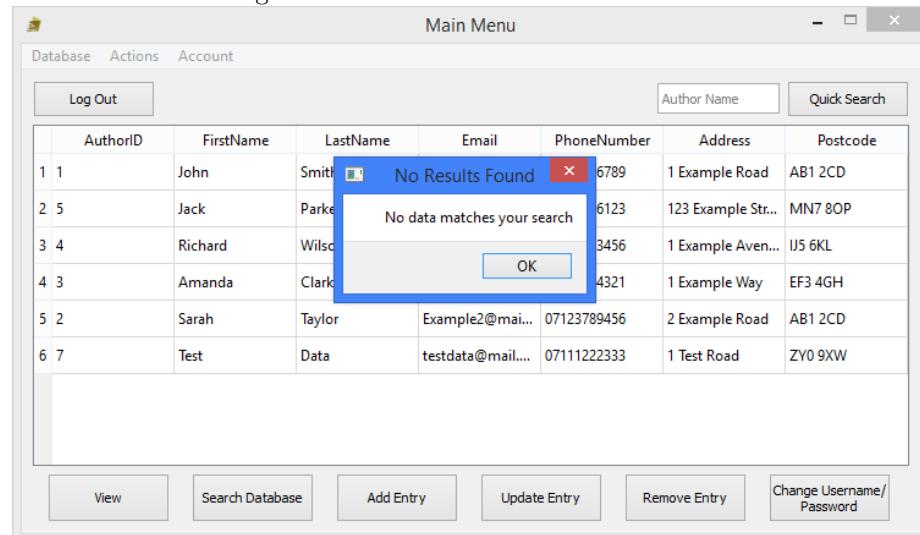
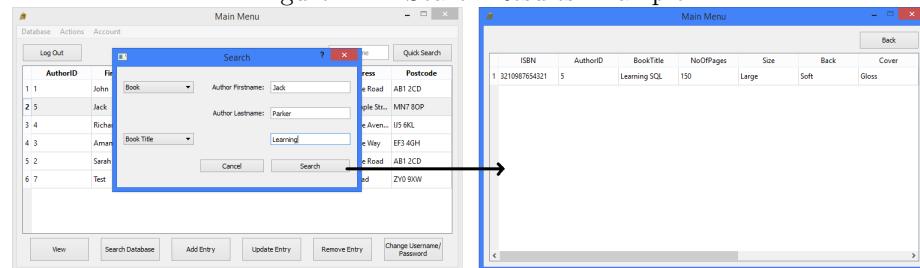
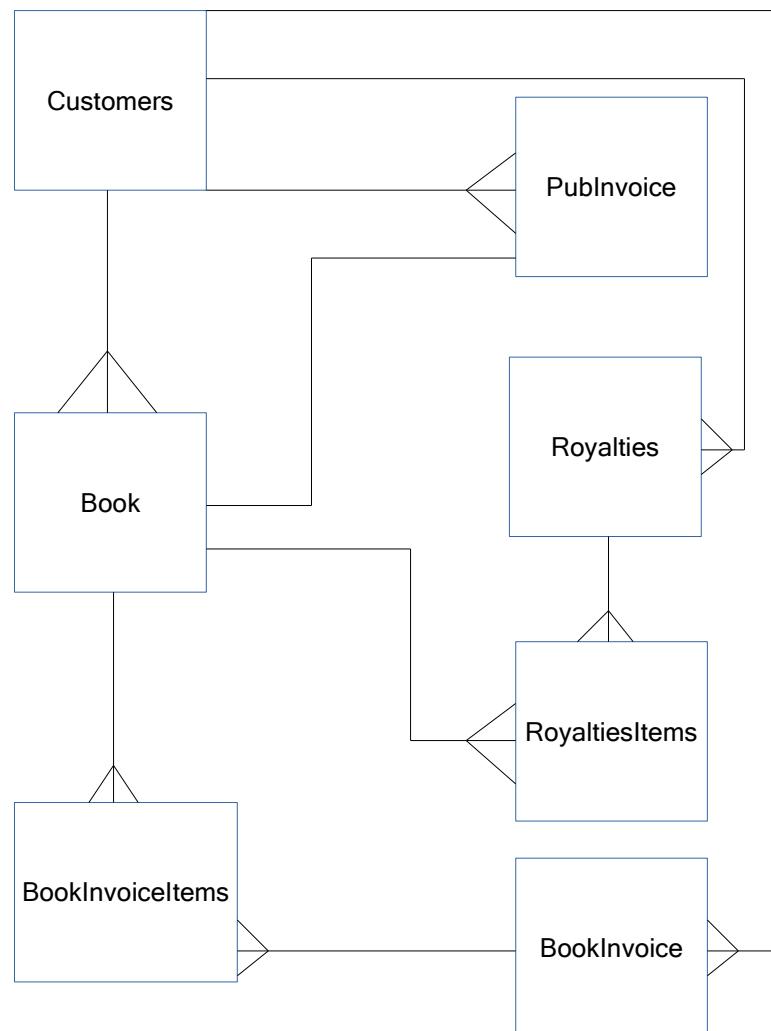


Figure 4.42: Search Results Example



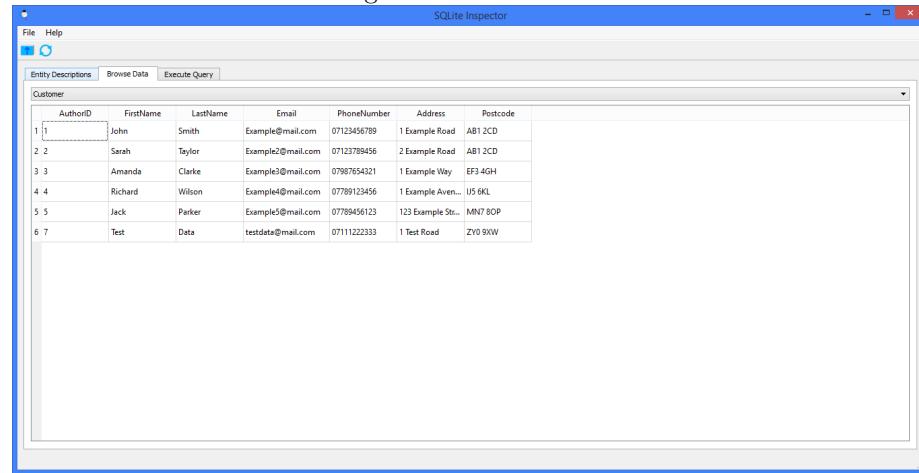
#### 4.5.2 ER Diagram

Figure 4.43: ER Diagram



#### 4.5.3 Database Table Views

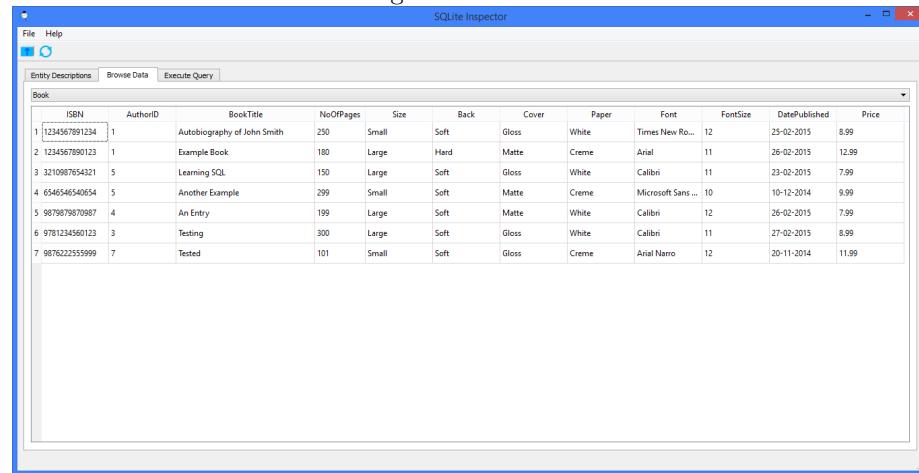
Figure 4.44: Customer



The screenshot shows the SQLite Inspector application window with the 'Customer' table selected. The table has columns: AuthorID, FirstName, LastName, Email, PhoneNumber, Address, and Postcode. There are 6 rows of data.

	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...	U5 6KL
5	5	Jack	Parker	Example5@mail.com	07789456123	123 Example Str...	MN7 8OP
6	7	Test	Data	testdata@mail.com	07111223333	1 Test Road	ZY0 9XW

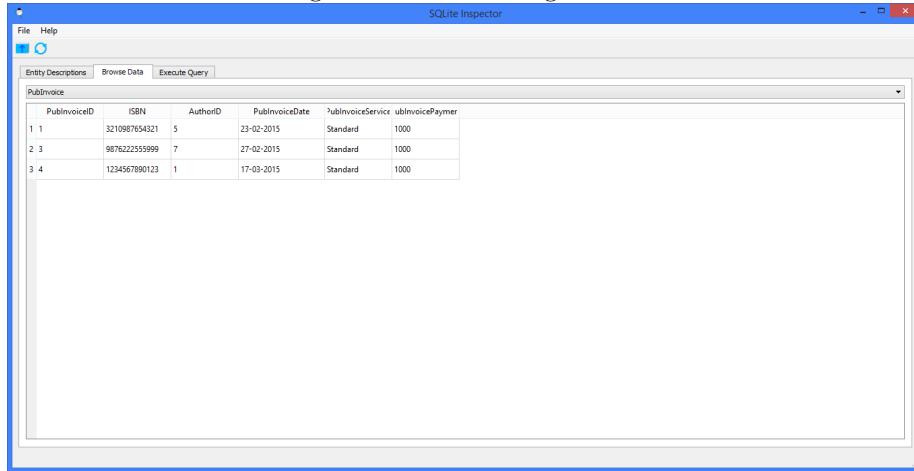
Figure 4.45: Book



The screenshot shows the SQLite Inspector application window with the 'Book' table selected. The table has columns: ISBN, AuthorID, BookTitle, NoOfPages, Size, Back, Cover, Paper, Font, FontSize, DatePublished, and Price. There are 7 rows of data.

	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	987622255999	7	Tested	101	Small	Soft	Gloss	Creme	Arial Narrow	12	20-11-2014	11.99

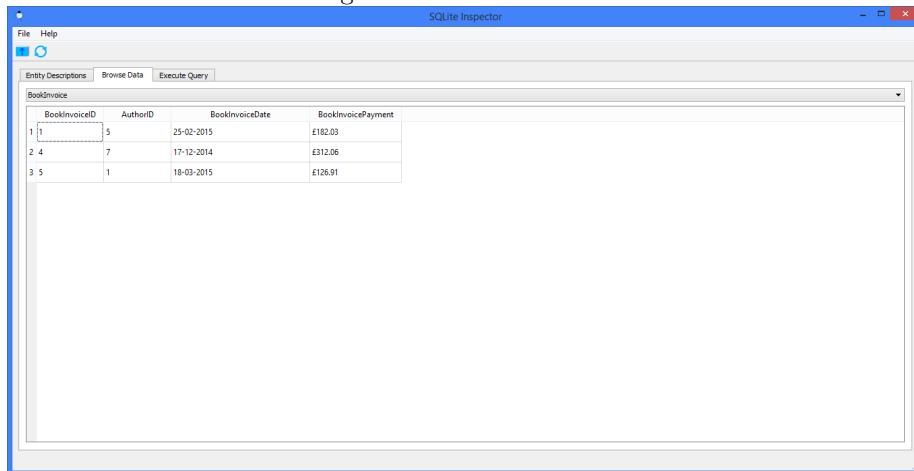
Figure 4.46: Publishing Invoice



The screenshot shows a Windows application window titled "SQLite Inspector". The menu bar includes "File" and "Help". Below the menu is a toolbar with three buttons: "Entity Descriptions", "Browse Data", and "Execute Query". The "Browse Data" button is currently selected. A dropdown menu labeled "PubInvoice" is open, showing a table structure with columns: PubInvoiceID, ISBN, AuthorID, PubInvoiceDate, PubInvoiceService, and PubInvoicePaymer. The table contains three rows of data:

	PubInvoiceID	ISBN	AuthorID	PubInvoiceDate	PubInvoiceService	PubInvoicePaymer
1	1	3210987654321	5	23-02-2015	Standard	1000
2	3	987622355999	7	27-02-2015	Standard	1000
3	4	1234567890123	1	17-03-2015	Standard	1000

Figure 4.47: Book Invoice



The screenshot shows a Windows application window titled "SQLite Inspector". The menu bar includes "File" and "Help". Below the menu is a toolbar with three buttons: "Entity Descriptions", "Browse Data", and "Execute Query". The "Browse Data" button is currently selected. A dropdown menu labeled "BookInvoice" is open, showing a table structure with columns: BookInvoiceID, AuthorID, BookInvoiceDate, and BookInvoicePayment. The table contains three rows of data:

	BookInvoiceID	AuthorID	BookInvoiceDate	BookInvoicePayment
1	1	5	25-02-2015	£182.03
2	4	7	17-12-2014	£312.06
3	5	1	18-03-2015	£126.91

Figure 4.48: Book Invoice Items

The screenshot shows a Windows application window titled "SQLite Inspector". The menu bar includes "File" and "Help". Below the menu is a toolbar with three buttons: "Entity Descriptions", "Browse Data", and "Execute Query". The "Browse Data" button is highlighted. A dropdown menu is open under "Entity Descriptions" showing "BookInvoiceItems". The main area displays a table with the following data:

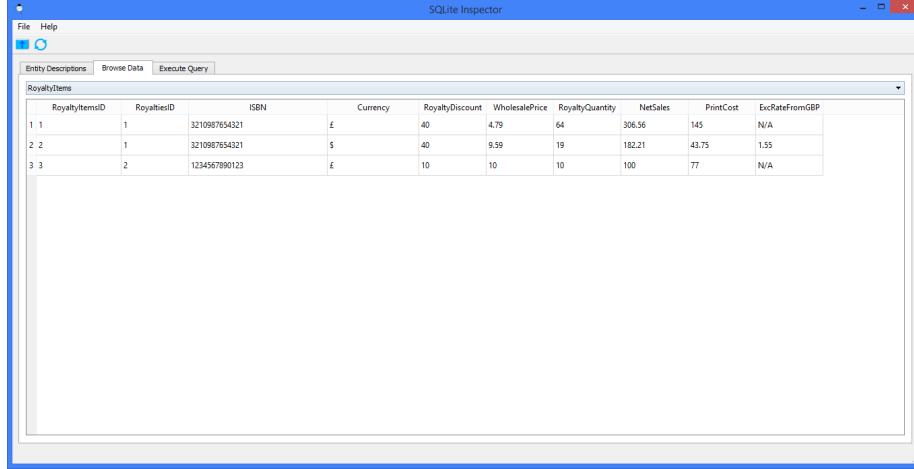
	BookInvoiceItemID	BookInvoiceID	ISBN	BookInvoiceQuantity	BookInvoiceDiscount	ShippingType	ShippingPrice
1	1	1	3210987654321	25	40	Premium	5.25
2	5	1	3210987654321	10	40	Rush	8.99
3	6	1	6546546540654	30	40	Economy	2.99
4	7	4	98762255999	51	50	Rush	6.32

Figure 4.49: Royalties

The screenshot shows a Windows application window titled "SQLite Inspector". The menu bar includes "File" and "Help". Below the menu is a toolbar with three buttons: "Entity Descriptions", "Browse Data", and "Execute Query". The "Browse Data" button is highlighted. A dropdown menu is open under "Entity Descriptions" showing "Royalties". The main area displays a table with the following data:

	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



The screenshot shows the SQLite Inspector application window. The title bar says "SQLite Inspector". The menu bar has "File" and "Help". Below the menu is a toolbar with three buttons: "Entity Descriptions", "Browse Data", and "Execute Query". The "Browse Data" button is highlighted. A dropdown menu is open under "Entity Descriptions" showing "RoyaltyItems". The main area is a table titled "RoyaltyItems" with the following data:

RoyaltyItemID	RoyaltiesID	ISBN	Currency	RoyaltyDiscount	WholesalePrice	RoyaltyQuantity	NetSales	PrintCost	ExcRateFromGBP
1 1	1	3210967654321	£	40	4.79	64	306.56	145	N/A
2 2	1	3210967654321	\$	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:

---

```

1 CREATE TABLE Customer
2   (AuthorID integer,
3    FirstName text,
4    LastName text,
5    Email text,
6    PhoneNumber text,
7    Address text,
8    Postcode text,
9    primary key(AuthorID))

```

---

Creating Book Table:

---

```

1 CREATE TABLE Book
2   (ISBN text,
3    AuthorID integer,
4    BookTitle text,
5    NoOfPages integer,
6    Size text,
7    Back text,
8    Cover text,
9    Paper text,
10   Font text,
11   FontSize real,

```

---

```

12          DatePublished date ,
13          Price real ,
14          primary key (ISBN) ,
15          foreign key (AuthorID) references
                        Customer (AuthorID))

```

---

Creating Publishing Invoice Table:

```

1 CREATE TABLE PubInvoice
2             (PubInvoiceID integer ,
3              ISBN text ,
4              AuthorID integer ,
5              PubInvoiceDate date ,
6              PubInvoiceService text ,
7              PubInvoicePayment real ,
8              primary key (PubInvoiceID) ,
9              foreign key (AuthorID) references
                        Customer (AuthorID) ,
10             foreign key (ISBN) references
                            Book (ISBN))

```

---

Creating Book Invoice Table:

```

1 CREATE TABLE BookInvoice
2             (BookInvoiceID integer ,
3              AuthorID integer ,
4              BookInvoiceDate date ,
5              BookInvoicePayment real ,
6              primary key (BookInvoiceID) ,
7              foreign key (AuthorID) references
                        Customer (AuthorID))

```

---

Creating Book Invoice Items Table:

```

1 CREATE TABLE BookInvoiceItems
2             (BookInvoiceItemsID integer ,
3              BookInvoiceID integer ,
4              ISBN text ,
5              BookInvoiceQuantity integer ,
6              BookInvoiceDiscount real ,
7              ShippingType text ,
8              ShippingPrice real ,
9              primary key (BookInvoiceItemsID) ,
10             foreign key (BookInvoiceID)
                            references
                            BookInvoice (BookInvoiceID) ,

```

```

11      foreign key(ISBN) references
          Book(ISBN))

```

---

Creating Royalties Table:

---

```

1 CREATE TABLE Royalties
2             (RoyaltiesID integer,
3              AuthorID integer,
4              RoyaltiesDate date,
5              RoyaltyPayment real,
6              primary key(RoyaltiesID),
7              foreign key(AuthorID) references
                  Customer(AuthorID))

```

---

Creating Royalty Items Table:

---

```

1 CREATE TABLE RoyaltyItems
2             (RoyaltyItemsID integer,
3              RoyaltiesID integer,
4              ISBN text,
5              Currency text,
6              RoyaltyDiscount real,
7              WholesalePrice real,
8              RoyaltyQuantity integer,
9              NetSales real,
10             PrintCost real,
11             ExcRateFromGBP real,
12             primary key(RoyaltyItemsID),
13             foreign key(RoyaltiesID) references
                  Royalties(RoyaltiesID),
14             foreign key(ISBN) references
                  Book(ISBN))

```

---

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

---

```

1 CREATE TABLE if not exists LoginDetails
2             (Username text
3              Password text)

```

---

#### 4.5.5 SQL Queries

Checking whether the login table is existent:

---

```

1 select name
2 from sqlite_master

```

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

---

Reference: Module 1, subsection 4.10.1, line 35.

Creating the default Username and Password.

---

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

---

Reference: Module 1, subsection 4.10.1, line 47.

Fetching the current username:

---

```
1 select Username
2 from LoginDetails
```

---

Reference: Module 1, subsection 4.10.1, line 108.

Fetching the current password:

---

```
1 select Password
2 from LoginDetails
```

---

Reference: Module 1, subsection 4.10.1, line 110.

Fetching all Customer data:

---

```
1 select *
2 from Customer
```

---

Reference: Module 2, subsection 4.10.2, line 38.

Fetching all Book data:

---

```
1 select *
2 from Book
```

---

Reference: Module 2, subsection 4.10.2, line 111.

Selecting data to be entered by user for PubInvoice:

---

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

---

Reference: Module 2, subsection 4.10.2, line 115.

Selecting data to be entered by user for BookInvoice:

---

```
1 select AuthorID, BookInvoiceDate
2 from BookInvoice
```

---

Reference: Module 2, subsection 4.10.2, line 119.

Selecting data to be entered by user for Royalties:

---

```
1 select AuthorID, RoyaltiesDate
2 from Royalties
```

---

Reference: Module 2, subsection 4.10.2, line 123.

Selecting data to be entered by user for BookInvoiceItems:

---

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

---

Reference: Module 2, subsection 4.10.2, line 127.

Selecting data to be entered by user for RoyaltyItems:

---

```
1 select RoyaltiesID, ISBN, Currency, RoyaltyDiscount,
      WholesalePrice, RoyaltyQuantity, PrintCost,
      ExcRateFromGBP
2 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:

---

```

1     self.input_data = []
2     if self.CurrentTable == "Book":
3         self.TableValues = "Book (ISBN, AuthorID, BookTitle, NoOfPages, Size, Back, Cover, Paper, Font,
4             FontSize, DatePublished, Price)"
5         self.Placeholders = "(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)"
6         self.NoOfEntries = 12
7
8     elif self.CurrentTable == "PubInvoice":
9         self.TableValues = "PubInvoice (ISBN, AuthorID, PubInvoiceDate, PubInvoiceService,
10             PubInvoicePayment)"
11        self.Placeholders = "(?, ?, ?, ?, ?)"
12        self.NoOfEntries = 5
13
14    elif self.CurrentTable == "BookInvoice":
15        self.TableValues = "BookInvoice (AuthorID, BookInvoiceDate)"
16        self.Placeholders = "(?, ?)"
17        self.NoOfEntries = 2
18
19    elif self.CurrentTable == "Royalties":
20        self.TableValues = "Royalties (AuthorID, RoyaltiesDate)"
21        self.Placeholders = "(?, ?)"
22        self.NoOfEntries = 2
23
24    elif self.CurrentTable == "BookInvoiceItems":
25        self.TableValues = "BookInvoiceItems (BookInvoiceID, ISBN, BookInvoiceQuantity,
26             BookInvoiceDiscount, ShippingType, ShippingPrice)"
27        self.Placeholders = "(?, ?, ?, ?, ?, ?, ?)"
28        self.NoOfEntries = 6
29
30    elif self.CurrentTable == "RoyaltyItems":
31        self.TableValues = "RoyaltyItems (RoyaltiesID, ISBN, Currency, RoyaltyDiscount, WholesalePrice,
32             RoyaltyQuantity, PrintCost, ExcRateFromGBP, NetSales)"
33        self.Placeholders = "(?, ?, ?, ?, ?, ?, ?, ?, ?)"
34        self.NoOfEntries = 9
35
36    for count in range(0, self.NoOfEntries):
37        try: #putting the input data in a list
38            self.input_data.append(str(self.AddWindow.inputList[count].currentText()))
39        except:
40            if count == 8 and self.CurrentTable == "RoyaltyItems":
41                self.input_data.append(self.NetSales) #Net sales are calculated, as opposed to being
42                entered
43            else:
44                self.input_data.append(self.AddWindow.inputList[count].text())
45
46    with sqlite3.connect("PP.db") as db:
47        cursor = db.cursor()
48        cursor.execute("PRAGMA foreign_keys = ON")
49        self.sql = "insert into {} values {}".format(self.TableValues, self.Placeholders)
50        cursor.execute(self.sql, self.input_data)
51        db.commit()

```

---

Reference: Module 2, subsection 4.10.2, line 168-214. Line 44 of the extract 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:

---

```

1     sql = "select ISBN from Book where AuthorID =
2         {}".format(self.ConfirmDialog.SelectedAuthorID)

```

---

Reference: Module 2, subsection 4.10.2, line 239.

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

---

```

1         sql = "delete from BookInvoiceItems where ISBN =
2             {}".format(list(list(self.ISBNDeletion)[count])[0])
3         cursor.execute(sql)
4         sql = "delete from RoyaltyItems where ISBN =
5             {}".format(list(list(self.ISBNDeletion)[count])[0])
6         cursor.execute(sql)
7         sql = "delete from PubInvoice where AuthorID =
8             {}".format(self.ConfirmDialog.SelectedAuthorID)

```

---

---

```

6         cursor.execute(sql)
7     sql = "delete from BookInvoice where AuthorID =
8         {}".format(self.ConfirmDialog.SelectedAuthorID)
9     cursor.execute(sql)
10    sql = "delete from Royalties where AuthorID =
11        {}".format(self.ConfirmDialog.SelectedAuthorID)
12    cursor.execute(sql)
13    sql = "delete from Book where AuthorID = {}".format(self.ConfirmDialog.SelectedAuthorID)
14    cursor.execute(sql)
15    sql = "delete from Customer where AuthorID =
16        {}".format(self.ConfirmDialog.SelectedAuthorID)

```

---

Reference: Module 2, subsection 4.10.2, lines 243-255.

Deleting a selected entry:

---

```

1     sql = "delete from {} where {} = '{}'".format(self.CurrentTable,
2             self.SelectedIDName, self.SelectedID)

```

---

Reference: Module 2, subsection 4.10.2, line 326.

Fetching the price of a selected book:

---

```

1             cursor.execute("select Price from
2                 Book where ISBN =
3                     {}".format(self.ISBN))

```

---

Reference: Module 2, subsection 4.10.2, line 491.

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

---

```

1             self.selectText = "Firstname, Lastname,
2                 Email, Phononenumber, Address, Postcode"
3             self.UpdateEntryWindow.table.sql =
4                 "select {} from Customer where
5                     AuthorID = {}".format(self.selectText,
6                         self.SelectedAuthorID)

```

---

Reference: Module 2, subsection 4.10.2, lines 556-557.

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

---

```

1             sql = "select * from Book where ISBN =
2                 {}".format(self.ViewWindow.table.item(self.ViewWindow.SelectedRow, 0).text())

```

---

Reference: Module 2, subsection 4.10.2, line 587.

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

---

```
1     sql = "select ISBN, AuthorID, PubInvoiceDate, PubInvoiceService, PubInvoicePayment from
        PubInvoice where PubInvoiceID =
        {}".format(self.PubInvoiceWindow.table.item(self.PubInvoiceWindow.SelectedRow,
        0).text())
```

---

Reference: Module 2, subsection 4.10.2, line 600.

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

---

```
1     sql = "select AuthorID, BookInvoiceDate from BookInvoice where BookInvoiceID =
        {}".format(self.BookInvoiceWindow.table.item(self.BookInvoiceWindow.SelectedRow,
        0).text())
```

---

Reference: Module 2, subsection 4.10.2, line 613.

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

---

```
1     sql = "select AuthorID, RoyaltiesDate from Royalties where RoyaltiesID =
        {}".format(self.RoyaltiesWindow.table.item(self.RoyaltiesWindow.SelectedRow,
        0).text())
```

---

Reference: Module 2, subsection 4.10.2, line 626.

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

---

```
1     sql = "select BookInvoiceID, ISBN, BookInvoiceQuantity, BookInvoiceDiscount,
        ShippingType, ShippingPrice from BookInvoiceItems where BookInvoiceItemsID =
        {}".format(self.BookInvoiceItemsWindow.table.item(self.BookInvoiceItemsWindow.SelectedRow,
        0).text())
```

---

Reference: Module 2, subsection 4.10.2, line 642.

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

---

```
1     sql = "select RoyaltiesID, ISBN, Currency, RoyaltyDiscount, WholesalePrice,
        RoyaltyQuantity, PrintCost, ExRateFromGBP from RoyaltyItems where
        RoyaltyItemsID =
        {}".format(self.RoyaltyItemsWindow.table.item(self.RoyaltyItemsWindow.SelectedRow,
        0).text())
```

---

Reference: Module 2, subsection 4.10.2, line 658.

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

---

```

1             sql = "update PubInvoice set ISBN
2                 = {0} where ISBN =
3                 {1}" .format(self.UpdateList[0],
4                           self.SelectedID)
5             cursor.execute(sql)
6             sql = "update BookInvoiceItems
7                 set ISBN = {0} where ISBN =
8                 {1}" .format(self.UpdateList[0],
9                           self.SelectedID)
10            cursor.execute(sql)
11            sql = "update RoyaltyItems set
12                ISBN = {0} where ISBN =
13                {1}" .format(self.UpdateList[0],
14                           self.SelectedID)

```

---

Reference: Module 2, subsection 4.10.2, lines 766-770.

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

```

---

The update string:

---

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

---

Reference: Module 2, subsection 4.10.2, lines 758-761, and 774.

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             cursor.execute("select Book.BookTitle ,
2                                 BookInvoiceItems.ISBN, Firstname, Lastname,
3                                 BookInvoiceItems.BookInvoiceID from Book,
4                                 BookInvoiceItems, Customer, BookInvoice where
5                                 BookInvoiceItems.BookInvoiceID = {} and
6                                 BookInvoiceItems.BookInvoiceItemsID = {} and
7                                 Book.ISBN = BookInvoiceItems.ISBN and
8                                 BookInvoice.BookInvoiceID =
9                                 BookInvoiceItems.BookInvoiceID and
10                                Customer.AuthorID =
11                                BookInvoice.AuthorID".format(self.SelectedID,
12                                              self.ID[count]))

```

---

Reference: Module 2, subsection 4.10.2, line 868.

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     cursor.execute("select Book.BookTitle, RoyaltyItems.ISBN, F firstname, L lastname,
                    RoyaltyItems.RoyaltiesID from Book, RoyaltyItems, Customer, Royalties where
                    RoyaltyItems.RoyaltiesID = {} and RoyaltyItems.RoyaltiesID = {} and
                    Book.ISBN = RoyaltyItems.ISBN and Royalties.RoyaltiesID =
                    RoyaltyItems.RoyaltiesID and Customer.AuthorID =
                    Royalties.AuthorID".format(self.SelectedID, self.ID[count]))
```

---

Reference: Module 2, subsection 4.10.2, line 895.

Fetching all Customers that match the firstname or lastname entered.

---

```
1     self.TableWidget.sql = "select * from
        Customer where F firstname like
        '{0}%' or L lastname like
        '{0}%'" .format(self.QSText[0])
```

---

Reference: Module 2, subsection 4.10.2, line 922

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.

---

```
1 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.

---

```
1 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.

---

```

1      Selection = "BookInvoiceItems.BookInvoiceQuantity,
2          BookInvoiceItems.BookInvoiceDiscount, BookInvoiceItems.ShippingPrice,
3          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.

---

```

1      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.

---

```

1      Selection = "RoyaltyItems.Currency, RoyaltyItems.NetSales,
2          RoyaltyItems.ExcRateFromGBP, RoyaltyItems.RoyaltyQuantity, Book.NoOfPages,
3          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,
2          Book.AuthorID, Book.ISBN,
3          {0}.ISBN, {0}ID from Customer,
4          Book, {0} where
5          (Customer.Firstname like '{1}%' or
6          Customer.Lastname like '{2}%') and
7          {0}.{3} like '{4}%' and
8          Customer.AuthorID = Book.AuthorID
9          and Book.ISBN =
10         {0}.ISBN".format(self.Table,
11                         self.Firstname, self.Lastname,
12                         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
2          where Firstname like '{0}%' or
3          Lastname like
4          '{1}%'" .format(self.Firstname,
5                           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.

---

```

1      def UpdateChanges(self):
2          UL = [] #UL = Update List
3          for count in range(0, 6):
4              UL.append(self.table.item(0,
5                               count).text())
6          self.Update = "Firstname = '{}', Lastname = "
7          self.Update += "{}', Email = '{}', Phonenumber = '{}',
8          Address = '{}', Postcode = "
9          self.Update += "{}".format(UL[0], UL[1], UL[2], UL[3],
10                                UL[4], UL[5])
11      with sqlite3.connect("PP.db") as db:
12          cursor = db.cursor()
13          cursor.execute("PRAGMA foreign_keys = ON")

```

---

```
9      sql = "update {} set {} where {} =  
10     {}".format(self.TableName,  
11                   self.Update, self.ID, self.selectedID)  
12     cursor.execute(sql)  
13     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 reliability 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

#### Calculating the Full Book Invoice Payment

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 ="
10         sql += "{}".format(self.selectedID)
11         cursor.execute(sql)
12         self.IDListTuple = list(cursor.fetchall())
13         for count in range(0, len(self.IDListTuple)): #conversion from tuple
14             self.IDList.append(list(self.IDListTuple[count])[0])
15
16         self.currentID = 0
17         for count in range(0, len(self.IDList)):
18             if self.currentID < self.IDList[count]: #finding biggest ID no.
19                 self.currentID = self.IDList[count]
20
21         sql = "select ISBN from BookInvoiceItems where BookInvoiceID = {}".format(self.selectedID)
22         cursor.execute(sql)
23         self.ISBNs = list(cursor.fetchall())
24
25         for count2 in range(0, len(self.ISBNs)):
26             if self.currentID != 0:
27                 for count in range(0, self.currentID + 1):
28                     self.Empty = False
29                     Selection = "BookInvoiceItems.BookInvoiceQuantity,"
30                     Selection += "BookInvoiceItems.BookInvoiceDiscount, BookInvoiceItems.ShippingPrice,"
31                     Selection += "Book.Price"
32                     Tables = "BookInvoiceItems, Book"
33                     sql = "select {} from {} where BookInvoiceItems.BookInvoiceID = {} and"
34                     sql += "BookInvoiceItems.BookInvoiceItemsID = {} and BookInvoiceItems.ISBN = {} and"
35                     sql += "Book.ISBN = BookInvoiceItems.ISBN".format(Selection, Tables,
36                     self.selectedID, count+1, list(self.ISBNs[count2])[0])
37                     cursor.execute(sql)
38                     try:
39                         self.SelectionList = list(cursor.fetchone())
40                     except:
41                         self.Empty = True
42
43                     if self.Empty != True:
44                         self.Quantity = self.SelectionList[0]
45                         self.Discount = self.SelectionList[1] / 100
46                         self.ShippingPrice = self.SelectionList[2]
47                         self.Price = self.SelectionList[3]
48
49                         self.TempPayment = (self.Quantity * self.Price)
50                         self.Discount *= self.TempPayment
51                         self.TempPayment -= self.Discount
52                         self.TempPayment += self.ShippingPrice
53
54                         self.BookInvoicePayment += self.TempPayment
55
56             elif self.currentID == 0:
57                 self.BookInvoicePayment = None

```

---

```

53         sql = "update BookInvoice set BookInvoicePayment = '{}' where BookInvoiceID =
54             {}".format(self.BookInvoicePayment, self.selectedID)
55         cursor.execute(sql)
56         db.commit()
57
58     if self.currentID != 0:
59         self.BookInvoicePayment = "{0:.2f}".format(self.BookInvoicePayment)
60         sql = "update BookInvoice set BookInvoicePayment = '{}' where BookInvoiceID =
61             {}".format(self.BookInvoicePayment, self.selectedID)
62         cursor.execute(sql)
63         db.commit()

```

---

Lines 12 to 18 of the extract 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 30 defines the sql statement to fetch all the necessary data to conduct the calculations. Then, upto line 48, 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.

## Calculating the Full Royalty 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,
29                                     RoyaltyItems.ExRateFromGBP, RoyaltyItems.RoyaltyQuantity, Book.NoOfPages,
30                                     Book.Size, Book.Cover, Book.Back"
31                        Tables = "RoyaltyItems, Book"
32                        sql = "select {} from {} where RoyaltyItems.RoyaltiesID = {} and
33                               RoyaltyItems.RoyaltyItemsID = {} and RoyaltyItems.ISBN = {} and Book.ISBN =
34                               RoyaltyItems.ISBN".format(Selection, Tables, self.selectedID, count+1,
35                               list(self.ISBNs[count2])[0])
36                        cursor.execute(sql)
37
38                try:
39                    self.SelectionList = list(cursor.fetchone())
40                except:
41                    self.Empty = True
42
43                if self.Empty != True:
44                    self.Currency = self.SelectionList[0]
45                    self.NetSales = float(self.SelectionList[1])
46                    self.Quantity = self.SelectionList[3]
47                    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 ="
71             sql += "{}".format(self.RoyaltyPayment, self.selectedID)
72             cursor.execute(sql)
73             db.commit()
74
75         if self.currentID != 0:
76             self.RoyaltyPayment = "{0:.2f}".format(self.RoyaltyPayment)
77             sql = "update Royalties set RoyaltyPayment = '{}' where RoyaltiesID ="
78             sql += "{}".format(self.RoyaltyPayment, self.selectedID)
79             cursor.execute(sql)
80             db.commit()

```

---

Lines 15 to 19 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 30 defines the sql statement to fetch all the necessary data to conduct the calculations. Then, upto line 66, 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 47-60. The payment for the single set items is then added on to the current total, which starts at £0. Lines 74-78 format the full final payment into a string, and update the payment in the royalty entry with the newly calculated payment.

### Emailing the User their Login Credentials

Lines 165-183 of Module 1, subsection 4.10.1.

---

```

1         self.sender = "pp.loginhelp@gmail.com"
2         self.recipient = [str(self.Email)]
3         self.server = smtplib.SMTP('smtp.gmail.com', 587)
4         self.server.ehlo()
5         self.server.starttls()
6         self.server.ehlo()
7         self.server.login("pp.loginhelp@gmail.com", "DB1061NH31P")
8
9         self.Subject = "Forgotten Login Details"
10        self.message = "From: {}\\r\\nTo: {}\\r\\nSubject: {}\\r\\n\\r\\n".format(self.sender, ", "
11                    .join(self.recipient), self.Subject)
12        self.message += "Your Password is: {}\\nPlease change this upon login for security
13        reasons.".format(self.CurrentPassword)
14
15        self.server.sendmail(self.sender, self.recipient, self.message)
16        self.server.close()
17
18        self.Msg = QMessageBox()
19        self.Msg.setWindowTitle("Email Sent") #email is sent if email is existent in the database,
19        with password details
19        self.Msg.setText("You have been sent an email with the corresponding password details")
19        self.Msg.exec_()

```

---

This extract of Module 1 uses the SMTP server to create and send an email to the user containing their forgotten password. The sender, recipient of the email are defined in lines 1 and 2 of the extract. The system then logs the email in to the server using the email and password of the customer service email used to send to the user. The contents of the email are defined in lines 9-11 of the extract. In line 12 of the extract, the Email is sent to the user.

#### 4.7.2 Self-created Algorithms

##### Calculations upon interaction with the book invoice items

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
2
3      try:
4          if self.Editing == False:
5              self.Quantity = int(self.AddWindow.inputList[2].text())
6              self.Discount = float(self.AddWindow.inputList[3].text()) / 100
7              self.ShippingPrice = float(self.AddWindow.inputList[5].text())
8              self.ISBN = self.AddWindow.inputList[1].text()
9          elif self.Editing == True:
10             self.Quantity = int(self.EditWindow.inputList[2].text())
11             self.Discount = float(self.EditWindow.inputList[3].text()) / 100
12             self.ShippingPrice = float(self.EditWindow.inputList[5].text())
13             self.ISBN = self.EditWindow.inputList[1].text()
14
15             with sqlite3.connect("PP.db") as db: #fetching data from db
16                 cursor = db.cursor()
17                 cursor.execute("select Price from Book where ISBN = {}".format(self.ISBN))
18                 self.Price = list(cursor.fetchone())[0]
19                 db.commit()
20
21             self.BookInvoiceItemPayment = (self.Quantity * self.Price)
22             self.Discount = self.BookInvoiceItemPayment * self.Discount
23             self.BookInvoiceItemPayment -= self.Discount
24             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.

##### Calculations upon interaction with the Royalty items

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.

---

```

1  def RoyaltyItemCalculation(self): #calculating the royalty payment
2
3      try:
4          if self.Editing == False:
5              self.Currency = self.AddWindow.inputList[2].text()
6              self.WholesalePrice = float(self.AddWindow.inputList[4].text())
7              self.Quantity = int(self.AddWindow.inputList[5].text())
8              self.PrintCost = float(self.AddWindow.inputList[6].text())
9          elif self.Editing == True:
10             self.Currency = self.EditWindow.inputList[2].text()
11             self.WholesalePrice = float(self.EditWindow.inputList[4].text())
12             self.Quantity = int(self.EditWindow.inputList[5].text())
13             self.PrintCost = float(self.EditWindow.inputList[6].text())
14
15             self.NetSales = self.WholesalePrice * self.Quantity

```

---

---

```
15     self.RoyaltyItemPayment = self.NetSales - self.PrintCost
16     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 to be able to run my program, the system itself must be installed on to the machine being used. Installation of the packages are not required as installing the system creates an executable file which does not require the packages to have been installed beforehand.

## 4.9 Acknowledgements

The following pieces of code used regular expressions which were found on [www.regexlib.com](http://www.regexlib.com).

I used this regular expression to validate the entry of emails. The user would not be able to enter values that did not follow the expression.

---

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

---

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

I used this regular expression to validate the entry of phone numbers.

---

```
1     self.regexp =
2     """((\+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})|((\+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)

I used this regular expression to validate the entry of postcodes.

---

```
1     self.inputList[5].setValidator(QRegExpValidator(QRegExp("^\w{1,2}\w{0-9}\w{1,1}\w{0-9}\w{1-2}\$")))
```

---

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

I used this regular expression to validate the entry of emails.

---

```

1 self.leNewUsername.setValidator(QRegExpValidator(QRegExp("^\w+\.\w+@\w+\.\w+\.\w+(2,4)$")))
2 self.leOldUsername.setValidator(QRegExpValidator(QRegExp("^\w+\.\w+@\w+\.\w+\.\w+(2,4)$")))

```

---

(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>.

---

```

1 import sys
2 from PyQt4.QtCore import *
3 from PyQt4.QtGui import *
4
5 class dbCalendarWidget(QDialog):
6     def __init__(self):
7         super().__init__()
8
9     def Calendar(self):
10        self.setFixedSize(265, 275)
11        self.setWindowTitle('Calendar')
12        calendar = QCalendarWidget(self)
13        calendar.setGridVisible(True)
14        calendar.clicked[QDate].connect(self.date)
15        date = calendar.selectedDate()
16        self.lblInstruction = QLabel(self)
17        self.lblInstruction.setText("Please select a date")
18        self.lblInstruction.setAlignment(Qt.AlignCenter)
19        self.qle = QLineEdit(self)
20        self.qle.setText(date.toString("dd-MM-yyyy"))
21        self.qle.setFixedSize(85, 20)
22        self.qle.setAlignment(Qt.AlignCenter)
23        self.btnSelect = QPushButton("Select", self)
24        self.btnCancel = QPushButton("Cancel", self)
25
26        self.horizontalTop = QHBoxLayout()
27        self.horizontalTop.addWidget(self.lblInstruction)
28        self.horizontalTop.addStretch(1)
29        self.horizontalTop.addWidget(self.btnCancel)
30        self.horizontalTop.addStretch(1)
31
32        self.horizontalMid1 = QHBoxLayout()
33        self.horizontalMid1.addWidget(calendar)
34        self.horizontalMid1.addStretch(1)
35
36        self.horizontalMid2 = QHBoxLayout()
37        self.horizontalMid2.addWidget(self.qle)
38        self.horizontalMid2.addStretch(1)
39
40        self.horizontalBottom = QHBoxLayout()
41        self.horizontalBottom.addWidget(self.btnCancel)
42        self.horizontalBottom.addStretch(1)
43        self.horizontalBottom.addWidget(self.btnSelect)
44
45        self.vertical = QVBoxLayout()
46        self.vertical.setLayout(self.horizontalTop)
47        self.vertical.setLayout(self.horizontalMid1)
48        self.vertical.setLayout(self.horizontalMid2)
49        self.vertical.setLayout(self.horizontalBottom)
50        self.setLayout(self.vertical)
51
52    def DisplayCalendar(self):
53        self.setModal(True)
54        self.btnSelect.clicked.connect(self.accept)
55        self.btnCancel.clicked.connect(self.reject)
56        self.exec_()
57
58    def date(self, date):
59        self.qle.setText(date.toString("dd-MM-yyyy"))

```

---

I used the following icon for the window icon in the login screen.



I found this image on <http://dryicons.com/icon/shine-icon-set/lock/>.

The following images were used from [www.perfectpublishers.co.uk](http://www.perfectpublishers.co.uk), which is the company of the client.



Figure 4.51: Company Logo



Figure 4.52: Company Icon, from Logo

These were used as the splash screen image and the main window icon respectively.

253

```
1 from PyQt4.QtCore import *
2 from PyQt4.QtGui import *
3 import sqlite3
4 import sys
5 import smtplib
6 import email
7 import time
8 from MainMenu import *
9
10 class dbLogin(QMainWindow):
11     """db for login"""
12
13     def __init__(self):
14         self.initSplashScreen()
15         self.initDetails()
16         if self.details == ("Username", "Password"):
17             self.customer_table()
18             self.book_table()
19             self.pub_invoice_table()
20             self.book_invoice_table()
21             self.book_invoice_items_table()
22             self.royalties_table()
23             self.royalty_items_table()
24             self.MainProgram = MainWindow("Username") #runs main window if Username/Password haven't been changed before
25             self.MainProgram.show()
26         else:
27             super().__init__() #runs login screen if they have been changed
28             self.initLoginScreen()
29
30     def initDetails(self):
31         with sqlite3.connect("dbLogin.db") as db:
32             cursor = db.cursor()
33
34             self.sql = "select name from sqlite_master WHERE type='table' and name='LoginDetails'"
35             cursor.execute(self.sql) #checking whether the login table exists
36             try:
```

## 4.10 Code Listing

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

```
37         self.Exists = list(cursor.fetchone())[0]
38     except:
39         self.Exists = False
40
41     self.sql = "create table if not exists LoginDetails (Username text, Password text)"
42     cursor.execute(self.sql) #creates login table if it doesn't exist
43
44     if self.Exists == False:
45         self.details = "Username", "Password"
46         self.sql = "insert into LoginDetails (Username, Password) values (?, ?)"
47         cursor.execute(self.sql, self.details) #adds default Username and Password
48     else:
49         self.details = True
50
51     def initLoginScreen(self):
52         self.setWindowTitle("Login")
53         self.setWindowIcon(QIcon("LoginIcon.png"))
54         self.setFixedSize(300, 190)
55         self.lblLogin = QLabel("Please Log In", self)
56         self.lblLogin.setFont(QFont("Calibri", 14))
57         self.lblLogin.setAlignment(Qt.AlignHCenter)
58         self.btnLogin = QPushButton("Login", self)
59         self.btnLogin.setFixedSize(self.btnLogin.sizeHint())
60         self.lblUsername = QLabel("Username:", self)
61         self.lblUsername.setFont(QFont("Calibri", 10))
62         self.lblPassword = QLabel("Password:", self)
63         self.lblPassword.setFont(QFont("Calibri", 10))
64         self.leUsername = QLineEdit(self)
65         self.leUsername.setPlaceholderText("Username")
66         self.lePassword = QLineEdit(self)
67         self.lePassword.setEchoMode(self.lePassword.Password)
68         self.lePassword.setPlaceholderText("Password")
69         self.lblForgot = QLabel("Help/Forgotten Password?", self)
70         self.Underline = QFont("Calibri", 10)
71         self.Underline.setUnderline(True)
72         self.lblForgot.setFont(self.Underline)
73         self.lblForgot.setAlignment(Qt.AlignHCenter)
74         self.lblForgot.mousePressEvent = self.getEmail
75         self.lblVertical = QVBoxLayout()
76         self.leVertical = QVBoxLayout()
77         self.horizontalLogin = QHBoxLayout()
78         self.lblVertical.addWidget(self.lblUsername)
```

```
79         self.leVertical.addWidget(self.leUsername)
80         self.lblVertical.addWidget(self.lblPassword)
81         self.leVertical.addWidget(self.lePassword)
82         self.horizontalLogin.addStretch(1)
83         self.horizontalLogin.addWidget(self.btnLogin)
84         self.horizontalLogin.addStretch(1)
85         self.vertical = QVBoxLayout()
86         self.vertical.addWidget(self.lblLogin)
87         self.horizontalEntry = QHBoxLayout()
88         self.horizontalEntry.addLayout(self.lblVertical)
89         self.horizontalEntry.addLayout(self.leVertical)
90         self.vertical.addLayout(self.horizontalEntry)
91         self.vertical.addWidget(self.lblForgot)
92         self.vertical.addLayout(self.horizontalLogin)
93         self.vertical.addStretch(1)
94         self.horizontal = QHBoxLayout()
95         self.horizontal.addStretch(1)
96         self.horizontal.addLayout(self.vertical)
97         self.horizontal.addStretch(1)
98         self.CentralWidget = QWidget()
99         self.CentralWidget.setLayout(self.horizontal)
100        self.setCentralWidget(self.CentralWidget)
101        self.btnLogin.clicked.connect(self.Login)
102        self.lblInvalid = None
103
104    def Login(self):
105        with sqlite3.connect("dbLogin.db") as db:
106            cursor = db.cursor()
107            cursor.execute("select Username from LoginDetails")
108            self.Username = list(cursor.fetchall()) #fetches original username and password
109            cursor.execute("select Password from LoginDetails")
110            self.Password = list(cursor.fetchall())
111            self.Valid = False
112
113        for count in range(0, len(self.Username)):
114            if self.leUsername.text().lower() == list(self.Username[count])[0].lower():
115                if self.lePassword.text() == list(self.Password[count])[0]:
116                    self.Valid = True
117                    self.customer_table() #checking all tables to see if they're existent
118                    self.book_table()
119                    self.pub_invoice_table()
120                    self.book_invoice_table()
```

```
121             self.book_invoice_items_table()
122             self.royalties_table()
123             self.royalty_items_table()
124             self.hide() #Username and password match, so the user is logged in
125             self.MainProgram = MainWindow(list(self.Username)[count])[0]
126             self.MainProgram.show()
127             break
128         else:
129             self.Valid = False
130
131     if self.Valid == False:
132         if self.lblInvalid == None: #Username and password do not match, user is rejected
133             self.lblInvalid = QLabel("Invalid Username or Password - Please try again.", self)
134             self.lblInvalid.setWordWrap(True)
135             self.lblInvalid.setAlignment(Qt.AlignHCenter)
136             self.horizontalInvalid = QHBoxLayout()
137             self.horizontalInvalid.addStretch(1)
138             self.horizontalInvalid.addWidget(self.lblInvalid)
139             self.horizontalInvalid.addStretch(1)
140             self.vertical.addLayout(self.horizontalInvalid)
141     else:
142         self.lblInvalid.show()
143
144 def getEmail(self, QMouseEvent):
145     with sqlite3.connect("dbLogin.db") as db:
146         cursor = db.cursor()
147         cursor.execute("select Username from LoginDetails")
148         self.Username = list(cursor.fetchone())[0]
149         cursor.execute("select Password from LoginDetails")
150         self.Password = list(cursor.fetchone())[0]
151
152     if self.Username == "Username" and self.Password == "Password":
153         self.Msg = QMessageBox()
154         self.Msg.setWindowTitle("First Time")
155         self.Msg.setText("This is your first time using this application.\n Your Username is 'Username' and your
156                         Password is 'Password'.\n Please change these once logged in.")
157         self.Msg.exec_() #default username and password is shown to the user
158     else:
159         self.Email, ok = QInputDialog.getText(self, 'Forgotten Password', 'Enter your Email:')
160         self.Email = self.Email.lower() #email is received from the user
161         if self.Email == self.Username:
162             with sqlite3.connect("dbLogin.db") as db:
```

```
162         cursor = db.cursor()
163         cursor.execute("select Password from LoginDetails")
164         self.CurrentPassword = list(cursor.fetchone())[0] #gets password from database
165         self.sender = "pp.loginhelp@gmail.com"
166         self.recipient = [str(self.Email)]
167         self.server = smtplib.SMTP('smtp.gmail.com', 587)
168         self.server.ehlo()
169         self.server.starttls()
170         self.server.ehlo()
171         self.server.login("pp.loginhelp@gmail.com", "DB1061NH31P")
172
173         self.Subject = "Forgotten Login Details"
174         self.message = "From: {}\\r\\nTo: {}\\r\\nSubject: {}\\r\\n\\r\\n".format(self.sender, "", ".join(self.recipient),
175                                         self.Subject)
176         self.message += "Your Password is: {}\\nPlease change this upon login for security
177                                         reasons.".format(self.CurrentPassword)
178
179         self.server.sendmail(self.sender, self.recipient, self.message)
180         self.server.close()
181
182         self.Msg = QMessageBox()
183         self.Msg.setWindowTitle("Email Sent") #email is sent if email is existent in the database, with password
184         details
185         self.Msg.setText("You have been sent an email with the corresponding password details")
186         self.Msg.exec_()
187
188     elif self.Email != self.Username and ok == True:
189         self.Msg = QMessageBox()
190         self.Msg.setWindowTitle("No Match found") #email is not found, so email is not sent
191         self.Msg.setText("No matching Email was found")
192         self.Msg.exec_()
193
194     def keyReleaseEvent(self, QKeyEvent):
195         if self.lblInvalid != None:
196             self.lblInvalid.hide()
197
198     def create_table(self):
199         with sqlite3.connect("PP.db") as db:
200             cursor = db.cursor()
201             cursor.execute("PRAGMA foreign_keys = ON")
202             self.Exists = True
203             self.FindTable = "select name from sqlite_master WHERE type='table' and name='{}'".format(self.TableName)
```

```
201     cursor.execute(self.FindTable) #checks if the tables exist
202     try:
203         self.Exists = list(cursor.fetchone())
204     except:
205         self.Exists = False
206
207     if self.Exists == False: #creates tables for the database if they don't already exist
208         cursor.execute(self.sql)
209         db.commit()
210
211     def customer_table(self):
212         self.sql = """create table Customer
213             (AuthorID integer,
214              FirstName text,
215              LastName text,
216              Email text,
217              PhoneNumber text,
218              Address text,
219              Postcode text,
220              primary key(AuthorID))"""
221         self.TableName = "Customer"
222         self.create_table()
223
224     def book_table(self):
225         self.sql = """create table Book
226             (ISBN text,
227              AuthorID integer,
228              BookTitle text,
229              NoOfPages integer,
230              Size text,
231              Back text,
232              Cover text,
233              Paper text,
234              Font text,
235              FontSize real,
236              DatePublished date,
237              Price real,
238              primary key(ISBN),
239              foreign key(AuthorID) references Customer(AuthorID))"""
240         self.TableName = "Book"
241         self.create_table()
242
```

```
243     def pub_invoice_table(self):
244         self.sql = """create table PubInvoice
245             (PubInvoiceID integer,
246              ISBN text,
247              AuthorID integer,
248              PubInvoiceDate date,
249              PubInvoiceService text,
250              PubInvoicePayment real,
251              primary key(PubInvoiceID),
252              foreign key(AuthorID) references Customer(AuthorID),
253              foreign key(ISBN) references Book(ISBN))"""
254         self.TableName = "PubInvoice"
255         self.create_table()
256
257     def book_invoice_table(self):
258         self.sql = """create table BookInvoice
259             (BookInvoiceID integer,
260              AuthorID integer,
261              BookInvoiceDate date,
262              BookInvoicePayment real,
263              primary key(BookInvoiceID),
264              foreign key(AuthorID) references Customer(AuthorID))"""
265         self.TableName = "BookInvoice"
266         self.create_table()
267
268     def book_invoice_items_table(self):
269         self.sql = """create table BookInvoiceItems
270             (BookInvoiceItemsID integer,
271              BookInvoiceID integer,
272              ISBN text,
273              BookInvoiceQuantity integer,
274              BookInvoiceDiscount real,
275              ShippingType text,
276              ShippingPrice real,
277              primary key(BookInvoiceItemsID),
278              foreign key(BookInvoiceID) references BookInvoice(BookInvoiceID),
279              foreign key(ISBN) references Book(ISBN))"""
280         self.TableName = "BookInvoiceItems"
281         self.create_table()
282
283     def royalties_table(self):
284         self.sql = """create table Royalties
```

```
285             (RoyaltiesID integer,
286              AuthorID integer,
287              RoyaltiesDate date,
288              RoyaltyPayment real,
289              primary key(RoyaltiesID),
290              foreign key(AuthorID) references Customer(AuthorID))"""
291         self.TableName = "Royalties"
292         self.create_table()
293
294     def royalty_items_table(self):
295         self.sql = """create table RoyaltyItems
296                     (RoyaltyItemsID integer,
297                      RoyaltiesID integer,
298                      ISBN text,
299                      Currency text,
300                      RoyaltyDiscount real,
301                      WholesalePrice real,
302                      RoyaltyQuantity integer,
303                      NetSales real,
304                      PrintCost real,
305                      ExcRateFromGBP real,
306                      primary key(RoyaltyItemsID),
307                      foreign key(RoyaltiesID) references Royalties(RoyaltiesID),
308                      foreign key(ISBN) references Book(ISBN))"""
309         self.TableName = "RoyaltyItems"
310         self.create_table()
311
312     def initSplashScreen(self):
313         self.pixmap = QPixmap("PerfectPublishersLtd.png")
314         self.Splashscreen = QSplashScreen(self.pixmap, Qt.WindowStaysOnTopHint)
315         self.Splashscreen.setMask(self.pixmap.mask())
316         self.Splashscreen.show()
317         time.sleep(2)
318         self.Splashscreen.finish(self.Splashscreen)
319
320
321     def main():
322         app = QApplication(sys.argv)
323         launcher = dbLogin()
324         try:
325             launcher.raise_()
326             launcher.show()
```

```
327     except RuntimeError:
328         pass
329     app.exec_()
330
331
332
333
334 if __name__ == "__main__":
335     main()
```

---

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

---

```
1  from PyQt4.QtCore import *
2  from PyQt4.QtGui import *
3  import sqlite3
4  import sys
5  import subprocess
6
7  from MenuBar import *
8  from initMainMenuBar import *
9  from AddEntryWindow import *
10 from TableWidget import *
11 from ConfirmationDialog import *
12 from ViewWindow import *
13 from AddItemWindow import *
14 from ViewRoyaltiesAndInvoices import *
15 from UpdateEntryWindow import *
16 from CalendarWidget import *
17 from Items import *
18 from SearchDatabase import *
19 from LoginDB import *
20 from ChangePassword import *
21 from UsernameOrPassword import *
22 from ChangeUsername import *
23 from SearchResults import *
24
25 class MainWindow(QMainWindow):
26     """main window"""
27
```

```
28     def __init__(self, Username):
29         super().__init__()
30         self.Username = Username
31         self.setWindowTitle("Main Menu")
32         self.setWindowIcon(QIcon("PPIcon.png"))
33         self.setFixedSize(735,400)
34         self.MenuBar = dbMenuBar()
35         self.setMenuBar(self.MenuBar)
36
37         self.TableWidget = dbTableWidget()
38         self.TableWidget.sql = "select * from Customer"
39         self.TableWidget.initTable()
40         self.MainMenuButtons = initMainMenuButtons()
41         self.MainMenuButtons.vertical.addLayout(self.MainMenuButtons.horizontalTop)
42         self.MainMenuButtons.vertical.addWidget(self.TableWidget)
43         self.MainMenuButtons.vertical.addLayout(self.MainMenuButtons.horizontalBottom)
44         self.CurrentTable = "Customer"
45         self.MainMenuButtons.setLayout(self.MainMenuButtons.vertical)
46
47         self.StackedLayout = QStackedLayout()
48
49         self.centralWidget = QWidget()
50         self.centralWidget.setLayout(self.StackedLayout)
51         self.setCentralWidget(self.centralWidget)
52
53         self.StackedLayout.addWidget(self.MainMenuButtons)
54
55         self.ViewWindow = dbViewWindow() #instantiating view window
56         self.ViewWindow.View()
57         self.ViewWindow.vertical.addLayout(self.ViewWindow.horizontalTop)
58         self.ViewWindow.table = dbTableWidget()
59         self.ViewWindow.vertical.addWidget(self.ViewWindow.table)
60         self.ViewWindow.vertical.addLayout(self.ViewWindow.horizontalBottom)
61         self.ViewWindow.setLayout(self.ViewWindow.vertical)
62
63         self.StackedLayout.addWidget(self.ViewWindow)
64
65         self.SearchResults = initSearchResultsMenu()
66         self.StackedLayout.addWidget(self.SearchResults)
67         self.SearchTable = dbTableWidget()
68         self.SearchResults.vertical.addWidget(self.SearchTable)
69
```

```
70 #connections
71 self.MainMenuButtons.btnAddEntry.clicked.connect(self.AddEntry)
72 self.MainMenuButtons.btnRemoveEntry.clicked.connect(self.RemoveEntry)
73 self.MainMenuButtons.btnView.clicked.connect(self.ViewCustomer)
74 self.MainMenuButtons.btnUpdateEntry.clicked.connect(self.UpdateCustomerEntry)
75 self.MainMenuButtons.btnQuickSearch.clicked.connect(self.QuickSearch)
76 self.MainMenuButtons.btnSearchdb.clicked.connect(self.Search)
77 self.MainMenuButtons.btnLogOut.clicked.connect(self.LogOut)
78 self.MainMenuButtons.btnChangePassword.clicked.connect(self.ChangeUsernameOrPassword)
79 self.MenuBar.search_database.triggered.connect(self.Search)
80 self.MenuBar.add_entry.triggered.connect(self.AddEntry)
81 self.MenuBar.remove_entry.triggered.connect(self.RemoveEntry)
82 self.MenuBar.update_entry.triggered.connect(self.UpdateCustomerEntry)
83 self.MenuBar.log_out.triggered.connect(self.LogOut)
84 self.MenuBar.change_password.triggered.connect(self.ChangeUsernameOrPassword)
85 self.ViewWindow.btnAddBook.clicked.connect(self.AddItem)
86 self.ViewWindow.btnUpdateBook.clicked.connect(self.UpdateEntry)
88 self.ViewWindow.btnDeleteBook.clicked.connect(self.RemoveFromDB)
89 self.ViewWindow.btnViewPubInvoice.clicked.connect(self.ViewPubInvoice)
90 self.ViewWindow.btnViewBookInvoices.clicked.connect(self.ViewBookInvoice)
91 self.ViewWindow.btnViewRoyalties.clicked.connect(self.ViewRoyalties)
92 self.SearchResults.btnBack.clicked.connect(self.Back)
93
94
95 def AddEntry(self): #adding customer entry
96     self.AddEntryWindow = dbAddEntryWindow()
97     self.AddEntryWindow.Added = False
98     self.AddEntryWindow.initAddEntryWindow()
99     self.RefreshTables()
100
101
102
103 def AddItem(self): #initialising an add window for getting inputs for other entries
104     self.AddWindow = dbAddItemWindow()
105     self.AddWindow.setFixedSize(360,200)
106     self.AddWindow.AddType = self.CurrentTable
107     self.AddWindow.AnswerButtons()
108     self.AddWindow.Editing = False
109
110     if self.CurrentTable == "Book":
111         self.AddWindow.sql = "select * from Book"
```

```
112
113     elif self.CurrentTable == "PubInvoice":
114         self.AddWindow.setFixedSize(400,150)
115         self.AddWindow.sql = "select ISBN, AuthorID, PubInvoiceDate, PubInvoiceService, PubInvoicePayment from
116             PubInvoice"
117         self.AddWindow.selectedISBN = self.SelectedISBN
118
119     elif self.CurrentTable == "BookInvoice":
120         self.AddWindow.sql = "select AuthorID, BookInvoiceDate from BookInvoice"
121         self.AddWindow.setFixedSize(350,100)
122
123     elif self.CurrentTable == "Royalties":
124         self.AddWindow.sql = "select AuthorID, RoyaltiesDate from Royalties"
125         self.AddWindow.setFixedSize(350,100)
126
127     elif self.CurrentTable == "BookInvoiceItems":
128         self.AddWindow.sql = "select BookInvoiceID, ISBN, BookInvoiceQuantity, BookInvoiceDiscount, ShippingType,
129             ShippingPrice from BookInvoiceItems"
130         self.AddWindow.setFixedSize(450, 150)
131         self.AddWindow.selectedISBN = self.SelectedISBN
132         self.Editing = False
133         self.AddWindow.btnAdd.clicked.connect(self.BookInvoiceItemCalculation)
134
135     elif self.CurrentTable == "RoyaltyItems":
136         self.AddWindow.sql = "select RoyaltiesID, ISBN, Currency, RoyaltyDiscount, WholesalePrice, RoyaltyQuantity,
137             PrintCost, ExcRateFromGBP from RoyaltyItems"
138         self.AddWindow.setFixedSize(450, 250)
139         self.AddWindow.selectedISBN = self.SelectedISBN
140         self.Editing = False
141         self.AddWindow.btnAdd.clicked.connect(self.RoyaltyItemCalculation)
142
143         self.AddWindow.selectedID = self.SelectedID
144         self.AddWindow.CalendarWidget = dbCalendarWidget()
145         self.AddWindow.CalendarWidget.Calendar()
146
147         self.AddWindow.initAddItemWindow()
148     try:
149         if self.AddWindow.Valid == True: #inputs must pass validation before being added
150             self.AddToDB()
151     except AttributeError:
152         pass #exception of window being closed
153         self.RefreshTables()
```

```
151
152     def RecalculateItems(self): #recalculates after changes
153         if self.CurrentTable == "BookInvoiceItems":
154             self.BookInvoiceItemsWindow.CalculateBookInvoiceItems()
155             self.CurrentTable = "BookInvoice"
156             self.RefreshTables()
157             self.CurrentTable = "BookInvoiceItems"
158             self.RefreshTables()
159     elif self.CurrentTable == "RoyaltyItems" or self.BookEdited == True:
160         self.RoyaltyItemsWindow.CalculateRoyaltyItems()
161         self.CurrentTable = "Royalties"
162         self.RefreshTables()
163         self.CurrentTable = "RoyaltyItems"
164         self.RefreshTables()
165
166
167     def AddToDB(self): #Adding Entries to database
168         self.input_data = []
169         if self.CurrentTable == "Book":
170             self.TableValues = "Book (ISBN, AuthorID, BookTitle, NoOfPages, Size, Back, Cover, Paper, Font, FontSize,
171             DatePublished, Price)"
172             self.Placeholders = "(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)"
173             self.NoOfEntries = 12
174
175         elif self.CurrentTable == "PubInvoice":
176             self.TableValues = "PubInvoice (ISBN, AuthorID, PubInvoiceDate, PubInvoiceService, PubInvoicePayment)"
177             self.Placeholders = "(?, ?, ?, ?, ?)"
178             self.NoOfEntries = 5
179
180         elif self.CurrentTable == "BookInvoice":
181             self.TableValues = "BookInvoice (AuthorID, BookInvoiceDate)"
182             self.Placeholders = "(?, ?)"
183             self.NoOfEntries = 2
184
185         elif self.CurrentTable == "Royalties":
186             self.TableValues = "Royalties (AuthorID, RoyaltiesDate)"
187             self.Placeholders = "(?, ?)"
188             self.NoOfEntries = 2
189
190         elif self.CurrentTable == "BookInvoiceItems":
191             self.TableValues = "BookInvoiceItems (BookInvoiceID, ISBN, BookInvoiceQuantity, BookInvoiceDiscount,
192             ShippingType, ShippingPrice)"
```

```
191         self.Placeholders = "(?, ?, ?, ?, ?, ?, ?)"
192         self.NoOfEntries = 6
193
194     elif self.CurrentTable == "RoyaltyItems":
195         self.TableValues = "RoyaltyItems (RoyaltiesID, ISBN, Currency, RoyaltyDiscount, WholesalePrice,
196                         RoyaltyQuantity, PrintCost, ExcRateFromGBP, NetSales)"
197         self.Placeholders = "(?, ?, ?, ?, ?, ?, ?, ?, ?)"
198         self.NoOfEntries = 9
199
200     for count in range(0, self.NoOfEntries):
201         try: #putting the input data in a list
202             self.input_data.append(str(self.AddWindow.inputList[count].currentText()))
203         except:
204             if count == 8 and self.CurrentTable == "RoyaltyItems":
205                 self.input_data.append(self.NetSales) #Net sales are calculated, as opposed to being entered
206             else:
207                 self.input_data.append(self.AddWindow.inputList[count].text())
208
208     with sqlite3.connect("PP.db") as db:
209         cursor = db.cursor()
210         cursor.execute("PRAGMA foreign_keys = ON")
211         self.sql = "insert into {} values {}".format(self.TableValues, self.Placeholders)
212         cursor.execute(self.sql, self.input_data)
213         db.commit()
214
215     self.RefreshTables()
216
217     try:
218         self.RecalculateItems()
219     except:
220         pass
221
222 def RemoveEntry(self): #removing a customer
223     self.SelectedRow = self.TableWidget.currentRow()
224     self.ConfirmDialog = dbConfirmationDialog()
225     self.ConfirmDialog.Username = self.Username
226     self.ConfirmDialog.SelectedAuthorID = QTableWidgetItem(self.TableWidget.item(self.SelectedRow, 0)).text() #getting
227     AuthorID of a row
228     if self.ConfirmDialog.SelectedAuthorID != "":
229         self.Firstname = QTableWidgetItem(self.TableWidget.item(self.SelectedRow, 1)).text()
230         self.Lastname = QTableWidgetItem(self.TableWidget.item(self.SelectedRow, 2)).text()
231         self.ConfirmDialog.Name = "{} {}".format(self.Firstname, self.Lastname)
```

```
231     self.ConfirmDialog.Msg = "Are you sure you want to delete {} and all records about  
232         them?".format(self.ConfirmDialog.Name)  
233     self.ConfirmDialog.ConfirmedMsg = "{}'s records have been successfully erased.".format(self.ConfirmDialog.Name)  
234     self.ConfirmDialog.VerifyDlg()  
235  
236     if self.ConfirmDialog.ConfirmedDialog.Accepted == True:  
237         with sqlite3.connect("PP.db") as db:  
238             cursor = db.cursor()  
239             cursor.execute("PRAGMA foreign_keys = ON")  
240             sql = "select ISBN from Book where AuthorID = {}".format(self.ConfirmDialog.SelectedAuthorID)  
241             cursor.execute(sql)  
242             self.ISBNDeletion = list(cursor.fetchall())  
243             for count in range(0, len(self.ISBNDeletion)): #removes all customer details, starting with foreign  
244                 keys for referential integrity  
245                 sql = "delete from BookInvoiceItems where ISBN =  
246                     {}".format(list(list(self.ISBNDeletion)[count])[0])  
247                 cursor.execute(sql)  
248                 sql = "delete from RoyaltyItems where ISBN = {}".format(list(list(self.ISBNDeletion)[count])[0])  
249                 cursor.execute(sql)  
250                 sql = "delete from PubInvoice where AuthorID = {}".format(self.ConfirmDialog.SelectedAuthorID)  
251                 cursor.execute(sql)  
252                 sql = "delete from BookInvoice where AuthorID = {}".format(self.ConfirmDialog.SelectedAuthorID)  
253                 cursor.execute(sql)  
254                 sql = "delete from Royalties where AuthorID = {}".format(self.ConfirmDialog.SelectedAuthorID)  
255                 cursor.execute(sql)  
256                 sql = "delete from Book where AuthorID = {}".format(self.ConfirmDialog.SelectedAuthorID)  
257                 cursor.execute(sql)  
258                 db.commit()  
259                 self.RefreshTables()  
260  
261  
262     def RemoveFromDB(self): #removing other entries  
263         #getting primary key of the row  
264         self.ConfirmDialog = dbConfirmationDialog()  
265         self.ConfirmDialog.Username = self.Username  
266         self.ConfirmDialog.DeleteMsg = self.CurrentTable  
267         self.SelectedAuthorID = self.SelectedID  
268  
269         if self.CurrentTable == "Book":
```

```
270     self.ForeignKeyMsg = "Royalties and Invoices"
271     self.SelectedRow = self.ViewWindow.table.currentRow()
272     self.SelectedID = QTableWidgetItem(self.ViewWindow.table.item(self.SelectedRow, 0)).text()
273     self.SelectedIDName = "ISBN"
274     self.ConfirmDialog.Msg = "Are you sure you want to delete this book?"
275     self.ConfirmDialog.ConfirmedMsg = "Book was successfully deleted"
276
277 elif self.CurrentTable == "PubInvoice":
278     self.SelectedRow = self.PubInvoiceWindow.table.currentRow()
279     self.OldID = self.SelectedID
280     self.SelectedID = QTableWidgetItem(self.PubInvoiceWindow.table.item(self.SelectedRow, 0)).text()
281     self.SelectedIDName = "PubInvoiceID"
282     self.ConfirmDialog.Msg = "Are you sure you want to delete this Invoice?"
283     self.ConfirmDialog.ConfirmedMsg = "Invoice was successfully deleted"
284
285 elif self.CurrentTable == "BookInvoice":
286     self.ForeignKeyMsg = "Book Invoice Items"
287     self.SelectedRow = self.BookInvoiceWindow.table.currentRow()
288     self.OldID = self.SelectedID
289     self.SelectedID = QTableWidgetItem(self.BookInvoiceWindow.table.item(self.SelectedRow, 0)).text()
290     self.SelectedIDName = "BookInvoiceID"
291     self.ConfirmDialog.Msg = "Are you sure you want to delete this Invoice?"
292     self.ConfirmDialog.ConfirmedMsg = "Invoice was successfully deleted"
293
294 elif self.CurrentTable == "Royalties":
295     self.ForeignKeyMsg = "Royalty Items"
296     self.SelectedRow = self.RoyaltiesWindow.table.currentRow()
297     self.OldID = self.SelectedID
298     self.SelectedID = QTableWidgetItem(self.RoyaltiesWindow.table.item(self.SelectedRow, 0)).text()
299     self.SelectedIDName = "RoyaltiesID"
300     self.ConfirmDialog.Msg = "Are you sure you want to delete this Entry?"
301     self.ConfirmDialog.ConfirmedMsg = "Entry was successfully deleted"
302
303 elif self.CurrentTable == "BookInvoiceItems":
304     self.SelectedRow = self.BookInvoiceItemsWindow.table.currentRow()
305     self.OldID = self.SelectedID
306     self.SelectedID = QTableWidgetItem(self.BookInvoiceItemsWindow.table.item(self.SelectedRow, 0)).text()
307     self.SelectedIDName = "BookInvoiceItemsID"
308     self.ConfirmDialog.Msg = "Are you sure you want to delete this Item?"
309     self.ConfirmDialog.ConfirmedMsg = "Item was successfully deleted"
310
311 elif self.CurrentTable == "RoyaltyItems":
```

```

312     self.SelectedRow = self.RoyaltyItemsWindow.table.currentRow()
313     self.OldID = self.SelectedID
314     self.SelectedID = QTableWidgetItem(self.RoyaltyItemsWindow.table.item(self.SelectedRow, 0)).text()
315     self.SelectedIDName = "RoyaltyItemsID"
316     self.ConfirmDialog.Msg = "Are you sure you want to delete this Item?"
317     self.ConfirmDialog.ConfirmedMsg = "Item was successfully deleted"
318     self.ConfirmDialog.Prevention = True #deletion may be prevented if integrity error occurs
319     if self.SelectedRow != -1:
320         self.ConfirmDialog.VerifyDlg() #verification
321     try:
322         if self.ConfirmDialog.ConfirmedDialog.Accepted == True:
323             with sqlite3.connect("PP.db") as db:
324                 cursor = db.cursor()
325                 cursor.execute("PRAGMA foreign_keys = ON")
326                 sql = "delete from {} where {} = '{}'".format(self.CurrentTable, self.SelectedIDName,
327                                                 self.SelectedID)
328                 cursor.execute(sql)
329                 db.commit()
330                 self.ConfirmDialog.ConfirmedDialog.Confirmed()
331                 self.SelectedID = self.SelectedAuthorID
332                 self.RefreshTables()
333             except sqlite3.IntegrityError:
334                 self.Msg = QMessageBox()
335                 self.Msg.setWindowTitle("Error")
336                 self.Msg.setText("You must delete all the {} first.".format(self.ForeignKeyMsg))
337                 self.Msg.exec_() #informs user that selected entry cannot be deleted and what must be done for it to be
338                 deleted
339             if self.CurrentTable in ["Royalties", "BookInvoice", "PubInvoice", "BookInvoiceItems", "RoyaltyItems"]:
340                 self.SelectedID = self.OldID
341             try:
342                 self.RecalculateItems()
343             except:
344                 pass
345
346
347     def ViewCustomer(self): #displaying customer data
348         self.SelectedRow = self.TableWidget.currentRow()
349         self.SelectedID = QTableWidgetItem(self.TableWidget.item(self.SelectedRow, 0)).text()
350         self.Firstname = QTableWidgetItem(self.TableWidget.item(self.SelectedRow, 1)).text()
351

```

```
352         self.Lastname = QTableWidgetItem(self.TableWidget.item(self.SelectedRow, 2)).text()
353
354     if self.SelectedID != "":
355         self.SelectedAuthorID = self.SelectedID
356         self.CurrentTable = "Book"
357         self.RefreshTables()
358         self.StackedLayout.setCurrentIndex(1)
359         self.MenuBar.setVisible(False)
360
361
362
363 def ViewPubInvoice(self): #initialising the view publishing invoice window
364     self.CurrentTable = "PubInvoice"
365     self.ViewWindow.SelectedRow = self.ViewWindow.table.currentRow()
366     self.SelectedISBN = QTableWidgetItem(self.ViewWindow.table.item(self.ViewWindow.SelectedRow, 0)).text()
367     self.SelectedID = QTableWidgetItem(self.TableWidget.item(self.TableWidget.currentRow(), 0)).text()
368     self.SelectedAuthorID = self.SelectedID
369     if self.SelectedISBN != "":
370         self.PubInvoiceWindow = dbRoyaltiesAndInvoices()
371         self.PubInvoiceWindow.PubInvoiceButtons()
372         self.PubInvoiceWindow.btnAddPubInvoice.clicked.connect(self.AddItem)
373         self.PubInvoiceWindow.btnUpdatePubInvoice.clicked.connect(self.UpdateEntry)
374         self.PubInvoiceWindow.btnDeleteEntry.clicked.connect(self.RemoveFromDB)
375         self.PubInvoiceWindow.table = dbTableWidget()
376         self.RefreshTables()
377         self.PubInvoiceWindow.table.setFixedSize(620, 150)
378         self.PubInvoiceWindow.PubInvoice()
379     self.CurrentTable = "Book"
380
381
382
383 def ViewBookInvoice(self): #initialising the view book invoice window
384     self.CurrentTable = "BookInvoice"
385     self.ViewWindow.SelectedRow = self.ViewWindow.table.currentRow()
386     self.SelectedISBN = QTableWidgetItem(self.ViewWindow.table.item(self.ViewWindow.SelectedRow, 0)).text()
387     self.SelectedAuthorID = QTableWidgetItem(self.TableWidget.item(self.TableWidget.currentRow(), 0)).text()
388
389     if self.SelectedISBN != "":
390         self.BookInvoiceWindow = dbRoyaltiesAndInvoices()
391         self.BookInvoiceWindow.BookInvoiceButtons()
392         self.BookInvoiceWindow.btnViewBookInvoiceItems.clicked.connect(self.ViewBookInvoiceItems)
393         self.BookInvoiceWindow.btnAddBookInvoice.clicked.connect(self.AddItem)
```

```
394         self.BookInvoiceWindow.btnUpdateBookInvoice.clicked.connect(self.UpdateEntry)
395         self.BookInvoiceWindow.btnDeleteEntry.clicked.connect(self.RemoveFromDB)
396         self.BookInvoiceWindow.table = dbTableWidget()
397         self.RefreshTables()
398         self.BookInvoiceWindow.table.setFixedSize(620, 150)
399         self.BookInvoiceWindow.BookInvoice()
400         self.CurrentTable = "Book"
401
402
403
404     def ViewBookInvoiceItems(self): #initialising the view book invoice items window
405         self.CurrentTable = "BookInvoiceItems"
406         self.BookInvoiceWindow.SelectedRow = self.BookInvoiceWindow.table.currentRow()
407         self.holder = self.SelectedAuthorID
408         self.SelectedAuthorID = self.SelectedID
409         self.BookInvoiceWindow.selectedISBN = self.SelectedISBN
410         self.SelectedID = QTableWidgetItem(self.BookInvoiceWindow.table.item(self.BookInvoiceWindow.SelectedRow, 0)).text()
411
412         if self.SelectedID != "":
413             self.BookInvoiceItemsWindow = dbItems()
414             self.BookInvoiceItemsWindow.selectedID = self.SelectedID
415             self.BookInvoiceItemsWindow.selectedISBN = self.SelectedISBN
416             self.BookInvoiceItemsWindow.BookInvoiceItemsButtons()
417             self.BookInvoiceItemsWindow.btnCalculate.clicked.connect(self.AddItem)
418             self.BookInvoiceItemsWindow.btnUpdateBookInvoiceItems.clicked.connect(self.UpdateEntry)
419             self.BookInvoiceItemsWindow.btnDeleteEntry.clicked.connect(self.RemoveFromDB)
420             self.BookInvoiceItemsWindow.table = dbTableWidget()
421             self.RefreshTables()
422             self.BookInvoiceItemsWindow.table.setFixedSize(620, 150)
423             self.BookInvoiceItemsWindow.BookInvoiceItems()
424             self.SelectedID = self.SelectedAuthorID
425             self.SelectedAuthorID = self.holder
426             self.CurrentTable = "BookInvoice"
427             self.RefreshTables()
428
429
430
431     def ViewRoyalties(self): #initialising the view royalties window
432         self.CurrentTable = "Royalties"
433         self.ViewWindow.SelectedRow = self.ViewWindow.table.currentRow()
434         self.SelectedISBN = QTableWidgetItem(self.ViewWindow.table.item(self.ViewWindow.SelectedRow, 0)).text()
435         self.SelectedAuthorID = QTableWidgetItem(self.TableWidget.item(self.TableWidget.currentRow(), 0)).text()
```

```
436     if self.SelectedISBN != "":
437         self.RoyaltiesWindow = dbRoyaltiesAndInvoices()
438         self.RoyaltiesWindow.RoyaltiesButtons()
439         self.RoyaltiesWindow.btnViewRoyaltyItems.clicked.connect(self.ViewRoyaltyItems)
440         self.RoyaltiesWindow.btnAddRoyalties.clicked.connect(self.AddItem)
441         self.RoyaltiesWindow.btnUpdateRoyalties.clicked.connect(self.UpdateEntry)
442         self.RoyaltiesWindow.btnDeleteEntry.clicked.connect(self.RemoveFromDB)
443         self.RoyaltiesWindow.table = dbTableWidget()
444         self.RefreshTables()
445         self.RoyaltiesWindow.table.setFixedSize(620, 150)
446         self.RoyaltiesWindow.Royalties()
447     self.CurrentTable = "Book"
448
449 def ViewRoyaltyItems(self): #initialising the view royalty items window
450     self.CurrentTable = "RoyaltyItems"
451     self.RoyaltiesWindow.SelectedRow = self.RoyaltiesWindow.table.currentRow()
452     self.holder = self.SelectedAuthorID
453     self.SelectedAuthorID = self.SelectedID
454     self.RoyaltiesWindow.selectedISBN = self.SelectedISBN
455     self.SelectedID = QTableWidgetItem(self.RoyaltiesWindow.table.item(self.RoyaltiesWindow.SelectedRow, 0)).text()
456
457     if self.SelectedID != "":
458         self.RoyaltyItemsWindow = dbItems()
459         self.RoyaltyItemsWindow.selectedID = self.SelectedID
460         self.RoyaltyItemsWindow.selectedISBN = self.SelectedISBN
461         self.RoyaltyItemsWindow.RoyaltyItemsButtons()
462         self.RoyaltyItemsWindow.btnCalculate.clicked.connect(self.AddItem)
463         self.RoyaltyItemsWindow.btnUpdateRoyaltyItems.clicked.connect(self.UpdateEntry)
464         self.RoyaltyItemsWindow.btnDeleteEntry.clicked.connect(self.RemoveFromDB)
465         self.RoyaltyItemsWindow.table = dbTableWidget()
466         self.RefreshTables()
467         self.RoyaltyItemsWindow.table.setFixedSize(620, 150)
468         self.RoyaltyItemsWindow.RoyaltiesItems()
469     self.SelectedID = self.SelectedAuthorID
470     self.SelectedAuthorID = self.holder
471     self.CurrentTable = "Royalties"
472     self.RefreshTables()
473
474
475 def BookInvoiceItemCalculation(self): #calculating the bookinvoicedpayment
476     try:
```

```
478     if self.Editing == False:
479         self.Quantity = int(self.AddWindow.inputList[2].text())
480         self.Discount = float(self.AddWindow.inputList[3].text()) / 100
481         self.ShippingPrice = float(self.AddWindow.inputList[5].text())
482         self.ISBN = self.AddWindow.inputList[1].text()
483     elif self.Editing == True:
484         self.Quantity = int(self.EditWindow.inputList[2].text())
485         self.Discount = float(self.EditWindow.inputList[3].text()) / 100
486         self.ShippingPrice = float(self.EditWindow.inputList[5].text())
487         self.ISBN = self.EditWindow.inputList[1].text()
488
489     with sqlite3.connect("PP.db") as db: #fetching data from db
490         cursor = db.cursor()
491         cursor.execute("select Price from Book where ISBN = {}".format(self.ISBN))
492         self.Price = list(cursor.fetchone())[0]
493         db.commit()
494
495     self.BookInvoiceItemPayment = (self.Quantity * self.Price)
496     self.Discount = self.BookInvoiceItemPayment * self.Discount
497     self.BookInvoiceItemPayment -= self.Discount
498     self.BookInvoiceItemPayment += self.ShippingPrice
499     self.BookInvoiceItemPayment = "{:.2f}".format(self.BookInvoiceItemPayment)
500     if self.Editing == False:
501         self.AddWindow.btnConfirm.clicked.connect(self.AddWindow.accept)
502         self.AddWindow.qleCalculation.setText(self.BookInvoiceItemPayment)
503     elif self.Editing == True:
504         if self.EditWindow.Calculated == False:
505             self.EditWindow.Calculated = True #prevents duplicate connections
506             self.EditWindow.btnConfirm.clicked.connect(self.VerifyUpdate)
507             self.EditWindow.qleCalculation.setText(self.BookInvoiceItemPayment)
508     except:
509         self.Msg = QMessageBox()
510         self.Msg.setWindowTitle("Error")
511         self.Msg.setText("You must fill all fields before clicking 'Calculate'.")
512         self.Msg.exec_()
513
514 def RoyaltyItemCalculation(self): #calculating the royalty payment
515     try:
516         if self.Editing == False:
517             self.Currency = self.AddWindow.inputList[2].text()
518             self.WholesalePrice = float(self.AddWindow.inputList[4].text())
519             self.Quantity = int(self.AddWindow.inputList[5].text())
```

```
520         self.PrintCost = float(self.AddWindow.inputList[6].text())
521     elif self.Editing == True:
522         self.Currency = self.EditWindow.inputList[2].text()
523         self.WholesalePrice = float(self.EditWindow.inputList[4].text())
524         self.Quantity = int(self.EditWindow.inputList[5].text())
525         self.PrintCost = float(self.EditWindow.inputList[6].text())
526
527     self.NetSales = self.WholesalePrice * self.Quantity
528     self.RoyaltyItemPayment = self.NetSales - self.PrintCost
529     self.RoyaltyItemPayment = "{0:.2f}".format(self.RoyaltyItemPayment)
530     if self.Editing == False:
531         self.AddWindow.btnConfirm.clicked.connect(self.AddWindow.accept)
532         self.AddWindow.qleCalculation.setText("{}{}".format(self.Currency, self.RoyaltyItemPayment))
533         self.AddWindow.NetSales = self.NetSales
534     elif self.Editing == True:
535         if self.EditWindow.Calculated == False:
536             self.EditWindow.Calculated = True #prevents duplicate connections
537             self.EditWindow.btnConfirm.clicked.connect(self.VerifyUpdate)
538             self.EditWindow.qleCalculation.setText("{}{}".format(self.Currency, self.RoyaltyItemPayment))
539             self.EditWindow.NetSales = self.NetSales
540     except:
541         self.Msg = QMessageBox()
542         self.Msg.setWindowTitle("Error")
543         self.Msg.setText("You must fill all fields before clicking 'Calculate'.")
544         self.Msg.exec_()
545
546
547 def UpdateCustomerEntry(self): #updating customer entries only
548     self.SelectedRow = self.TableWidget.currentRow()
549     self.SelectedAuthorID = QTableWidgetItem(self.TableWidget.item(self.SelectedRow, 0)).text()
550
551     if self.SelectedAuthorID != "":
552         self.UpdateEntryWindow = dbUpdateEntryWindow()
553         self.UpdateEntryWindow.setFixedSize(640, 115)
554         self.UpdateEntryWindow.selectedID = self.SelectedAuthorID
555         self.UpdateEntryWindow.table = dbTableWidget()
556         self.selectText = "Firstname, Lastname, Email, Phonenumer, Address, Postcode"
557         self.UpdateEntryWindow.table.sql = "select {} from Customer where AuthorID = {}".format(self.selectText,
558                                         self.SelectedAuthorID)
558         self.SelectedID = self.SelectedAuthorID
559         self.UpdateEntryWindow.table.initTable()
560         self.UpdateEntryWindow.table.setFixedSize(617, 55)
```

```
561         self.UpdateEntryWindow.Verify = dbConfirmationDialog()
562         self.UpdateEntryWindow.Verify.Username = self.Username
563         self.UpdateEntryWindow.initConfirmBtn()
564         self.UpdateEntryWindow.btnConfirm.clicked.connect(self.VerifyCustomerUpdate)
565         self.UpdateEntryWindow.initUpdateEntryWindowDlg()
566         self.TableWidget.sql = "select * from Customer"
567         self.RefreshTables()
568
569
570     def UpdateEntry(self): #getting the update input
571         self.Selection = False
572         self.EditWindow = dbAddItemWindow() #uses the add window to init same interface but fill boxes with data
573         self.EditWindow.setFixedSize(360, 200)
574         self.EditWindow.AddType = self.CurrentTable
575         self.EditWindow.AnswerButtons()
576         self.EditWindow.ReadyToVerify = False
577         self.EditWindow.originalItemList = []
578
579         if self.CurrentTable == "Book":
580             self.EditWindow.sql = "select * from Book"
581             self.ViewWindow.SelectedRow = self.ViewWindow.table.currentRow()
582             if self.ViewWindow.SelectedRow != -1:
583                 self.Selection = True
584                 with sqlite3.connect("PP.db") as db:
585                     cursor = db.cursor() #fetching data from database for the edit window
586                     sql = "select * from Book where ISBN = "
587                     sql = sql.format(self.ViewWindow.table.item(self.ViewWindow.SelectedRow, 0).text())
588                     cursor.execute(sql)
589                     self.EditWindow.originalItemList = list(cursor.fetchone())
590
591         elif self.CurrentTable == "PubInvoice":
592             self.EditWindow.setFixedSize(400,150)
593             self.EditWindow.selectedISBN = self.SelectedISBN
594             self.EditWindow.sql = "select ISBN, AuthorID, PubInvoiceDate, PubInvoiceService, PubInvoicePayment from
595             PubInvoice"
596             self.PubInvoiceWindow.SelectedRow = self.PubInvoiceWindow.table.currentRow()
597             if self.PubInvoiceWindow.SelectedRow != -1:
598                 self.Selection = True
599                 with sqlite3.connect("PP.db") as db:
                      cursor = db.cursor() #fetching data from database for the edit window
```

```
600         sql = "select ISBN, AuthorID, PubInvoiceDate, PubInvoiceService, PubInvoicePayment from PubInvoice  
601             where PubInvoiceID =  
602                 {}".format(self.PubInvoiceWindow.table.item(self.PubInvoiceWindow.SelectedRow, 0).text())  
603             cursor.execute(sql)  
604             self.EditWindow.originalItemList = list(cursor.fetchone())  
605  
606     elif self.CurrentTable == "BookInvoice":  
607         self.EditWindow.setFixedSize(350, 100)  
608         self.EditWindow.selectedID = self.SelectedID  
609         self.EditWindow.sql = "select AuthorID, BookInvoiceDate from BookInvoice"  
610         self.BookInvoiceWindow.SelectedRow = self.BookInvoiceWindow.table.currentRow()  
611         if self.BookInvoiceWindow.SelectedRow != -1:  
612             self.Selection = True  
613             with sqlite3.connect("PP.db") as db:  
614                 cursor = db.cursor() #fetching data from database for the edit window  
615                 sql = "select AuthorID, BookInvoiceDate from BookInvoice where BookInvoiceID =  
616                     {}".format(self.BookInvoiceWindow.table.item(self.BookInvoiceWindow.SelectedRow, 0).text())  
617                 cursor.execute(sql)  
618                 self.EditWindow.originalItemList = list(cursor.fetchone())  
619  
620     elif self.CurrentTable == "Royalties":  
621         self.EditWindow.setFixedSize(350, 100)  
622         self.EditWindow.selectedID = self.SelectedID  
623         self.EditWindow.sql = "select AuthorID, RoyaltiesDate from Royalties"  
624         self.RoyaltiesWindow.SelectedRow = self.RoyaltiesWindow.table.currentRow()  
625         if self.RoyaltiesWindow.SelectedRow != -1:  
626             self.Selection = True  
627             with sqlite3.connect("PP.db") as db:  
628                 cursor = db.cursor() #fetching data from database for the edit window  
629                 sql = "select AuthorID, RoyaltiesDate from Royalties where RoyaltiesID =  
630                     {}".format(self.RoyaltiesWindow.table.item(self.RoyaltiesWindow.SelectedRow, 0).text())  
631                 cursor.execute(sql)  
632                 self.EditWindow.originalItemList = list(cursor.fetchone())  
633  
634     elif self.CurrentTable == "BookInvoiceItems":  
635         self.EditWindow.setFixedSize(450, 150)  
636         self.EditWindow.selectedID = self.SelectedID  
637         self.Editing = True  
638         self.EditWindow.btnCalculate.clicked.connect(self.BookInvoiceItemCalculation)  
639         self.EditWindow.selectedISBN = self.SelectedISBN  
640         self.EditWindow.sql = "select BookInvoiceID, ISBN, BookInvoiceQuantity, BookInvoiceDiscount, ShippingType,  
641             ShippingPrice from BookInvoiceItems"
```

```
637     self.BookInvoiceItemsWindow.SelectedRow = self.BookInvoiceItemsWindow.table.currentRow()
638     if self.BookInvoiceItemsWindow.SelectedRow != -1:
639         self.Selection = True
640         with sqlite3.connect("PP.db") as db:
641             cursor = db.cursor() #fetching data from database for the edit window
642             sql = "select BookInvoiceID, ISBN, BookInvoiceQuantity, BookInvoiceDiscount, ShippingType,
643                   ShippingPrice from BookInvoiceItems where BookInvoiceItemsID =
644                   {}".format(self.BookInvoiceItemsWindow.table.item(self.BookInvoiceItemsWindow.SelectedRow,
645                               0).text())
646             cursor.execute(sql)
647             self.EditWindow.originalItemList = list(cursor.fetchone())
648
649 elif self.CurrentTable == "RoyaltyItems":
650     self.EditWindow.setFixedSize(450, 250)
651     self.EditWindow.selectedID = self.SelectedID
652     self.Editing = True
653     self.EditWindow.btnCalculate.clicked.connect(self.RoyaltyItemCalculation)
654     self.EditWindow.selectedISBN = self.SelectedISBN
655     self.EditWindow.sql = "select RoyaltiesID, ISBN, Currency, RoyaltyDiscount, WholesalePrice, RoyaltyQuantity,
656                           PrintCost, ExcRateFromGBP from RoyaltyItems"
657     self.RoyaltyItemsWindow.SelectedRow = self.RoyaltyItemsWindow.table.currentRow()
658     if self.RoyaltyItemsWindow.SelectedRow != -1:
659         self.Selection = True
660         with sqlite3.connect("PP.db") as db:
661             cursor = db.cursor() #fetching data from database for the edit window
662             sql = "select RoyaltiesID, ISBN, Currency, RoyaltyDiscount, WholesalePrice, RoyaltyQuantity,
663                   PrintCost, ExcRateFromGBP from RoyaltyItems where RoyaltyItemsID =
664                   {}".format(self.RoyaltyItemsWindow.table.item(self.RoyaltyItemsWindow.SelectedRow, 0).text())
665             cursor.execute(sql)
666             self.EditWindow.originalItemList = list(cursor.fetchone())
667
668     self.EditWindow.Editing = True
669
670     if self.Selection == True: #initialising the edit window
671         self.EditWindow.btnConfirm.clicked.connect(self.EditWindow.Validate)
672         if self.CurrentTable in ["RoyaltyItems", "BookInvoiceItems"]:
673             self.EditWindow.btnConfirm.clicked.connect(self.EditWindow.CheckCalculated)
674         else:
675             self.EditWindow.btnConfirm.clicked.connect(self.VerifyUpdate)
676         self.EditWindow.AddType = self.CurrentTable
677         self.EditWindow.selectedID = self.SelectedID
```

```
673         self.EditWindow.CalendarWidget = dbCalendarWidget()
674         self.EditWindow.CalendarWidget.Calendar()
675         self.EditWindow.initAddItemWindow()
676
677
678     def VerifyCustomerUpdate(self):
679         self.UpdateEntryWindow.Verify = dbConfirmationDialog()
680         #new instance for customer verification
681
682
683
684     def VerifyUpdate(self): #verification dialog
685         if self.EditWindow.ReadyToVerify == True:
686             self.EditWindow.Verify = dbConfirmationDialog()
687             self.EditWindow.Verify.Msg = "Insert Password to confirm all changes"
688             self.EditWindow.Verify.ConfirmedMsg = "Update successful"
689             self.EditWindow.Verify.VerifyDlg()
690             if self.EditWindow.Verify.ConfirmedDialog.Accepted == True:
691                 self.UpdateChanges()
692                 self.EditWindow.accept()
278
693
694
695
696     def UpdateChanges(self): #committing changes
697         self.UpdateList = []
698
699         with sqlite3.connect("PP.db") as db:
700             cursor = db.cursor()
701             if self.CurrentTable == "Book":
702                 self.NoOfEntries = 12
703                 cursor.execute("select * from Book")
704                 self.ID = "ISBN"
705                 self.SelectedISBN = QTableWidgetItem(self.ViewWindow.table.item(self.ViewWindow.SelectedRow, 0)).text()
706                 self.SelectedID = self.SelectedISBN
707                 self.BookEdited = True
708
709             elif self.CurrentTable == "PubInvoice":
710                 cursor.execute("select ISBN, AuthorID, PubInvoiceDate, PubInvoiceService, PubInvoicePayment from
711                               PubInvoice")
712                 self.NoOfEntries = 5
713                 self.ID = "PubInvoiceID"
```

```
714     self.OldID = self.SelectedID
715     self.SelectedID = QTableWidgetItem(self.PubInvoiceWindow.table.item(self.PubInvoiceWindow.SelectedRow,
716                               0)).text()
717
718     elif self.CurrentTable == "BookInvoice":
719         cursor.execute("select AuthorID, BookInvoiceDate from BookInvoice")
720         self.NoOfEntries = 2
721         self.ID = "BookInvoiceID"
722         self.OldID = self.SelectedID
723         self.SelectedID = QTableWidgetItem(self.BookInvoiceWindow.table.item(self.BookInvoiceWindow.SelectedRow,
724                               0)).text()
725
726     elif self.CurrentTable == "Royalties":
727         cursor.execute("select AuthorID, RoyaltiesDate from Royalties")
728         self.NoOfEntries = 2
729         self.ID = "RoyaltiesID"
730         self.OldID = self.SelectedID
731         self.SelectedID = QTableWidgetItem(self.RoyaltiesWindow.table.item(self.RoyaltiesWindow.SelectedRow,
732                               0)).text()
733
734     elif self.CurrentTable == "BookInvoiceItems":
735         cursor.execute("select BookInvoiceID, ISBN, BookInvoiceQuantity, BookInvoiceDiscount, ShippingType,
736                         ShippingPrice from BookInvoiceItems")
737         self.NoOfEntries = 6
738         self.ID = "BookInvoiceItemsID"
739         self.OldID = self.SelectedID
740         self.SelectedID =
741             QTableWidgetItem(self.BookInvoiceItemsWindow.table.item(self.BookInvoiceItemsWindow.SelectedRow,
742                           0)).text()
743
744     elif self.CurrentTable == "RoyaltyItems":
745         cursor.execute("select RoyaltiesID, ISBN, Currency, RoyaltyDiscount, WholesalePrice, RoyaltyQuantity,
746                         PrintCost, ExcRateFromGBP from RoyaltyItems")
747         self.NoOfEntries = 6
748         self.ID = "RoyaltyItemsID"
749         self.OldID = self.SelectedID
750         self.SelectedID = QTableWidgetItem(self.RoyaltyItemsWindow.table.item(self.RoyaltyItemsWindow.SelectedRow,
751                               0)).text()
752
753     for count in range(0, self.NoOfEntries): #creating the update string for sql
754         try:
755             self.UpdateList.append(str(self.EditWindow.inputList[count].currentText()))
```

```
748     except:
749         if count == 8 and self.CurrentTable == "RoyaltyItems":
750             self.UpdateList.append(str(self.NetSales))
751         else:
752             self.UpdateList.append(self.EditWindow.inputList[count].text())
753
754     self.Update = ""
755
756     self.Headers = list(cursor.description)
757     for count in range(0, len(self.UpdateList)):
758         self.Update += "{} = '{}'".format(list(self.Headers[count])[0], self.UpdateList[count])
759         if count != len(self.UpdateList) - 1:
760             self.Update += ", "
761
762     with sqlite3.connect("PP.db") as db: #update
763         cursor = db.cursor()
764         if self.CurrentTable == "Book": #ISBN is primary key which may be changed, so all foreign keys will change
765             first for it
766             try:
767                 sql = "update PubInvoice set ISBN = {0} where ISBN = {1}".format(self.UpdateList[0], self.SelectedID)
768                 cursor.execute(sql)
769                 sql = "update BookInvoiceItems set ISBN = {0} where ISBN = {1}".format(self.UpdateList[0],
770                     self.SelectedID)
771                 cursor.execute(sql)
772                 sql = "update RoyaltyItems set ISBN = {0} where ISBN = {1}".format(self.UpdateList[0], self.SelectedID)
773                 cursor.execute(sql)
774             except:
775                 pass # error if no entry is there for pubinvoice/bookinvoiceitems/royaltyitems
776             sql = "update {} set {} where {} = {}".format(self.CurrentTable, self.Update, self.ID, self.SelectedID)
777             cursor.execute(sql)
778             db.commit()
779
780         if self.CurrentTable in ["Royalties", "BookInvoice", "PubInvoice", "BookInvoiceItems", "RoyaltyItems"]:
781             self.SelectedID = self.OldID
782
783         self.RefreshTables()
784
785         try:
786             self.RecalculateItems()
787             self.BookEdited = False
788         except:
789             pass
```

```
788
789
790
791     def RefreshTables(self): #refreshing tables
792         if self.CurrentTable == "Customer":
793             self.TableWidget.sql = "select * from Customer"
794             self.TableWidget.initTable()
795
796         if self.CurrentTable == "Book":
797             self.ViewWindow.table.sql = "select * from Book where AuthorID = {}".format(self.SelectedAuthorID)
798             self.ViewWindow.table.initTable()
799             with sqlite3.connect("PP.db") as db:
800                 cursor = db.cursor()
801                 cursor.execute("select Firstname, Lastname from Customer where AuthorID = "
802                               "{}".format(self.SelectedAuthorID))
803                 self.Name = list(cursor.fetchone())
804                 self.Name = "{} , {}".format(self.Name[1], self.Name[0])
805                 for count in range(0, self.TableWidget.rowCount()+1):
806                     self.ViewWindow.table.setItem(count, 1, QTableWidgetItem(self.Name))
807                     self.ViewWindow.table.setHorizontalHeaderItem(1, QTableWidgetItem("Author"))
808                     db.commit()
809
810         elif self.CurrentTable == "PubInvoice":
811             self.PubInvoiceWindow.table.sql = "select * from PubInvoice where ISBN = {}".format(self.SelectedISBN)
812             self.PubInvoiceWindow.table.initTable()
813             with sqlite3.connect("PP.db") as db:
814                 cursor = db.cursor()
815                 cursor.execute("select BookTitle from Book where ISBN = {}".format(self.SelectedISBN))
816                 self.Title = "{}".format(list(cursor.fetchone())[0])
817                 for count in range(0, self.TableWidget.rowCount()+1):
818                     self.PubInvoiceWindow.table.setItem(count, 1, QTableWidgetItem(self.Title))
819                     self.PubInvoiceWindow.table.setHorizontalHeaderItem(1, QTableWidgetItem("BookTitle"))
820                     cursor.execute("select Firstname, Lastname from Customer where AuthorID = "
821                                   "{}".format(self.SelectedAuthorID))
822                     self.Name = list(cursor.fetchone())
823                     self.Name = "{} , {}".format(self.Name[1], self.Name[0])
824                     for count in range(0, self.TableWidget.rowCount()+1):
825                         self.PubInvoiceWindow.table.setItem(count, 2, QTableWidgetItem(self.Name))
826                         self.PubInvoiceWindow.table.setHorizontalHeaderItem(2, QTableWidgetItem("Author"))
827                         db.commit()
828
829         elif self.CurrentTable == "BookInvoice":
```

```
827     self.BookInvoiceWindow.table.sql = "select * from BookInvoice where AuthorID =
828         {}".format(self.SelectedAuthorID)
829     self.BookInvoiceWindow.table.initTable()
830     with sqlite3.connect("PP.db") as db:
831         cursor = db.cursor()
832         cursor.execute("select Firstname, Lastname from Customer where AuthorID =
833             {}".format(self.SelectedAuthorID))
834         self.Name = list(cursor.fetchone())
835         self.Name = "{}, {}".format(self.Name[1], self.Name[0])
836         for count in range(0, self.TableWidget.rowCount()+1):
837             self.BookInvoiceWindow.table.setItem(count, 1, QTableWidgetItem(self.Name))
838             self.BookInvoiceWindow.table.setHorizontalHeaderItem(1, QTableWidgetItem("Author"))
839             db.commit()
840
841     elif self.CurrentTable == "Royalties":
842         self.RoyaltiesWindow.table.sql = "select * from Royalties where AuthorID = {}".format(self.SelectedAuthorID)
843         self.RoyaltiesWindow.table.initTable()
844         with sqlite3.connect("PP.db") as db:
845             cursor = db.cursor()
846             cursor.execute("select Firstname, Lastname from Customer where AuthorID =
847                 {}".format(self.SelectedAuthorID))
848             self.Name = list(cursor.fetchone())
849             self.Name = "{}, {}".format(self.Name[1], self.Name[0])
850             for count in range(0, self.TableWidget.rowCount()+1):
851                 self.RoyaltiesWindow.table.setItem(count, 1, QTableWidgetItem(self.Name))
852                 self.RoyaltiesWindow.table.setHorizontalHeaderItem(1, QTableWidgetItem("Author"))
853                 db.commit()
854
855     elif self.CurrentTable == "BookInvoiceItems":
856         self.BookInvoiceItemsWindow.table.sql = "select * from BookInvoiceItems where BookInvoiceID =
857             {}".format(self.SelectedID)
858
859         self.BookInvoiceItemsWindow.table.initTable()
860         with sqlite3.connect("PP.db") as db:
861             cursor = db.cursor()
862             self.ID = []
863             sql = "select BookInvoiceItemsID from BookInvoiceItems where BookInvoiceID = {}".format(self.SelectedID)
864             cursor.execute(sql)
865             self.IDList = list(cursor.fetchall())
866             for count in range(0, len(self.IDList)):
867                 self.temp = list(self.IDList[count])[0]
868                 self.ID.append(self.temp)
```

```
865
866     for count in range(0, len(self.ID)):
867         try:
868             cursor.execute("select Book.BookTitle, BookInvoiceItems.ISBN, Firstname, Lastname,
869                             BookInvoiceItems.BookInvoiceID from Book, BookInvoiceItems, Customer, BookInvoice where
870                             BookInvoiceItems.BookInvoiceID = {} and BookInvoiceItems.BookInvoiceItemsID = {} and
871                             Book.ISBN = BookInvoiceItems.ISBN and BookInvoice.BookInvoiceID =
872                             BookInvoiceItems.BookInvoiceID and Customer.AuthorID =
873                             BookInvoice.AuthorID".format(self.SelectedID, self.ID[count]))
874             self.Title = list(cursor.fetchone())
875             self.Name = "{}, {}".format(self.Title[3], self.Title[2])
876             self.Title = self.Title[0]
877             self.BookInvoiceItemsWindow.table.setItem(count, 2, QTableWidgetItem(str(self.Title)))
878             self.BookInvoiceItemsWindow.table.setItem(count, 1, QTableWidgetItem(str(self.Name)))
879         except:
880             pass
881         self.BookInvoiceItemsWindow.table.setHorizontalHeaderItem(2, QTableWidgetItem("BookTitle"))
882         self.BookInvoiceItemsWindow.table.setHorizontalHeaderItem(1, QTableWidgetItem("Author"))
883         db.commit()
884
885     elif self.CurrentTable == "RoyaltyItems":
886         self.RoyaltyItemsWindow.table.sql = "select * from RoyaltyItems where RoyaltiesID = {}".format(self.SelectedID)
887         self.RoyaltyItemsWindow.table.initTable()
888         with sqlite3.connect("PP.db") as db:
889             cursor = db.cursor()
890             self.ID = []
891             sql = "select RoyaltyItemsID from RoyaltyItems where RoyaltiesID = {}".format(self.SelectedID)
892             cursor.execute(sql)
893             self.IDList = list(cursor.fetchall())
894             for count in range(0, len(self.IDList)):
895                 self.temp = list(self.IDList[count])[0]
896                 self.ID.append(self.temp)
897
898             for count in range(0, len(self.ID)):
899                 try:
900                     cursor.execute("select Book.BookTitle, RoyaltyItems.ISBN, Firstname, Lastname,
901                                     RoyaltyItems.RoyaltiesID from Book, RoyaltyItems, Customer, Royalties where
902                                     RoyaltyItems.RoyaltiesID = {} and RoyaltyItems.RoyaltyItemsID = {} and Book.ISBN =
903                                     RoyaltyItems.ISBN and Royalties.RoyaltiesID = RoyaltyItems.RoyaltiesID and Customer.AuthorID
904                                     = Royalties.AuthorID".format(self.SelectedID, self.ID[count]))
905                     self.Title = list(cursor.fetchone())
906                     self.Name = "{}, {}".format(self.Title[3], self.Title[2])
```

```
898         self.Title = self.Title[0]
899         self.RoyaltyItemsWindow.table.setItem(count, 2, QTableWidgetItem(str(self.Title)))
900         self.RoyaltyItemsWindow.table.setItem(count, 1, QTableWidgetItem(str(self.Name)))
901     except:
902         pass
903     self.RoyaltyItemsWindow.table.setHorizontalHeaderItem(2, QTableWidgetItem("BookTitle"))
904     self.RoyaltyItemsWindow.table.setHorizontalHeaderItem(1, QTableWidgetItem("Author"))
905     db.commit()
906
907 def Back(self): #going back from the view window to the main menu
908     self.ViewWindow.table.selectedID = None
909     self.CurrentTable = "Customer"
910     self.StackedLayout.setCurrentIndex(0)
911     self.MenuBar.setVisible(True)
912
913
914
915 def QuickSearch(self): #quick search from main menu
916     self.QSText = self.MainMenuButtons.leQuickSearch.text()
917     self.QSText = self.QSText.split(' ')
918     with sqlite3.connect("PP.db") as db:
919         cursor = db.cursor()
920
921         if len(self.QSText) == 1:
922             self.TableWidget.sql = "select * from Customer where Firstname like '{0}%' or Lastname like
923             '{0}%'.format(self.QSText[0])
924         elif len(self.QSText) == 0:
925             self.TableWidget.sql = "select * from Customer"
926         else:
927             for count in range(1, len(self.QSText)):
928                 if count != 1:
929                     self.QSText[1] += " {}".format(self.QSText[count])
930             self.TableWidget.sql = "select * from Customer where Firstname like '{0}%' and Lastname like
931             '{1}%'.format(self.QSText[0], self.QSText[1])
932             self.TableWidget.initTable()
933
934
935 def Search(self): #main search interface
936     self.SearchDatabase = dbSearchDatabase()
937     self.SearchDatabase.Table = None
938     self.SearchDatabase.Valid = False
```

```
938     self.SearchDatabase.CalendarWidget = dbCalendarWidget()
939     self.SearchDatabase.CalendarWidget.Calendar()
940     self.SearchDatabase.initLayout()
941     self.errorIndex = False
942     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 =
950                                     '{}'.format(self.SearchDatabase.Table,
951                                     list(self.SearchDatabase.Results[count])[2])
952                             else:
953                                 self.SearchTable.sql += " or ISBN =
954                                     '{}'.format(list(self.SearchDatabase.Results[count])[2])
955
956
957
958
959
285
960
961
962
963
964
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 =
978                 '{}'.format(list(self.SearchDatabase.Results[count])[0])"
979         else:
980             self.SearchTable.sql += " or AuthorID =
981                 '{}'.format(list(self.SearchDatabase.Results[count])[0])"
982         self.SearchTable.initTable()
983
984     except IndexError:
985         self.errorIndex = True
986         self.SearchTable.initTable()
987
988 elif self.SearchDatabase.Table != "Book" and self.SearchDatabase.Table != None:
989     for count in range(0, len(self.SearchDatabase.Results)+1):
990         try: #using the results to fetch the correct data
991             if count == 0:
992                 self.SearchTable.sql = "select * from {} where {}ID =
993                     '{}'.format(self.SearchDatabase.Table,
994                         list(self.SearchDatabase.Results[count])[index])"
995             else:
996                 self.SearchTable.sql += " or {}ID = '{}'".format(self.SearchDatabase.Table,
997                     list(self.SearchDatabase.Results[count])[index])
998             self.SearchTable.initTable()
999
1000         except IndexError:
1001             self.errorIndex = True
1002             with sqlite3.connect("PP.db") as db:
1003                 cursor = db.cursor() #fetching firstnames and lastnames and book titles using foreign
1004                 keys
1005                 if self.SearchDatabase.Table in ["Royalties", "BookInvoice"]:
1006                     self.SearchTable.setHorizontalHeaderItem(1, QTableWidgetItem("Author"))
1007                     for count2 in range(0, self.SearchTable.rowCount()):
1008                         cursor.execute("select Firstname, Lastname, {} .AuthorID, Customer.AuthorID
1009                             from Customer, {} where Customer.AuthorID = {} and {} .AuthorID =
1010                             Customer.AuthorID".format(self.SearchDatabase.Table,
1011                             list(self.SearchDatabase.Results[count2])[0]))
1012                         self.Name = list(cursor.fetchone())
1013                         self.Name = "{} , {}".format(self.Name[1], self.Name[0])
1014                         self.SearchTable.setItem(count2, 1, QTableWidgetItem(self.Name))
```

```
1008     elif self.SearchDatabase.Table in ["RoyaltyItems", "BookInvoiceItems"]:
1009         self.SearchTable.setHorizontalHeaderItem(1, QTableWidgetItem("Author"))
1010         self.SearchTable.setHorizontalHeaderItem(2, QTableWidgetItem("Book Title"))
1011         if self.SearchDatabase.Table == "RoyaltyItems":
1012             self.IDType = "Royalties"
1013         else:
1014             self.IDType = "BookInvoice"
1015         for count2 in range(0, self.SearchTable.rowCount()):
1016             cursor.execute("select BookTitle, Book.ISBN, {0}.ISBN, {1}.AuthorID,
1017                             Firstname, LastName from Book, {0}, {1}, Customer where Book.ISBN = {2}
1018                             and {0}.ISBN = Book.ISBN and {1}.{1}ID = {0}.{1}ID and Customer.AuthorID
1019                             = {1}.AuthorID".format(self.SearchDatabase.Table, self.IDType,
1020                             list(self.SearchDatabase.Results[count2])[2]))
1021             self.Title = cursor.fetchone()
1022             self.Name = "{}".format(self.Title[5], self.Title[4])
1023             self.Title = "{}".format(list(self.Title)[0])
1024             self.SearchTable.setItem(count2, 2, QTableWidgetItem(self.Title))
1025             self.SearchTable.setItem(count2, 1, QTableWidgetItem(self.Name))
1026     elif self.SearchDatabase.Table == "PubInvoice":
1027         self.SearchTable.setHorizontalHeaderItem(1, QTableWidgetItem("Book Title"))
1028         self.SearchTable.setHorizontalHeaderItem(2, QTableWidgetItem("Author"))
1029         for count2 in range(0, self.SearchTable.rowCount()):
1030             cursor.execute("select Firstname, Lastname, BookTitle, {0}.AuthorID,
1031                             Customer.AuthorID, Book.ISBN, {0}.ISBN from Customer, {0}, Book where
1032                             Customer.AuthorID = {1} and {0}.AuthorID = Customer.AuthorID and
1033                             Book.AuthorID = Customer.AuthorID and {0}.ISBN =
1034                             Book.ISBN".format(self.SearchDatabase.Table,
1035                             list(self.SearchDatabase.Results[count2])[0]))
1036             self.Name = list(cursor.fetchone())
1037             self.Title = "{}".format(self.Name[2])
1038             self.Name = "{}".format(self.Name[1], self.Name[0])
1039             self.SearchTable.setItem(count2, 2, QTableWidgetItem(self.Name))
1040             self.SearchTable.setItem(count2, 1, QTableWidgetItem(self.Title))

          db.commit()

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

  except:
      self.errorIndex = True
```

```
1041         if self.errorIndex == True:
1042             self.StackedLayout.setCurrentIndex(0)
1043             self.Msg = QMessageBox()
1044             self.Msg.setWindowTitle("No Results Found")
1045             self.Msg.setText("No data matches your search")
1046             self.Msg.exec_()
1047
1048
1049     def keyReleaseEvent(self, QKeyEvent):
1050         if self.MainMenuButtons.leQuickSearch.text() == "":
1051             self.TableWidget.sql = "select * from Customer"
1052             self.TableWidget.initTable()
1053
1054
1055
1056     def ChangeUsernameOrPassword(self):
1057
1058         self.ChangeWhat = dbUsernameOrPassword()
1059         self.ChangeWhat.Selection = None
1060         self.ChangeWhat.ChangeSelection()
1061         if self.ChangeWhat.Selection == "Username":
1062             self.UsernameChange = dbChangeUsername()
1063             self.UsernameChange.Username = self.Username
1064             self.UsernameChange.initChangeUsernameScreen()
1065         elif self.ChangeWhat.Selection == "Password":
1066             self.PasswordChange = dbChangePassword()
1067             self.PasswordChange.Username = self.Username
1068             self.PasswordChange.initChangePasswordScreen()
1069
1070     def LogOut(self):
1071         self.hide()
1072         subprocess.call("LoginDB.py", shell=True)
```

---

#### 4.10.3 Module 3 - Initialising Main Menu Layout

---

```
1 from PyQt4.QtCore import *
2 from PyQt4.QtGui import *
3 import sqlite3
4 import sys
```

```
5
6
7 class initMainMenuButtons(QWidget):
8     """initialising the main menu buttons"""
9
10    def __init__(self):
11        super().__init__()
12
13    #creating the layout
14
15    self.vertical = QVBoxLayout()
16
17    self.btnLogOut = QPushButton("Log Out", self)
18    self.btnLogOut.setFixedSize(100, 30)
19
20    self.leQuickSearch = QLineEdit(self)
21    self.leQuickSearch.setPlaceholderText("Author Name")
22    self.leQuickSearch.setFixedSize(100, 25)
23
24    self.btnQuickSearch = QPushButton("Quick Search", self)
25    self.btnQuickSearch.setFixedSize(100, 30)
26
27    self.horizontalTop = QHBoxLayout()
28    self.horizontalTop.addWidget(self.btnLogOut)
29    self.horizontalTop.addStretch(1)
30    self.horizontalTop.addWidget(self.leQuickSearch)
31    self.horizontalTop.addWidget(self.btnQuickSearch)
32
33    self.btnView = QPushButton("View", self)
34    self.btnView.setFixedSize(100, 40)
35
36    self.btnSearchdb = QPushButton("Search Database", self)
37    self.btnSearchdb.setFixedSize(100, 40)
38
39    self.btnAddEntry = QPushButton("Add Entry", self)
40    self.btnAddEntry.setFixedSize(100, 40)
41
42    self.btnUpdateEntry = QPushButton("Update Entry", self)
43    self.btnUpdateEntry.setFixedSize(100, 40)
44
45    self.btnRemoveEntry = QPushButton("Remove Entry", self)
46    self.btnRemoveEntry.setFixedSize(100, 40)
```

289

Imran Rahman

Candidate No. 30928

Centre No. 22151

```
47         self.btnExit = QPushButton("Change Username/\nPassword", self)
48         self.btnExit.setFixedSize(100, 40)
49
50         self.horizontalBottom = QHBoxLayout()
51
52         self.horizontalBottom.addWidget(self.btnView)
53         self.horizontalBottom.addWidget(self.btnSearchdb)
54         self.horizontalBottom.addWidget(self.btnAddEntry)
55         self.horizontalBottom.addWidget(self.btnUpdateEntry)
56         self.horizontalBottom.addWidget(self.btnRemoveEntry)
57         self.horizontalBottom.addWidget(self.btnExit)
58         self.horizontalBottom.addWidget(self.btnChangePassword)
```

#### 4.10.4 Module 4 - Initialising Menu Bar

```
1  from PyQt4.QtCore import *
2  from PyQt4.QtGui import *
3  import sys
4
5  class dbMenuBar(QMenuBar):
6      """the menu bar"""
7
8      def __init__(self):
9          super().__init__()
10
11         self.search_database = QAction("Search Database", self) #creating actions
12         self.add_entry = QAction("Add Entry", self)
13         self.update_entry = QAction("Update Entry", self)
14         self.remove_entry = QAction("Remove Entry", self)
15         self.change_password = QAction("Change Username/Password", self)
16         self.log_out = QAction("Log Out", self)
17
18         self.database_menu = self.addMenu("Database") #adding menus
19         self.database_menu.addAction(self.search_database) #adding actions to menus
20
21         self.actions_menu = self.addMenu("Actions")
22         self.actions_menu.addAction(self.add_entry)
23         self.actions_menu.addAction(self.update_entry)
24         self.actions_menu.addAction(self.remove_entry)
```

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

#### 4.10.5 Module 5 - Initialising a Table Widget

```
1  from PyQt4.QtCore import *
2  from PyQt4.QtGui import *
3  import sys
4  import sqlite3
5
6  class dbTableWidget(QTableWidget):
7      """main table widget"""
8
9      def __init__(self):
10          super().__init__()
11
12
13      def initTable(self):
14          self.clear()
15          self.columns = []
16          self.setSelectionBehavior(QAbstractItemView.SelectRows)
17          self.setEditTriggers(QAbstractItemView.NoEditTriggers)
18          self.setSortingEnabled(False)
19          with sqlite3.connect("PP.db") as db: #fetching data from db
20              cursor = db.cursor()
21              cursor.execute(self.sql)
22              self.ColumnNames = cursor.description
23              for count in range(0, len(self.ColumnNames)):
24                  self.columns.append(list(list(self.ColumnNames)[count])[0])
25          self.setRowCount(0)
26          self.setColumnCount(len(self.columns))
27          self.setHorizontalHeaderLabels(self.columns)
28          for self.row, self.form in enumerate(cursor):
29              self.insertRow(self.row)
30              for self.column, self.unit in enumerate(self.form):
31                  self.setItem(self.row, self.column, QTableWidgetItem(str(self.unit)))
32          self.setSortingEnabled(True)
```

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

```

1 from PyQt4.QtCore import *
2 from PyQt4.QtGui import *
3 import sqlite3
4 import sys
5 import re
6
7 class dbAddEntryWindow(QDialog):
8     """add entry window dialog"""
9
10    def __init__(self):
11        super().__init__()
12
13    def initAddEntryWindow(self):
14        self.setWindowTitle("Add Entry")
15        self.setFixedSize(640,115)
16        self.setModal(True) #modal window
17        self.AddEntryTable = QTableWidget(self) #table for adding customer entry
18        self.AddEntryTable.setRowCount(1)
19        self.AddEntryTable.setColumnCount(6)
20        self.AddEntryTable.setFixedSize(617, 55)
21        self.inputList = []
22        for count in range(0, 6):
23            #0 firstname, 1 lastname, 2 email, 3 phoneno, 4 address, 5 postcode
24            self.input = QLineEdit(self)
25            self.input setFrame(False)
26            self.inputList.append(self.input)
27            self.AddEntryTable.setCellWidget(0, count, self.input)
28
29            self.inputList[0].setValidator(QRegExpValidator(QRegExp("[a-zA-Z\\-\\!]+")))
30            self.inputList[1].setValidator(QRegExpValidator(QRegExp("[a-zA-Z\\-\\!]+")))
31            self.inputList[2].setValidator(QRegExpValidator(QRegExp("^(\\w-\\.]+@[\\w-]+\\.\\w-[2,4]$")))
32            self.regexp = """^(((\\+44\\s?\\d{4}\\\\(\\?\\d{4}\\)\\\\?)\\s?\\d{3})\\s?\\d{3})|((\\+44\\s?\\d{3}\\\\(\\?\\d{3}\\)\\\\?)\\s?\\d{3})\\s?\\d{4})|((\\+44\\s?\\d{2}\\\\(\\?\\d{2}\\)\\\\?)\\s?\\d{4})\\s?\\d{4}))\\s?\\#(\\d{4})\\d{3}))?$$"
33            self.inputList[3].setValidator(QRegExpValidator(QRegExp(self.regexp)))
34            self.inputList[4].setValidator(QRegExpValidator(QRegExp("[a-zA-Z \\d\\-\\.]+")))
35            self.inputList[5].setValidator(QRegExpValidator(QRegExp("^[a-zA-Z]{1,2}[0-9][0-9A-Za-z]{0,1}
36            {0,1}[0-9][A-Za-z]{2}$")))
37            #2, 3 & 5 from www.regexlib.com
38            self.CustomerHeaders = ["Firstname", "Lastname", "Email", "Phonenumber", "Address", "Postcode"]
39            self.AddEntryTable.setHorizontalHeaderLabels(self.CustomerHeaders)

```

```
40     self.btnExit = QPushButton("Confirm", self) #buttons
41     self.btnExit = QPushButton("Cancel", self)
42     self.horizontal = QHBoxLayout()
43     self.horizontal.addStretch(1)
44     self.horizontal.addWidget(self.btnExit)
45     self.horizontal.addWidget(self.btnExit)
46
47     self.vertical = QVBoxLayout() #vbox layout
48     self.vertical.addWidget(self.AddEntryTable)
49     self.vertical.addStretch(1)
50     self.vertical.setLayout(self.horizontal)
51     self.setLayout(self.vertical)
52
53     self.btnExit.clicked.connect(self.reject) #reject on clicking cancel
54     self.btnExit.clicked.connect(self.AddEntryTodb) #call function after clicking confirm
55     self.exec_()
56
57 def AddEntryTodb(self):
58     #fetching inputs from table
59     self.Valid = True
60     self.Message = "All Fields must be filled."
61     self.Firstname = self.inputList[0].text()
62     self.Lastname = self.inputList[1].text()
63     self.Email = self.inputList[2].text()
64     self.Phonenumber = self.inputList[3].text()
65     self.Address = self.inputList[4].text()
66     self.Postcode = self.inputList[5].text()
67
68     self.input_data = (self.Firstname, self.Lastname, self.Email, self.Phonenumber, self.Address, self.Postcode)
69     for count in range(0, len(list(self.input_data))):
70         if list(self.input_data)[count].replace(" ", "") == "":
71             self.Valid = False
72     try:
73         float(self.input_data[4])
74         self.Valid = False
75         self.Message = "Invalid Address"
76     except:
77         pass
78
79     if self.Valid == True:
80         with sqlite3.connect("PP.db") as db:
81             cursor = db.cursor()
```

```
82         cursor.execute("PRAGMA foreign_keys = ON")
83         sql = "insert into Customer (FirstName, LastName, Email, PhoneNumber, Address, Postcode) values (?, ?, ?, ?, ?, ?)"
84         cursor.execute(sql, self.input_data)
85         db.commit()
86         self.accept()
87     else:
88         self.Msg = QMessageBox()
89         self.Msg.setWindowTitle("Invalid Entry")
90         self.Msg.setText(self.Message)
91         self.Msg.exec_()
```

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

```
1 from PyQt4.QtCore import *
2 from PyQt4.QtGui import *
3 import sqlite3
4 import sys
5 from ConfirmationDialog import *
6
7 class dbUpdateEntryWindow(QDialog):
8     """update entry window dialog"""
9
10    def __init__(self):
11        super().__init__()
12
13    def initUpdateEntryWindowDlg(self):
14        self.table.setSelectionBehavior(QAbstractItemView.SelectItems)
15        self.setWindowTitle("Update Entry")
16
17        self.setModal(True)
18        self.btnEdit = QPushButton("Edit", self)
19        #self.btnConfirm = QPushButton("Confirm", self)
20        self.horizontal = QHBoxLayout()
21        self.horizontal.addWidget(self.btnEdit)
22        self.horizontal.addStretch(1)
23        self.horizontal.addWidget(self.btnConfirm)
24        self.vertical = QVBoxLayout()
25        self.vertical.addWidget(self.table)
```

```
26         self.vertical.addLayout(self.horizontal)
27         self.setLayout(self.vertical)
28         self.btnEdit.clicked.connect(self.Edit)
29         self.btnConfirm.clicked.connect(self.Verification)
30         self.TableName = "Customer"
31         self.ID = "AuthorID"
32         self.exec_()
33
34     def initConfirmBtn(self): #creating confirm btn for instantiation of verification window
35         self.btnConfirm = QPushButton("Confirm", self)
36
37     def Verification(self):
38
39         self.Verify.Msg = "Insert Password to confirm all changes"
40         self.Verify.ConfirmedMsg = "Update successful"
41         self.Verify.VerifyDlg()
42         if self.Verify.ConfirmedDialog.Accepted == True:
43             self.UpdateChanges()
44             self.accept()
45
46     def Edit(self):
47         self.SelectedItem = self.table.currentItem()
48         self.SelectedRow = self.table.currentRow()
49         self.SelectedColumn = self.table.currentColumn()
50         if self.SelectedItem != None:
51             self.EditDlg = dbUpdateEntryWindow()
52             self.EditDlg.setModal(True)
53             self.EditDlg.setWindowTitle("Input Text")
54             self.EditDlg.setFixedSize(210, 120)
55             self.EditDlg.lbl = QLabel("Insert text to save over current entry", self)
56             self.EditDlg.qle = QLineEdit(self)
57             self.EditDlg.qle.setPlaceholderText("Insert text here")
58             self.EditDlg.btnConfirm = QPushButton("Confirm", self)
59             self.EditDlg.btnConfirm.setFixedSize(60, 25)
60
61             self.EditDlg.qlehorizontal = QHBoxLayout()
62             self.EditDlg.btnhorizontal = QHBoxLayout()
63             self.EditDlg.qlehorizontal.addStretch(1)
64             self.EditDlg.qlehorizontal.addWidget(self.EditDlg.qle)
65             self.EditDlg.qlehorizontal.addStretch(1)
66             self.EditDlg.btnhorizontal.addStretch(1)
67             self.EditDlg.btnhorizontal.addWidget(self.EditDlg.btnConfirm)
```

```
68         self.EditDlg.btnhorizontal.addStretch(1)
69         self.EditDlg.vertical = QVBoxLayout()
70         self.EditDlg.vertical.addWidget(self.EditDlg.lbl)
71         self.EditDlg.vertical.addLayout(self.EditDlg.qlehorizontal)
72         self.EditDlg.vertical.addLayout(self.EditDlg.btnhorizontal)
73         self.EditDlg.setLayout(self.EditDlg.vertical)
74         self.EditDlg.btnConfirm.clicked.connect(self.EditDlg.accept)
75         self.EditDlg.btnConfirm.clicked.connect(self.GetInput)
76         self.EditDlg.exec_()
77
78     def GetInput(self):
79         self.EditInput = self.EditDlg.qle.text()
80         self.table.setItem(self.SelectedRow, self.SelectedColumn, QTableWidgetItem(self.EditInput))
81
82
83     def UpdateChanges(self):
84         UL = [] #UL = Update List
85         for count in range(0, 6):
86             UL.append(self.table.item(0, count).text())
87         self.Update = "Firstname = '{}', Lastname = '{}', Email = '{}', Phonenumber = '{}', Address = '{}', Postcode = "
88         self.Update += '{}'.format(UL[0], UL[1], UL[2], UL[3], UL[4], UL[5])
89         with sqlite3.connect("PP.db") as db:
90             cursor = db.cursor()
91             cursor.execute("PRAGMA foreign_keys = ON")
92             sql = "update {} set {} where {} = {}".format(self.TableName, self.Update, self.ID, self.selectedID)
93             cursor.execute(sql)
94             db.commit()
```

#### 4.10.8 Module 8 - Initialising View Menu Layout

```
1 from PyQt4.QtCore import *
2 from PyQt4.QtGui import *
3 import sqlite3
4 import sys
5
6 class dbViewWindow(QWidget):
7     """generic view window"""
8
9     def __init__(self):
```

```
10     super().__init__()
11
12     def View(self):
13         self.setWindowTitle("View Menu")
14         self.setFixedSize(735,400)
15
16         self.btnExit = QPushButton("Back", self)
17         self.btnExit.setFixedSize(100, 30)
18         self.btnViewRoyalties = QPushButton("View Royalties", self)
19         self.btnViewRoyalties.setFixedSize(100, 40)
20         self.btnViewBookInvoices = QPushButton("View Book \n Invoices", self)
21         self.btnViewBookInvoices.setFixedSize(100, 40)
22         self.btnViewPubInvoice = QPushButton("View Publishing \n Invoice", self)
23         self.btnViewPubInvoice.setFixedSize(100, 40)
24         self.btnAddBook = QPushButton("Add Book", self)
25         self.btnAddBook.setFixedSize(100, 40)
26         self.btnUpdateBook = QPushButton("Update Book", self)
27         self.btnUpdateBook.setFixedSize(100,40)
28         self.btnDeleteBook = QPushButton("Delete Book", self)
29         self.btnDeleteBook.setFixedSize(100, 40)
29
30         self.horizontalTop = QHBoxLayout()
31         self.horizontalTop.addStretch(1)
32         self.horizontalTop.addWidget(self.btnExit)
33
34         self.horizontalBottom = QHBoxLayout()
35         self.horizontalBottom.addWidget(self.btnViewPubInvoice)
36         self.horizontalBottom.addWidget(self.btnViewBookInvoices)
37         self.horizontalBottom.addWidget(self.btnViewRoyalties)
38         self.horizontalBottom.addWidget(self.btnAddBook)
39         self.horizontalBottom.addWidget(self.btnUpdateBook)
40         self.horizontalBottom.addWidget(self.btnDeleteBook)
41
42
43         self.vertical = QVBoxLayout()
```

---

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

---

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

```
3 import sqlite3
4 import sys
5 import re
6
7
8 class dbAddItemWindow(QDialog):
9     """add entry window dialog"""
10
11     def __init__(self):
12         super().__init__()
13
14     def initAddItemWindow(self):
15         self.setWindowTitle("Add {}".format(self.AddType))
16         if self.Edition == True:
17             self.setWindowTitle("Edit {}".format(self.AddType))
18         self.ReadyToVerify = True
19         self.setModal(True) #modal window
20         self.Calculated = False
21         with sqlite3.connect("PP.db") as db: #fetching data from db
22             cursor = db.cursor()
23             cursor.execute(self.sql)
24             db.commit()
25
26         self.Columns = []
27         self.ColumnNames = cursor.description
28
29         for count in range(0, len(self.ColumnNames)):
30             self.Columns.append(list(list(self.ColumnNames)[count])[0])
31         self.coordinates = []
32
33         for count in range(int(round(len(self.Columns)/2, 1) + 1)):
34             for count2 in range(4):
35                 self.coordinates.append((count, count2))
36
37         self.ColumnLength = len(self.Columns)
38         for count in range(0, self.ColumnLength):
39             self.Columns.insert((count * 2) + 1, "")
40
41         db.close()
42         self.inputList = []
43         self.qlabelList = []
44         self.gridLayout = QGridLayout()
```

298

```
45     count = 0
46
47     for self.coordinate, self.columnHeader in zip(self.coordinates, self.Columns):
48
49         if self.columnHeader == "": #replacing spaces with line edits
50
51             if self.AddType == "Book" and count in [0, 1, 3, 4, 5, 6, 7, 9, 10, 11]:
52                 if count == 0:
53                     self.input = QLineEdit(self)
54                     self.input.setValidator(QRegExpValidator(QRegExp("[\w]+")))
55                 elif count == 1: #exceptions for book
56                     self.input = QLineEdit(self) #new line edit
57                     self.input.setReadOnly(True)
58                     self.input.setText(self.selectedID)
59                 elif count in [3, 9, 11]:
60                     self.input = QLineEdit(self)
61                     if count == 3:
62                         self.input.setValidator(QIntValidator())
63                     else:
64                         self.input.setValidator(QDoubleValidator())
65                 elif count in [4, 5, 6, 7]: #exceptions for book where combobox is needed
66                     self.input = QComboBox(self) #new combo box
67
68             elif count == 10:
69                 self.input = QLineEdit(self)
70                 self.input.setReadOnly(True)
71
72         elif self.AddType == "PubInvoice" and count in [0, 1, 2, 3, 4]:
73
74             if count == 0: #setting isbn ineditable
75                 self.input = QLineEdit(self)
76                 self.input.setReadOnly(True)
77                 self.input.setText(self.selectedISBN)
78
79             elif count == 1: #setting authorID ineditable
80                 self.input = QLineEdit(self)
81                 self.input.setReadOnly(True)
82                 self.input.setText(self.selectedID)
83
84             elif count == 2:
85                 self.input = QLineEdit(self)
86                 self.input.setReadOnly(True)
```

299

Imran Rahman

Candidate No. 30928

Centre No. 22151

```
87
88         elif count == 3:
89             self.input = QComboBox(self)
90
91         elif count == 4:
92             self.input = QLineEdit(self)
93             self.input.setValidator(QDoubleValidator())
94
95     elif self.AddType in ["BookInvoice", "Royalties"] and count in [0, 1]:
96
97         self.input = QLineEdit(self)
98         self.input.setReadOnly(True)
99
100        if count == 0:
101            self.input.setText(self.selectedID)
102
103    elif self.AddType == "BookInvoiceItems" and count in [0, 1, 2, 3, 4, 5]:
104
105        if count == 0:
106            self.input = QLineEdit(self)
107            self.input.setReadOnly(True)
108            self.input.setText(self.selectedID)
109
110        elif count == 1:
111            self.input = QLineEdit(self)
112            self.input.setReadOnly(True)
113            self.input.setText(self.selectedISBN)
114
115        elif count in [2, 3, 5]:
116            self.input = QLineEdit(self)
117            if count == 2:
118                self.input.setValidator(QIntValidator())
119            else:
120                self.input.setValidator(QDoubleValidator())
121
122    elif self.AddType == "RoyaltyItems" and count in [0, 1, 3, 4, 5, 6, 7]:
123        with sqlite3.connect("PP.db") as db:
124            cursor = db.cursor()
125            Selection = "NoOfPages, Size, Back"
126            sql = "select {} from Book where ISBN = {}".format(Selection, self.selectedISBN)
127            cursor.execute(sql)
128            self.SelectionList = list(cursor.fetchone())
129            self.NoOfPages = int(self.SelectionList[0])
```

```
129         self.Size = self.SelectionList[1]
130         self.Back = self.SelectionList[2]
131
132         if self.Size == "Large":
133             self.PagePrice = 0.015 * self.NoOfPages
134             if self.Back == "Hard":
135                 self.CoverPrice = 5
136             elif self.Back == "Soft":
137                 self.CoverPrice = 1
138
139         elif self.Size == "Small":
140             self.PagePrice = 0.01 * self.NoOfPages
141             if self.Back == "Hard":
142                 self.CoverPrice = 4
143             elif self.Back == "Soft":
144                 self.CoverPrice = 0.7
145
146         self.input = QLineEdit(self)
147
148         if count == 0:
149             self.input.setText(self.selectedID)
150             self.input.setReadOnly(True)
151         elif count == 1:
152             self.input.setText(self.selectedISBN)
153             self.input.setReadOnly(True)
154         elif count in [3, 4, 5, 6, 7]:
155             if count == 5:
156                 self.input.setValidator(QIntValidator())
157             elif count == 6:
158                 self.input.setReadOnly(True)
159                 self.input.setValidator(QDoubleValidator())
160             else:
161                 self.input.setValidator(QDoubleValidator())
162         else:
163             self.input = QLineEdit(self) #new line edit #standard input method
164
165         self.inputList.append(self.input) #line edits/combo boxes appended to list for further reference
166         self.gridLayout.addWidget(self.inputList[count], *self.coordinate)
167
168         count += 1
169
170     else: #adding qlabels with the line edits
```

```
171     self.DateEntry = False
172     if self.AddType == "Book" and count == 10: #adding date button instead of qlabel
173         self.DateEntry = True
174     elif self.AddType == "PubInvoice" and count == 2: #data button instead of qlabel
175         self.DateEntry = True
176     elif self.AddType == "BookInvoice" and count == 1:
177         self.DateEntry = True
178     elif self.AddType == "Royalties" and count == 1:
179         self.DateEntry = True
180
181 else:
182     if str(self.columnHeader) == "PubInvoiceService":
183         self.QLabel = QLabel("Service", self)
184     elif str(self.columnHeader) in ["PubInvoicePayment", "BookInvoicePayment", "RoyaltyPayment"]:
185         self.QLabel = QLabel("Payment", self)
186     elif str(self.columnHeader) in ["BookInvoiceDiscount", "RoyaltyDiscount"]:
187         self.QLabel = QLabel("Discount", self)
188     elif str(self.columnHeader) in ["BookInvoiceQuantity", "RoyaltyQuantity"]:
189         self.QLabel = QLabel("Quantity", self)
190     elif str(self.columnHeader) == "ExcRateFromGBP":
191         self.QLabel = QLabel("£1 = ", self)
192     elif str(self.columnHeader)[-2:] in ["ID", "BN"]:
193         self.QLabel = QLabel(str(self.columnHeader))
194
195 else:
196     self.Label = str(self.columnHeader)
197     self.LabelLength = len(self.Label)
198     self.CamelCase = False
199     for count2 in range(1, self.LabelLength):
200         if self.CamelCase == True:
201             pass
202         else:
203             self.CamelCase = False
204         if self.Label[count2].isupper() == True:
205             self.String1 = re.sub('([A-Z][a-z]+)', r'\1 \2', self.Label[0:count2])
206             self.String2 = re.sub('([A-Z][a-z]+)', r'\1 \2', self.Label[count2:self.LabelLength])
207             self.tempLabel = "{} {}".format(self.String1, self.String2)
208             self.CamelCase = True
209
210     if self.CamelCase == True:
211         self.Label = self.tempLabel
212
```

```
213             self.QLabel = QLabel(str(self.Label), self)
214             self.QLabelList.append(self.QLabel)
215             self.gridLayout.addWidget(self.QLabelList[count], *self.coordinate)
216
217         if self.DateEntry == True:
218             self.btnDate = QPushButton("Date", self)
219             self.QLabelList.append(self.btnDate)
220             self.gridLayout.addWidget(self.QLabelList[count], *self.coordinate)
221             self.btnDate.clicked.connect(self.CalendarWidget.DisplayCalendar)
222             self.CalendarWidget.btnSelect.clicked.connect(self.getDate)
223
224         if self.AddType == "Book":
225             self.inputList[4].addItem("Large")
226             self.inputList[4].addItem("Small")
227             self.inputList[5].addItem("Hard")
228             self.inputList[5].addItem("Soft")
229             self.inputList[6].addItem("Matte")
230             self.inputList[6].addItem("Gloss")
231             self.inputList[7].addItem("White")
232             self.inputList[7].addItem("Creme")
233
234     elif self.AddType == "PubInvoice":
235         self.inputList[3].addItem("Standard")
236         self.inputList[3].addItem("Enhanced")
237         self.inputList[3].addItem("Colour Publishing")
238         self.inputList[3].addItem("Reprint")
239
240     elif self.AddType == "BookInvoiceItems":
241         self.inputList[4].addItem("Rush")
242         self.inputList[4].addItem("Premium")
243         self.inputList[4].addItem("Standard")
244         self.inputList[4].addItem("Economy")
245         self.inputList[4].addItem("International")
246
247     if self.Editing == True:
248
249         if self.AddType == "Book":
250
251             for count in range(0, 12):
252                 try: #for line edits
253                     self.inputList[count].setText(str(self.originalItemList[count]))
254                 except: #for comboboxes
```

```
255         self.originalIndex = self.inputList[count].findText(str(self.originalItemList[count]))
256         self.inputList[count].setCurrentIndex(self.originalIndex)
257
258     elif self.AddType == "PubInvoice":
259
260         for count in range(0, 5):
261             try:
262                 self.inputList[count].setText(str(self.originalItemList[count]))
263             except:
264                 self.originalIndex = self.inputList[count].findText(str(self.originalItemList[count]))
265                 self.inputList[count].setCurrentIndex(self.originalIndex)
266
267     elif self.AddType in ["BookInvoice", "Royalties"]:
268
269         for count in range(0, 2):
270             self.inputList[count].setText(str(self.originalItemList[count]))
271     elif self.AddType == "BookInvoiceItems":
272         for count in range(0, 6):
273             try:
274                 self.inputList[count].setText(str(self.originalItemList[count]))
275             except:
276                 self.originalIndex = self.inputList[count].findText(str(self.originalItemList[count]))
277                 self.inputList[count].setCurrentIndex(self.originalIndex)
278     elif self.AddType == "RoyaltyItems":
279         for count in range(0, 8):
280             self.inputList[count].setText(str(self.originalItemList[count]))
281
282     self.horizontal = QHBoxLayout()
283     if self.AddType in ["BookInvoiceItems", "RoyaltyItems"]:
284         self.horizontal.addWidget(self.btnCalculate)
285         self.horizontal.addWidget(self.qleCalculation)
286         self.horizontal.addStretch(1)
287         self.horizontal.addWidget(self.btnCancel)
288         self.horizontal.addWidget(self.btnConfirm)
289     if self.AddType in ["BookInvoiceItems", "RoyaltyItems"]:
290         if self.Editing == False:
291             self.btnConfirm.clicked.connect(self.CheckCalculated)
292     if self.Editing == False:
293         self.btnConfirm.clicked.connect(self.Validate)
294     self.vertical = QVBoxLayout()
295     self.vertical.addLayout(self.gridLayout)
296     self.vertical.addLayout(self.horizontal)
```

```

297         self.setLayout(self.vertical)
298
299
300     self.btnCancel.clicked.connect(self.reject) #reject on clicking cancel
301     self.exec_()
302
303     def CheckCalculated(self):
304         self.ReadyToVerify = False
305         if len(self.qleCalculation.text()) == 0:
306             self.Msg = QMessageBox()
307             self.Msg.setWindowTitle("Calculation")
308             self.Msg.setText("You must fill all fields and click 'Calculate' before attempting to add to the database.")
309             self.Msg.exec_()
310         else:
311             self.ReadyToVerify = True
312
313     def AnswerButtons(self): #so connections can be made outside of this class
314         self.btnConfirm = QPushButton("Confirm", self)
315         self.btnCancel = QPushButton("Cancel", self)
316         if self.AddType in ["BookInvoiceItems", "RoyaltyItems"]:
317             self.btnCalculate = QPushButton("Calculate", self)
318             self.qleCalculation = QLineEdit(self)
319             self.qleCalculation.setReadOnly(True)
320
321     def getDate(self):
322         self.CalendarWidget.date = self.CalendarWidget.qle.text()
323         if self.AddType == "Book":
324             self.inputList[10].setText(self.CalendarWidget.date)
325
326         elif self.AddType == "PubInvoice":
327             self.inputList[2].setText(self.CalendarWidget.date)
328
329         elif self.AddType in ["BookInvoice", "Royalties"]:
330             self.inputList[1].setText(self.CalendarWidget.date)
331
332     def keyReleaseEvent(self, QKeyEvent):
333         if self.AddType == "RoyaltyItems":
334             if self.inputList[2].text() == "£":
335                 self.inputList[7].setText("N/A")
336                 self.inputList[7].setReadOnly(True)
337             elif self.inputList[7].text() == "N/A":
338                 self.inputList[7].setText("")

```

```
339         self.inputList[7].setReadOnly(False)
340     if self.inputList[5].text() != "":
341         self.PrintCost = (self.PagePrice + self.CoverPrice) * int(self.inputList[5].text())
342         self.inputList[6].setText("{:.2f}".format(self.PrintCost))
343     elif self.inputList[5].text() == "":
344         self.inputList[6].clear()
345
346
347
348     def Validate(self):
349         if self.ReadyToVerify == True:
350             self.input_data = []
351             self.Valid = True
352             for count in range(0, len(self.inputList)):
353                 try: #gathering the input data
354                     self.input_data.append(str(self.inputList[count].currentText()))
355                 except:
356                     if count == 8 and self.AddType == "RoyaltyItems":
357                         self.input_data.append(self.NetSales)
358                     else:
359                         self.input_data.append(self.inputList[count].text())
360
361             for count in range(0, len(self.input_data)):
362
363                 if str(self.input_data[count]).replace(" ", "") == "": #presence check
364                     self.Valid = False
365                     self.ErrorMessage = "All Fields must be filled."
366                     break
367                 try:
368                     if float(self.input_data[count]) < 0: #range check
369                         self.Valid = False
370                         self.ErrorMessage = "Invalid Entry - Please check the fields."
371                         break
372                 except:
373                     pass
374
375                 if self.qlabelList[count].text() == "ISBN":
376                     if len(self.input_data[count]) != 13 and len(self.input_data[count]) != 10: #isbn must be 10 or 13
377                         digits
378                         self.Valid = False
379                         self.ErrorMessage = "Invalid ISBN - Must be 10 or 13 digits."
380                         break
```

```
380     elif self.qlabelList[count].text() == "No Of Pages":
381         if int(self.input_data[count]) > 2000:
382             self.Valid = False
383             self.ErrorMessage = "Invalid Entry - Please check the fields."
384             break
385         elif self.qlabelList[count].text() == "Discount": #%'s must be between 0 and 100
386             if float(self.input_data[count]) > 100 or float(self.input_data[count]) < 0:
387                 self.Valid = False
388                 self.ErrorMessage = "Invalid Entry - Please check the fields."
389                 break
390
391     if self.Valid == False:
392         self.Msg = QMessageBox()
393         self.Msg.setWindowTitle("Invalid Entry")
394         self.Msg.setText(self.ErrorMessage)
395         self.Msg.exec_()
396     else:
397         if self.AddType not in ["BookInvoiceItems", "RoyaltyItems"]:
398             if self.Editing == False:
399                 self.accept()
400             else:
401                 self.ReadyToVerify = True
```

#### 4.10.10 Module 10 - Initialising Calendar Widget

```
1 import sys
2 from PyQt4.QtCore import *
3 from PyQt4.QtGui import *
4
5 class dbCalendarWidget(QDialog):
6
7     def __init__(self):
8         super().__init__()
9
10    def Calendar(self):
11        self.setFixedSize(265, 275)
12        self.setWindowTitle('Calendar')
13        calendar = QCalendarWidget(self)
14        calendar.setGridVisible(True)
```

```
15     calendar.clicked[QDate].connect(self.date)
16     date = calendar.selectedDate()
17     self.lblInstruction = QLabel(self)
18     self.lblInstruction.setText("Please select a date")
19     self.lblInstruction.setAlignment(Qt.AlignCenter)
20     self.qle = QLineEdit(self)
21     self.qle.setText(date.toString("dd-MM-yyyy"))
22     self.qle.setFixedSize(85,20)
23     self.qle.setAlignment(Qt.AlignCenter)
24     self.btnSelect = QPushButton("Select", self)
25     self.btnCancel = QPushButton("Cancel", self)
26
27     self.horizontalTop = QHBoxLayout()
28     self.horizontalTop.addStretch(1)
29     self.horizontalTop.addWidget(self.lblInstruction)
30     self.horizontalTop.addStretch(1)
31
32     self.horizontalMid1 = QHBoxLayout()
33     self.horizontalMid1.addStretch(1)
34     self.horizontalMid1.addWidget(calendar)
35     self.horizontalMid1.addStretch(1)
36
37     self.horizontalMid2 = QHBoxLayout()
38     self.horizontalMid2.addStretch(1)
39     self.horizontalMid2.addWidget(self.qle)
40     self.horizontalMid2.addStretch(1)
41
42     self.horizontalBottom = QHBoxLayout()
43     self.horizontalBottom.addWidget(self.btnCancel)
44     self.horizontalBottom.addStretch(1)
45     self.horizontalBottom.addWidget(self.btnSelect)
46
47     self.vertical = QVBoxLayout()
48     self.vertical.setLayout(self.horizontalTop)
49     self.vertical.setLayout(self.horizontalMid1)
50     self.vertical.setLayout(self.horizontalMid2)
51     self.vertical.setLayout(self.horizontalBottom)
52     self.setLayout(self.vertical)
53
54
55     def DisplayCalendar(self):
56         self.setModal(True)
```

308

```
57         self.btnExit.clicked.connect(self.accept)
58         self.btnCancel.clicked.connect(self.reject)
59         self.exec_()
60
61     def date(self, date):
62         self.qle.setText(date.toString("dd-MM-yyyy"))
```

#### 4.10.11 Module 11 - Initialising a Confirmation Dialog

```
1  from PyQt4.QtCore import *
2  from PyQt4.QtGui import *
3  import sqlite3
4  import sys
5
6  class dbConfirmationDialog(QDialog):
7      """Creating confirmation modal dialogs"""
8
9      def __init__(self):
10          super().__init__()
11          self.Prevention = False
12
13      def VerifyDlg(self):
14          self.setWindowTitle("Verification")
15          self.setFixedSize(275, 150)
16          self.setModal(True)
17          self.lblWarningMsg = QLabel(self.Msg, self)
18          self.horizontal = QHBoxLayout()
19          self.horizontal.addStretch(1)
20          self.horizontal.addWidget(self.lblWarningMsg)
21          self.horizontal.addStretch(1)
22          self.lblWarningMsg.setFixedSize(250,30)
23          self.lblWarningMsg.setWordWrap(True)
24          self.lblWarningMsg.setAlignment(Qt.AlignHCenter)
25
26          self.qlePasswordBox = QLineEdit(self)
27          self.qlePasswordBox.setFixedSize(100, 25)
28          self.qlePasswordBox.setEchoMode(self.qlePasswordBox.Password)
29          self.lblPassword = QLabel("Password: ", self)
30
```

```
31         self.horizontal1 = QHBoxLayout()
32         self.horizontal1.addStretch(1)
33         self.horizontal1.addWidget(self.lblPassword)
34         self.horizontal1.addWidget(self.qlePasswordBox)
35         self.horizontal1.addStretch(1)
36
37         self.btnConfirm = QPushButton("Confirm", self)
38         self.btnCancel = QPushButton("Cancel", self)
39         self.horizontal2 = QHBoxLayout()
40         self.horizontal2.addWidget(self.btnCancel)
41         self.horizontal2.addWidget(self.btnConfirm)
42
43         self.vertical = QVBoxLayout()
44         self.vertical.setLayout(self.horizontal)
45         self.vertical.setLayout(self.horizontal1)
46         self.vertical.setLayout(self.horizontal2)
47         self.vertical.addStretch(1)
48         self.setLayout(self.vertical)
49
50
51         self.btnCancel.clicked.connect(self.reject)
52         self.ConfirmedDialog = dbConfirmationDialog()
53         self.ConfirmedDialog.ConfirmedMsg = self.ConfirmedMsg
54         self.lblInvalid = None
55         self.btnConfirm.clicked.connect(self.PasswordCheck)
56
57         self.ConfirmedDialog.Accepted = False
58         self.exec_()
59
60
61     def PasswordCheck(self):
62         with sqlite3.connect("dbLogin.db") as db:
63             cursor = db.cursor()
64             cursor.execute("select Password from LoginDetails")
65             self.Password = list(cursor.fetchall())
66             self.Valid = False
67
68             for count in range(0, len(self.Password)):
69                 if self.qlePasswordBox.text() == list(self.Password[count])[0]:
70                     self.accept()
71                     self.ConfirmedDialog.Accepted = True
72                     if self.Prevention == False:
```

```
73             self.ConfirmedDialog.Confirmed()
74         break
75     else:
76         self.Valid = False
77
78     if self.Valid == False:
79         if self.lblInvalid == None:
80             self.lblInvalid = QLabel("Invalid Username or Password - Please try again.", self)
81             self.lblInvalid.setWordWrap(True)
82             self.lblInvalid.setAlignment(Qt.AlignHCenter)
83             self.horizontalInvalid = QHBoxLayout()
84             self.horizontalInvalid.addStretch(1)
85             self.horizontalInvalid.addWidget(self.lblInvalid)
86             self.horizontalInvalid.addStretch(1)
87             self.vertical.addLayout(self.horizontalInvalid)
88         else:
89             self.lblInvalid.show()
90
91
92     def Confirmed(self):
93         self.setWindowTitle("Confirmation")
94         self.setFixedSize(275, 100)
95         self.setModal(True)
96
97         self.lblConfirmed = QLabel(self.ConfirmedMsg, self)
98         self.lblConfirmed.setFixedSize(250, 50)
99         self.lblConfirmed.setWordWrap(True)
100        self.lblConfirmed.setAlignment(Qt.AlignHCenter)
101
102        self.btnOk = QPushButton("OK", self)
103        self.btnOk.setFixedSize(75, 30)
104        self.btnOk.clicked.connect(self.accept)
105
106        self.horizontal = QHBoxLayout()
107        self.horizontal.addStretch(1)
108        self.horizontal.addWidget(self.btnOk)
109        self.horizontal.addStretch(1)
110
111        self.vertical = QVBoxLayout()
112        self.vertical.addWidget(self.lblConfirmed)
113        self.vertical.addStretch(1)
114        self.vertical.addLayout(self.horizontal)
```

```
115         self.setLayout(self.vertical)
116         self.exec_()
117
118     def keyReleaseEvent(self, QKeyEvent):
119         if self.lblInvalid != None:
120             self.lblInvalid.hide()
```

---

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

```
1 from PyQt4.QtCore import *
2 from PyQt4.QtGui import *
3 import sqlite3
4 import sys
5
6 class dbItems(QDialog):
7     """viewing royalty and invoice items"""
8
9     def __init__(self):
10         super().__init__()
11         self.BookEdited = False
12     def BookInvoiceItems(self):
13         self.setWindowTitle("View Book Invoice Items")
14         self.setModal(True)
15         self.setFixedSize(640, 220)
16         self.vertical = QVBoxLayout(self)
17         self.vertical.addWidget(self.table)
18         self.btnCalculate.setFixedSize(100, 40)
19         self.btnUpdateBookInvoiceItems.setFixedSize(100, 40)
20         self.btnDeleteEntry.setFixedSize(100, 40)
21         self.horizontal = QHBoxLayout()
22         self.horizontal.addWidget(self.btnCalculate)
23         self.horizontal.addStretch(1)
24         self.horizontal.addWidget(self.btnUpdateBookInvoiceItems)
25         self.horizontal.addStretch(1)
26         self.horizontal.addWidget(self.btnDeleteEntry)
27         self.vertical.addLayout(self.horizontal)
28         self.setLayout(self.vertical)
29
```

```
30         self.exec_()
31
32     def BookInvoiceItemsButtons(self):
33         self.btnCalculate = QPushButton("Calculate", self)
34         self.btnUpdateBookInvoiceItems = QPushButton("Update Book \n Invoice Items", self)
35         self.btnDeleteEntry = QPushButton("Delete \n Entry", self)
36
37     def RoyaltiesItems(self):
38         self.setWindowTitle("View Royalty Items")
39         self.setModal(True)
40         self.setFixedSize(640, 220)
41         self.vertical = QVBoxLayout(self)
42         self.vertical.addWidget(self.table)
43         self.btnCalculate.setFixedSize(100, 40)
44         self.btnUpdateRoyaltyItems.setFixedSize(100, 40)
45         self.btnDeleteEntry.setFixedSize(100, 40)
46         self.horizontal = QHBoxLayout()
47         self.horizontal.addWidget(self.btnCalculate)
48         self.horizontal.addStretch(1)
49         self.horizontal.addWidget(self.btnUpdateRoyaltyItems)
50         self.horizontal.addStretch(1)
51         self.horizontal.addWidget(self.btnDeleteEntry)
52         self.vertical.addLayout(self.horizontal)
53         self.setLayout(self.vertical)
54
55         self.exec_()
56
57     def RoyaltyItemsButtons(self):
58         self.btnCalculate = QPushButton("Calculate", self)
59         self.btnUpdateRoyaltyItems = QPushButton("Update \n Royalty Items", self)
60         self.btnDeleteEntry = QPushButton("Delete \n Entry", self)
61
62
63     def CalculateBookInvoiceItems(self):
64         with sqlite3.connect("PP.db") as db:
65             cursor = db.cursor()
66             cursor.execute("PRAGMA foreign_keys_ = ON")
67             self.Empty = False
68             self.BookInvoicePayment = 0
69
70             self.IDList = []
71             sql = "select BookInvoiceItemsID from BookInvoiceItems where BookInvoiceID = {}".format(self.selectedID)
```

```
72     cursor.execute(sql)
73     self.IDListTuple = list(cursor.fetchall())
74     for count in range(0, len(self.IDListTuple)): #conversion from tuple
75         self.IDList.append(list(self.IDListTuple[count])[0])
76
77     self.currentID = 0
78     for count in range(0, len(self.IDList)):
79         if self.currentID < self.IDList[count]: #finding biggest ID no.
80             self.currentID = self.IDList[count]
81
82     sql = "select ISBN from BookInvoiceItems where BookInvoiceID = {}".format(self.selectedID)
83     cursor.execute(sql)
84     self.ISBNs = list(cursor.fetchall())
85
86     for count2 in range(0, len(self.ISBNs)):
87         if self.currentID != 0:
88             for count in range(0, self.currentID +1):
89                 self.Empty = False
90                 Selection = "BookInvoiceItems.BookInvoiceQuantity, BookInvoiceItems.BookInvoiceDiscount, "
91                 Selection += "BookInvoiceItems.ShippingPrice, Book.Price"
92                 Tables = "BookInvoiceItems, Book"
93                 sql = "select {} from {} where BookInvoiceItems.BookInvoiceID = {} and "
94                 sql += "BookInvoiceItems.BookInvoiceItemsID = {} and BookInvoiceItems.ISBN = {} and Book.ISBN = "
95                 sql += "list({})[{}])".format(Selection, Tables, self.selectedID, count+1,
96                 self.ISBNs[count2][0])
97                 cursor.execute(sql)
98             try:
99                 self.SelectionList = list(cursor.fetchone())
100            except:
101                self.Empty = True
102
103            if self.Empty != True:
104                self.Quantity = self.SelectionList[0]
105                self.Discount = self.SelectionList[1] / 100
106                self.ShippingPrice = self.SelectionList[2]
107                self.Price = self.SelectionList[3]
108
109                self.TempPayment = (self.Quantity * self.Price)
110                self.Discount *= self.TempPayment
111                self.TempPayment -= self.Discount
112                self.TempPayment += self.ShippingPrice
```

```
110                     self.BookInvoicePayment += self.TempPayment
111
112
113     elif self.currentID == 0:
114         self.BookInvoicePayment = None
115         sql = "update BookInvoice set BookInvoicePayment = '{}' where BookInvoiceID =
116             {}".format(self.BookInvoicePayment, self.selectedID)
117         cursor.execute(sql)
118         db.commit()
119
120     if self.currentID != 0:
121         self.BookInvoicePayment = "{0:.2f}".format(self.BookInvoicePayment)
122         sql = "update BookInvoice set BookInvoicePayment = '{}' where BookInvoiceID =
123             {}".format(self.BookInvoicePayment, self.selectedID)
124         cursor.execute(sql)
125         db.commit()
126
127     def CalculateRoyaltyItems(self):
128         with sqlite3.connect("PP.db") as db:
129             cursor = db.cursor()
130             cursor.execute("PRAGMA foreign_keys = ON")
131             self.Empty = False
132             self.RoyaltyPayment = 0
133             self.Currency = ""
134             self.IDList = []
135             sql = "select RoyaltyItemsID from RoyaltyItems where RoyaltiesID = {}".format(self.selectedID)
136             cursor.execute(sql)
137             self.IDListTuple = list(cursor.fetchall())
138             for count in range(0, len(self.IDListTuple)): #conversion from tuple
139                 self.IDList.append(list(self.IDListTuple[count])[0])
140
141             self.currentID = 0
142             for count in range(0, len(self.IDList)):
143                 if self.currentID < self.IDList[count]: #finding biggest ID no. for iteration limit
144                     self.currentID = self.IDList[count]
145
146             sql = "select ISBN from RoyaltyItems where RoyaltiesID = {}".format(self.selectedID)
147             cursor.execute(sql)
148             self.ISBNs = list(cursor.fetchall())
149
150             for count2 in range(0, len(self.ISBNs)):
151                 if self.currentID != 0:
```

```
150     for count in range(0, self.currentID+1):
151         self.Empty = False
152         Selection = "R royaltyItems.Currency, RoyaltyItems.NetSales, RoyaltyItems.ExcRateFromGBP,
153                     RoyaltyItems.RoyaltyQuantity, Book.NoOfPages, Book.Size, Book.Cover, Book.Back"
154         Tables = "RoyaltyItems, Book"
155         sql = "select {} from {} where RoyaltyItems.RoyaltiesID = {} and RoyaltyItems.RoyaltyItemsID = {}
156             and RoyaltyItems.ISBN = {} and Book.ISBN = RoyaltyItems.ISBN".format(Selection, Tables,
157             self.selectedID, count+1, list(self.ISBNs[count2])[0])
158         cursor.execute(sql)
159
160     try:
161         self.SelectionList = list(cursor.fetchone())
162     except:
163         self.Empty = True
164
165     if self.Empty != True:
166         self.Currency = self.SelectionList[0]
167         self.NetSales = float(self.SelectionList[1])
168         self.Quantity = self.SelectionList[3]
169         self.NoOfPages = int(self.SelectionList[4])
170         self.Size = self.SelectionList[5]
171         self.Cover = self.SelectionList[6]
172         self.Back = self.SelectionList[7]
173
174         if self.Size == "Large":
175             self.PagePrice = 0.015 * self.NoOfPages
176             if self.Back == "Hard":
177                 self.CoverPrice = 5
178             elif self.Back == "Soft":
179                 self.CoverPrice = 1
180             elif self.Size == "Small":
181                 self.PagePrice = 0.01 * self.NoOfPages
182                 if self.Back == "Hard":
183                     self.CoverPrice = 4
184                 elif self.Back == "Soft":
185                     self.CoverPrice = 0.7
186
187             self.PrintCost = (self.PagePrice + self.CoverPrice) * self.Quantity
188
189             self.TempPayment= self.NetSales - self.PrintCost
190             if self.Currency != "£":
191                 self.ExcRateFromGBP = float(self.SelectionList[2])
```

```
189             self.TempPayment /= self.ExcRateFromGBP
190             self.RoyaltyPayment += self.TempPayment
191
192         elif self.currentID == 0:
193             self.BookInvoicePayment = None
194             sql = "update Royalties set RoyaltyPayment = '{}' where RoyaltiesID = {}".format(self.RoyaltyPayment,
195                 self.selectedID)
196             cursor.execute(sql)
197             db.commit()
198
199         if self.currentID != 0:
200             self.RoyaltyPayment = "{0:.2f}".format(self.RoyaltyPayment)
201             sql = "update Royalties set RoyaltyPayment = '{}' where RoyaltiesID = {}".format(self.RoyaltyPayment,
202                 self.selectedID)
203             cursor.execute(sql)
204             db.commit()
```

---

#### 317 4.10.13 Module 13 - Initialising a dialog for viewing Royalties and Invoices

```
1 from PyQt4.QtCore import *
2 from PyQt4.QtGui import *
3 import sqlite3
4 import sys
5
6 class dbRoyaltiesAndInvoices(QDialog):
7     """viewing royalties and invoices"""
8
9     def __init__(self):
10         super().__init__()
11
12     def PubInvoice(self):
13         self.setWindowTitle("View Publishing Invoices")
14         self.setModal(True)
15         self.setFixedSize(640, 220)
16         self.vertical = QVBoxLayout(self)
17         self.vertical.addWidget(self.table)
18         self.btnAddPubInvoice.setFixedSize(100, 40)
19         self.btnUpdatePubInvoice.setFixedSize(100, 40)
20         self.btnDeleteEntry.setFixedSize(100, 40)
```

```
21         self.horizontal = QHBoxLayout()
22         self.horizontal.addWidget(self.btnAddPubInvoice)
23         self.horizontal.addStretch(1)
24         self.horizontal.addWidget(self.btnUpdatePubInvoice)
25         self.horizontal.addStretch(1)
26         self.horizontal.addWidget(self.btnDeleteEntry)
27         self.vertical.addLayout(self.horizontal)
28         self.setLayout(self.vertical)
29         self.exec_()
30
31     def PubInvoiceButtons(self):
32         self.btnAddPubInvoice = QPushButton("Add Publishing \n Invoice", self)
33         self.btnUpdatePubInvoice = QPushButton("Update Publishing \n Invoice", self)
34         self.btnDeleteEntry = QPushButton("Delete \n Entry", self)
35
36     def BookInvoice(self):
37         self.setWindowTitle("View Book Invoices")
38         self.setModal(True)
39         self.setFixedSize(640, 220)
40         self.vertical = QVBoxLayout(self)
41         self.vertical.addWidget(self.table)
42         self.btnAddBookInvoiceItems.setFixedSize(100, 40)
43         self.btnAddBookInvoice.setFixedSize(100, 40)
44         self.btnUpdateBookInvoice.setFixedSize(100, 40)
45         self.btnDeleteEntry.setFixedSize(100, 40)
46         self.horizontal = QHBoxLayout()
47         self.horizontal.addWidget(self.btnAddBookInvoiceItems)
48         self.horizontal.addStretch(1)
49         self.horizontal.addWidget(self.btnAddBookInvoice)
50         self.horizontal.addStretch(1)
51         self.horizontal.addWidget(self.btnUpdateBookInvoice)
52         self.horizontal.addStretch(1)
53         self.horizontal.addWidget(self.btnDeleteEntry)
54         self.vertical.addLayout(self.horizontal)
55         self.setLayout(self.vertical)
56         self.exec_()
57
58     def BookInvoiceButtons(self):
59         self.btnAddBookInvoiceItems = QPushButton("View Book \n Invoice Items", self)
60         self.btnAddBookInvoice = QPushButton("Add Book \n Invoice", self)
61         self.btnUpdateBookInvoice = QPushButton("Update Book \n Invoice", self)
62         self.btnDeleteEntry = QPushButton("Delete \n Entry", self)
```

```
63
64     def Royalties(self):
65         self.setWindowTitle("View Royalties")
66         self.setModal(True)
67         self.setFixedSize(640, 220)
68         self.vertical = QVBoxLayout(self)
69         self.vertical.addWidget(self.table)
70         self.btnViewRoyaltyItems.setFixedSize(100, 40)
71         self.btnAddRoyalties.setFixedSize(100, 40)
72         self.btnUpdateRoyalties.setFixedSize(100, 40)
73         self.btnDeleteEntry.setFixedSize(100, 40)
74         self.horizontal = QHBoxLayout()
75         self.horizontal.addWidget(self.btnViewRoyaltyItems)
76         self.horizontal.addStretch(1)
77         self.horizontal.addWidget(self.btnAddRoyalties)
78         self.horizontal.addStretch(1)
79         self.horizontal.addWidget(self.btnUpdateRoyalties)
80         self.horizontal.addStretch(1)
81         self.horizontal.addWidget(self.btnDeleteEntry)
82         self.vertical.addLayout(self.horizontal)
83         self.setLayout(self.vertical)
84         self.exec_()
85
86     def RoyaltiesButtons(self):
87         self.btnViewRoyaltyItems = QPushButton("View Royalty \n Items", self)
88         self.btnAddRoyalties = QPushButton("Add \n Royalties", self)
89         self.btnUpdateRoyalties = QPushButton("Update \n Royalties", self)
90         self.btnDeleteEntry = QPushButton("Delete \n Entry", self)
```

#### 4.10.14 Module 14 - Initialising a Search Window

```
1 from PyQt4.QtCore import *
2 from PyQt4.QtGui import *
3 import sqlite3
4 import sys
5
6
7 class dbSearchDatabase(QDialog):
8     """initialising the detailed search window"""
```

```
9
10    def __init__(self):
11        super().__init__()
12
13    def initLayout(self):
14        self.setWindowTitle("Search")
15        self.gridLayout = QGridLayout()
16        self.gridLayout.setVerticalSpacing(10)
17        self.gridLayout.setHorizontalSpacing(10)
18        self.setFixedSize(400, 200)
19        self.leFirstname = QLineEdit(self)
20        self.leLastname = QLineEdit(self)
21        self.cbTable = QComboBox(self)
22        self.cbCategory = QComboBox(self)
23        self.btnDate = QPushButton("Add Date", self)
24        self.leSearch = QLineEdit(self)
25        self.btnCancel = QPushButton("Cancel", self)
26        self.btnSearch = QPushButton("Search", self)
27        self.btnDate.setFixedSize(75, 20)
28        self.lblFirstname = QLabel("Author Firstname:")
29        self.lblLastname = QLabel("Author Lastname:")
30        self.gridLayout.addWidget(self.leFirstname, 0, 3)
31        self.gridLayout.addWidget(self.leLastname, 1, 3)
32        self.gridLayout.addWidget(self.lblFirstname, 0, 2, Qt.AlignRight)
33        self.gridLayout.addWidget(self.lblLastname, 1, 2, Qt.AlignRight)
34        self.gridLayout.addWidget(self.cbTable, 0, 0)
35        self.gridLayout.addWidget(self.cbCategory, 2, 0)
36        self.gridLayout.addWidget(self.btnCancel, 2, 2, Qt.AlignHCenter)
37        self.gridLayout.addWidget(self.leSearch, 2, 3)
38        self.gridLayout.addWidget(self.btnSearch, 3, 2)
39        self.gridLayout.addWidget(self.btnDate, 3, 3)
40        self.setLayout(self.gridLayout)
41        self.cbTable.addItem("Author")
42        self.cbTable.addItem("Book")
43        self.cbTable.addItem("Publishing Invoice")
44        self.cbTable.addItem("Book Invoice")
45        self.cbTable.addItem("Book Invoice Items")
46        self.cbTable.addItem("Royalties")
47        self.cbTable.addItem("Royalty Items")
48        self.cbTable.activated[str].connect(self.ChangeCategories)
49        self.cbCategory.activated[str].connect(self.DateButton)
50        self.btnDate.hide()
```

320

```
51         self.leSearch.hide()
52         self.CalendarWidget.btnSelect.clicked.connect(self.getDate)
53         self.btnExit.clicked.connect(self.CalendarWidget.DisplayCalendar)
54         self.leSearch.setFixedSize(self.leSearch.sizeHint())
55         self.leFirstname.setFixedSize(self.leFirstname.sizeHint())
56         self.leLastname.setFixedSize(self.leLastname.sizeHint())
57         self.btnExit.clicked.connect(self.getSearchData)
58         self.btnCancel.clicked.connect(self.reject)
59         self.exec_()
60
61     def ChangeCategories(self):
62         self.btnExit.hide()
63         self.leSearch.show()
64         self.cbCategory.clear()
65         if self.cbTable.currentText() == "Author":
66             self.leSearch.hide()
67         elif self.cbTable.currentText() == "Book":
68             self.cbCategory.addItem("Book Title")
69             self.cbCategory.addItem("Price")
70             self.cbCategory.addItem("Date Published")
71
72         elif self.cbTable.currentText() == "Publishing Invoice":
73             self.cbCategory.addItem("Service")
74             self.cbCategory.addItem("Date")
75
76         elif self.cbTable.currentText() == "Book Invoice":
77             self.cbCategory.addItem("Date")
78             self.btnExit.show()
79
80         elif self.cbTable.currentText() == "Book Invoice Items":
81             self.cbCategory.addItem("Quantity")
82             self.cbCategory.addItem("Discount")
83             self.cbCategory.addItem("Shipping Type")
84
85         elif self.cbTable.currentText() == "Royalties":
86             self.cbCategory.addItem("Date")
87             self.btnExit.show()
88
89         elif self.cbTable.currentText() == "Royalty Items":
90             self.cbCategory.addItem("Quantity")
91             self.cbCategory.addItem("Discount")
92             self.cbCategory.addItem("Currency")
```

321

Imran Rahman

Candidate No. 30928

Centre No. 22151

```
93
94     def DateButton(self):
95         if self.cbCategory.currentText()[:4] == "Date":
96             self.btnDate.show()
97             self.leSearch.setReadOnly(True)
98         else:
99             self.btnDate.hide()
100            self.leSearch.setReadOnly(False)
101
102    def getDate(self):
103        self.CalendarWidget.date = self.CalendarWidget.qlc.text()
104        self.leSearch.setText(self.CalendarWidget.date)
105
106    def getSearchData(self):
107
108        self.Valid = True
109        self.Firstname = self.leFirstname.text()
110        self.Lastname = self.leLastname.text()
111        self.Table = self.cbTable.currentText().replace(" ", " ")
112        if self.Table == "PublishingInvoice":
113            self.Table = "PubInvoice"
322
114
115        if self.Table != "Author":
116            self.Category = self.cbCategory.currentText().replace(" ", " ")
117            self.Search = self.leSearch.text()
118
119        if self.Firstname.replace(" ", "") == "" or self.Lastname.replace(" ", "") == "" or self.Category == "" or
120            self.Search.replace(" ", "") == "":
121            self.Msg = QMessageBox()
122            self.Msg.setWindowTitle("Invalid Entry")
123            self.Msg.setText("You must fill in all fields.")
124            self.Msg.exec_()
125            self.Valid = False
126
127        if self.Category in ["Discount", "Quantity"]:
128
129            if self.Table == "RoyaltyItems":
130                self.Category = "Royalty{}".format(self.Category)
131
132            elif self.Table == "BookInvoiceItems":
133                self.Category = "BookInvoice{}".format(self.Category)
```

```
134
135     elif self.Category == "Date":
136         self.Category = "{}Date".format(self.Table)
137
138     elif self.Category == "Service":
139         self.Category = "PubInvoiceService"
140
141     if self.Table not in ["BookInvoiceItems", "RoyaltyItems"]:
142         if self.Table in ["PubInvoice", "Royalties", "BookInvoice"]:
143             self.sql = "select Customer.AuthorID, {0}ID from Customer, {0} where (Customer.Ffirstname like '{1}%'
144                         or Customer.Lastname like '{2}%' and {0}.{3} like '{4}%' and {0}.AuthorID =
145                         Customer.AuthorID)".format(self.Table, self.Firstname, self.Lastname, self.Category, self.Search)
146     elif self.Table == "Book":
147         self.sql = "select Customer.AuthorID, Book.AuthorID, Book.ISBN from Customer, {0} where
148                         (Customer.Ffirstname like '{1}%' or Customer.Lastname like '{2}%' and {0}.{3} like '{4}%' and
149                         {0}.AuthorID = Customer.AuthorID)".format(self.Table, self.Firstname, self.Lastname,
150                                         self.Category, self.Search)
151
152     else:
153         self.sql = "select Customer.AuthorID, Book.AuthorID, Book.ISBN, {0}.ISBN, {0}ID from Customer, Book, {0}
154                         where (Customer.Ffirstname like '{1}%' or Customer.Lastname like '{2}%' and {0}.{3} like '{4}%' and
155                         Customer.AuthorID = Book.AuthorID and Book.ISBN = {0}.ISBN)".format(self.Table, self.Firstname,
156                                         self.Lastname, self.Category, self.Search)
157
158     else:
159         if self.Firstname.replace(" ", "") == "" or self.Lastname.replace(" ", "") == "":
160             self.Msg = QMessageBox()
161             self.Msg.setWindowTitle("Invalid Entry")
162             self.Msg.setText("You must fill enter the Ffirstname and Lastname")
163             self.Msg.exec_()
164             self.Valid = False
165
166         self.Table = "Customer" #Author table is referred to as 'Customer'
167         self.sql = "select AuthorID from Customer where Ffirstname like '{0}%' or Lastname like
168                         '{1}%'.format(self.Firstname, self.Lastname)
169
170     if self.Valid == True:
171         with sqlite3.connect("PP.db") as db:
172             cursor = db.cursor()
173             cursor.execute(self.sql)
174             self.Results = list(cursor.fetchall())
175
176     self.accept()
```

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

```
1 from PyQt4.QtCore import *
2 from PyQt4.QtGui import *
3 import sys
4
5
6 class initSearchResultsMenu(QWidget):
7     """main window"""
8
9     def __init__(self):
10         super().__init__()
11         self.btnExit = QPushButton("Back", self)
12         self.btnExit.setFixedSize(100, 30)
13         self.horizontalTop = QHBoxLayout()
14         self.horizontalTop.addStretch(1)
15         self.horizontalTop.addWidget(self.btnExit)
16
17         self.vertical = QVBoxLayout()
18         self.vertical.setLayout(self.horizontalTop)
19         self.setLayout(self.vertical)
```

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

```
1 from PyQt4.QtCore import *
2 from PyQt4.QtGui import *
3 import sqlite3
4 import sys
5
6 class dbUsernameOrPassword(QDialog):
7     """main window"""
8
9     def __init__(self):
10         super().__init__()
11
12     def ChangeSelection(self):
13         self.setModal(True)
14         self.setFixedSize(400, 50)
15         self.setWindowTitle("Selection")
```

```
16     self.btnExit = QPushButton("Change Username", self)
17     self.btnPassword = QPushButton("Change Password", self)
18     self.btnCancel = QPushButton("Cancel", self)
19     self.btnCancel.clicked.connect(self.reject)
20     self.btnExit.clicked.connect(self.UsernameSelected)
21     self.btnPassword.clicked.connect(self.PasswordSelected)
22     self.gridLayout = QGridLayout()
23     self.gridLayout.addWidget(self.btnExit, 0, 0)
24     self.gridLayout.addWidget(self.btnPassword, 0, 1)
25     self.gridLayout.addWidget(self.btnCancel, 0, 2)
26     self.setLayout(self.gridLayout)
27     self.exec_()
28
29     def UsernameSelected(self):
30         self.Selection = "Username"
31         self.accept()
32
33     def PasswordSelected(self):
34         self.Selection = "Password"
35         self.accept()
```

325

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

```
1 from PyQt4.QtCore import *
2 from PyQt4.QtGui import *
3 import sqlite3
4 import sys
5
6 class dbChangeUsername(QDialog):
7     """Change Username confirmation"""
8
9     def __init__(self):
10         super().__init__()
11
12     def initChangeUsernameScreen(self):
13         self.setWindowTitle("Change Username")
14         self.setModal(True)
15         self.leOldUsername = QLineEdit(self)
16         self.leNewUsername = QLineEdit(self)
```

```
17     self.leNewUsername.setValidator(QRegExpValidator(QRegExp("^\w+\.\w+@\w+\.\w+\.\w+{2,4}$")))
18     self.leOldUsername.setValidator(QRegExpValidator(QRegExp("^\w+\.\w+@\w+\.\w+\.\w+{2,4}$")))
19 #reg ex from www.regexlib.com
20     self.leRetype = QLineEdit(self)
21     self.lblOld = QLabel("Old Username:", self)
22     self.lblNew = QLabel("New Username:", self)
23     self.lblRetype = QLabel("Retype New Username:", self)
24     self.leOldUsername.setPlaceholderText("Old Username")
25     self.leNewUsername.setPlaceholderText("New Username")
26     self.leRetype.setPlaceholderText("Retype New Username")
27     self.btnConfirm = QPushButton("Confirm")
28     self.btnCancel = QPushButton("Cancel")
29     self.horizontal = QHBoxLayout()
30     self.horizontal.addWidget(self.btnCancel)
31     self.horizontal.addWidget(self.btnConfirm)
32     self.gridLayout = QGridLayout()
33     self.gridLayout.addWidget(self.lblOld, 0, 0)
34     self.gridLayout.addWidget(self.lblNew, 1, 0)
35     self.gridLayout.addWidget(self.lblRetype, 2, 0)
36     self.gridLayout.addWidget(self.leOldUsername, 0, 1)
37     self.gridLayout.addWidget(self.leNewUsername, 1, 1)
38     self.gridLayout.addWidget(self.leRetype, 2, 1)
39     self.gridLayout.setLayout(self.horizontal, 3, 2)
40     self.setLayout(self.gridLayout)
41     self.btnCancel.clicked.connect(self.reject)
42     self.btnConfirm.clicked.connect(self.Check)
43     self.exec_()
44
45 def Check(self):
46     if self.leNewUsername.text() == self.leRetype.text():
47
48         if len(self.leNewUsername.text()) < 5:
49             self.Msg = QMessageBox()
50             self.Msg.setWindowTitle("Username")
51             self.Msg.setText("New Username was too short")
52             self.Msg.exec_()
53
54         with sqlite3.connect("dbLogin.db") as db:
55             cursor = db.cursor()
56             cursor.execute("select Username from LoginDetails")
57             self.oldUsername = list(cursor.fetchone())[0]
58
```

```
59         if self.leOldUsername.text() == self.oldUsername:
60             self.NewUsername = self.leNewUsername.text()
61             cursor.execute("update LoginDetails set Username = '{}' where Username =
62                             '{}'.format(self.NewUsername, self.Username))")
63             self.Msg = QMessageBox()
64             self.Msg.setWindowTitle("Username")
65             self.Msg.setText("Username was successfully changed")
66             self.Msg.exec_()
67             self.accept()
68
68     else:
69         self.Msg = QMessageBox()
70         self.Msg.setWindowTitle("Username")
71         self.Msg.setText("Old Username was incorrect")
72         self.Msg.exec_()
73
73     else:
74         self.Msg = QMessageBox()
75         self.Msg.setWindowTitle("Username")
76         self.Msg.setText("New Usernames did not match")
77         self.Msg.exec_()
```

327

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

```
1 from PyQt4.QtCore import *
2 from PyQt4.QtGui import *
3 import sqlite3
4 import sys
5
6 class dbChangePassword(QDialog):
7     """Change password confirmation"""
8
9     def __init__(self):
10         super().__init__()
11
12     def initChangePasswordScreen(self):
13         self.setWindowTitle("Change Password")
14         self.setModal(True)
15         self.leOldPassword = QLineEdit(self)
16         self.leNewPassword = QLineEdit(self)
```

```
17     self.leRetype = QLineEdit(self)
18     self.leOldPassword.setEchoMode(self.leOldPassword.Password)
19     self.leNewPassword.setEchoMode(self.leNewPassword.Password)
20     self.leRetype.setEchoMode(self.leRetype.Password)
21     self.lblOld = QLabel("Old Password:", self)
22     self.lblNew = QLabel("New Password:", self)
23     self.lblRetype = QLabel("Retype New Password:", self)
24     self.leOldPassword.setPlaceholderText("Old Password")
25     self.leNewPassword.setPlaceholderText("New Password")
26     self.leRetype.setPlaceholderText("Retype New Password")
27     self.btnConfirm = QPushButton("Confirm")
28     self.btnCancel = QPushButton("Cancel")
29     self.horizontal = QHBoxLayout()
30     self.horizontal.addWidget(self.btnCancel)
31     self.horizontal.addWidget(self.btnConfirm)
32     self.gridLayout = QGridLayout()
33     self.gridLayout.addWidget(self.lblOld, 0, 0)
34     self.gridLayout.addWidget(self.lblNew, 1, 0)
35     self.gridLayout.addWidget(self.lblRetype, 2, 0)
36     self.gridLayout.addWidget(self.leOldPassword, 0, 1)
37     self.gridLayout.addWidget(self.leNewPassword, 1, 1)
38     self.gridLayout.addWidget(self.leRetype, 2, 1)
39     self.gridLayout.setLayout(self.horizontal, 3, 2)
40     self.setLayout(self.gridLayout)
41
42     self.btnCancel.clicked.connect(self.reject)
43     self.btnConfirm.clicked.connect(self.Check)
44     self.exec_()
45
46 def Check(self):
47     if self.leNewPassword.text() == self.leRetype.text():
48
49         if len(self.leNewPassword.text()) < 5:
50             self.Msg = QMessageBox()
51             self.Msg.setWindowTitle("Password")
52             self.Msg.setText("New Password was too short")
53             self.Msg.exec_()
54         else:
55             with sqlite3.connect("dbLogin.db") as db:
56                 cursor = db.cursor()
57                 cursor.execute("select Password from LoginDetails")
58                 self.oldPassword = list(cursor.fetchone())[0]
```

```
59     if self.leOldPassword.text() == self.oldPassword:
60         self.Password = self.leNewPassword.text()
61         cursor.execute("update LoginDetails set Password = '{}' where Username =
62             '{}'.format(self.Password, self.Username))
63         self.Msg = QMessageBox()
64         self.Msg.setWindowTitle("Password")
65         self.Msg.setText("Password was successfully changed")
66         self.Msg.exec_()
67         self.accept()
68     else:
69         self.Msg = QMessageBox()
70         self.Msg.setWindowTitle("Password")
71         self.Msg.setText("Old Password was incorrect")
72         self.Msg.exec_()
73     else:
74         self.Msg = QMessageBox()
75         self.Msg.setWindowTitle("Password")
76         self.Msg.setText("New Passwords did not match")
77         self.Msg.exec_()
```

# **Chapter 5**

# **User Manual**

# Contents

5.1	Introduction . . . . .	332
5.1.1	Purpose of System . . . . .	332
5.1.2	Intended Audience . . . . .	332
5.2	Installation . . . . .	332
5.2.1	Prerequisite Installation . . . . .	332
5.2.2	System Installation . . . . .	333
5.2.3	Running the System . . . . .	342
5.3	Tutorial . . . . .	347
5.3.1	Introduction . . . . .	347
5.3.2	Assumptions . . . . .	347
5.3.3	Tutorial Questions . . . . .	348
5.3.4	Saving . . . . .	370
5.3.5	Limitations . . . . .	370
5.4	Error Recovery . . . . .	371
5.4.1	Invalid entries when calculating payments . . . . .	371
5.5	System Recovery . . . . .	372
5.5.1	Backing-up Data . . . . .	372
5.5.2	Restoring Data . . . . .	374

## 5.1 Introduction

### 5.1.1 Purpose of System

The purpose of the system is to simplify the management of customer and book information and be able to perform calculations to find royalty and book invoice payments.

The following features are included in the program:

- Displaying customer and book details
- Adding customers, books of the customer and book details, making every entry unique with the use of ID's
- Editing data through the use of the user interface
- Deleting data through the use of the user interface
- Searching for data using the interface

### 5.1.2 Intended Audience

The intended audience for my system was the director of Perfect Publishers Ltd, which is a publishing company. The system is specifically designed for the company, as it uses fixed prices for calculations of book invoice and royalty payments, which were used from the company website.

## 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 to 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

The system was originally designed for a laptop with the following 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

As the laptop meets the specifications of the system, the program should not have any problems running on it. The laptop also runs on windows 8, and as the system was developed on windows 7 and windows 8, it is able to run on both operating systems.

### **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'.



Figure 5.1: Open 'My Computer'

The user will then need to right click the CD-R, and click open.

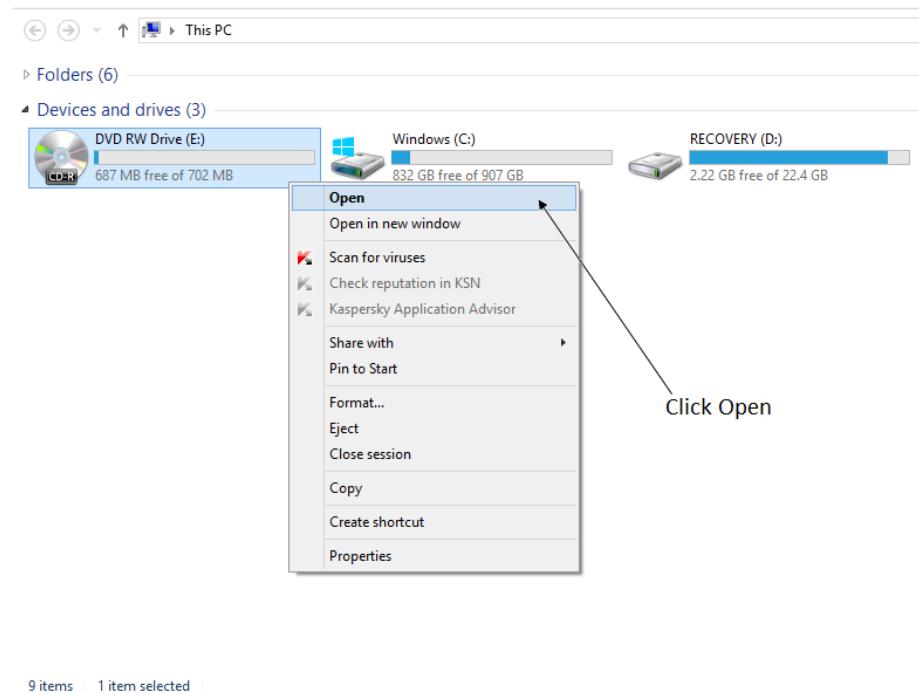


Figure 5.2: Opening the CD-R

Then right click on the installation file, and click 'Install'.

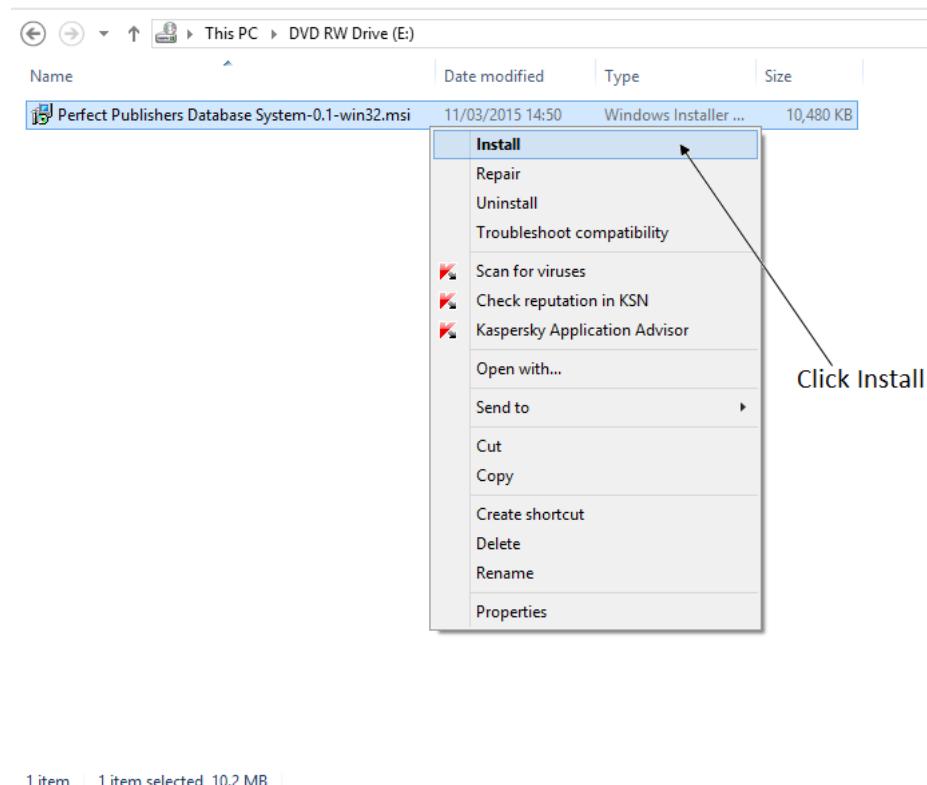


Figure 5.3: Clicking 'Install'

The Windows installer will commence installing. The user is required to select a destination to be installed at, with the default directory given.

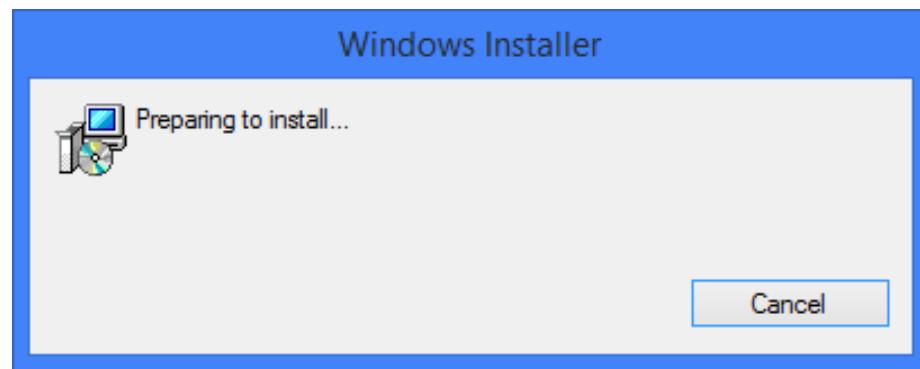


Figure 5.4: Installing begins

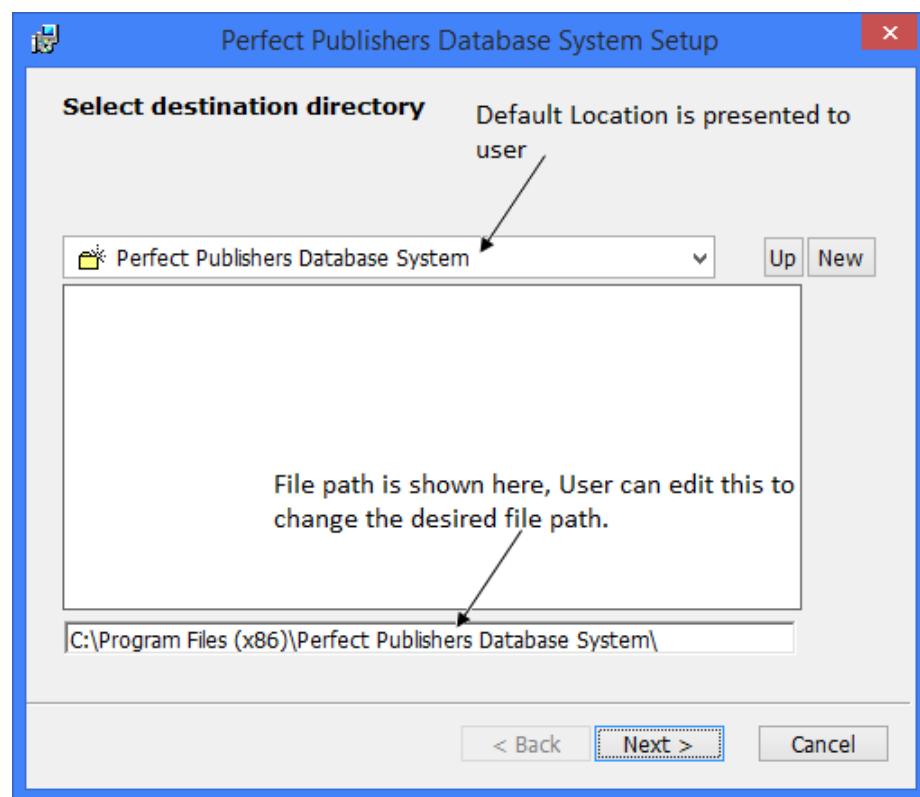


Figure 5.5: Setup Screen

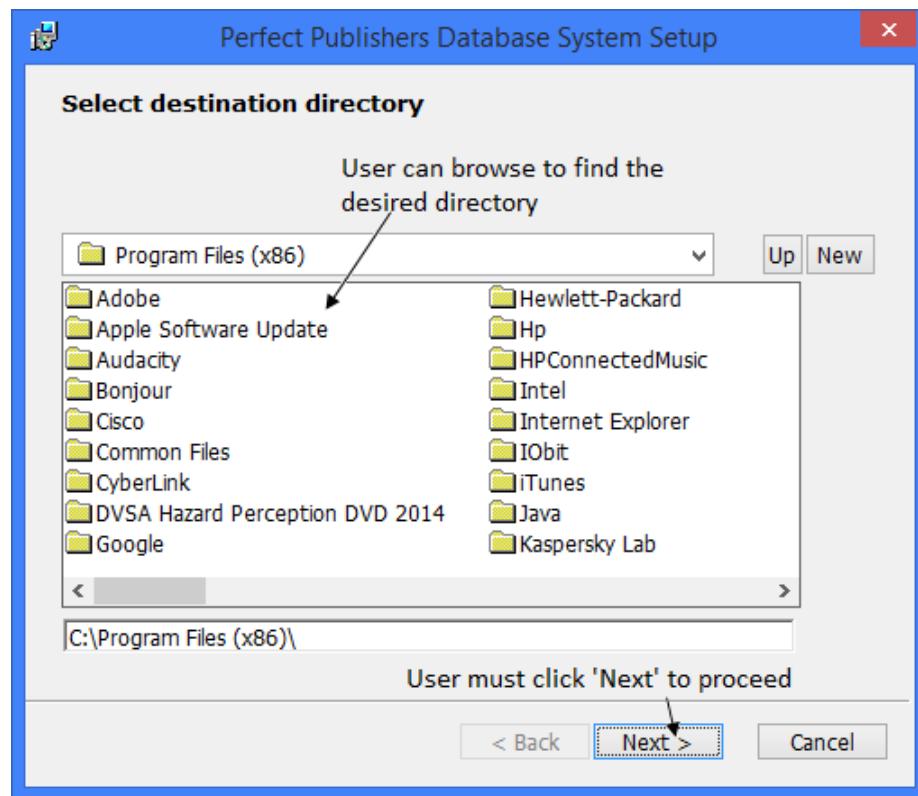


Figure 5.6: Setup Screen Browsing

The Installer initialises the screen for installation.

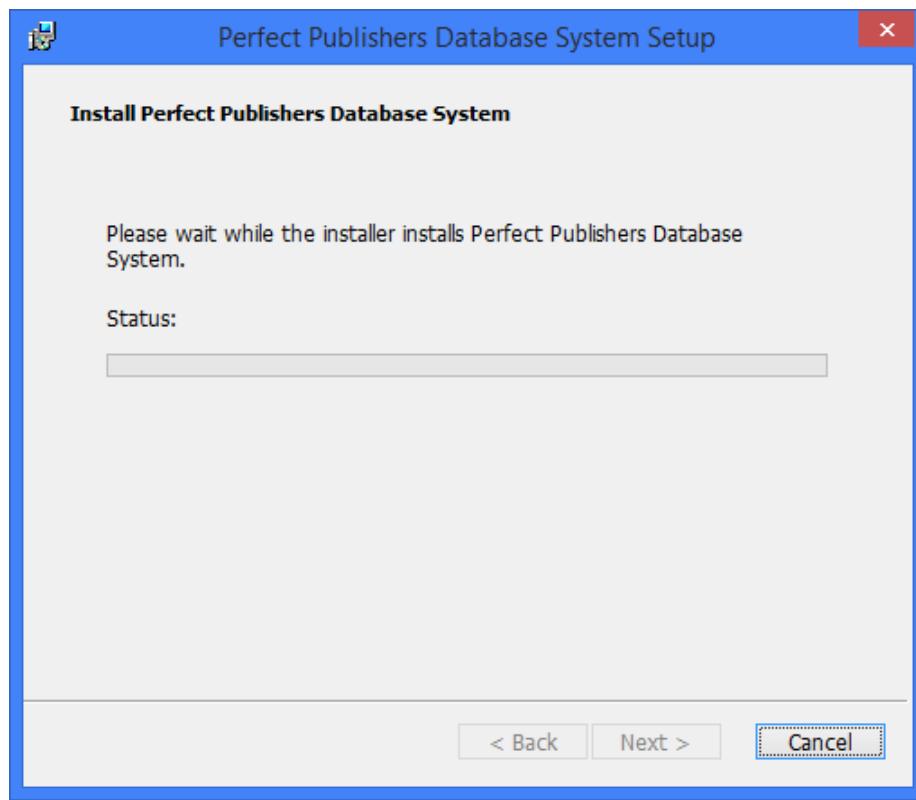


Figure 5.7: 'Please Wait' Screen

The user must grant the installer permission to complete installation. To do so, click 'Yes' when the prompt opens.



Figure 5.8: Prompt for permission

Once the user has granted the installer permission, the installation is completed.

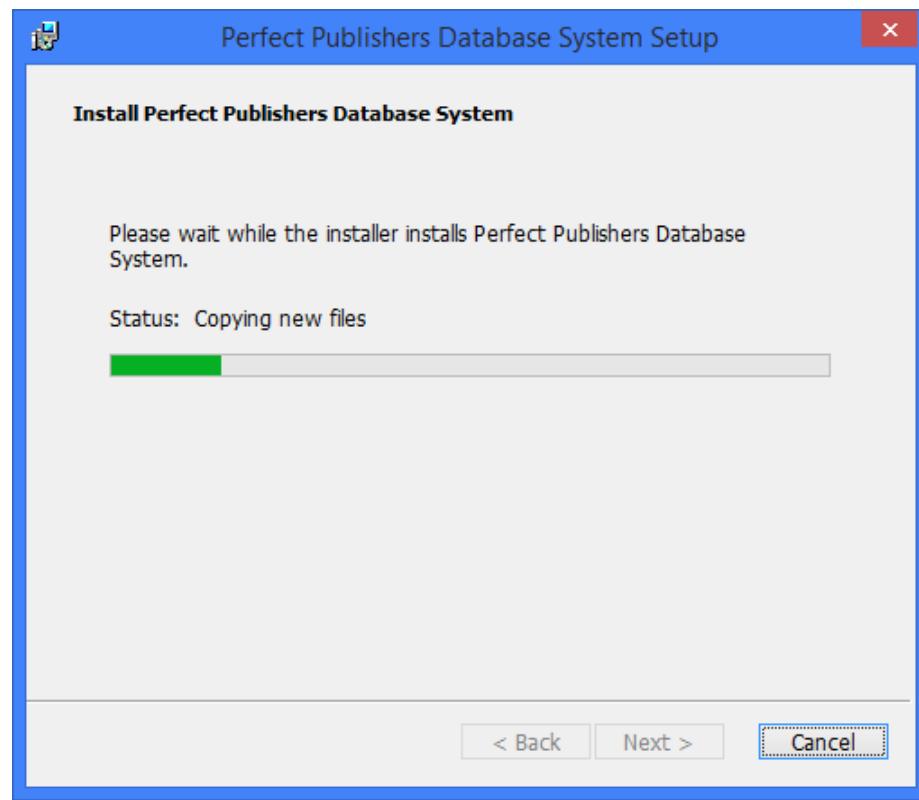


Figure 5.9: Installation process

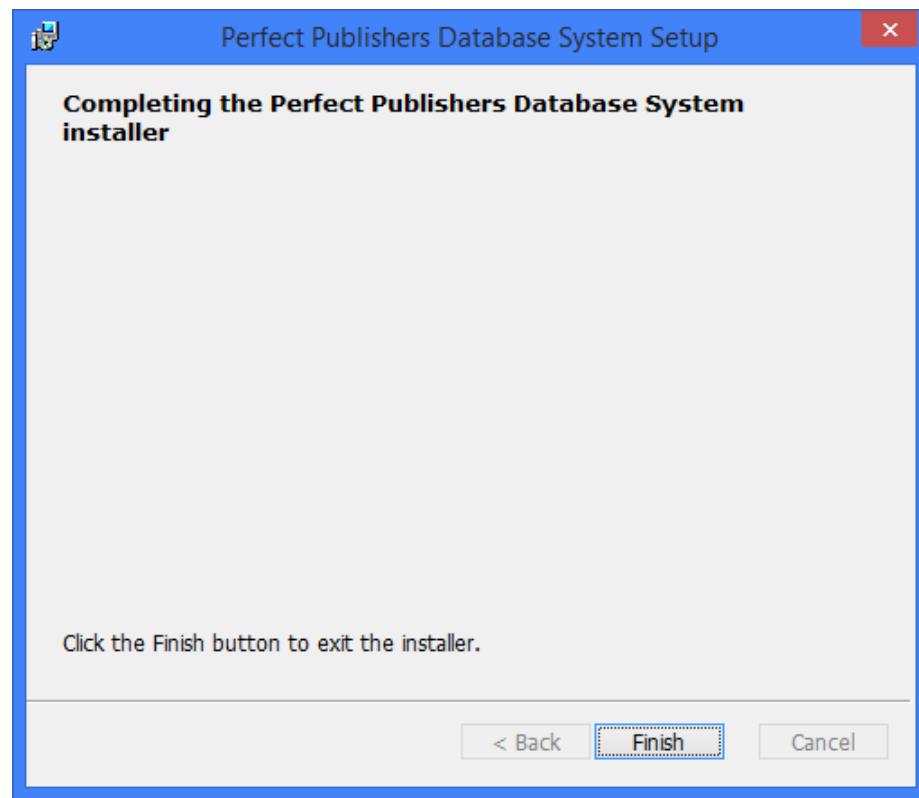


Figure 5.10: Finished Installing

### 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:

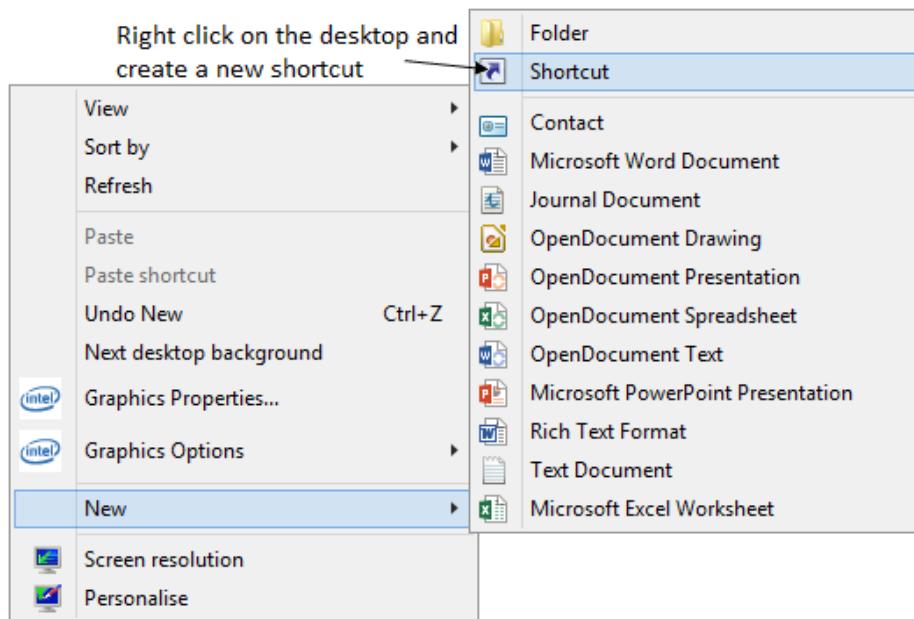


Figure 5.11: Creating a shortcut

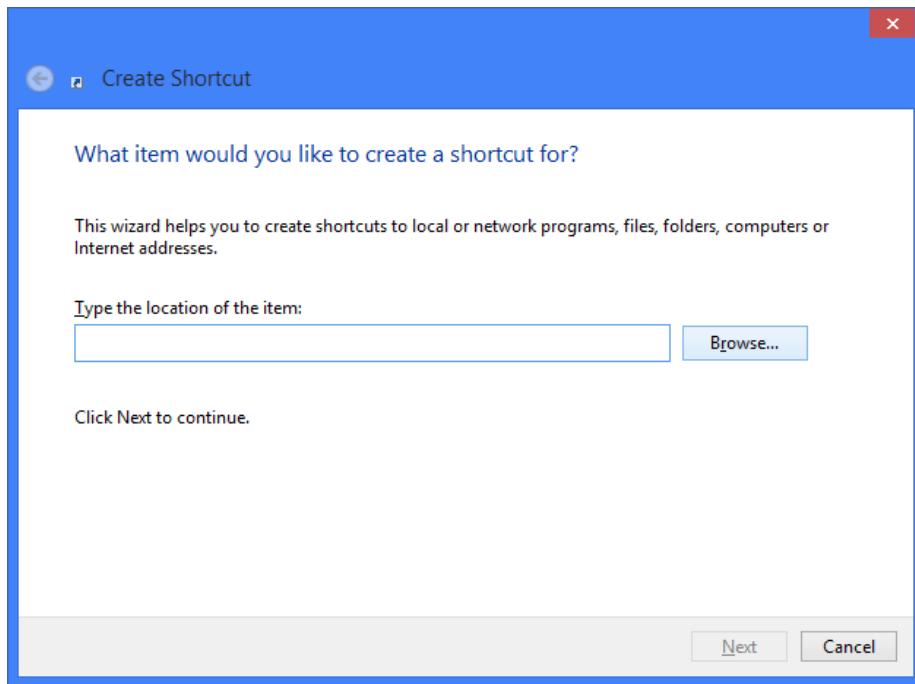


Figure 5.12: Create shortcut screen

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 5.13.

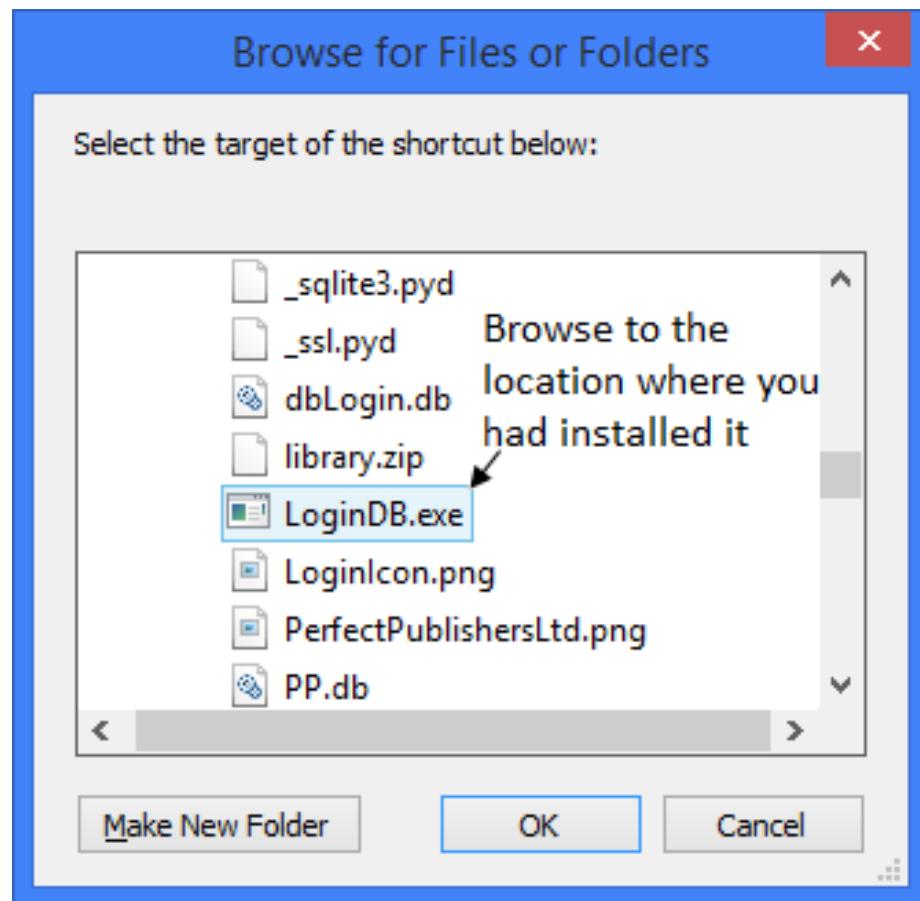


Figure 5.13: File Browsing

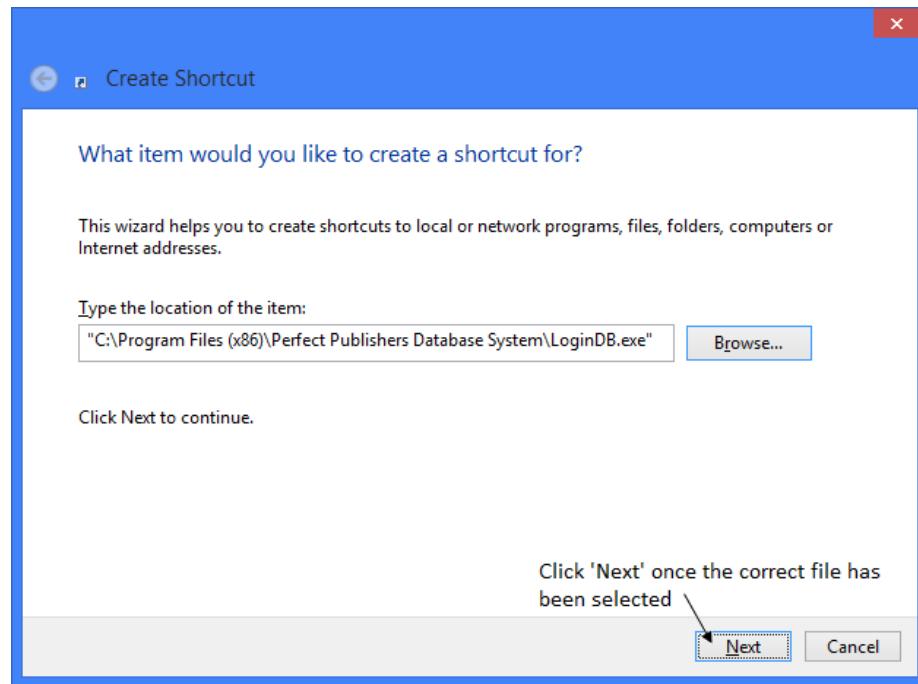


Figure 5.14: File Selected

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.

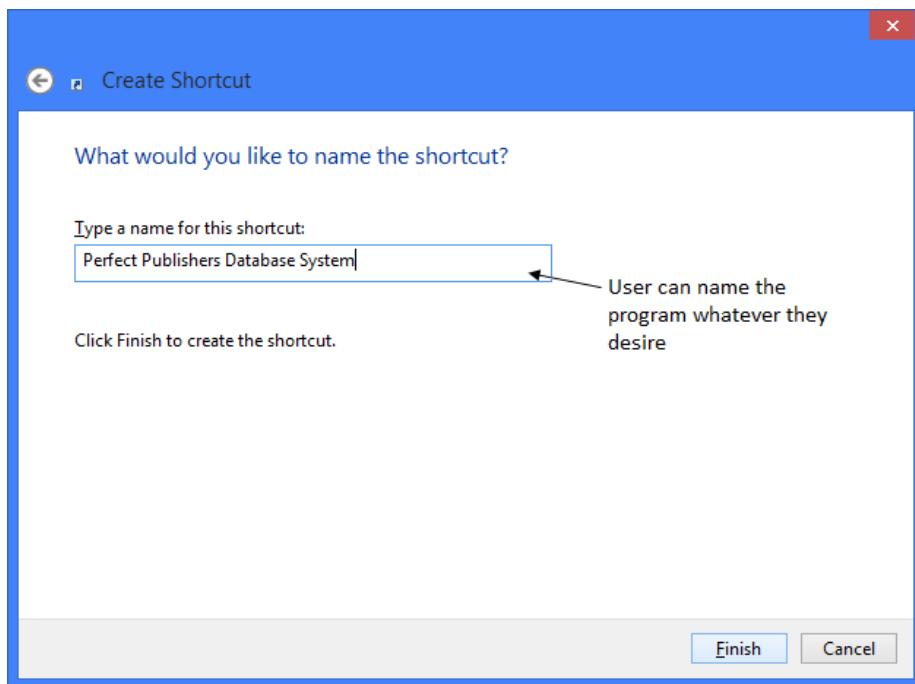


Figure 5.15: Naming shortcut

## 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.

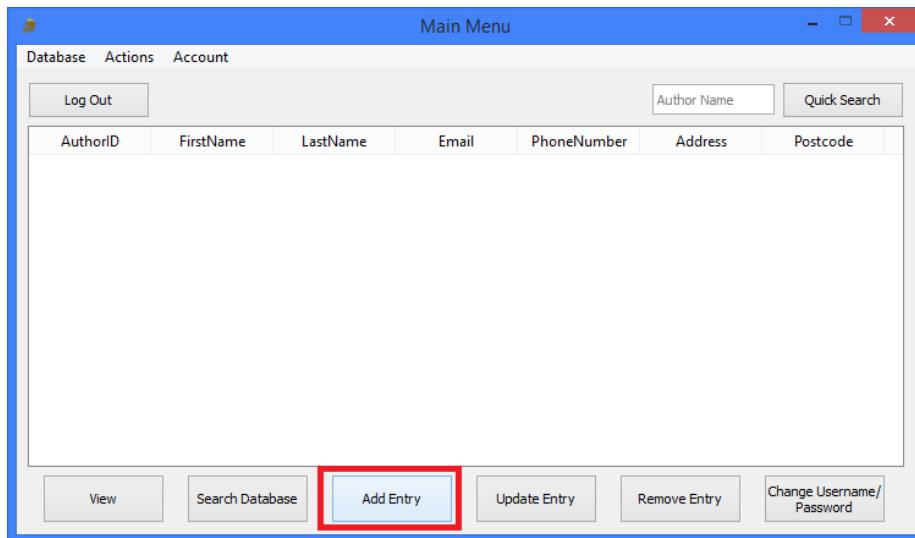
### 5.3.3 Tutorial Questions

The following sections will show how to use different parts of the system, step by step. I have simplified these down into Questions, which i have answered using these step by step tutorials. These can be referenced in the following table.

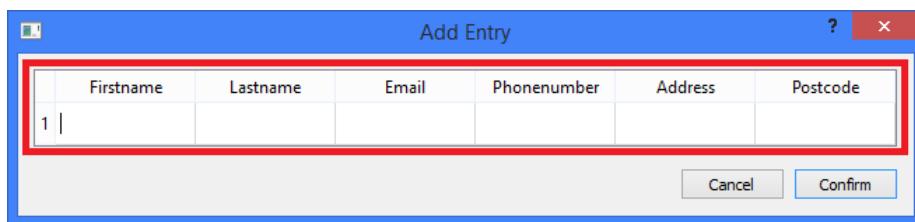
Section Number	Question	Page Number
1	"How do I add a customer?"	349
2	"How do I update customer details?"	350
3	"How do I delete a customer?"	353
4	"How do I view a customer's books?"	354
5	"How do I add a customer's book?"	355
6	"How do I update a customer's book?"	356
7	"How do I delete a customer's book?"	358
8	"How do I view customers' publishing invoices?"	359
9	"How do I view customers' book invoice items of a book?"	360
10	"How do I view customers' royalty items of a book?"	361
11	"How do I calculate a book invoice payment?"	362
12	"How do I calculate a royalty payment?"	363
13	"How do I search for an author?"	363
14	"How do I search for an author's book?"	365
15	"How do I change my Username/-Password?"	368

### 1 - How do I add a customer?

1. On the Main Menu, click on the 'Add Entry' button. A window to add a customer entry is displayed.



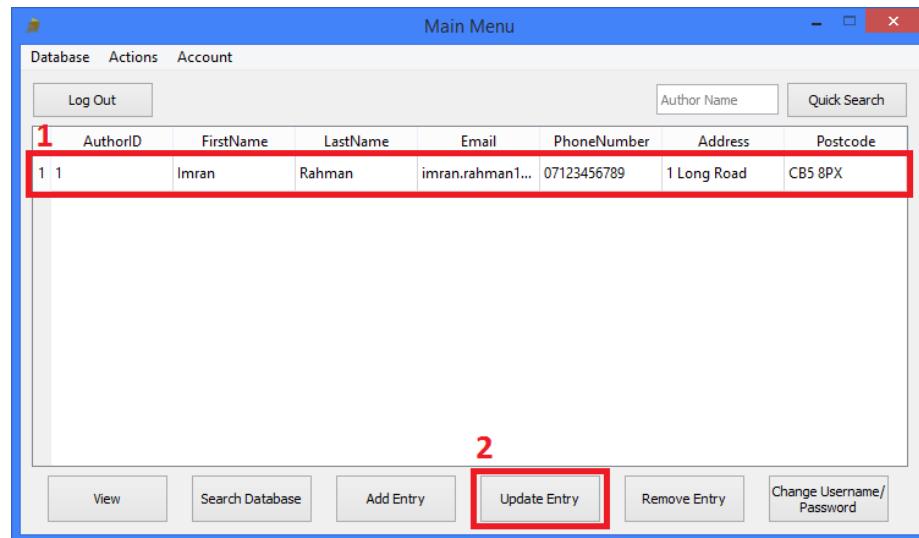
2. Fill in each box of the table with the appropriate data. Then click "Confirm".



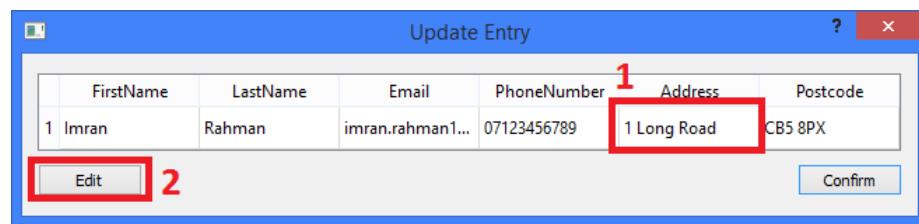
3. After clicking "Confirm", the data will be validated, then added to the system if valid.

## 2 - How do I update customer details?

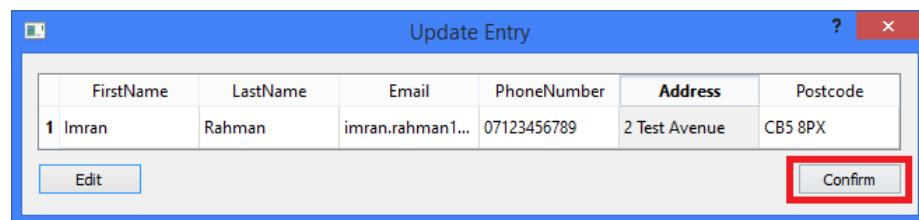
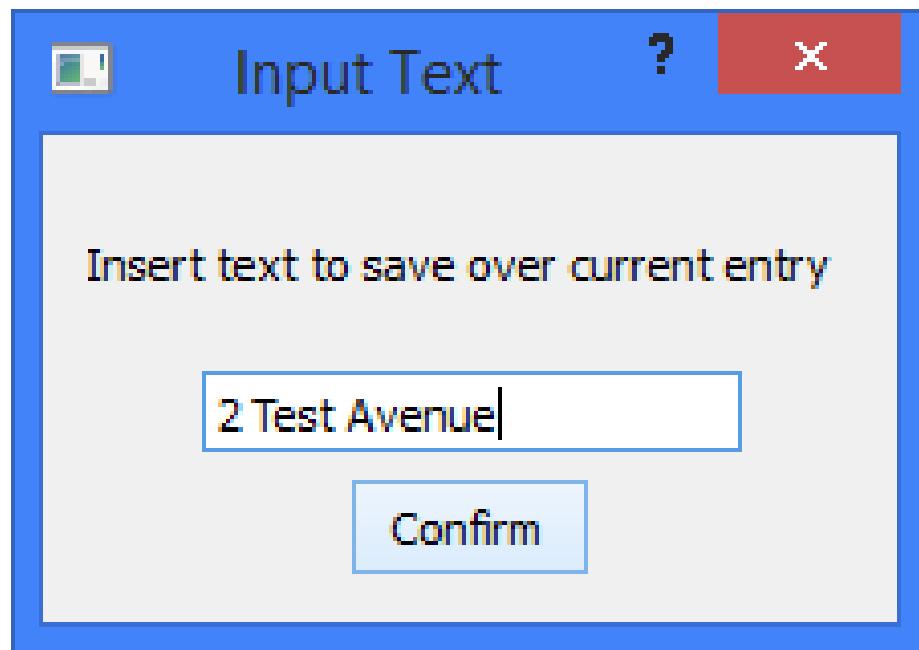
1. On the Main Menu, choose a customer, and click on him/her. Then click "Update Entry".



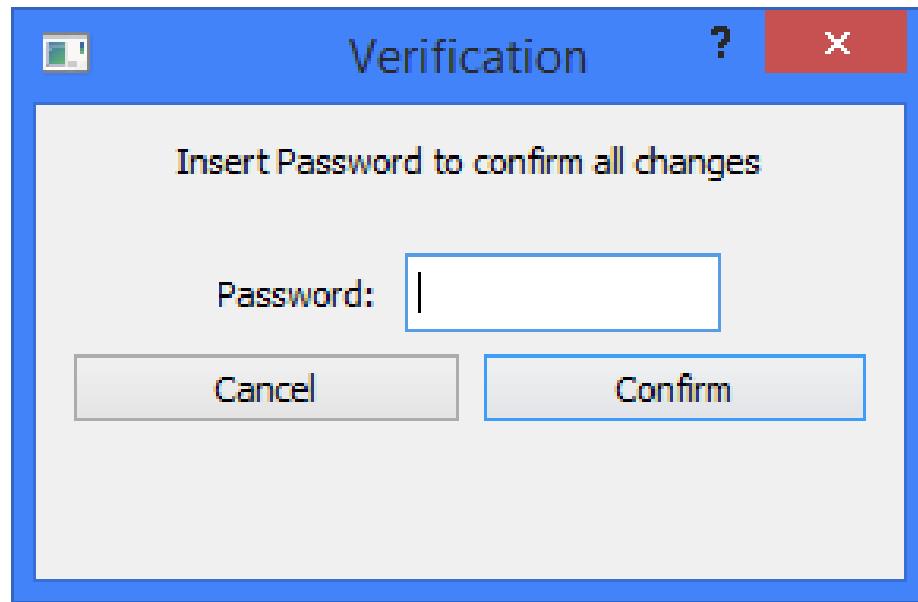
2. A window to update the selected customer entry is displayed. To edit a field, select one, then click "Edit".



3. Upon clicking edit, a window opens, prompting for a new entry for that field. After clicking "Confirm", the change is temporarily made, and more fields can be edited. Once finished, click "Confirm" to save changes.

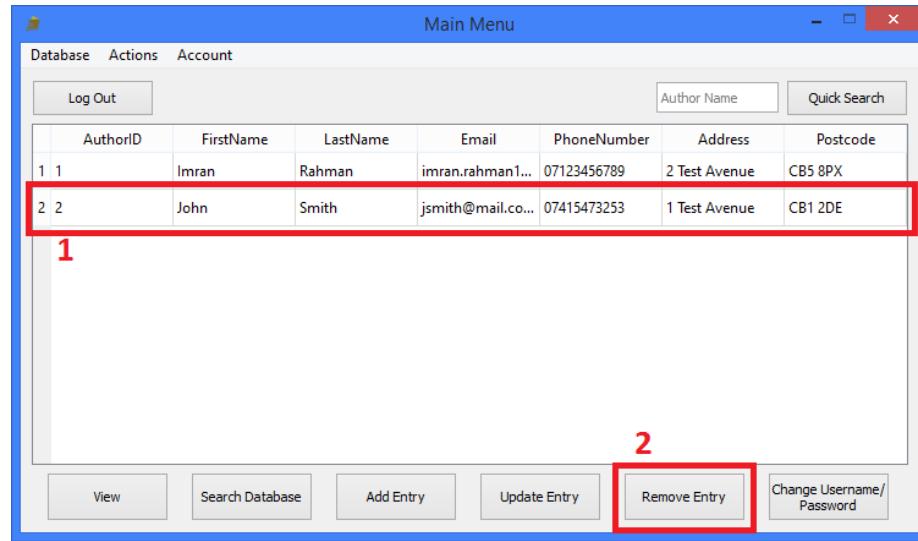


4. A window is displayed, asking for the password. Once the correct password has been entered and the user clicks "Confirm", the changes are saved to the database.



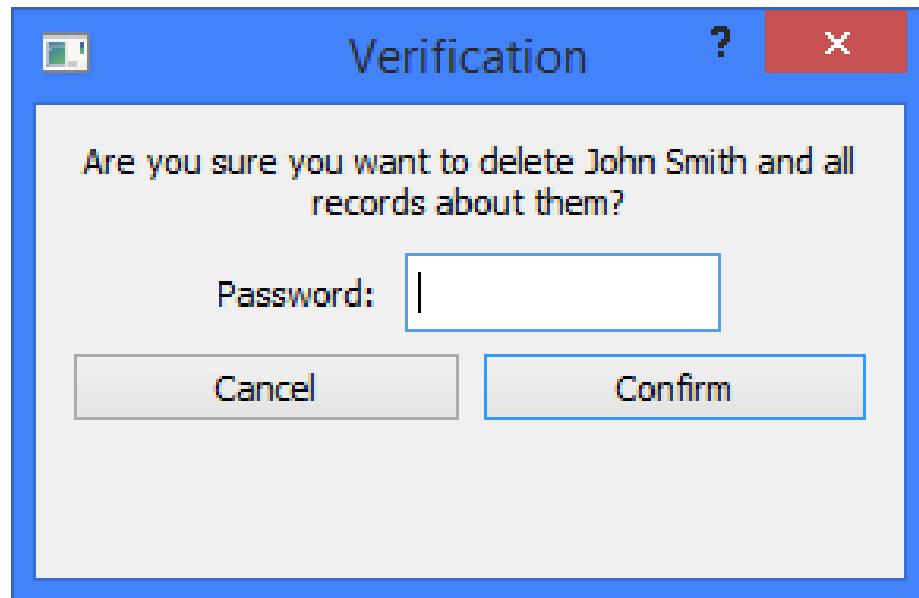
### 3 - How do I delete a customer?

1. On the Main Menu, choose a customer, and click on him/her. Then click "Remove Entry".



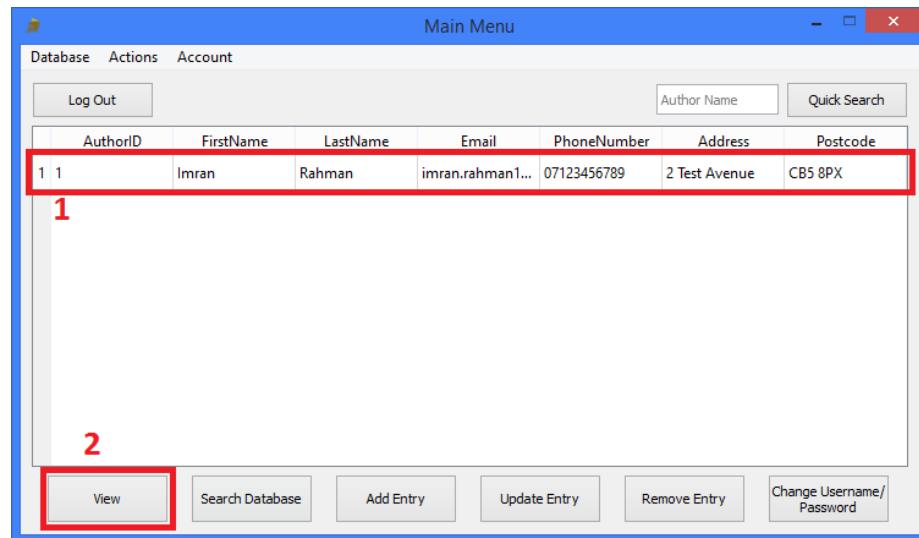
2. A window is displayed, asking for the password to confirm the deletion of the customer and all of their data. Once the correct password has been entered,

click "Confirm". The deletion is then completed.

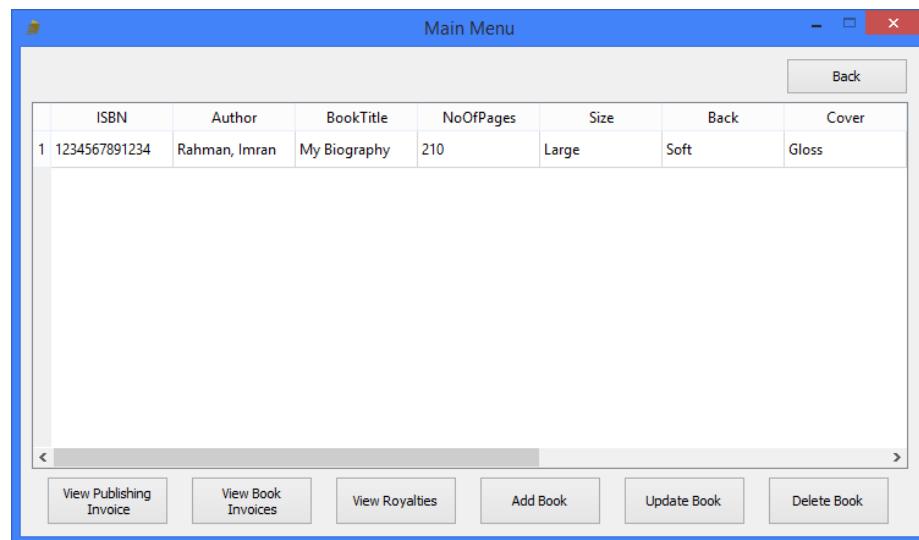


#### 4 - How do I view a customer's books?

1. On the Main Menu, choose a customer, and click on him/her. Then click "View".

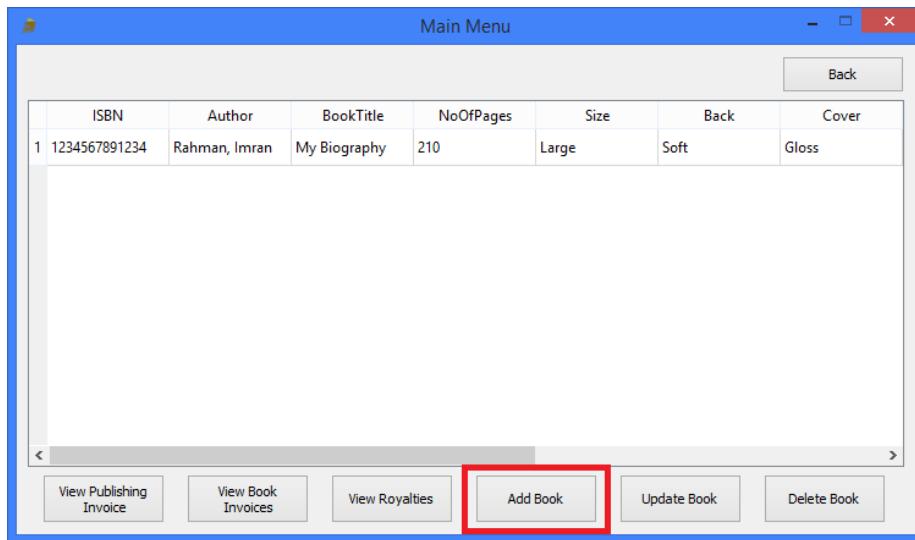


2. A new layout for interactions with books is displayed, and a list of the books from the selected customer.

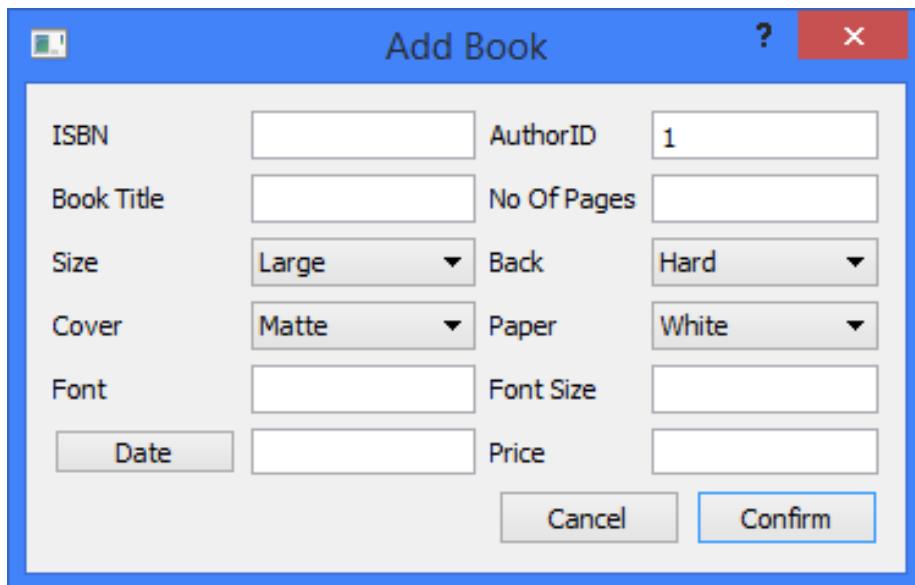


#### 5 - How do I add a customer's book?

1. Follow section 4 to select a customer to add books to.
2. On the Menu presenting a customer's books, click "Add Book".



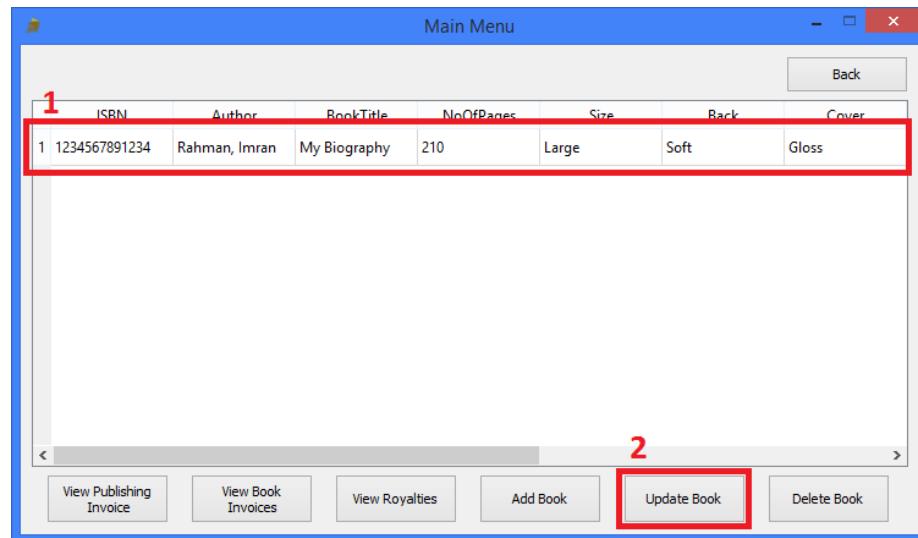
3. A window is opened, showing empty fields required to be filled in. After filling in the appropriate data, click "Confirm" to add the book to the database.



#### 6 - How do I update a customer's book?

1. Follow section 4 to select a customer to find their books.

2. On the Menu presenting a customer's books, choose a book, and click on it. Then click "Update Book".



3. A window is opened, showing the current details of the book in their respective fields. All fields can be freely edited where necessary. Once finished, click "Confirm".

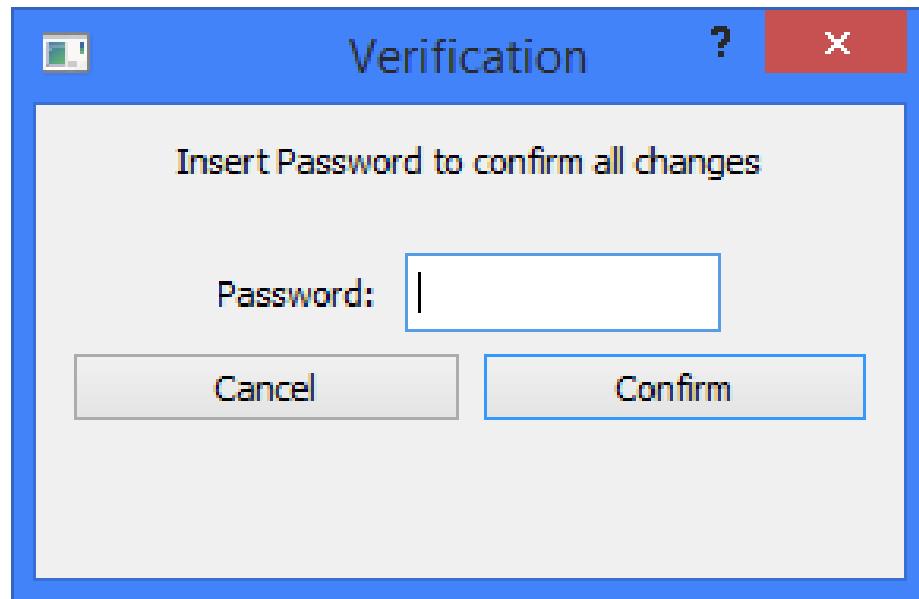
Edit Book

ISBN	1234567891234	AuthorID	1
Book Title	My Biography	No Of Pages	250
Size	Large	Back	Soft
Cover	Gloss	Paper	White
Font	Calibri	Font Size	10
Date	13-03-2015	Price	8.99

Cancel   **Confirm**

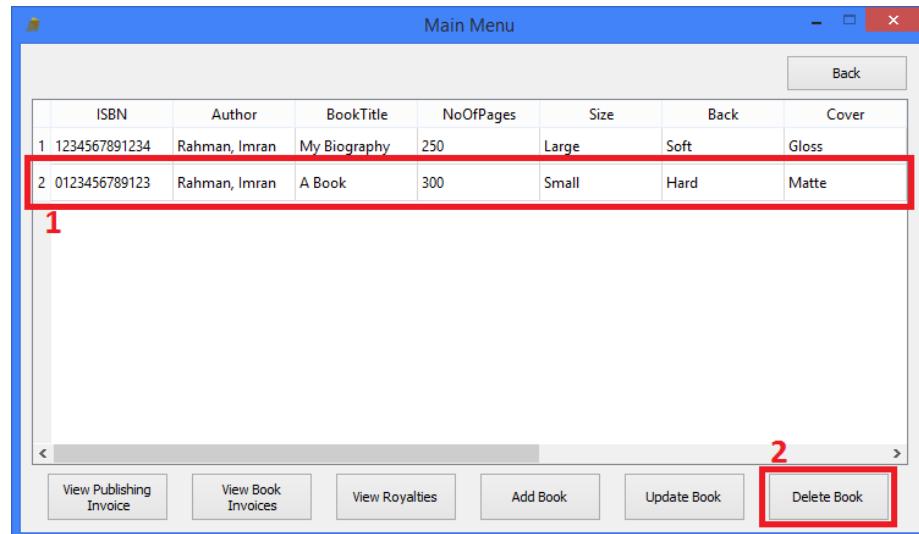
4. A window is displayed, asking for the password. Once the correct password

has been entered and the user clicks "Confirm", the changes are saved to the database.

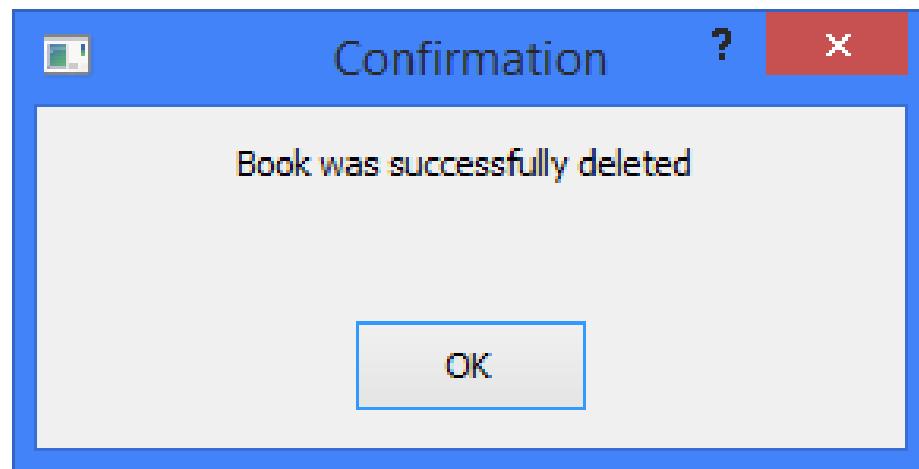
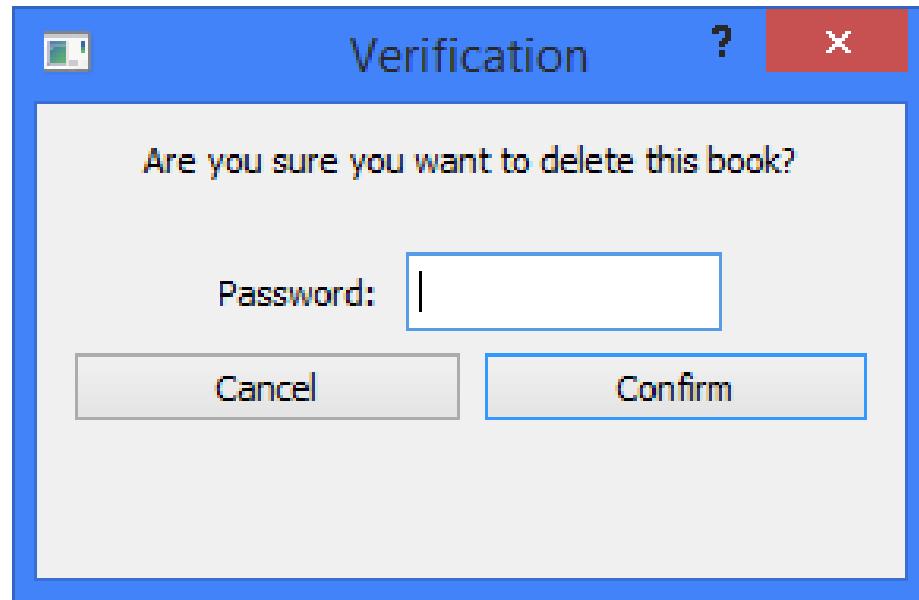


#### 7 - How do I delete a customer's book?

1. Follow section 4 to select a customer to find their books.
2. Click on a book entry, then click "Delete Book".

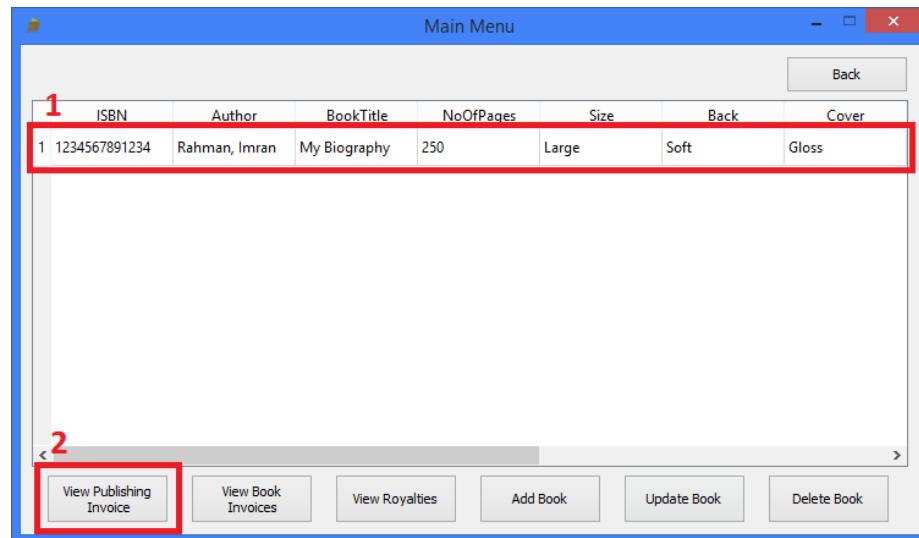


2. A window is displayed, asking for the password to confirm the deletion of the book. Once the correct password has been entered, click "Confirm". The deletion is then completed.

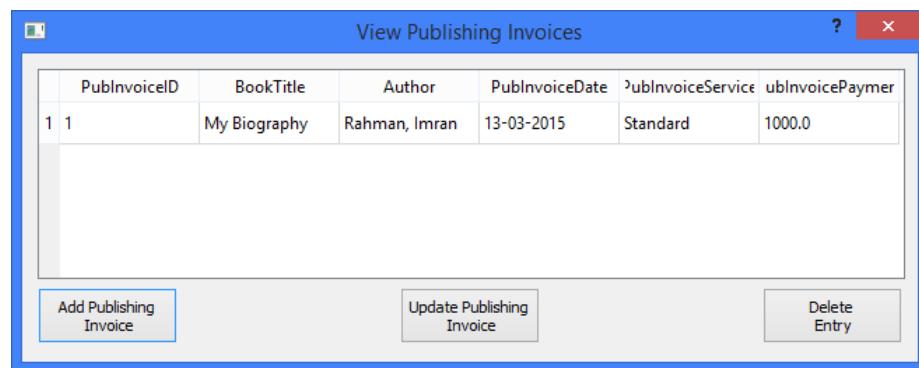


#### 8 - How do I view a customers' publishing invoices?

1. Follow section 4 to select a customer to find their books.
2. Click on a book entry, then click "View Publishing Invoice".

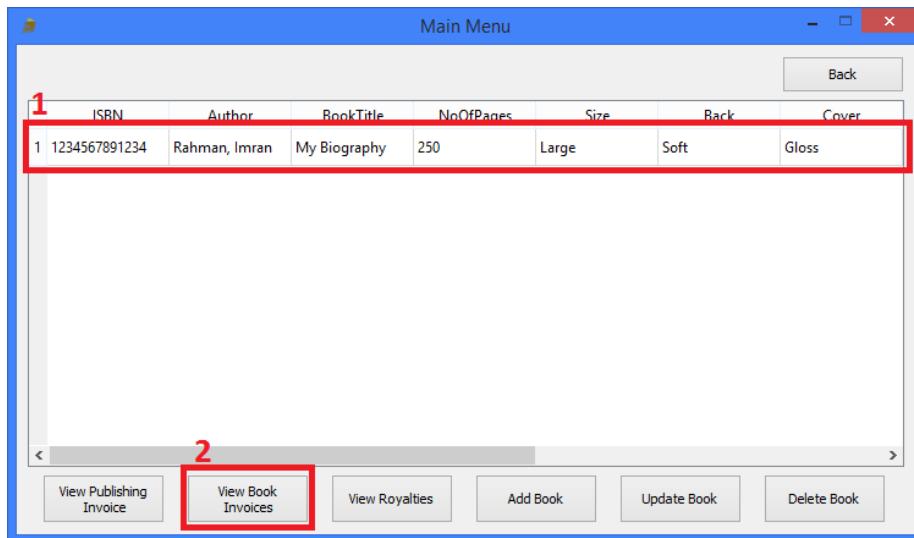


3. The selected book's publishing invoice(s) is displayed.

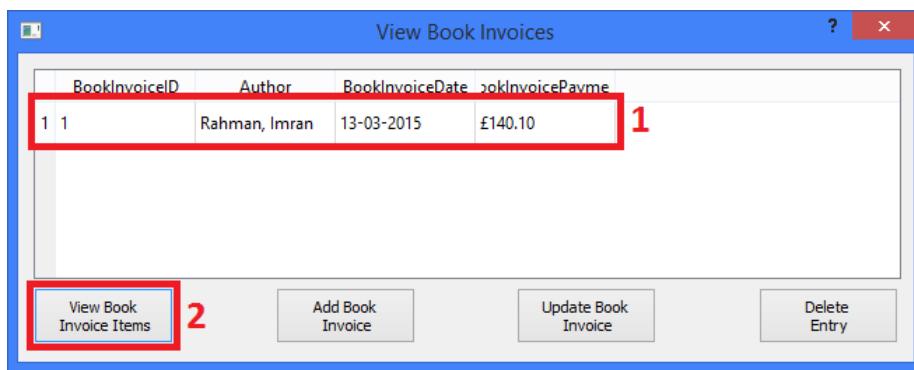


#### **9 - How do I view customers' book invoice items of a book?**

1. Follow section 4 to select a customer to find their books.
2. Click on a book entry, then click "View Book Invoices".



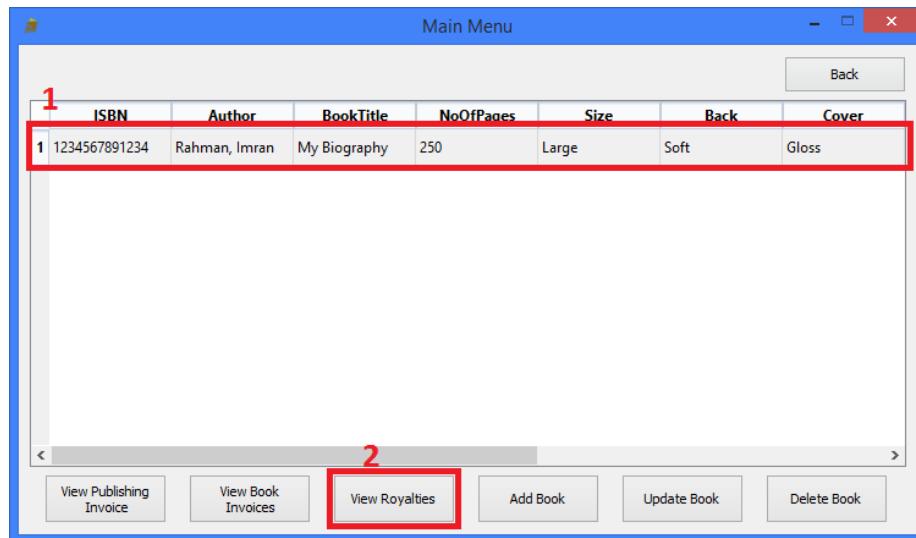
3. Click on a book invoice, then click "View Book Invoice Items".



4. The items of the selected invoice are displayed.

#### 10 - How do I view customers' royalty items of a book?

1. Follow section 4 to select a customer to find their books.
2. Click on a book entry, then click "View Royalties".



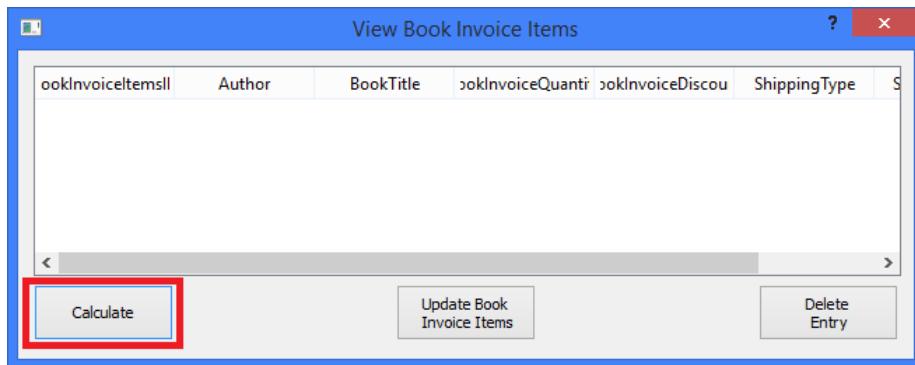
3. Click on a book invoice, then click "View Royalty Items".



4. The items of the selected royalty payment are displayed.

#### 11 - How do I calculate a book invoice payment?

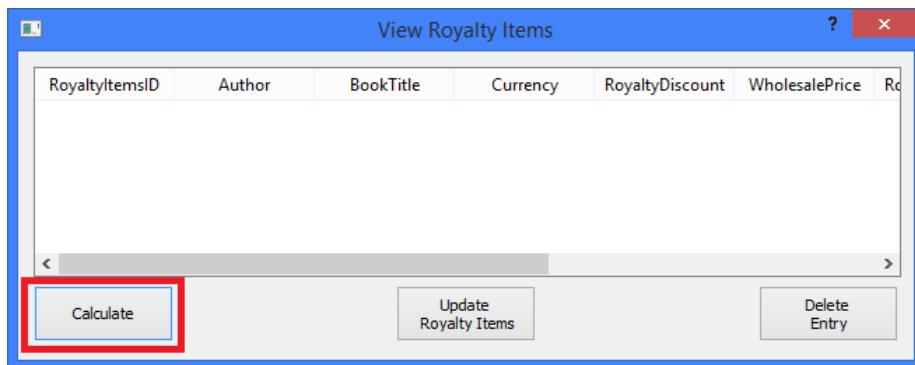
1. Follow section 9 to navigate to a customer's book invoice items.
2. Click "Calculate" in the bottom left corner, to conduct a calculation to add to the database.



3. Fill in the empty fields, then click "Calculate" in the bottom left corner. Afterwards, click "Confirm". This calculation will be added to the current book invoice payment and can be seen in the book invoice table.

### 12 - How do I calculate a royalty payment?

1. Follow section 10 to navigate to a customer's royalty items.
2. Click "Calculate" in the bottom left corner, to conduct a calculation to add to the database.



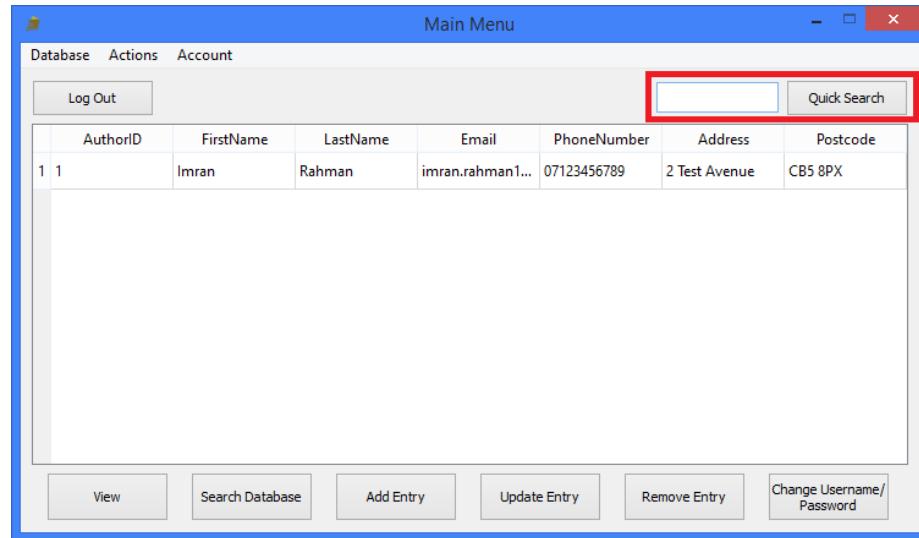
3. Fill in the empty fields, then click "Calculate" in the bottom left corner. Afterwards, click "Confirm". This calculation will be added to the current royalty payment and can be seen in the royalties table.

### 13 - How do I search for an author?

There are two ways to search for an author. The first way is recommended.

First method:

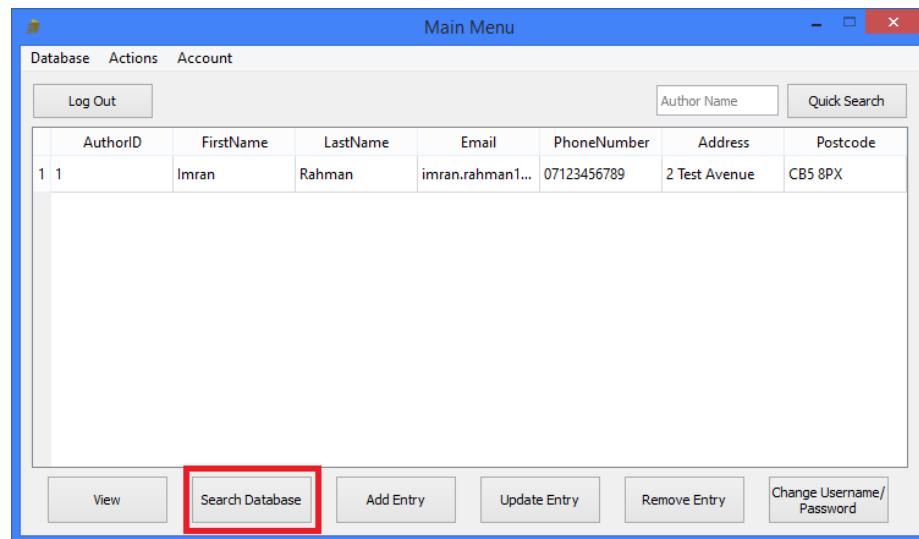
1. On the Main Menu, type in a customer's firstname, lastname, or both in the quick search bar in the top right corner, then click "Quick Search".



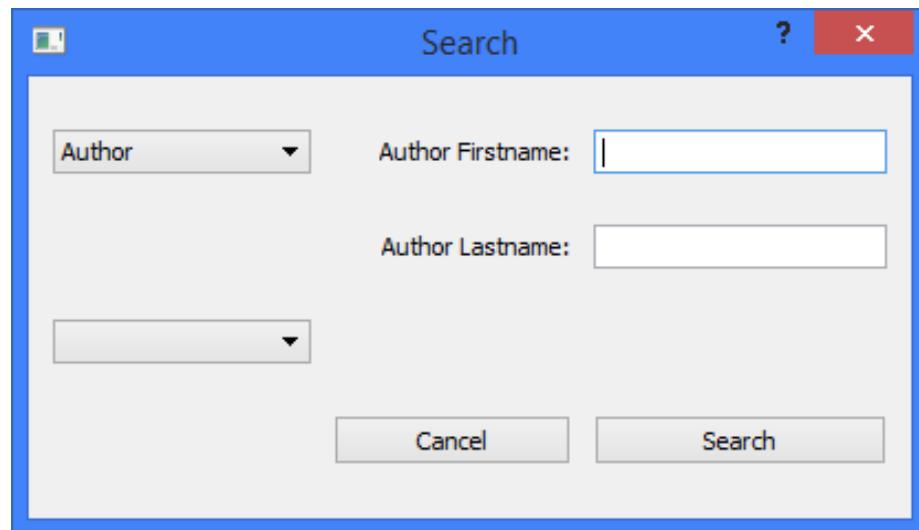
2. The results are displayed in the main window table.

Second method:

1. Click "Search Database" on the Main Menu.



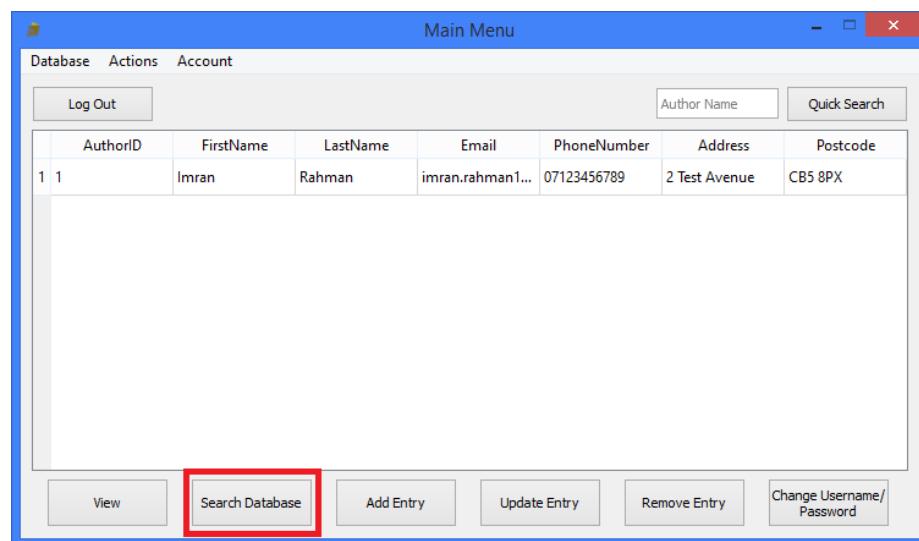
2. A new window will open. Type in the author's first and last names in their respective boxes, and select "Author" in the first combo box. Leave the last combo box blank. Click "Search".



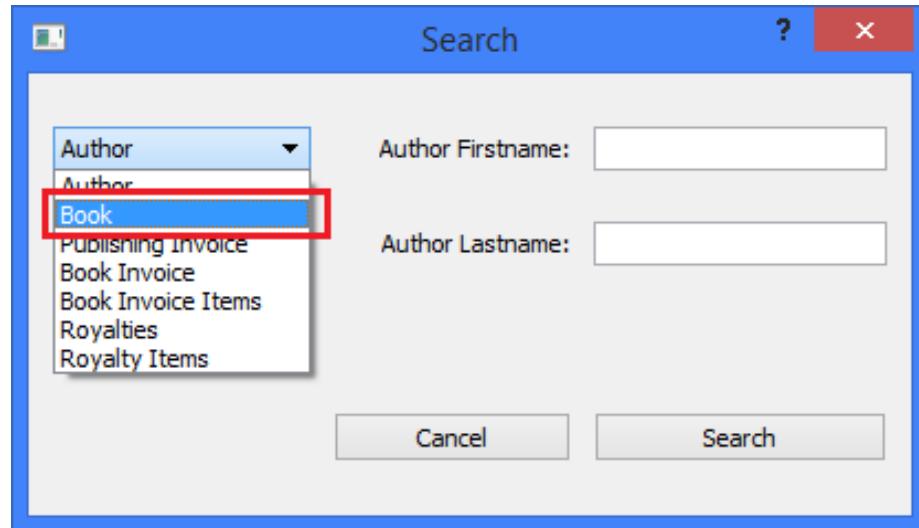
3. The results are displayed in the main window table.

#### 14 - How do I search for an author's book?

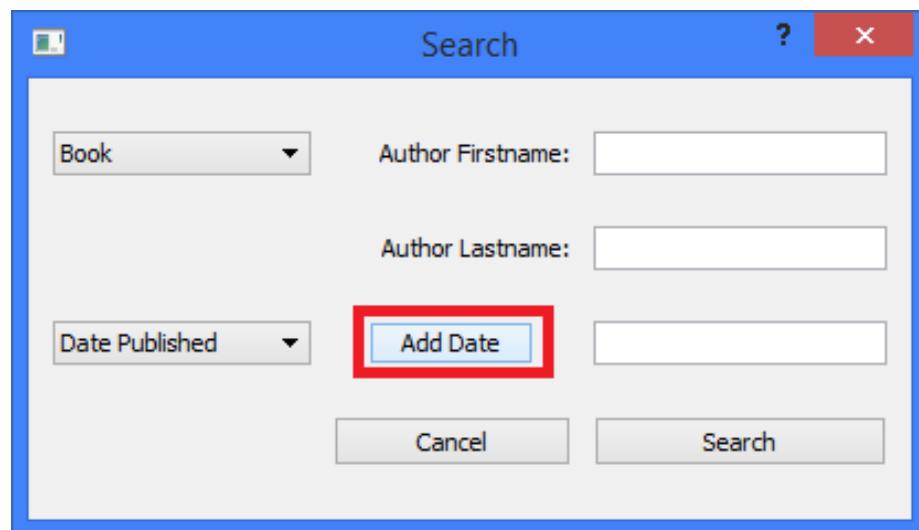
1. Click "Search Database" on the Main Menu.

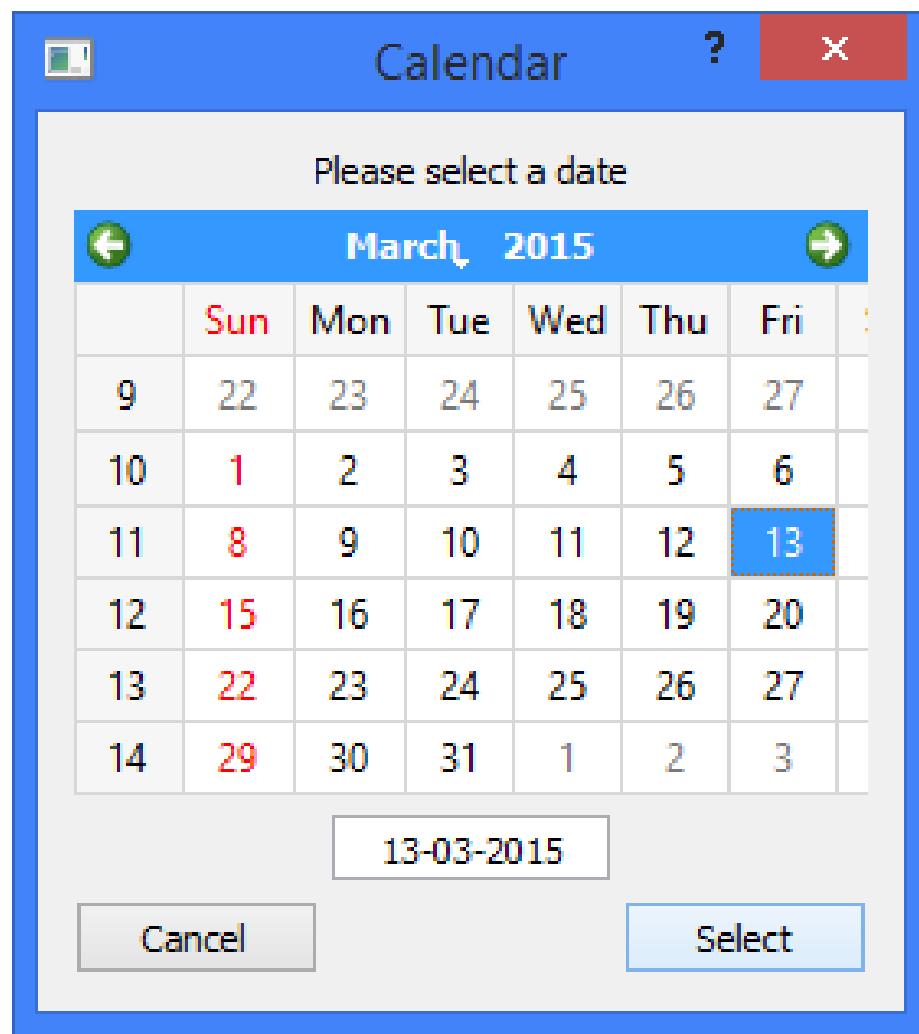


2. Select "Book" from the first combo box.

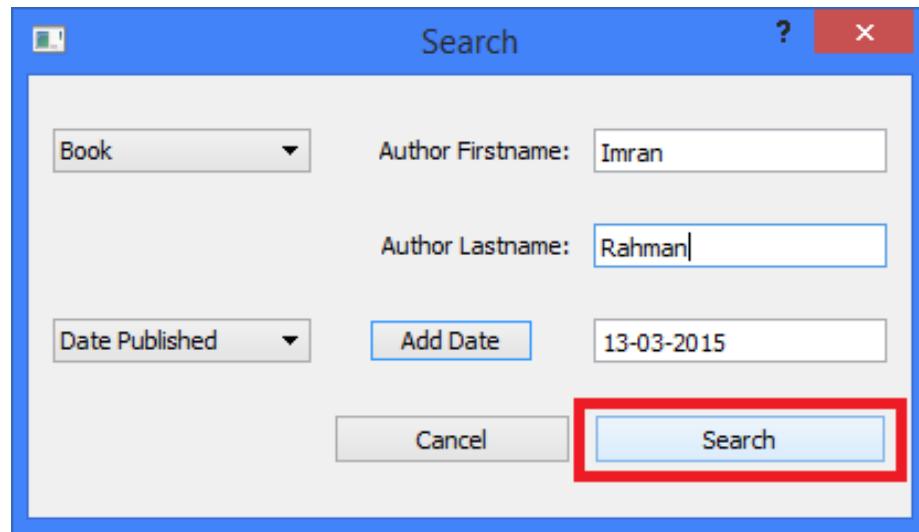


3. Select a category of the book you wish to search for from the second combo box, and then fill in the empty line edit opposite. (If a date category was selected, a date button will appear. Click it to open the calendar and select a date from it.)





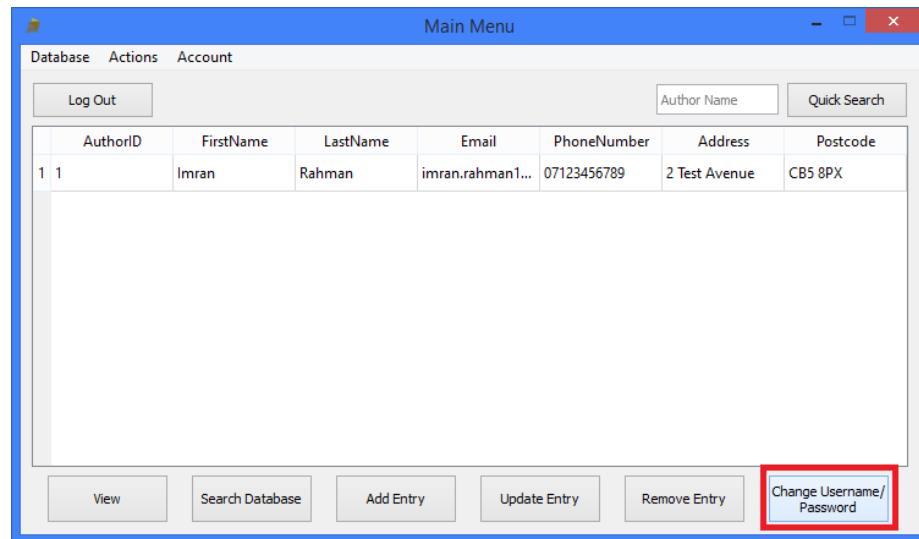
4. Fill in the Firstname and Lastname boxes. Click "Search".



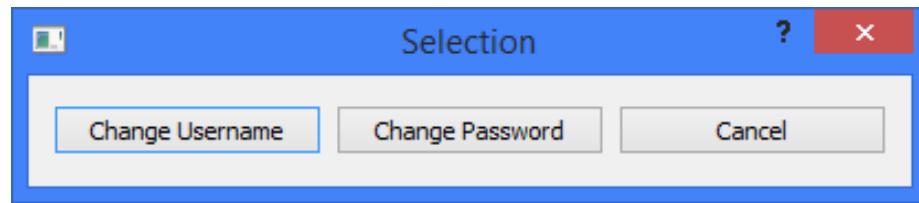
5. The results will be displayed in the main window table.

#### 15 - How do I change my Username/Password?

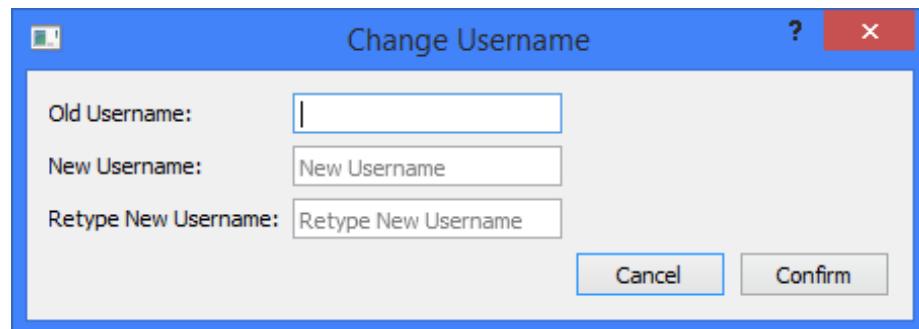
1. Click "Change Username/Password" on the Main Menu.



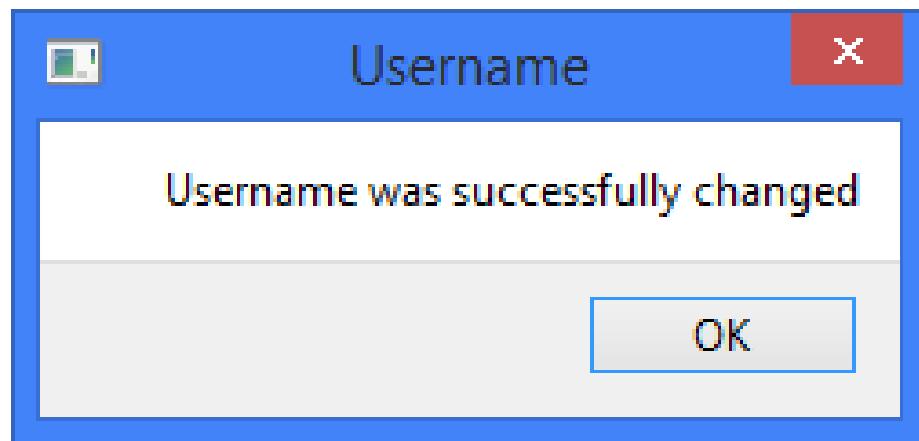
2. If you want to change your Password, click "Change Password" and skip to step 5. Otherwise, click "Change Username".



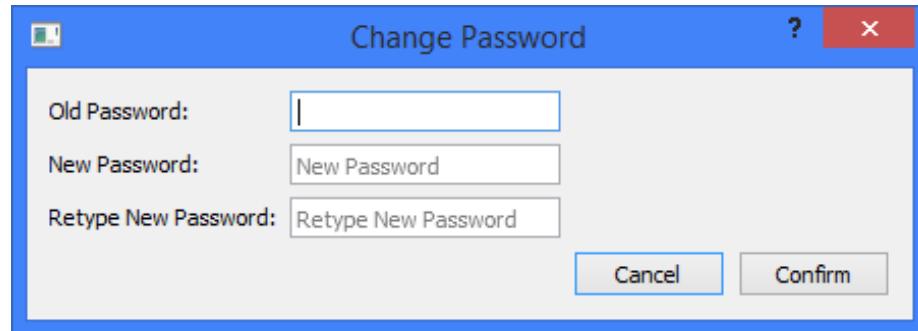
3. Enter your Old username in the top box, and your new username in middle box and last box for verification. Then click "Confirm".



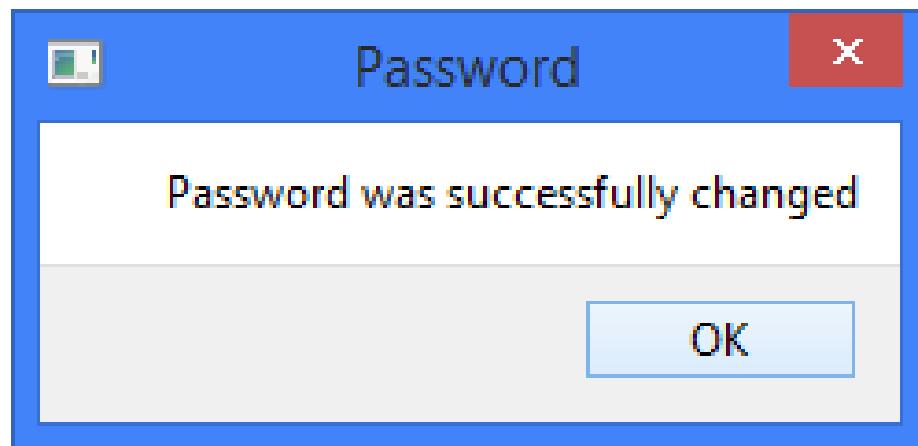
4. Your Username has been successfully changed.



5. Enter your Old Password in the top box, and your new Password in middle box and last box for verification. Then click "Confirm".



6. Your Password has been successfully changed.



#### 5.3.4 Saving

All saving in the system is done automatically, including calculations of royalty and invoice payments. This means that no saving is required to be manually done by the user. Data is saved after each interaction of adding, editing and deleting it, so that if there is a program crash, or the computer being used is switched off, data is not lost. However, data should only be backed up after usage of the system.

#### 5.3.5 Limitations

My system cannot edit data upon searching for it. This was not done because I did not have enough time to implement this into the system. However, this is not difficult to create, and could be included in future versions of the system. Also, when adding customer/book data, the data is not checked against the

database for duplications. This is because I did not have enough time to finish this element of the system. However, the clarity of the data in the system means that the user does not need to duplicate entries as data can easily be searched for, meaning that the user can check the existence of the data before adding it.

## 5.4 Error Recovery

### 5.4.1 Invalid entries when calculating payments

When calculating royalty/invoice payments, try to avoid entering invalid entries before calculations. This is because the window will prompt you of an invalid entry, and will then close, but still preventing the adding of the invalid data.

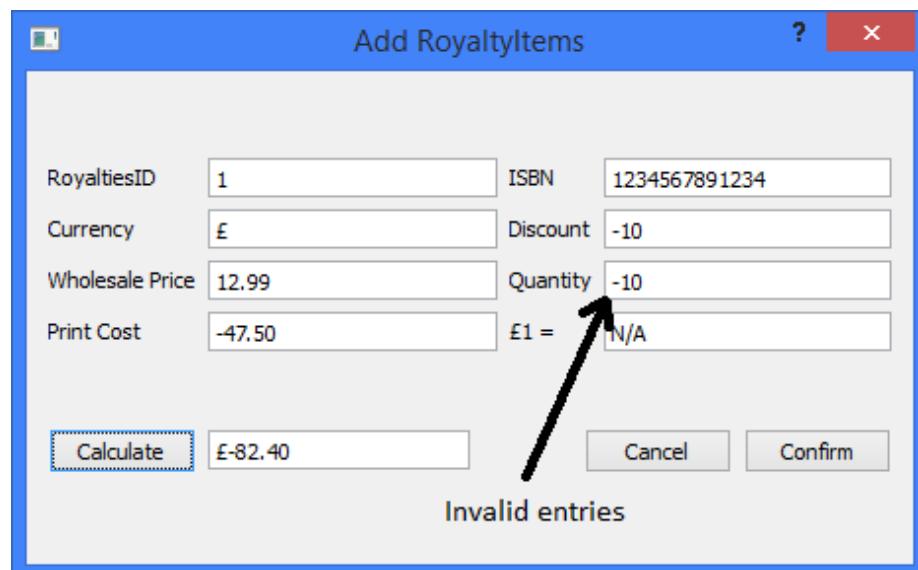


Figure 5.16: Invalid Royalty Items Entries

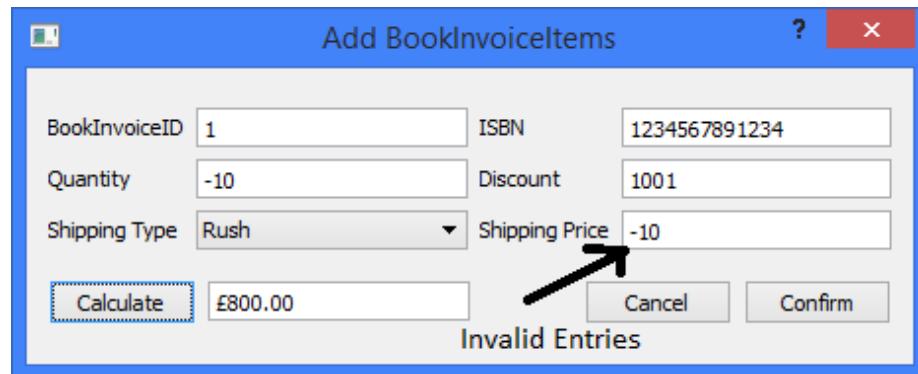


Figure 5.17: Invalid Book Invoice Items Entries

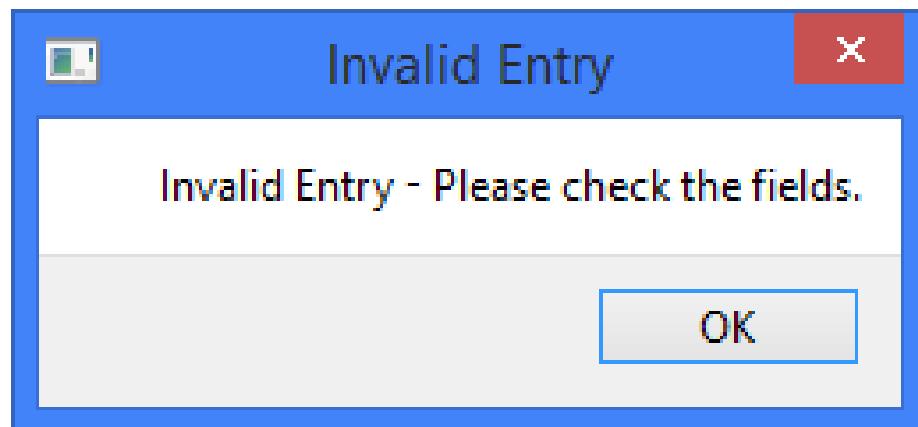


Figure 5.18: User is prompted

In order to solve these issues, just reopen the window to calculate the payments and reenter the data.

## 5.5 System Recovery

### 5.5.1 Backing-up Data

In order to backup my program, the user will preferably need a hard copy of the backup kept off site, perhaps through the usage of a USB drive. The user

should backup the database file, as this is the only data that requires being backed up.

1. First, you must navigate to where the executable file of the system is stored.
2. You must right click the "PP.db" file, and click "Copy".

This PC ▶ Windows (C:) ▶ Program Files (x86) ▶ Perfect Publishers Database System

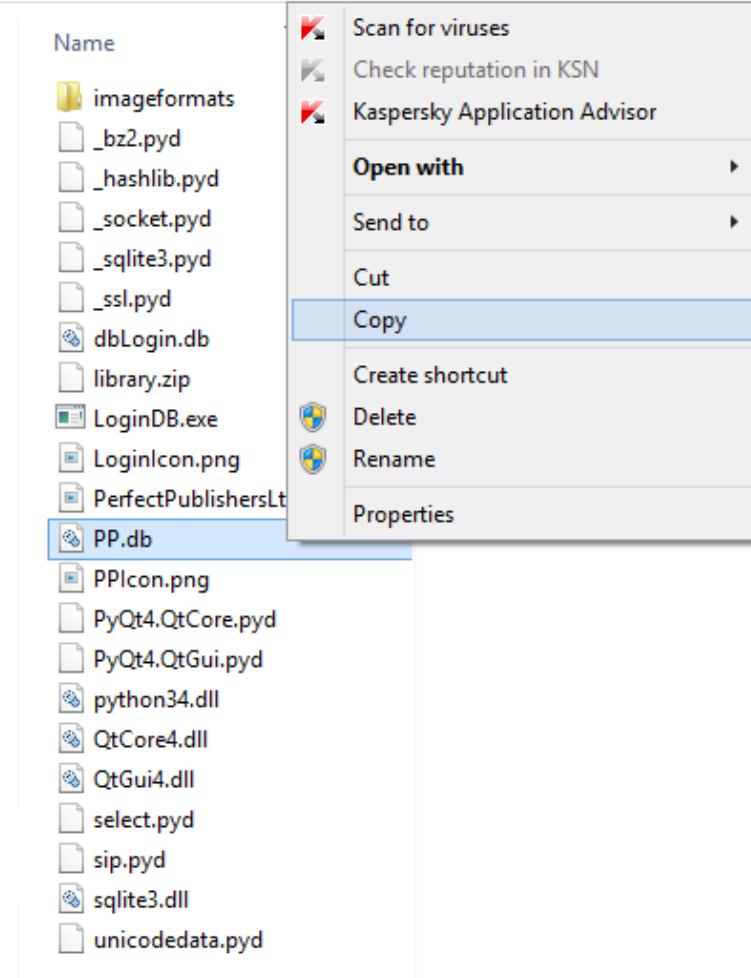


Figure 5.19: Copying the database file

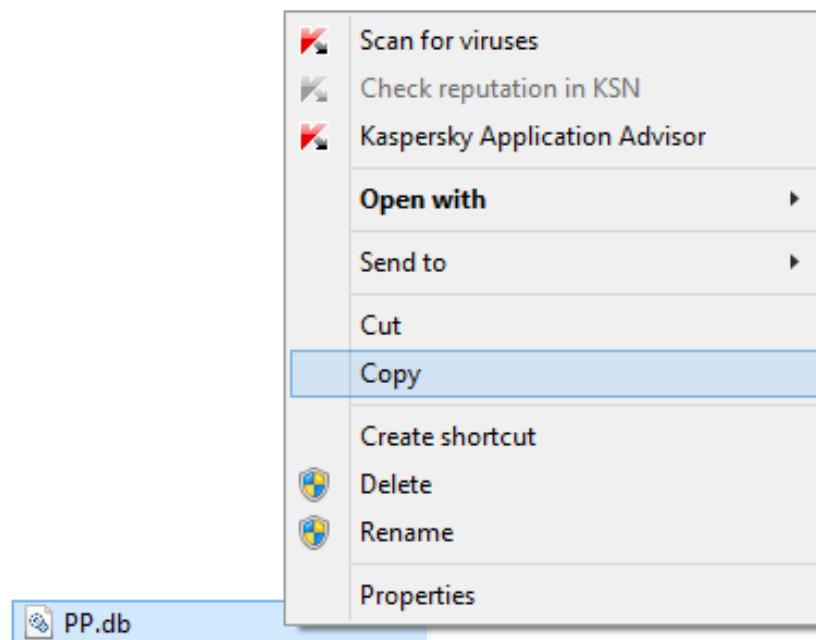
3. Now, the user must choose where the data should be stored. Once a destination has been chosen and navigated to, create a new folder and name it "Perfect Publishers System Database".

4. Inside the folder, paste the database file.
5. You have successfully backed up the database

### 5.5.2 Restoring Data

In order to restore my program, the user will need to use the hardware used to back up the database.

1. Find the directory where the file was backed up to.
2. Right click on the database file, "PP.db", and click "Copy".



3. Navigate to where the executable file of the system is stored.
4. Inside the folder, paste the database file. It should be automatically pasted as "PP - Copy.db"

PP - Copy.db	11/03/2015 14:45	Data Base File	13 KB
PP.db	11/03/2015 14:45	Data Base File	13 KB

5. Now, you can delete the original file and rename the newly pasted file to "PP.db".

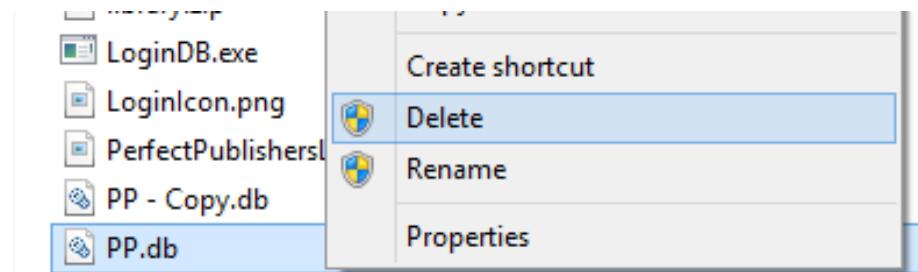


Figure 5.20: Deleting Original file

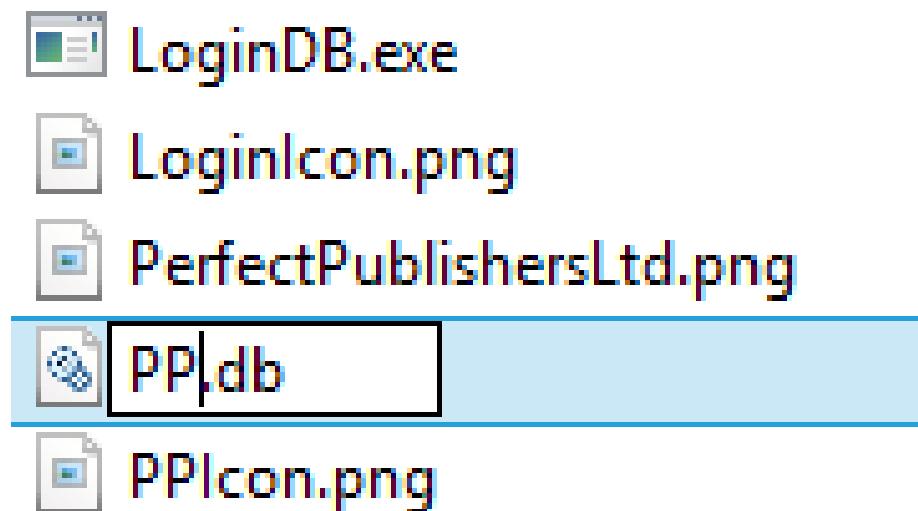


Figure 5.21: Renaming the new file

6. You have successfully restored the database.

# Chapter 6

## Evaluation

### 6.1 Customer Requirements

In this section I will evaluate whether I have fulfilled each objective which were set in my system specifications. This will determine whether the system meets the full customer requirements. Where the objectives may not have been met, I will evaluate and explain the reason for this being so. I will also provide evidence to prove that objectives have been met in other cases, which will determine that the system has achieved its specifications

#### 6.1.1 Objective Evaluation

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.
- Ability to calculate royalties using given data

##### Objective: Organised layout for the database

This objective has been fulfilled by clearly labelling each window and giving clear table headings for each table. For example, the customer table clearly shows the

following headings: "Author ID", "Firstname", "Lastname", "Email", "Phonenumber", "Address" and "Postcode". This can be seen in the image below.

AuthorID	FirstName	LastName	Email	PhoneNumber	Address	Postcode
1 1	John	Smith	Example@mail...	07123456789	1 Example Road	AB1 2CD
2 6	Jackie	Wilson	Example6@mail...	07111222333	3 Example Close	QR9 0ST
3 5	Jack	Parker	Example5@mail...	07789456123	123 Example Str...	MN7 8OP
4 4	Richard	Wilson	Example4@mail...	07789123456	1 Example Aven...	IJ5 6KL
5 3	Amanda	Clarke	Example3@mail...	07987654321	1 Example Way	EF3 4GH
6 2	Sarah	Taylor	Example2@mail...	07123789456	2 Example Road	AB1 2CD
7 7	Test	Data	testdata@mail....	07111222333	1 Test Road	ZY0 9XW

Figure 6.1: Interface is titled and tables are clearly labelled

Also, in question 1 of the questionnaire in page 384, the client has strongly agreed with the statement in the question and said that the titles on windows make the interface clear to them. The user has said that the large buttons make navigation easy as it is difficult to misclick a button. Furthermore, the headings can be clicked to sort the details by alphabetical order, ascending or descending. The tables each hold data from only 1 entity, so that the user does not get confused with the data in the table.

#### Objective: Preventing Duplication

This objective has not been fulfilled, as the system does not run checks to see whether the data is already existent in the database. The answer to question 12 of the questionnaire, on page 385 shows that the user disagrees with whether the system prevents unnecessary duplication efficiently. The client has said that the system does not warn the user of the duplication, which proves that the system does not run any of these checks. However, the use of ID's still make each author unique, and the client has said the searches can be easily conducted to check whether the data is existent or not.

#### Objective: Simple interface for entering data, meaning it can be conducted quickly.

This objective has been fulfilled, as entry boxes are clearly labelled, and there are only as many boxes as required for each entry. The entry boxes are all in line with each other, making it look tidy instead of looking cluttered. This can be seen in the image below.

**Labels are clearly shown for each box**

**Boxes are aligned with each other and with the corresponding labels**

Figure 6.2: Simple interface for entering data

The client has also strongly agreed with questions 2 and 3, which show that they think that there isn't a problem with the interfaces for adding data. Each different interface for adding an certain entry has a specific title to show what is being added, which makes the current entry clear to the user.

Objective: Search Function to find a specific customer in the database

This objective has been fulfilled, as there are search functions in the system which can search for customers. In the images below, it is evident that the system has working search functions.

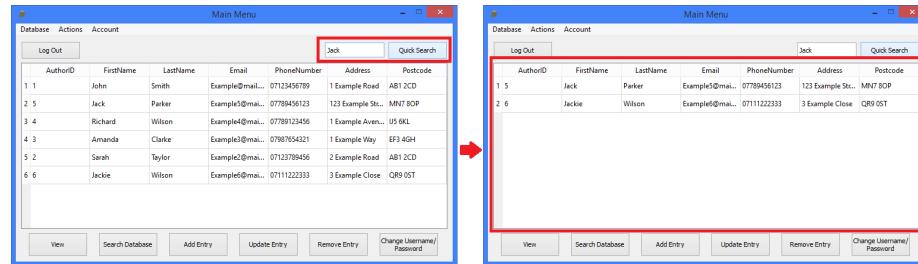


Figure 6.3: Search Function which can find specific customers

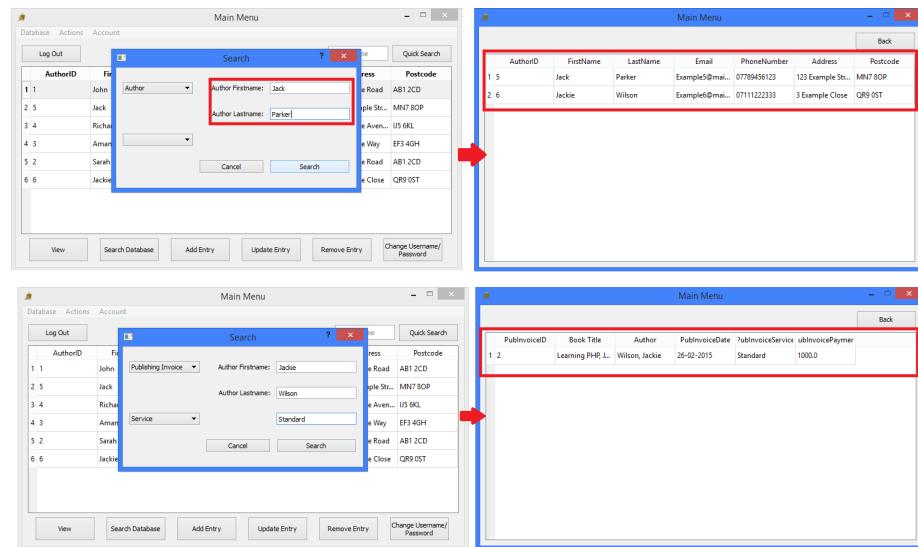


Figure 6.4: Search Function which can find specific customers and more

Furthermore, the client has agreed to the statement in question 4 of the questionnaire on page 384, meaning that they found the search functions up to the required standard and useful. This proves that the objective has been met. However, the data cannot be interacted with upon finding it, and this was part of the full objective of the search functionality, therefore i can conclude that this objective was partially met, even though the client was content with it.

Objective: Ability to edit existing data easily and quickly

This objective has been met, as all the user needs to do in the system is navigate to the data, click on it, click edit, make their changes and confirm them. This can be seen in the image below, where it is evident that the system is capable of editing data efficiently.

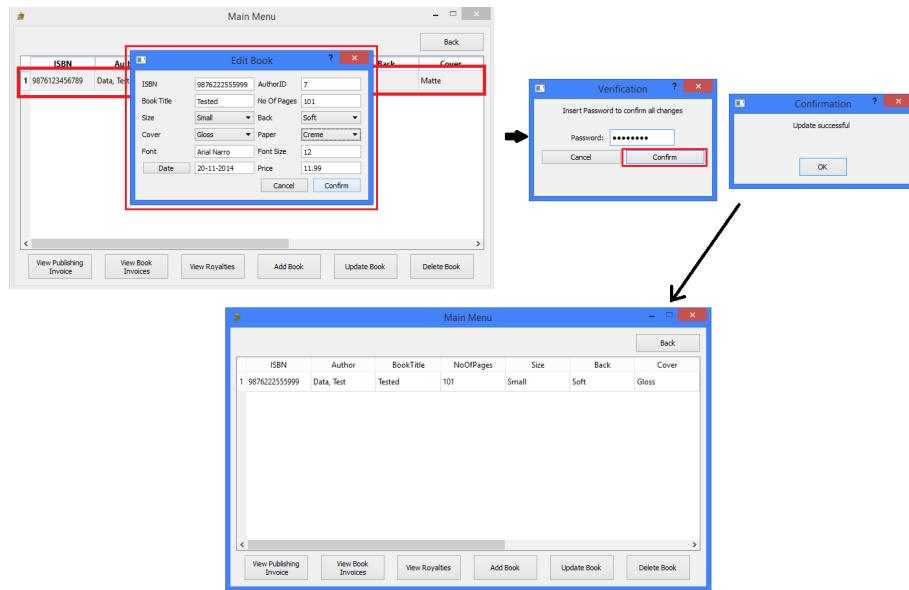


Figure 6.5: User finds data, clicks on it, clicks edit, makes changes, and confirms changes.

The client also strongly agreed with the statement in question 5, showing that they are content with the editing functionality in the system, and that it successfully edits and saves changes when the user verifies the changes with their credentials. This proves that the objective has been fulfilled.

#### Objective: Ability to calculate royalties using given data

This objective has been met, as the system calculates the price of a given set of items, then it sums up the prices of all the items into one royalty payment. This can be seen in the image below, where the calculations for a single set of items are displayed, and the full payment is also displayed on the royalties screen.

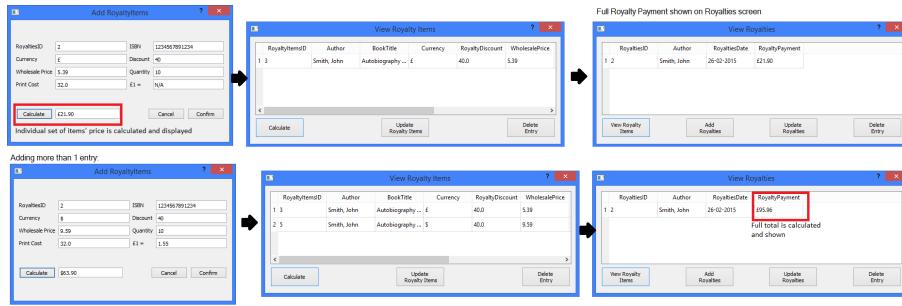


Figure 6.6: Individual calculations and full calculations are shown.

However, the client was merely satisfied with this functionality and this can be seen in question 14 of the questionnaire on page 385, where the client has said that they were satisfied with the fact that it completes the task set, but it is somewhat confusing as the user is required to keep track of which book they are currently interacting with. This could possibly be improved so that the customer is shown what book they are currently adding to, and can change between books somewhere on the interface, in order to avoid confusion over this. Therefore I can conclude that this objective has been mostly fulfilled.

### 6.1.2 Summary

The system has fulfilled half of the objectives required which were set in the system specification, and has failed to meet one of them, as seen in the graph below.



The main reason why half of these objectives were not completely fulfilled is that I did not have enough time to finish the development of my system, which caused the validation check for duplicates in the system and the interactions with search results to be left out. However, the reason why the objective for calculations of royalty payments was not fully fulfilled is that the client was not in agreement with whether the royalty/book invoice items organisation was clear or not. This was because of how the user had to remember what book they had selected instead of it being there in front of them. The system needed to be clear on this, which it seemingly was not.

To summarise, I have concluded that the system only just meets the requirements specified by my client, because it only failed to meet 1 objective out of 6, and the objectives that were partially met were more inclined to being fully completed. Also, the client has accepted the system as it is a major improvement on the current system that they use.

Imran Rahman

Candidate No. 30928

Centre No. 22151

---

## 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

1 – Strongly disagree    2 - Disagree    3 - Neither    4 - Agree    5 – Strongly agree

1. How far do you agree that the clarity and organisation of the interface layout is adequate for the user to navigate sufficiently?

5 Large buttons make navigation through interfaces, clear with titles and windows.

2. How far do you agree that the system has an effective method of adding customer data to the database?

5

3. How far do you agree that it is easy to add data about customers' books?

5

4. How far do you agree that the system has fulfilled the objective of being able to search for specific data in the database?

5

5. How far do you agree that data can be edited efficiently and is saved to the correct place?

5

6. How far do you agree that calculations are accurate and quickly performed?

384

4 Calculations are very quick and fairly simple  
may cause issues in future to conduct, But fixed prices could cause issues if printing prices change.

7. How far do you agree that the system gives a clear prompt upon finding an invalid entry during adding/editing data?

5

8. How far do you agree that the system is safe and secure to use?

9. How far do you agree that the system's use of combo boxes prevent human error when adding/editing entries?

5

10. How far do you agree that the system has a clear and effective way for the user to recover login credentials via email?

5

11. How far do you agree that the system has secure and effective login functionality?

5

12. How far do you agree that the system has an efficient method of preventing unnecessary duplication?

2 The system does not warn me when adding a duplicate but the clarity of the database tables and the search functions make it easy for me to check before adding.

13. How far do you agree that the system makes changing your username/password clear and simple?

5

14. How far do you agree that the system has a clear organisation and use of royalty/book invoice items?

3 A little confusing at first, with what book I'm currently adding items to but it does the job.

Figure 6.8: Page 2 of Questionnaire

**6.7.2 Graphs**

**6.7.3 Written Statements**