

Computing A2 Coursework

Imran Rahman

March 4, 2015

Contents

1 Analysis	5
1.1 Introduction	5
1.1.1 Client Identification	5
1.1.2 Define the current system	6
1.1.3 Describe the problems	6
1.1.4 Section appendix	7
1.2 Investigation	9
1.2.1 The current system	9
1.2.2 The proposed system	17
1.3 Objectives	22
1.3.1 General Objectives	22
1.3.2 Specific Objectives	22
1.3.3 Core Objectives	23
1.3.4 Other Objectives	23
1.4 ER Diagrams and Descriptions	25
1.4.1 ER Diagram	25
1.4.2 Entity Descriptions	26
1.5 Object Analysis	26
1.5.1 Object Listing	26
1.5.2 Relationship diagrams	28
1.5.3 Class definitions	29
1.6 Other Abstractions and Graphs	30
1.7 Constraints	31
1.7.1 Hardware	31
1.7.2 Software	31
1.7.3 Time	31
1.7.4 User Knowledge	31
1.7.5 Access restrictions	32
1.8 Limitations	32
1.8.1 Areas which will not be included in computerisation	32
1.8.2 Areas considered for future computerisation	32
1.9 Solutions	33
1.9.1 Alternative solutions	33

1.9.2	Justification of chosen solution	33
2	Design	35
2.1	Overall System Design	35
2.1.1	Short description of the main parts of the system	35
2.1.2	System flowcharts showing an overview of the complete system	37
2.2	User Interface Designs	45
2.3	Hardware Specification	55
2.4	Program Structure	55
2.4.1	Top-down design structure charts	55
2.4.2	Algorithms in pseudo-code for each data transformation process	57
2.4.3	Object Diagrams	63
2.4.4	Class Definitions	64
2.5	Prototyping	67
2.6	Definition of Data Requirements	71
2.6.1	Identification of all data input items	71
2.6.2	Identification of all data output items	72
2.6.3	Explanation of how data output items are generated	73
2.6.4	Data Dictionary	73
2.6.5	Identification of appropriate storage media	75
2.7	Database Design	76
2.7.1	Normalisation	76
2.7.2	SQL Queries	82
2.8	Security and Integrity of the System and Data	84
2.8.1	Security and Integrity of Data	84
2.8.2	System Security	84
2.9	Validation	84
2.10	Testing	88
2.10.1	Outline Plan	88
2.10.2	Detailed Plan	88
3	Testing	102
3.1	Test Plan	103
3.1.1	Original Outline Plan	103
3.1.2	Changes to Outline Plan	104
3.1.3	Original Detailed Plan	104
3.1.4	Changes to Detailed Plan	117
3.2	Test Data	143
3.2.1	Original Test Data	143
3.2.2	Changes to Test Data	144
3.3	Annotated Samples	144
3.3.1	Actual Results	144
3.3.2	Evidence	145
3.4	Evaluation	186

3.4.1	Approach to Testing	186
3.4.2	Problems Encountered	186
3.4.3	Strengths of Testing	186
3.4.4	Weaknesses of Testing	187
3.4.5	Reliability of Application	187
3.4.6	Robustness of Application	187
4	System Maintenance	188
4.1	Environment	188
4.1.1	Software	188
4.1.2	Usage Explanation	188
4.1.3	Features Used	189
4.2	System Overview	191
4.2.1	System Component	191
4.3	Code Structure	191
4.3.1	Particular Code Section	191
4.4	Variable Listing	191
4.5	System Evidence	191
4.5.1	User Interface	191
4.5.2	ER Diagram	191
4.5.3	Database Table Views	191
4.5.4	Database SQL	191
4.5.5	SQL Queries	191
4.6	Testing	191
4.6.1	Summary of Results	191
4.6.2	Known Issues	191
4.7	Code Explanations	191
4.7.1	Difficult Sections	191
4.7.2	Self-created Algorithms	191
4.8	Settings	191
4.9	Acknowledgements	191
4.10	Code Listing	192
4.10.1	Module 1	193
4.10.2	Module 2	206
4.10.3	Module 3	252
4.10.4	Module 4	255
4.10.5	Module 5	256
4.10.6	Module 6	257
4.10.7	Module 7	260
4.10.8	Module 8	264
4.10.9	Module 9	266
4.10.10	Module 10	281
4.10.11	Module 11	283
4.10.12	Module 12	288
4.10.13	Module 13	296
4.10.14	Module 14	299

4.10.15 Module 15	306
4.10.16 Module 16	307
4.10.17 Module 17	308
4.10.18 Module 18	311
5 User Manual	315
5.1 Introduction	316
5.2 Installation	316
5.2.1 Prerequisite Installation	316
5.2.2 System Installation	316
5.2.3 Running the System	316
5.3 Tutorial	316
5.3.1 Introduction	316
5.3.2 Assumptions	316
5.3.3 Tutorial Questions	316
5.3.4 Saving	316
5.3.5 Limitations	316
5.4 Error Recovery	316
5.4.1 Error 1	316
5.4.2 Error 2	316
5.5 System Recovery	316
5.5.1 Backing-up Data	316
5.5.2 Restoring Data	316
6 Evaluation	317
6.1 Customer Requirements	318
6.1.1 Objective Evaluation	318
6.2 Effectiveness	318
6.2.1 Objective Evaluation	318
6.3 Learnability	318
6.4 Usability	318
6.5 Maintainability	318
6.6 Suggestions for Improvement	318
6.7 End User Evidence	318
6.7.1 Questionnaires	318
6.7.2 Graphs	318
6.7.3 Written Statements	318

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, £5.00
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.

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

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

Algorithms

In the current system there are two Algorithms which are being used. The first sends an invoice to the customer and checks whether Shahida has 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

```

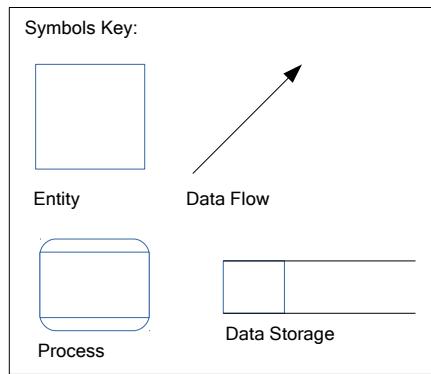
1: SET Payment TO false
2:
    Check Website for Price
    Create Invoice
    Send Invoice
3: WHILE Payment = false DO
    Check For Payment
4:     IF PaymentReceived THEN
        Payment = true
5: END IF
6: END WHILE

```

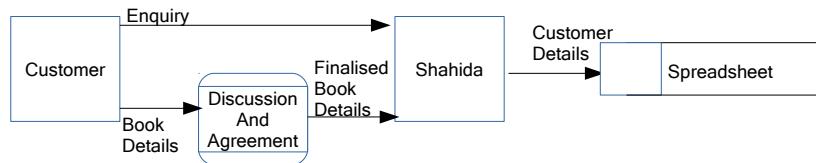
Algorithm 2 Second Algorithm - Completing Work and Checking If Work is Completed

```
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:

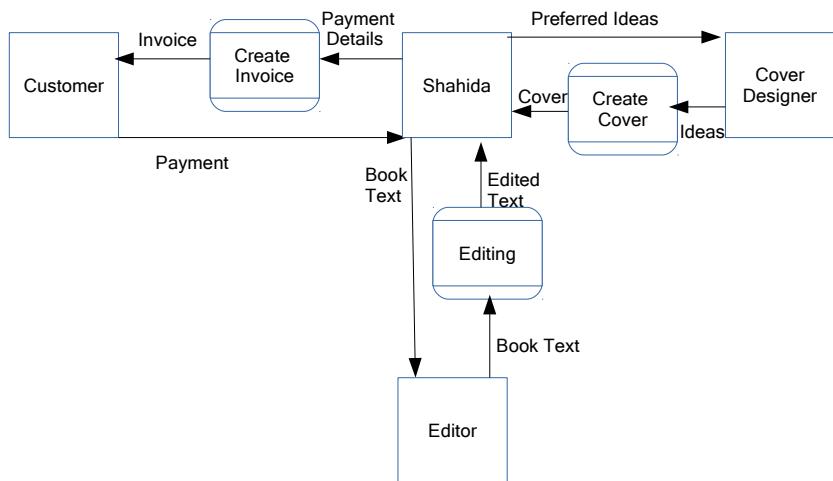


Figure 1.3: Data Flow Diagrams

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 Stoppani
via Al Marcadello 2
CH-6988 Ponte Tresa
Switzerland

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

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

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

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

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

Figure 1.4: Invoice Example

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

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



ROYALTY STATEMENT

Date: 01-01-14 – 30-06-14

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

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

 PRINT COST PER BOOK: £3.70 UK AND \$4.80 US

*\$91.01 = £53.19

1 GBP = 1.71565 USD 09-07-14

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

Figure 1.5: Royalty Statement Example

1.2.2 The proposed system

In the proposed system the Customer's information will still be received through the online form on the company's website, which Shahida receives via email. She will then enter this into the system using a new interface that will ask her for the details. This will be placed into a database. Each Customer's book will have a primary key, the ISBN number which Shahida assigns to the book. In a 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

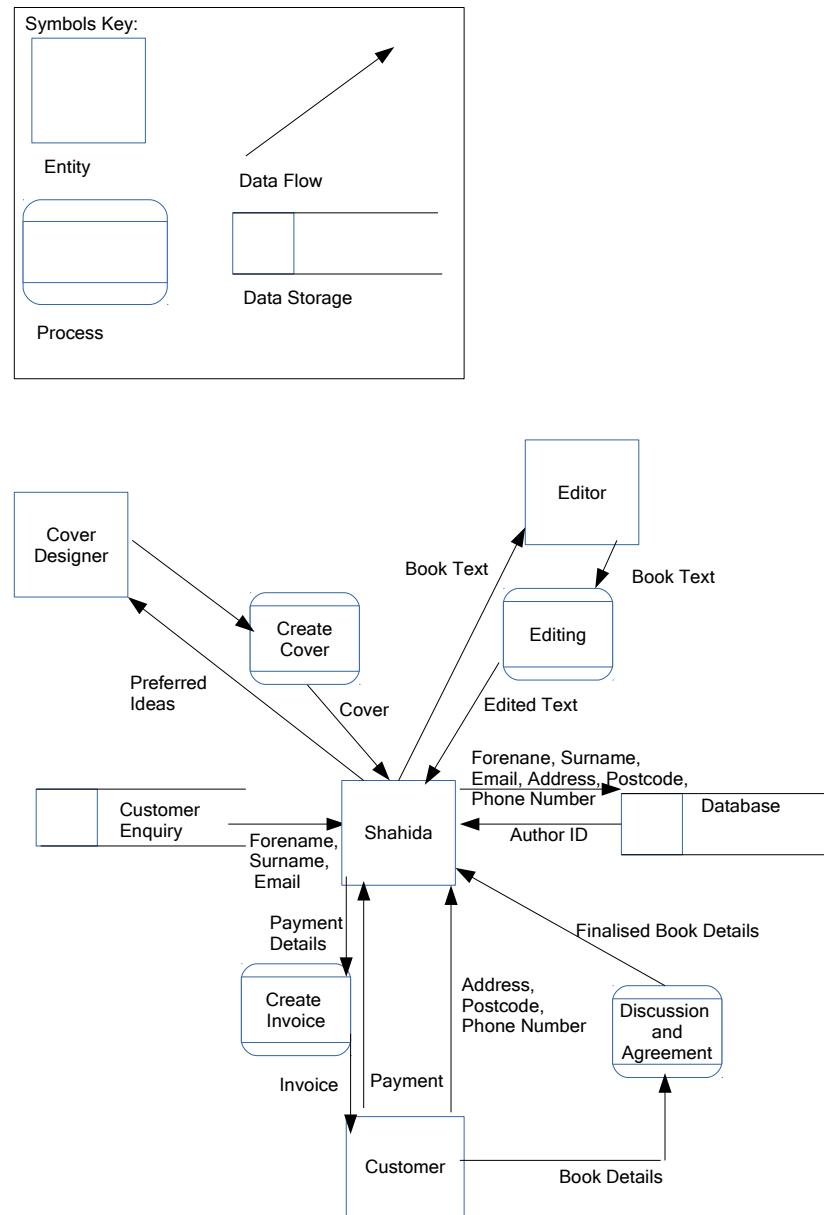


Figure 1.6: Data Flow Diagram

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.

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

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

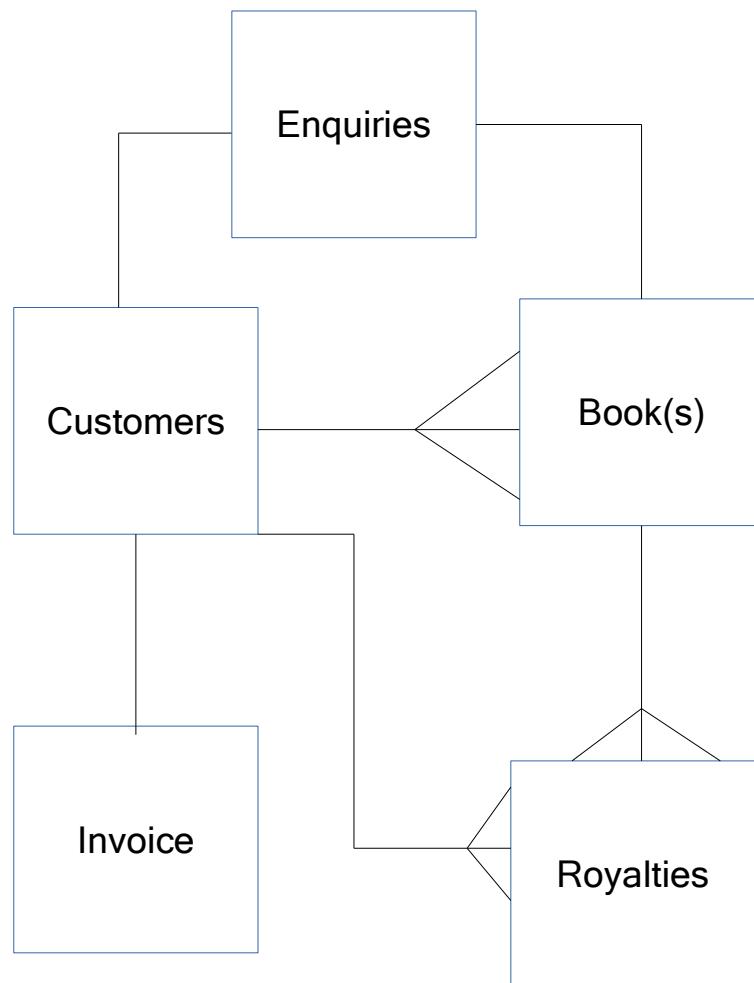
Candidate No. 30928

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

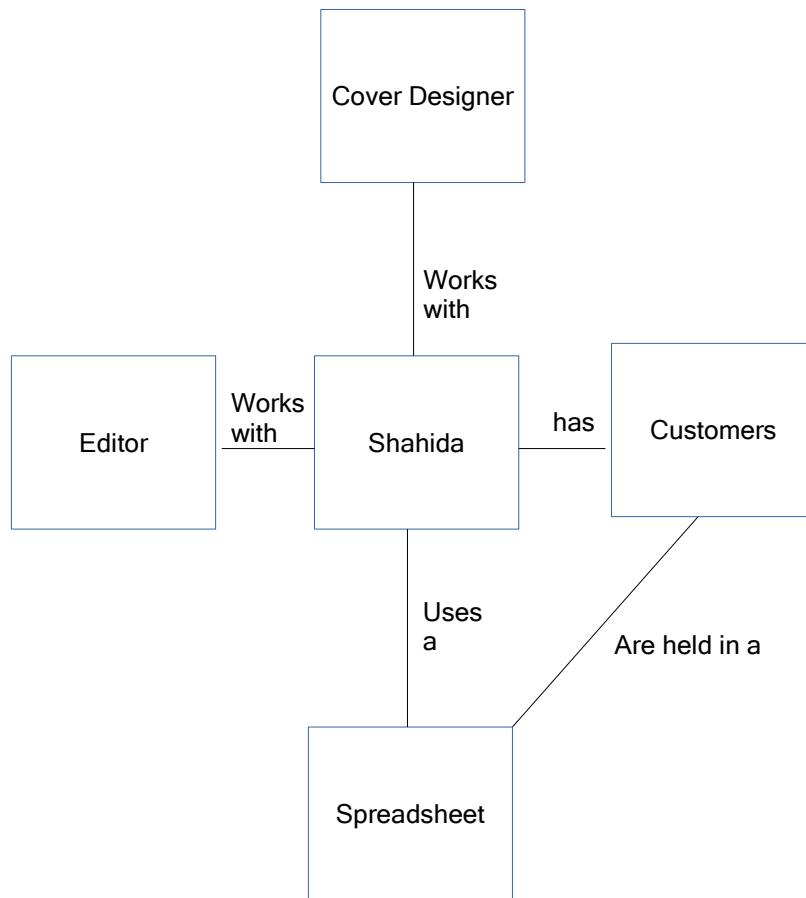


Figure 1.8: Relationship Diagram

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

Book
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

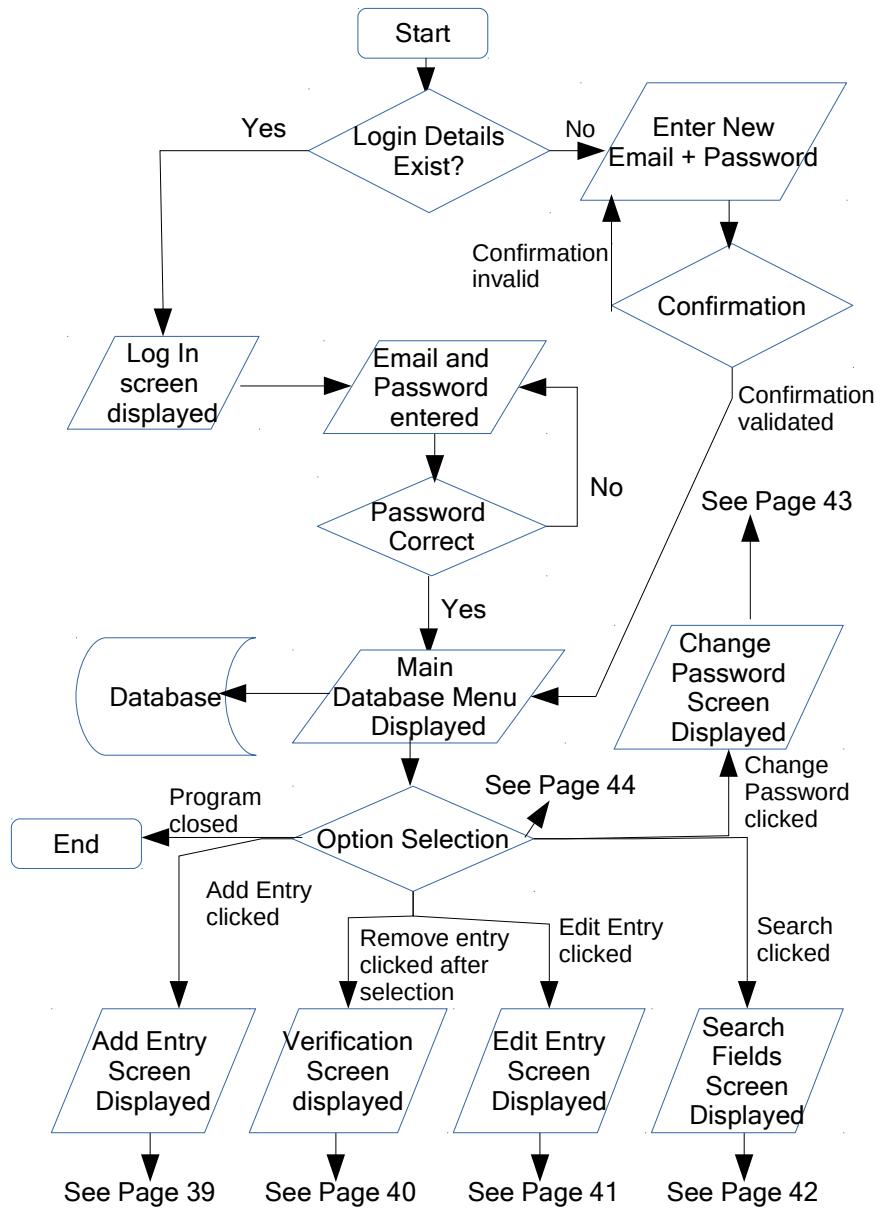


Figure 2.2: Flowchart 2

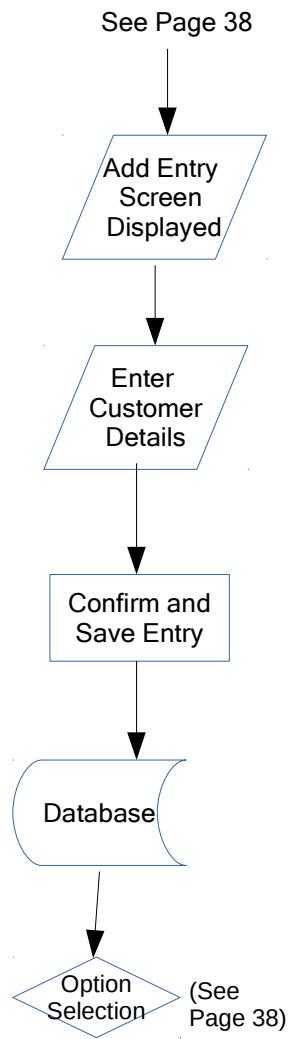


Figure 2.3: Flowchart 3

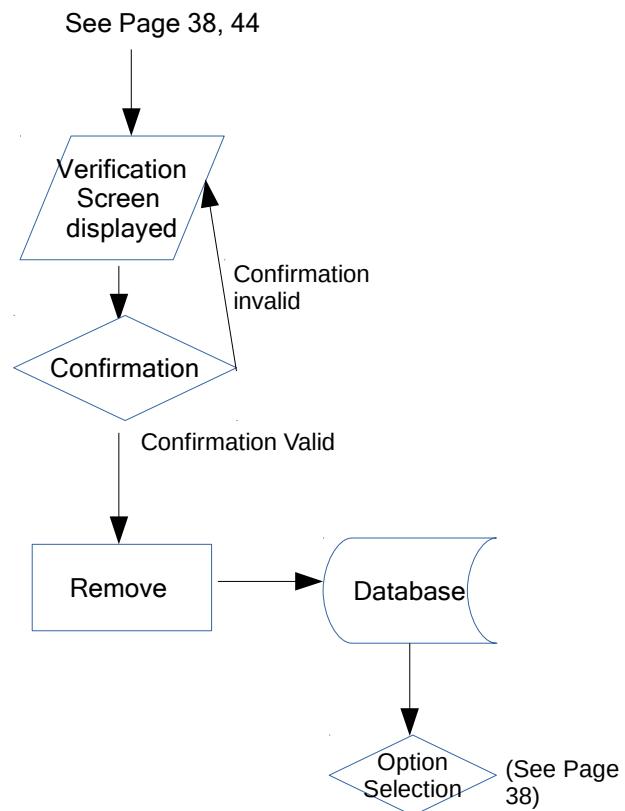


Figure 2.4: Flowchart 4

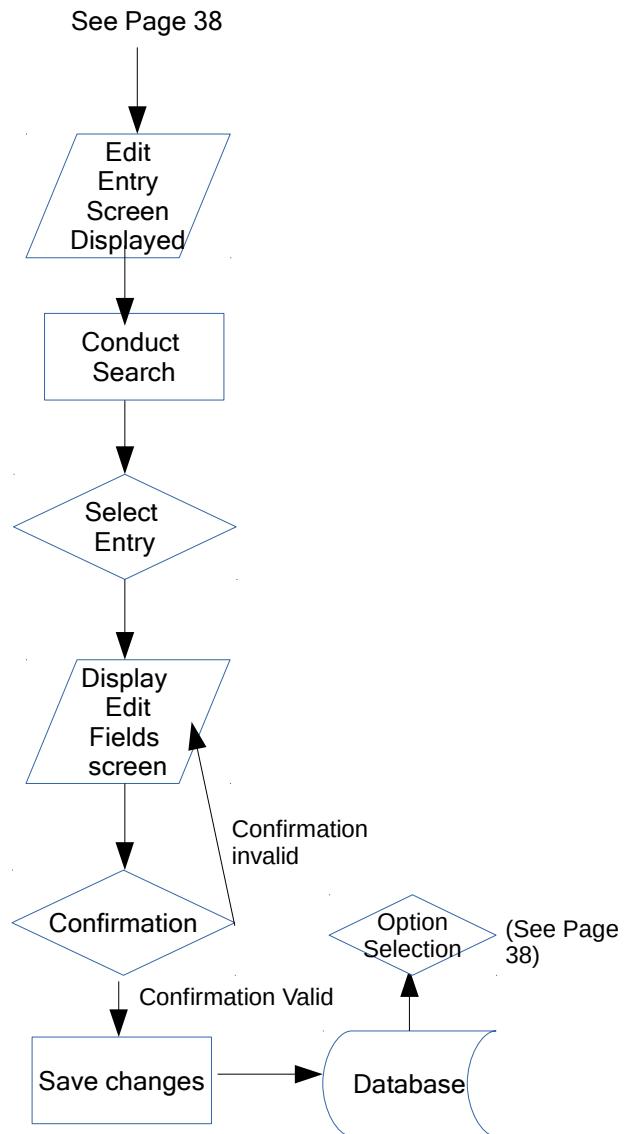


Figure 2.5: Flowchart 5

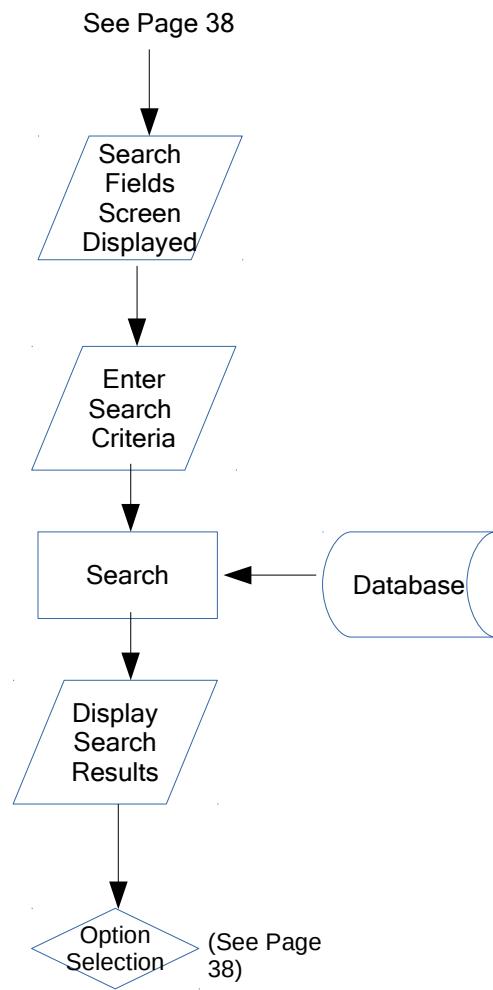


Figure 2.6: Flowchart 6

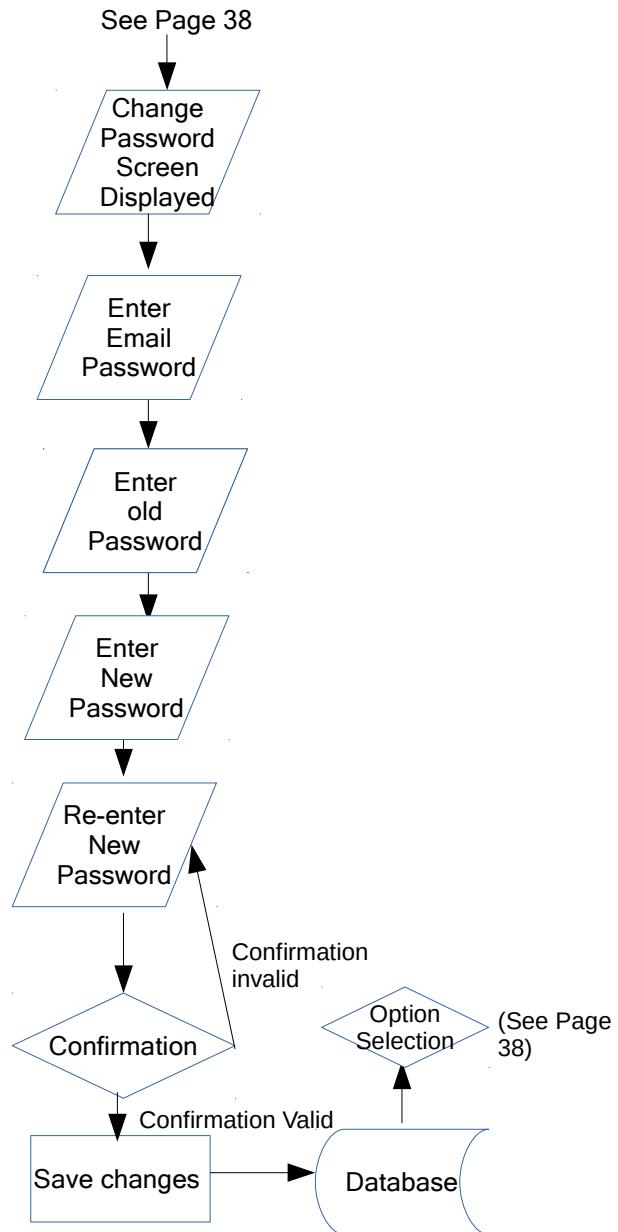
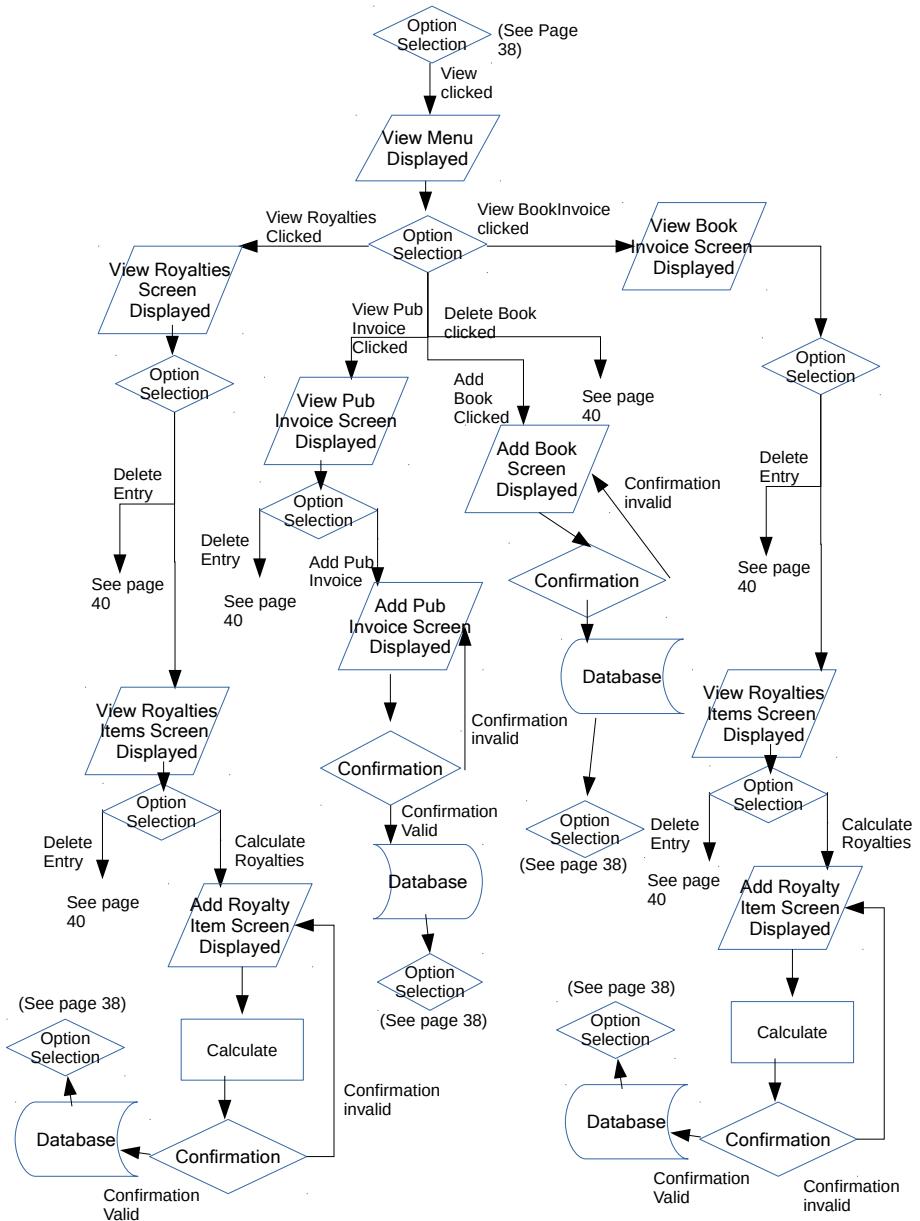


Figure 2.7: Flowchart 7



2.2 User Interface Designs

Figure 2.8: Login Screen and Main Menu

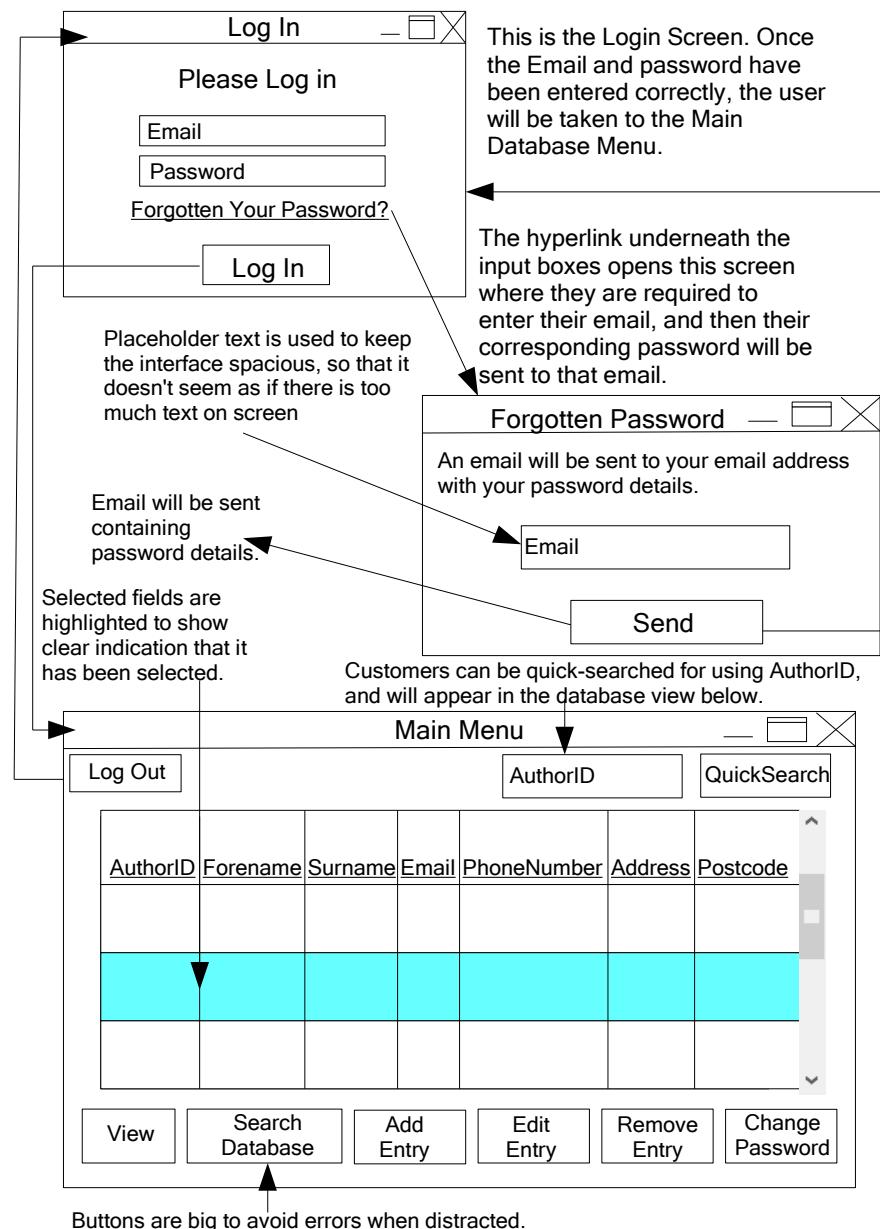


Figure 2.9: View Menu and Royalties

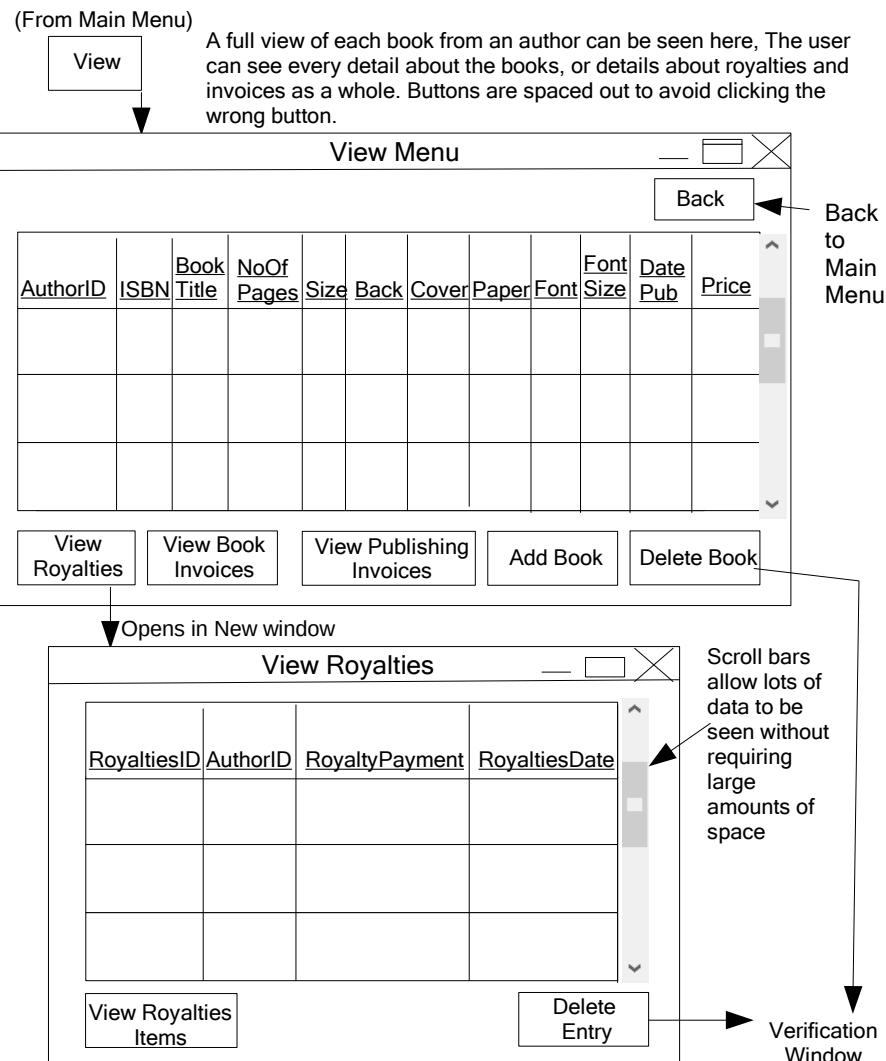


Figure 2.10: RoyaltiesItems and BookInvoice and Items

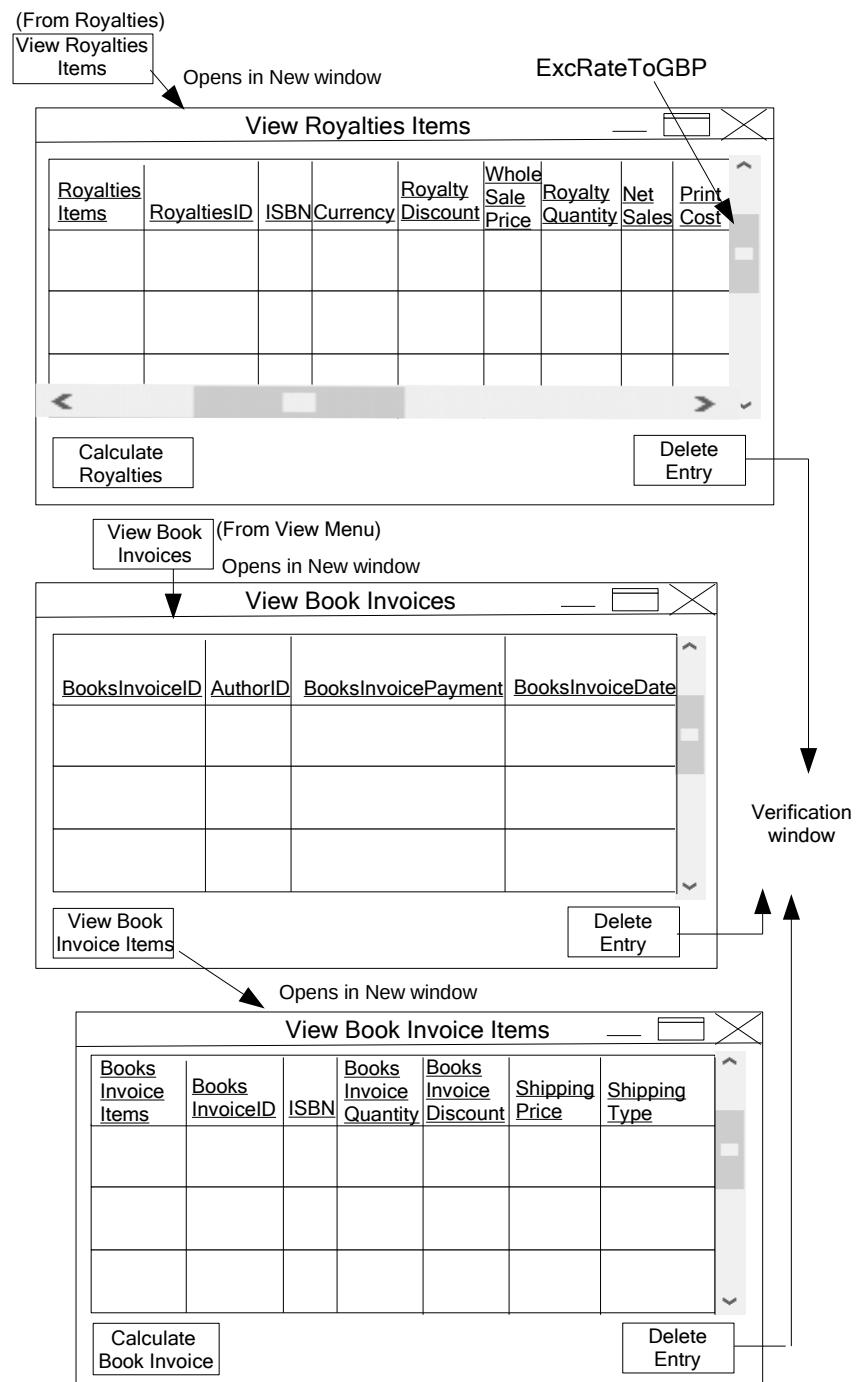


Figure 2.11: Publishing Invoice and Adding Entries

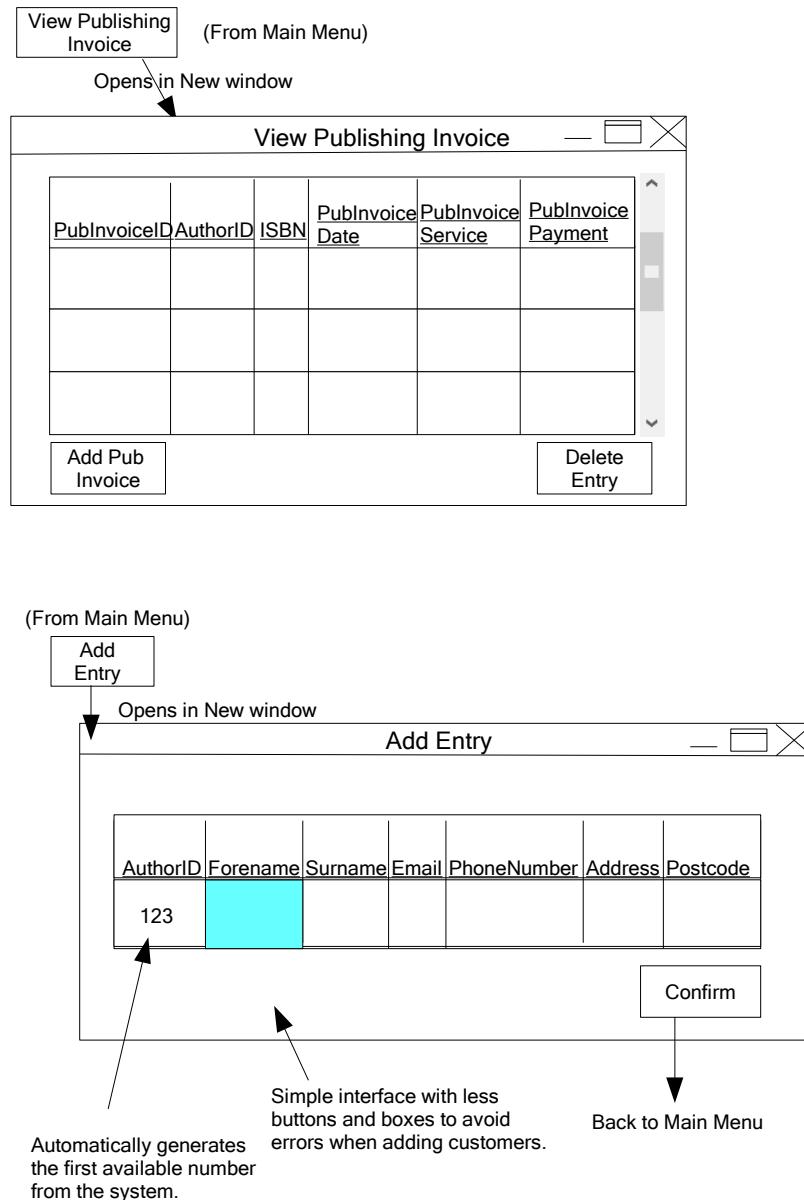


Figure 2.12: Search

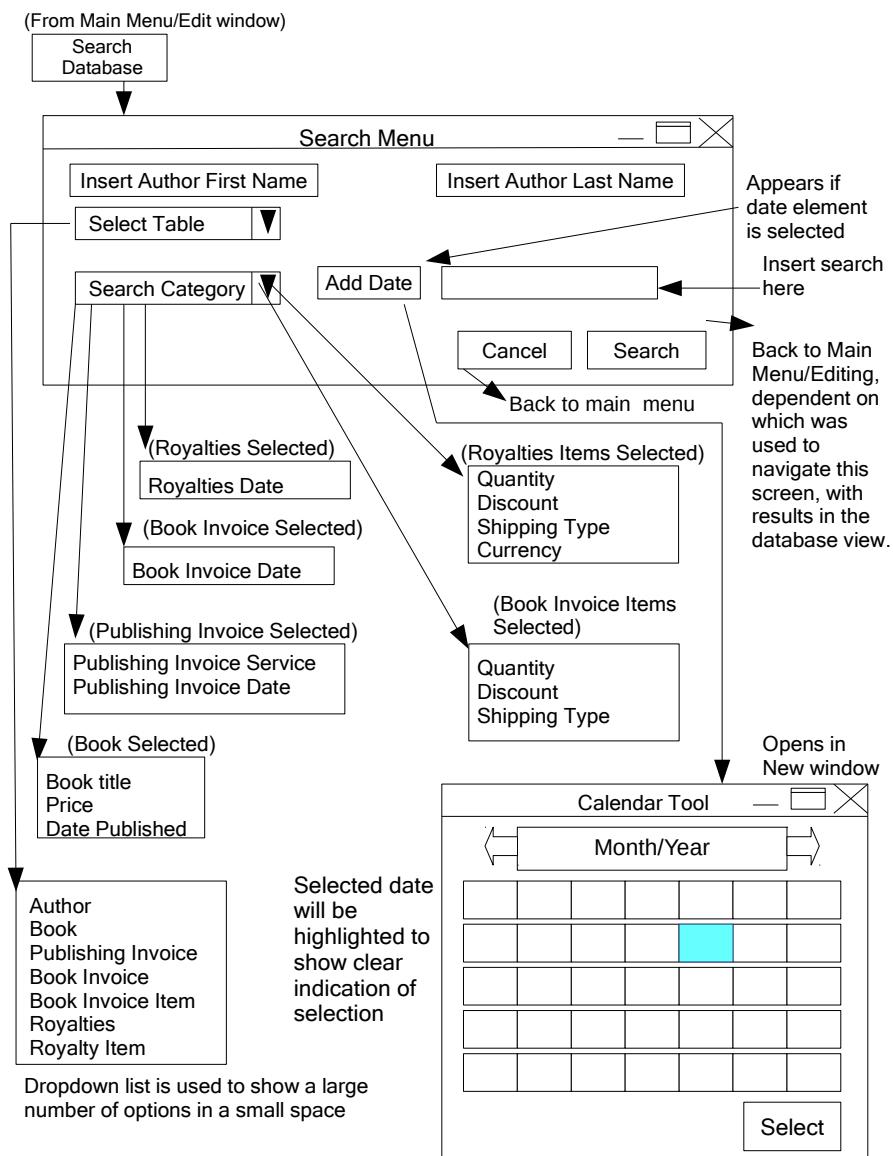


Figure 2.13: Editing Screens

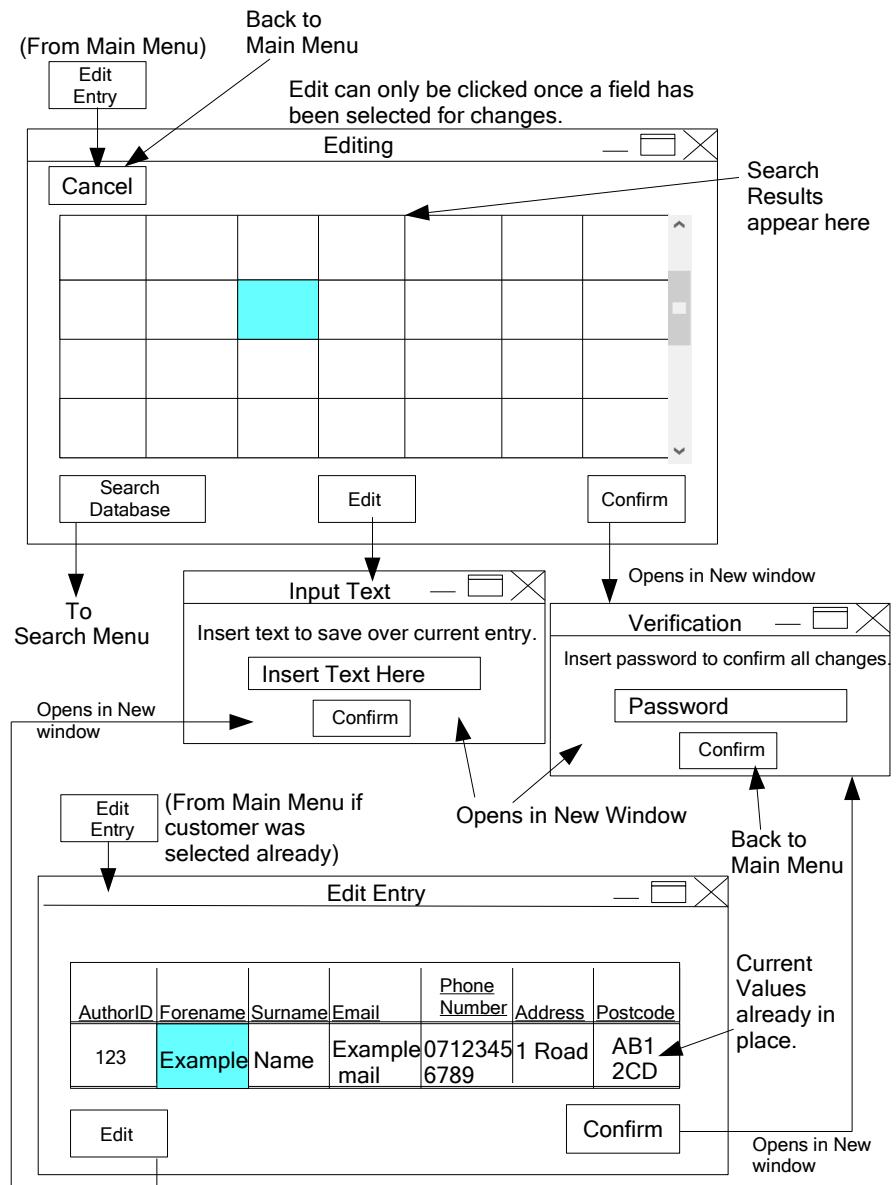


Figure 2.14: Remove Entry and Change Password

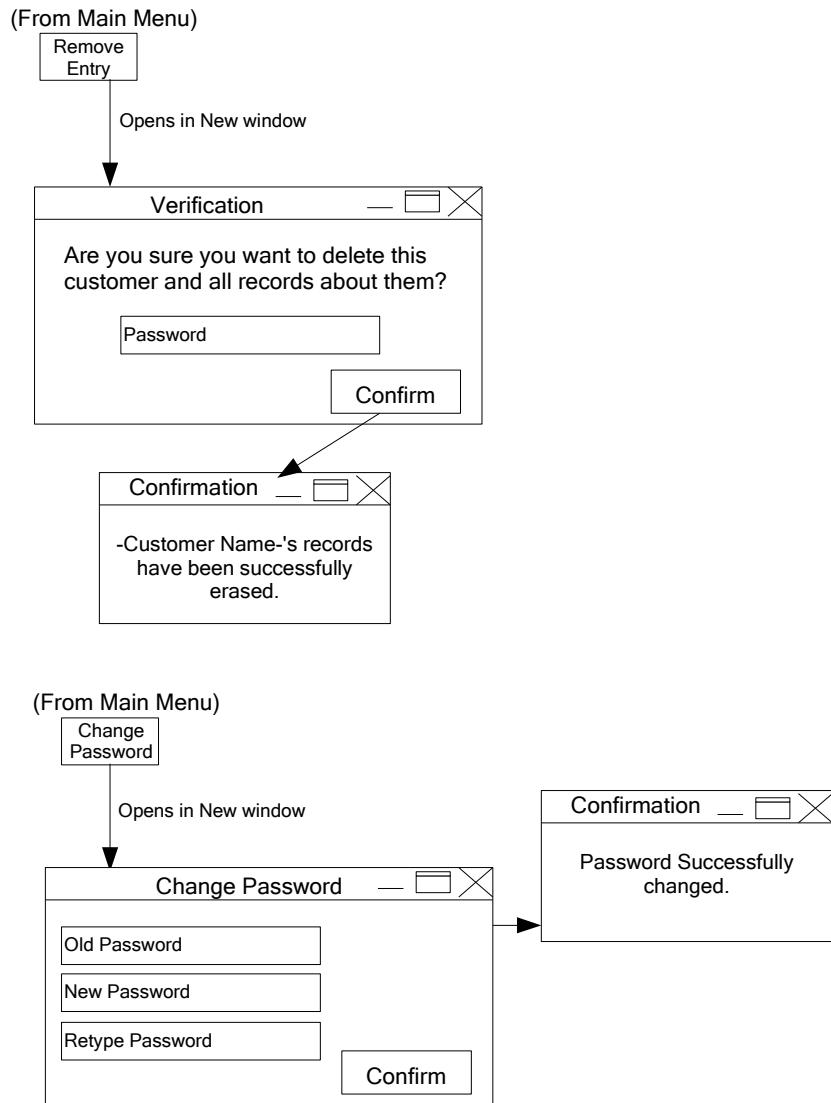


Figure 2.15: Calculations

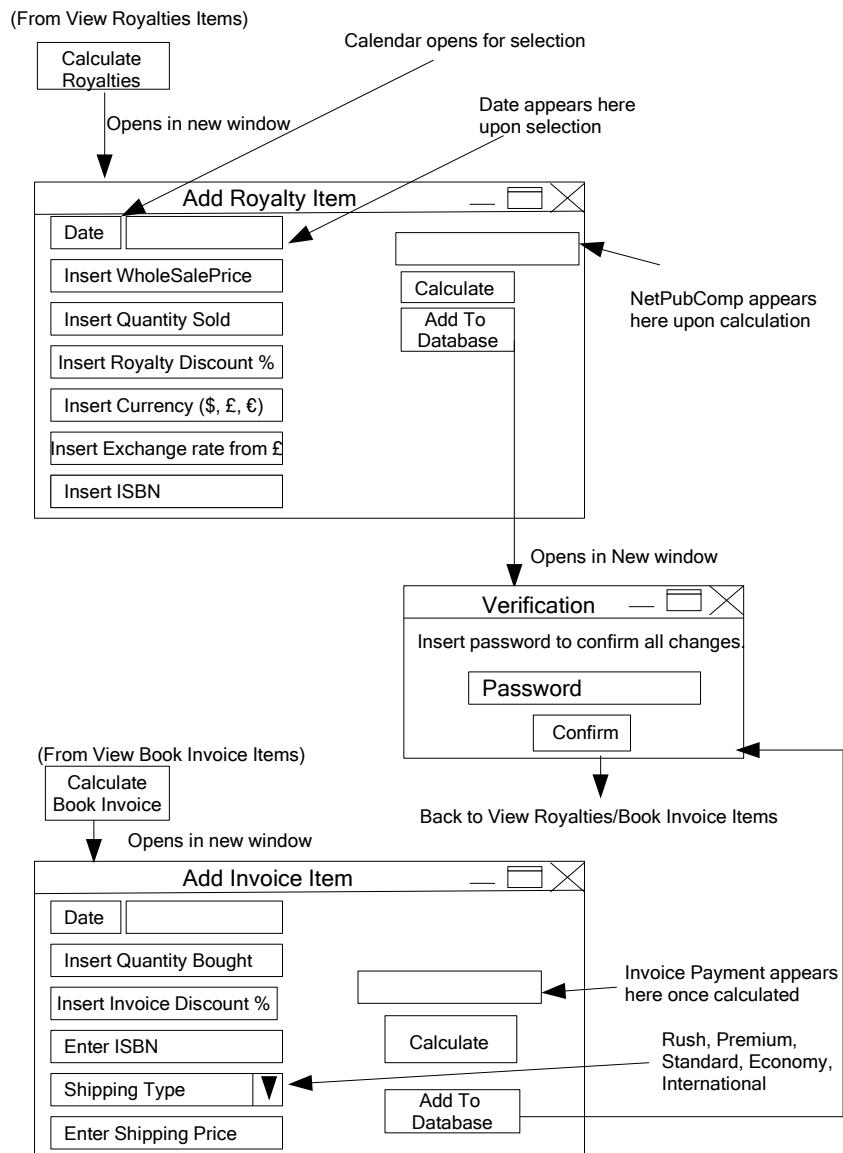


Figure 2.16: Adding Publishing Invoices

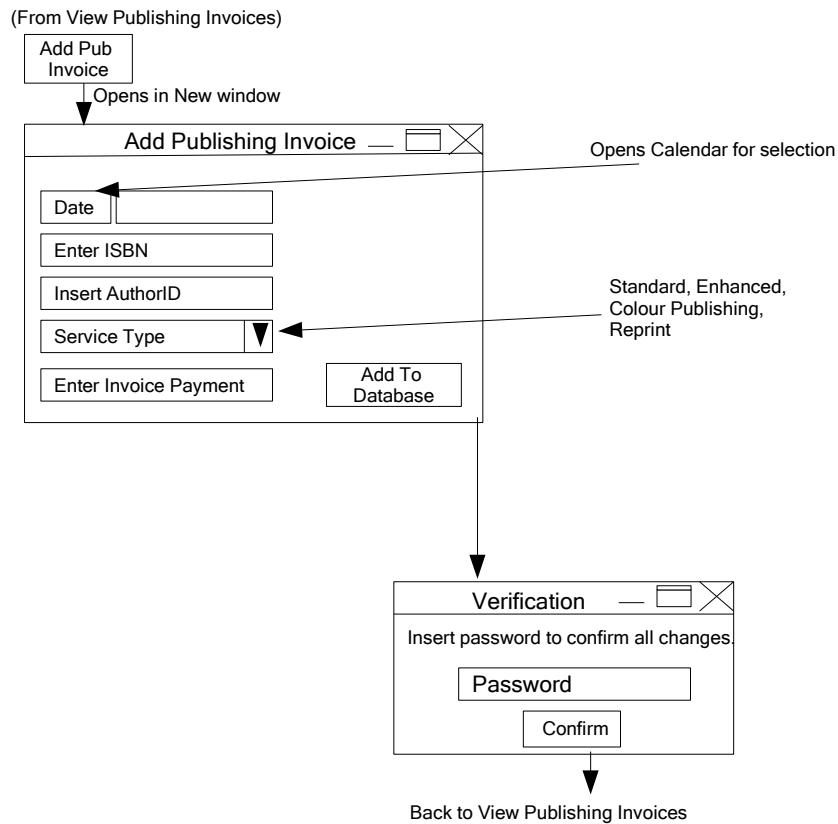
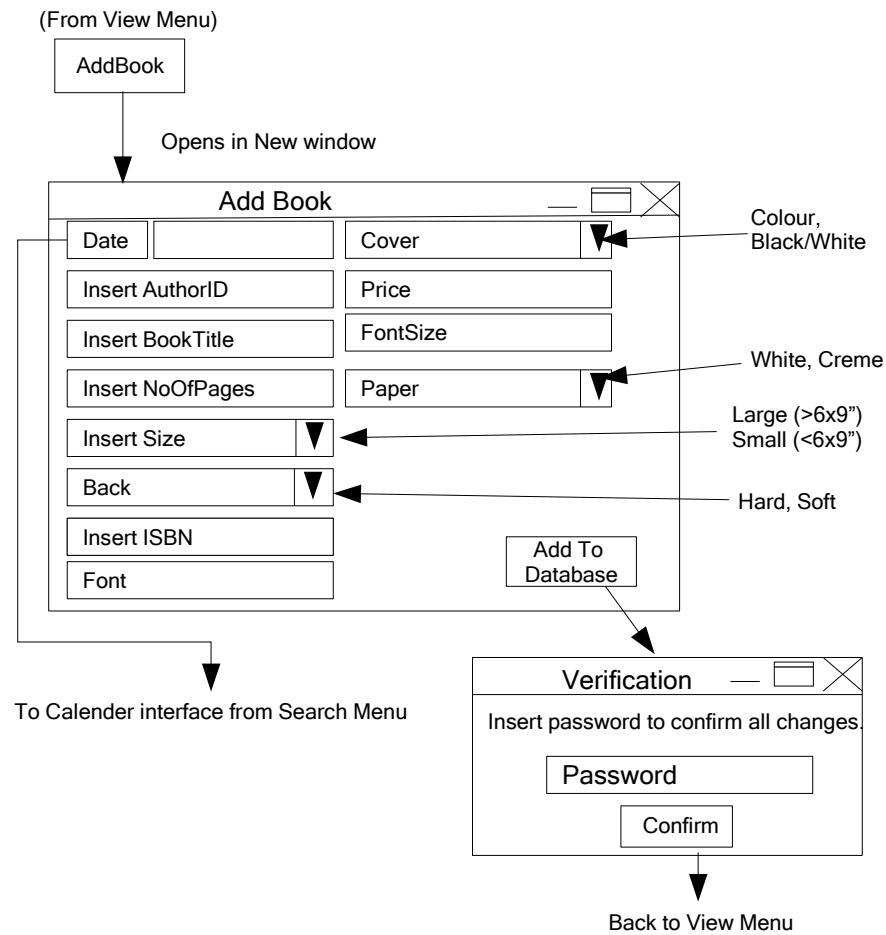


Figure 2.17: Add Book



2.3 Hardware Specification

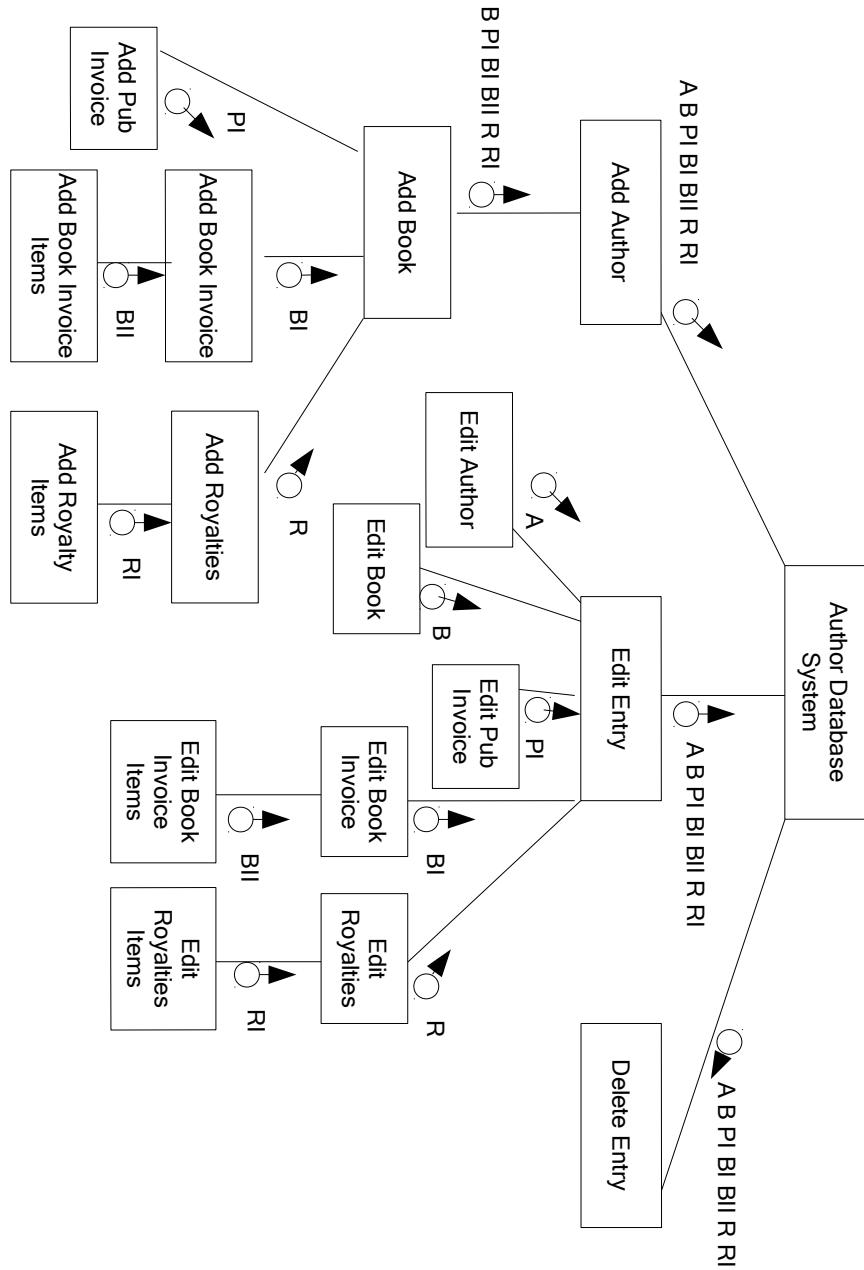
The system needs to be able to run on a laptop with a 1366 x 768, 16:9 aspect ratio screen which runs on Windows 8. This is imperative to the size of the application I will be creating because it must fit on the given screen size and can't be resizable. A mouse or touchpad will be used for navigational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

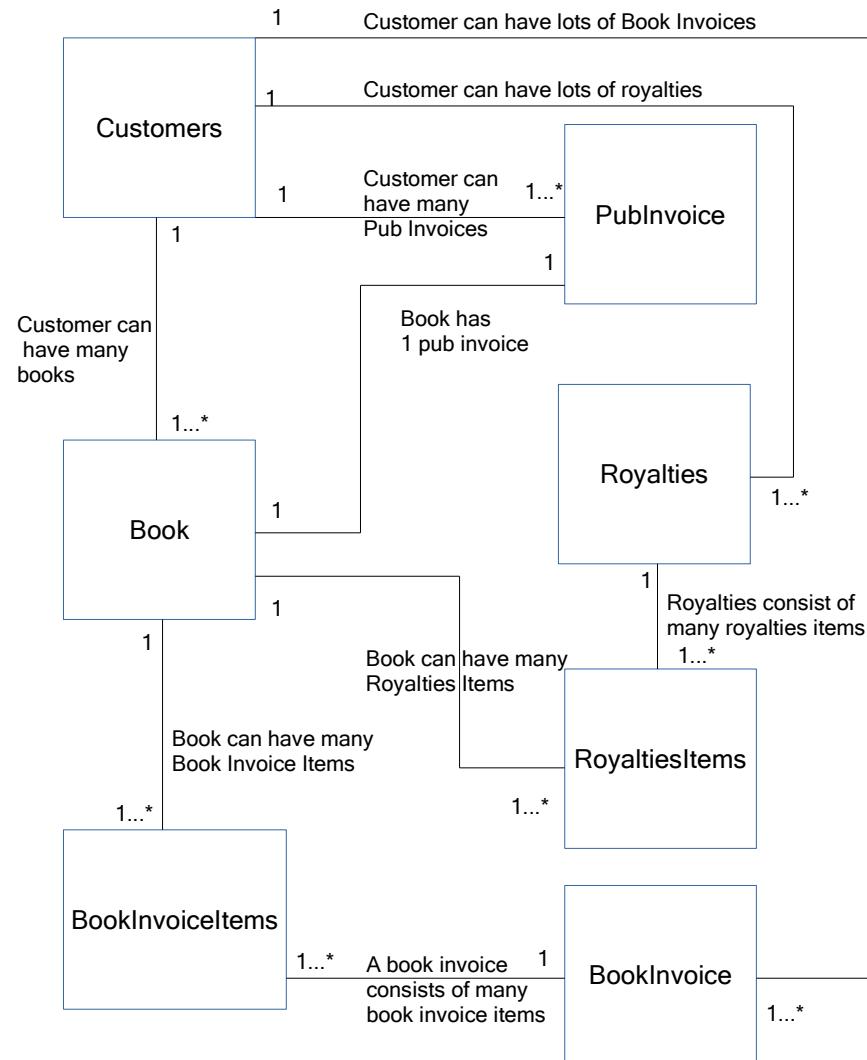
```
1: function ADDINVOICEITEM(CustomerTable)
2:   SET AddToDatabaseClicked TO False
   DatePublished = null
   ISBN = null
   AuthorID = null
   BookTitle = null
   NoOfPages = null
   Size = null
   Back = null
   Cover = null
   Paper = null
   Font = null
   FontSize = null
   Price = null
3:   WHILE len(DatePublished) <= 0 DO
   INPUT DatePublished
4:   END WHILE
5:   WHILE len(ISBN) <= 0 DO
   INPUT ISBN
6:   END WHILE
7:   WHILE len(BookTitle) <= 0 DO
   INPUT BookTitle
8:   END WHILE
9:   WHILE len(Size) <= 0 DO
   INPUT Size
10:  END WHILE
11:  WHILE len(BackType) <= 0 DO
   INPUT BackType
12:  END WHILE
13:  WHILE len(NoOfPages) <= 0 DO
   INPUT NoOfPages
14:  END WHILE
15:  WHILE len(Paper) <= 0 DO
   INPUT Paper
16:  END WHILE
17:  WHILE len(Font) <= 0 DO
   INPUT Font
18:  END WHILE
19:  WHILE len(FontSize) <= 0 DO
   INPUT FontSize
20:  END WHILE
21:  WHILE len(Price) <= 0 DO
   INPUT Price
22:  END WHILE
23:  IF AddToDatabaseClicked == True THEN
   CONNECT to Book Database
24:  END IF
25: 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

Book
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

Royalties
RoyaltiesID
AuthorID
RoyaltyPayment
RoyaltiesDate
Add AuthorID
Edit AuthorID
Add RoyaltiesDate
Edit Royalties Date

Royalties Items
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

BookInvoice
BookInvoiceID
AuthorID
BookInvoicePayment
BookInvoiceDate
Add AuthorID
Edit AuthorID
Add BookInvoiceDate
Edit BookInvoiceDaate

BookInvoiceItems
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

PubInvoice
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

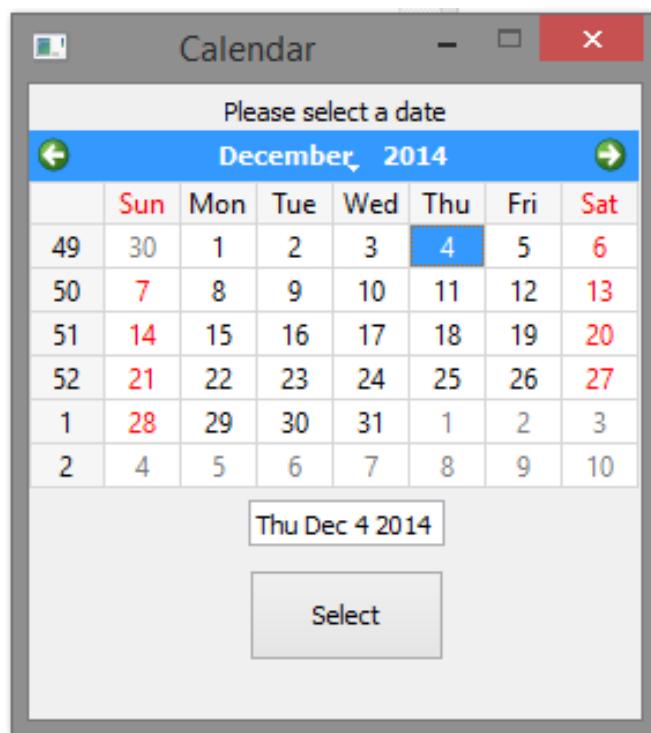
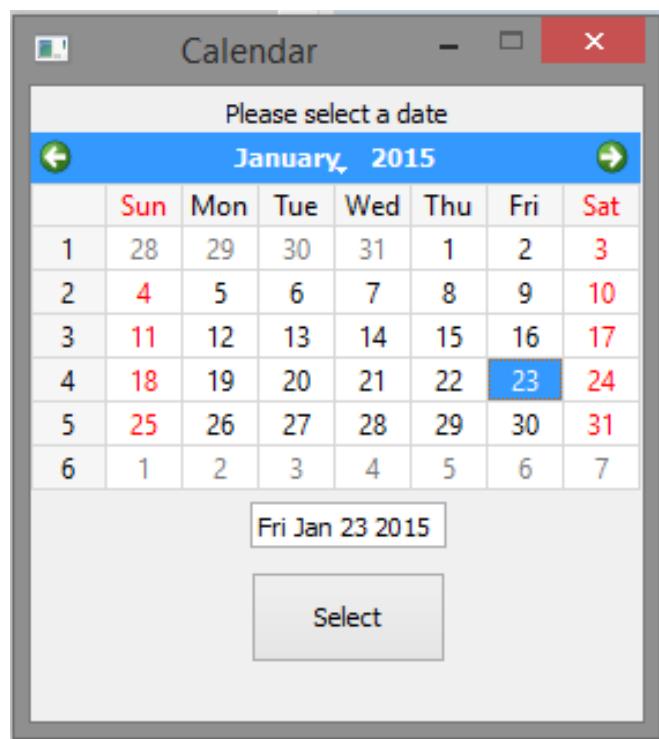


Figure 2.21: Calendar - Date Selection



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

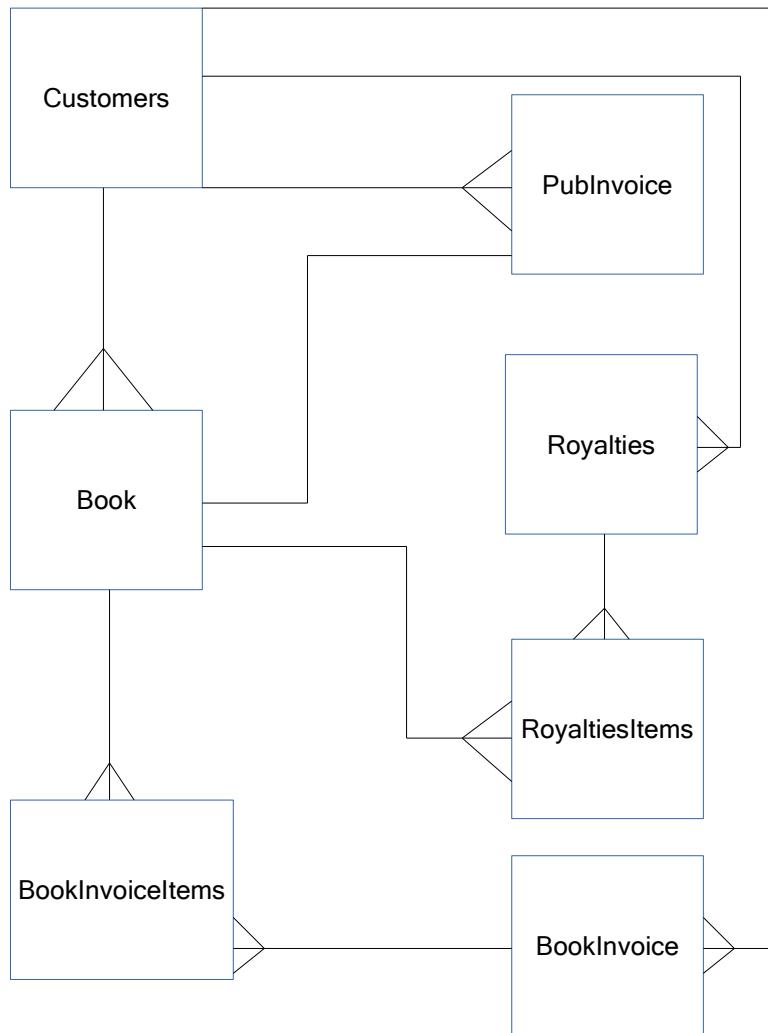
off site, as well as being able to hold all data files on it. The database will be needed to be kept for long term storage. This is the data will be required to be accessed a number of times.

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.

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

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

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

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

AuthorID	ISBN	RoyaltiesID	RoyaltiesItems
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		

PubInvoiceID	BooksInvoiceID	BooksInvoiceItems
<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.

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

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

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

2.10 Testing

2.10.1 Outline Plan

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

2.10.2 Detailed Plan

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

1.1	Test the Log in button on the log in 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

103

3.1 Test Plan

3.1.1 Original Outline Plan

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

3.1.2 Changes to Outline Plan

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

3.1.3 Original Detailed Plan

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

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

1.4	Testing the Search Database button on the Main Menu	This button should prompt a 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 (removed)	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

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 145, and Figure ?? on page ??.

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 148
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 ?? on page ??

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 150
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 151

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 152
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 153

1.10	Testing the Quick Search button on the Main Menu	This button returns the customer that matches the entered Name	Type in a Firstname, Lastname or both and click QuickSearch	Normal	The program should return any customers that match the entered name(s) and show it in the grid	Works as expected	Figure 3.10 on page 154
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 155

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 156
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 157

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 158
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 159

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 160
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 161

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 162
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 163

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 163
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 164

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 164
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 165
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 165

2.2 (removed)	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 166
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 166
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 166

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 166 and Figure ?? on page ??
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 3.26 on page 167

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 3.27 on page 168
-----	---	---	---	---	-----------------------------------	--	-------------------------

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 3.28 on page 169
-----	--	---	---	---	--	---	-------------------------

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 3.29 on page 170
------	---	---	---	---	--	---	-------------------------

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 3.30 on page 171
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 3.31 on page 172 and Figure 3.28 on page 169

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 3.32 on page 173 and Figure 3.29 on page 170
-----	--	--	---	----------------------------------	--	--	---

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 ?? on page ??
-----	--	---	--	----------------------------------	--------------------------	--	----------------------

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 3.34 on page 175
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 3.35 on page 175
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 3.36 on page 175

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 3.37 on page 176
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 3.38 on page 176
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 3.41 on page 178

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 3.39 on page 177
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 3.42 on page 178
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 3.40 on page 177

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 3.43 on page 179
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 3.44 on page 180

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 3.45 on page 181
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 3.46 on page 182
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 ?? on page ??

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 3.48 on page 184
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 3.49 on page 185

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

3.2 Test Data

3.2.1 Original Test Data

The data is shown in the previous table.

3.2.2 Changes to Test Data

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

3.3 Annotated Samples

3.3.1 Actual Results

The data is shown in the previous table.

3.3.2 Evidence

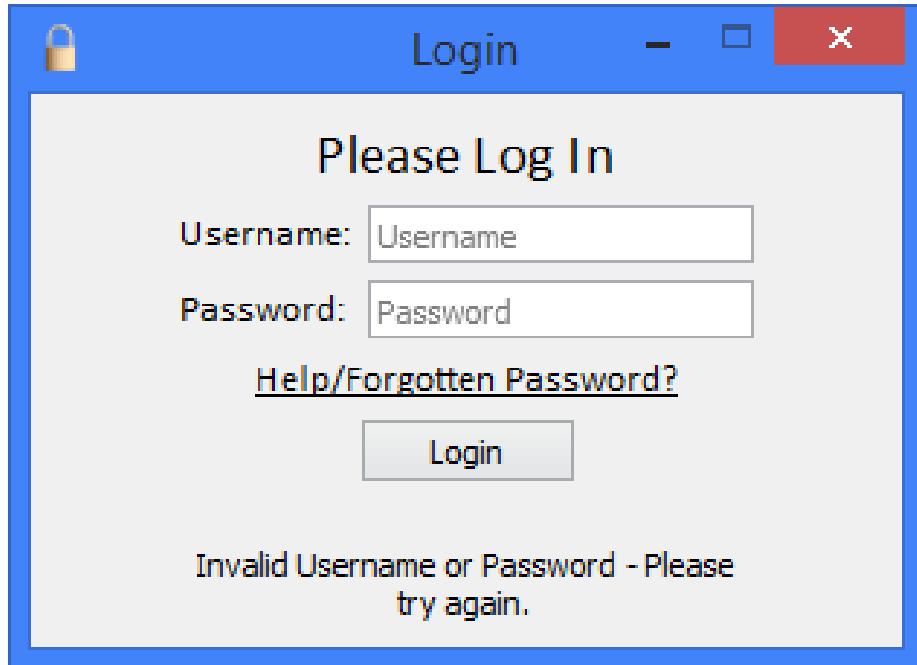


Figure 3.1: Test 1.1

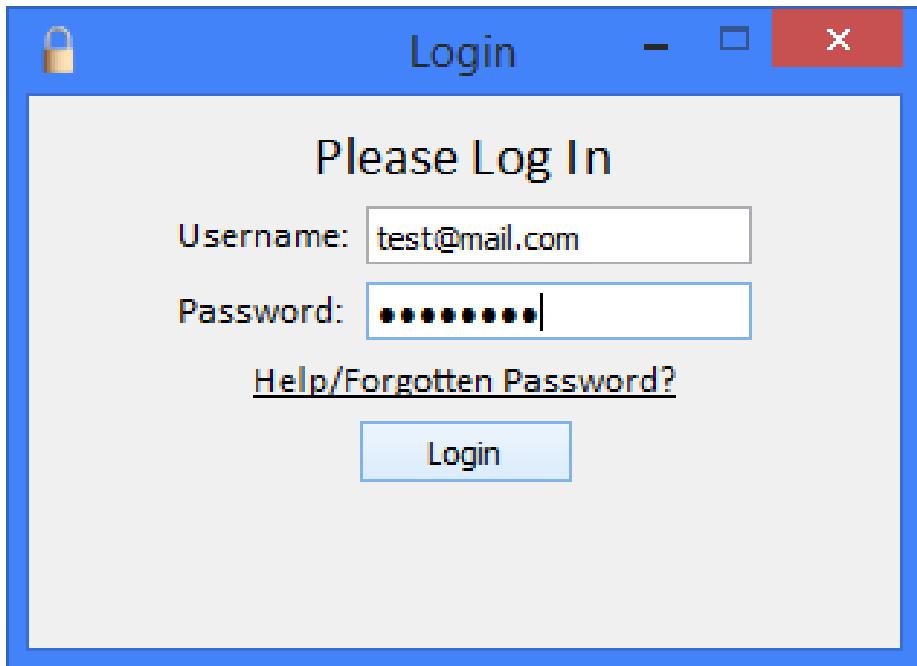


Figure 3.2: Test 1.1

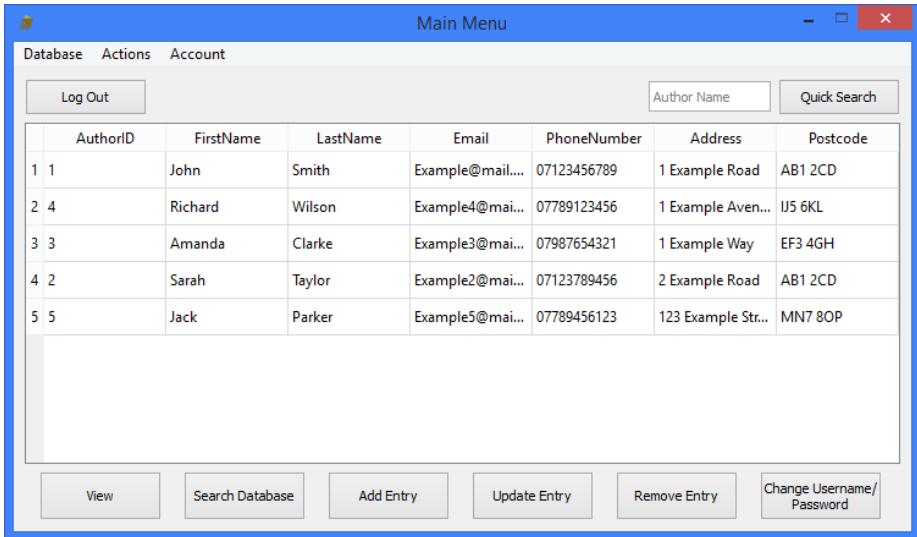


Figure 3.3: Test 1.1

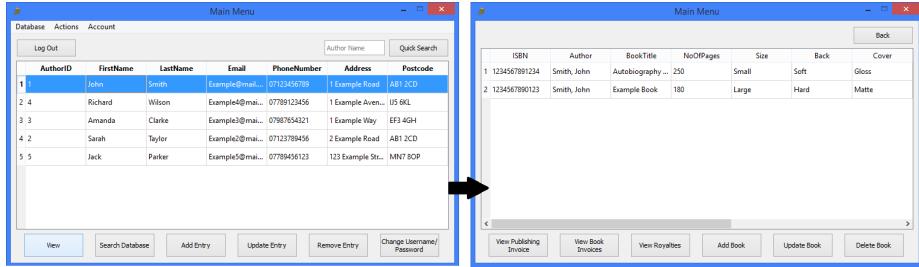


Figure 3.4: Test 1.2

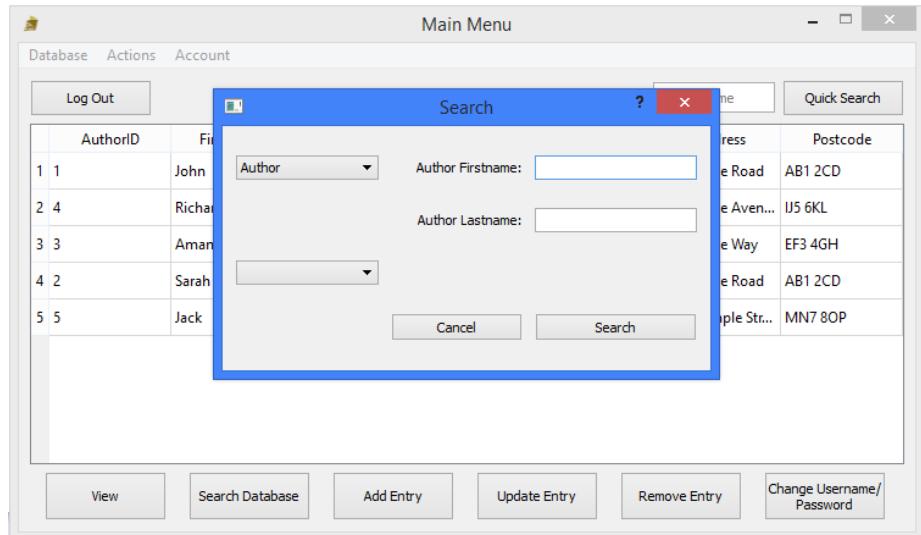


Figure 3.5: Test 1.4

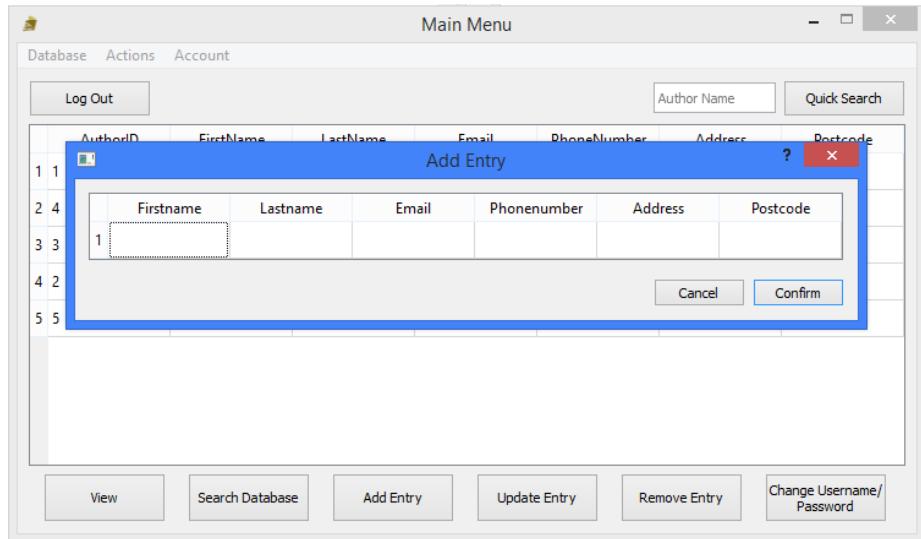


Figure 3.6: Test 1.5

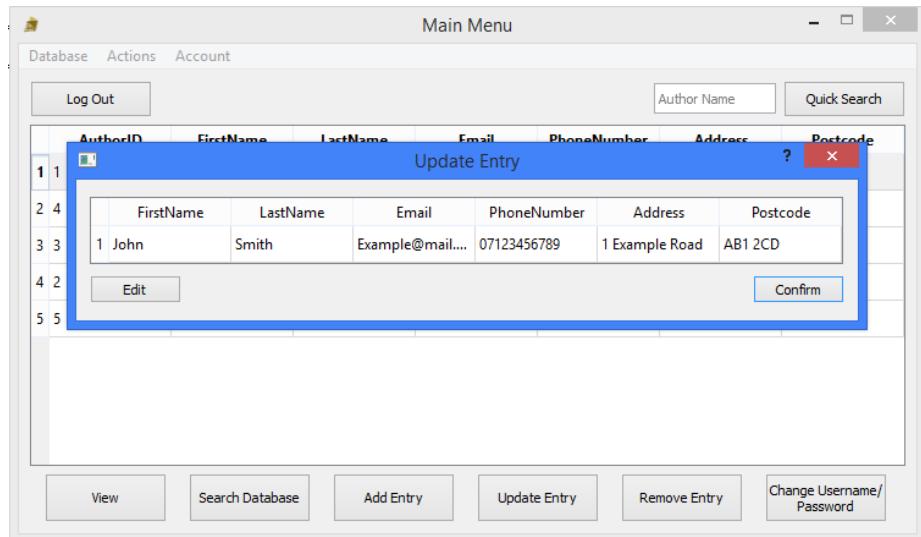


Figure 3.7: Test 1.7

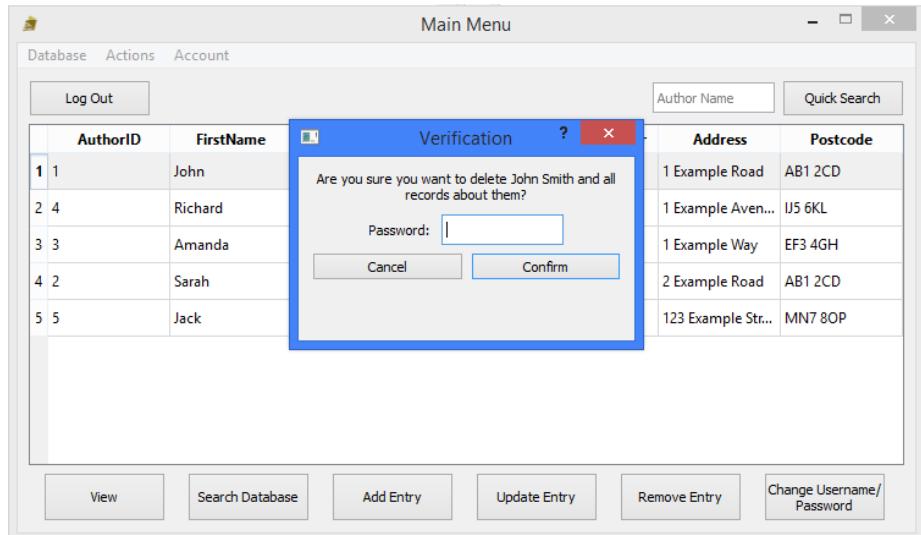


Figure 3.8: Test 1.8

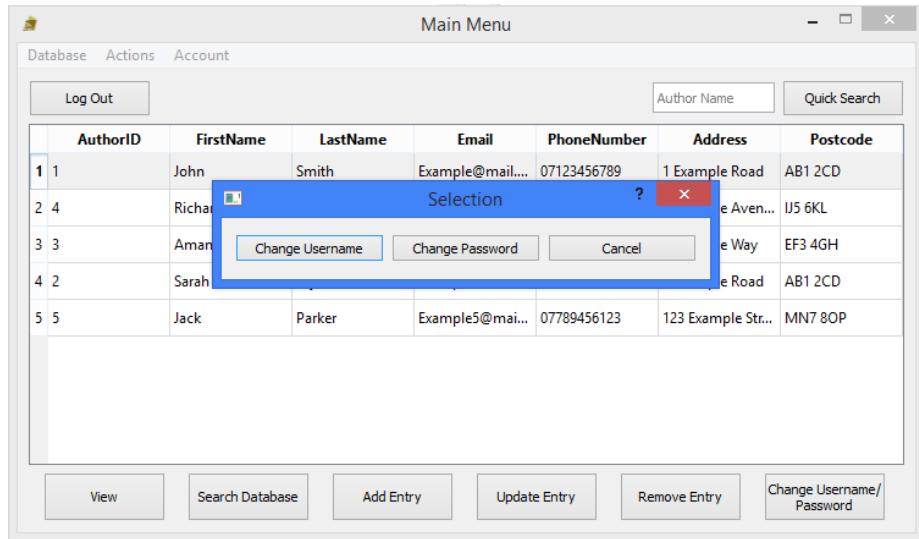


Figure 3.9: Test 1.9

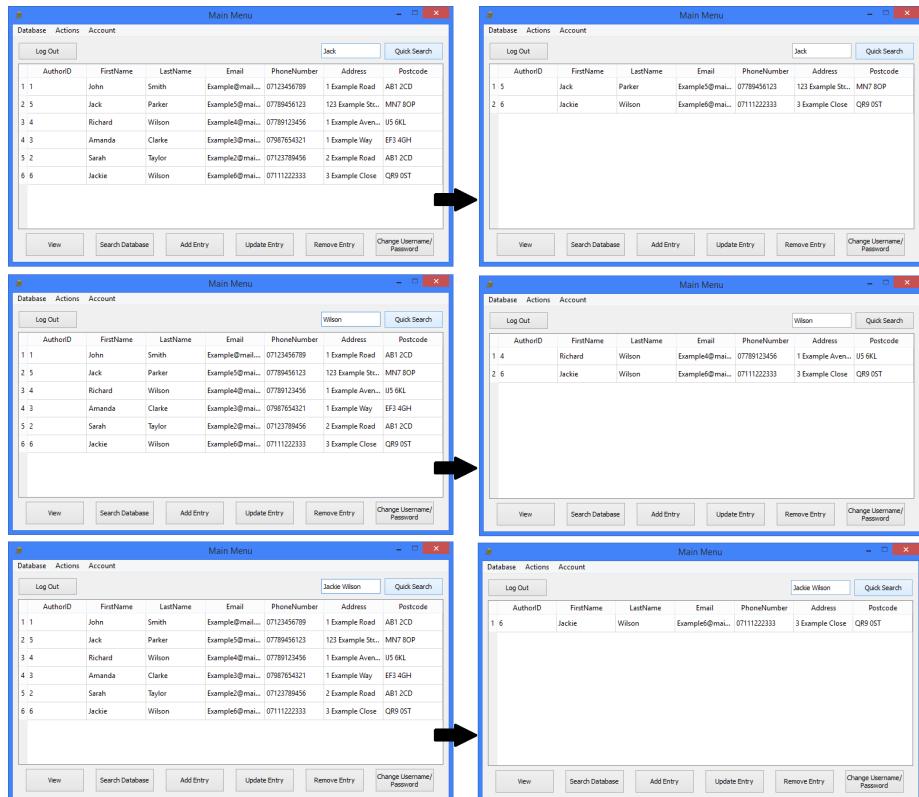


Figure 3.10: Test 1.10

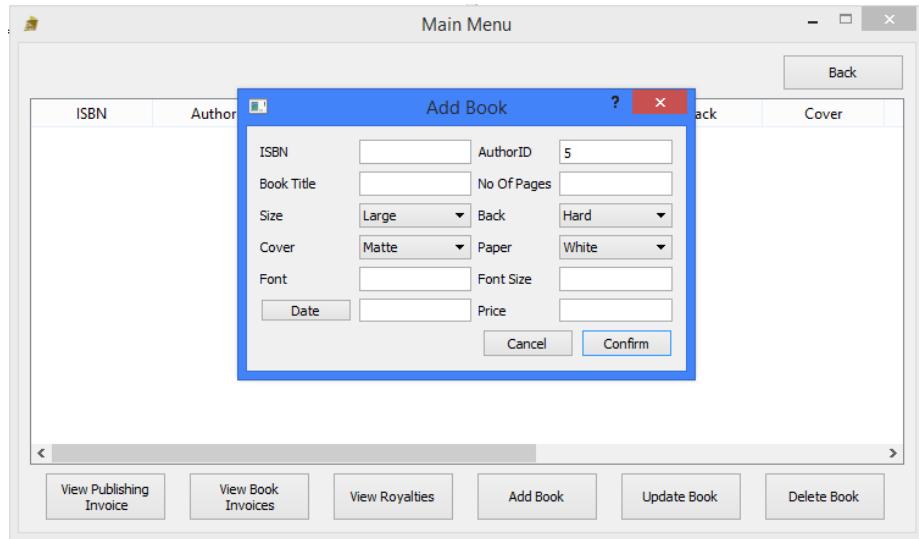


Figure 3.11: Test 1.12

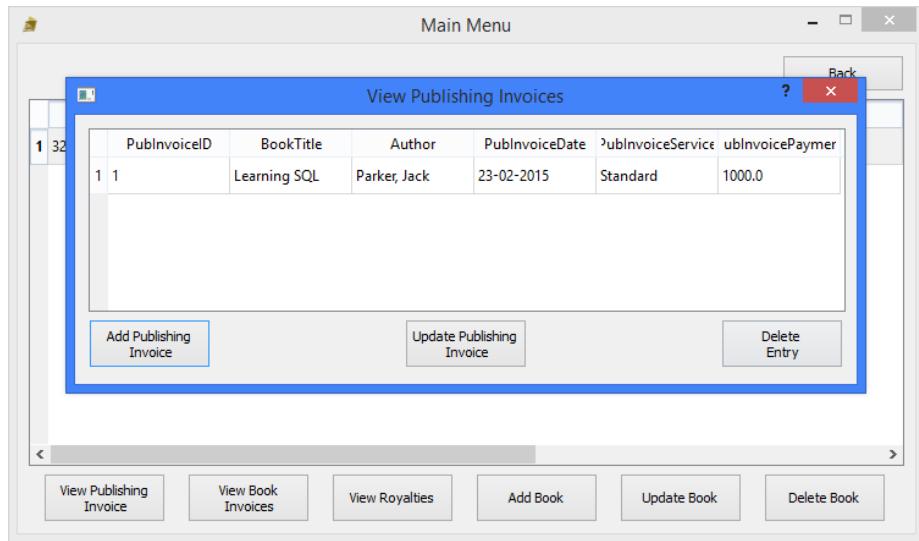


Figure 3.12: Test 1.13

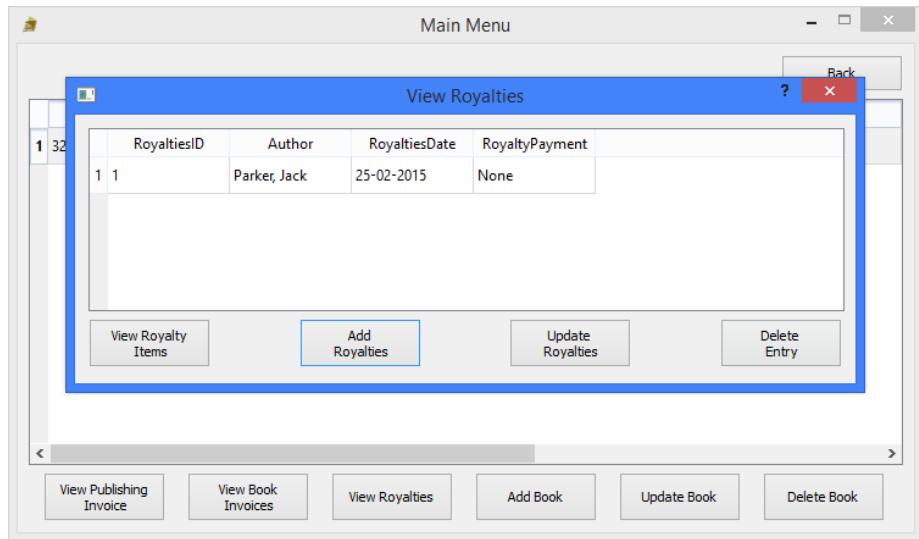


Figure 3.13: Test 1.14

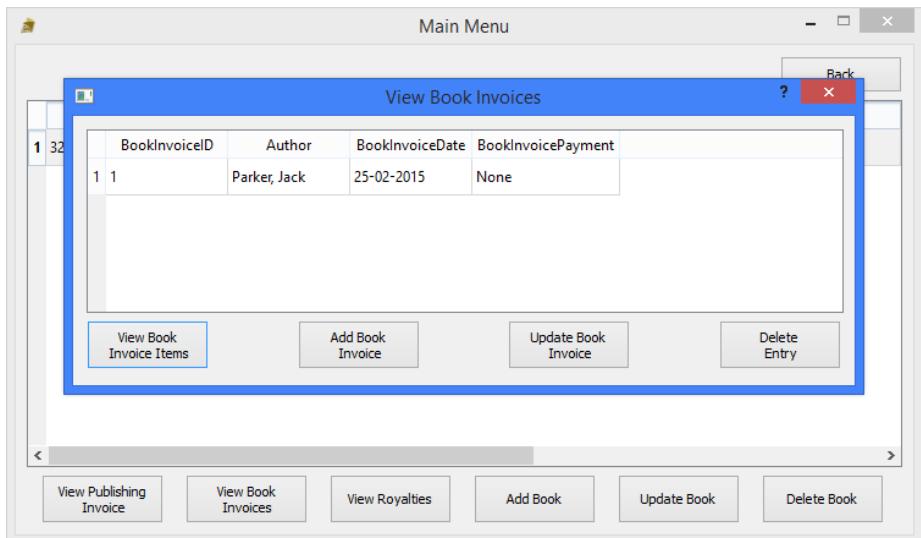


Figure 3.14: Test 1.15

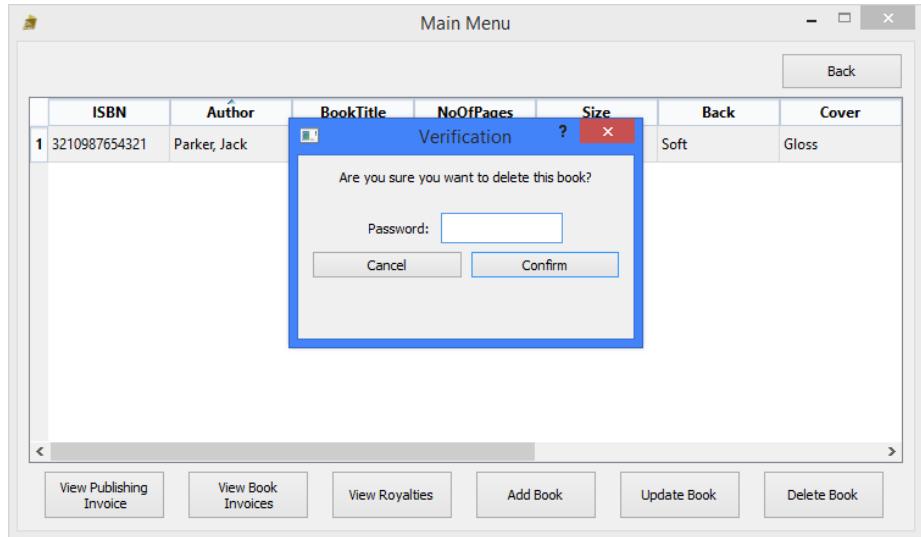


Figure 3.15: Test 1.16

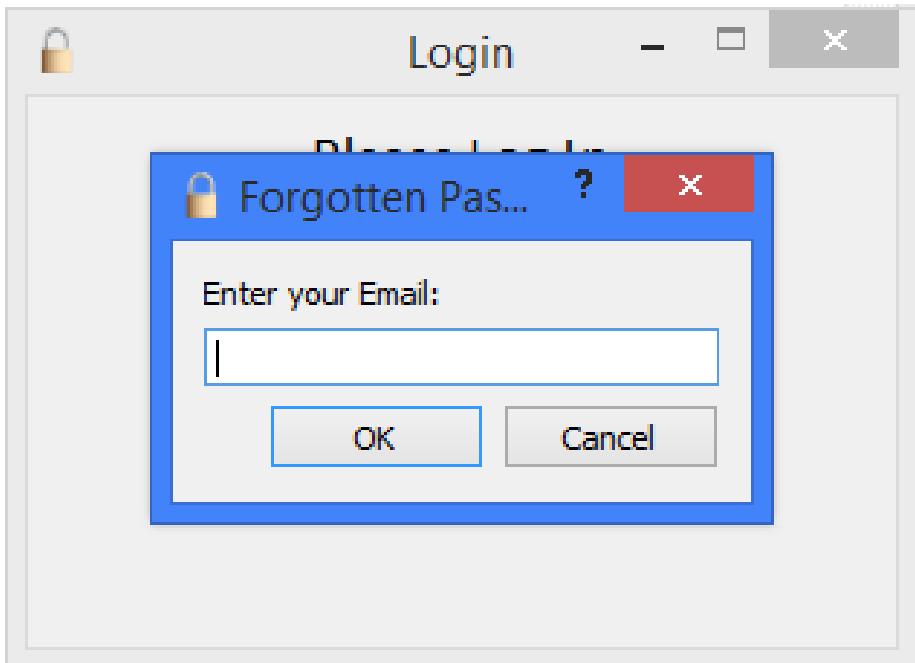


Figure 3.16: Test 1.17

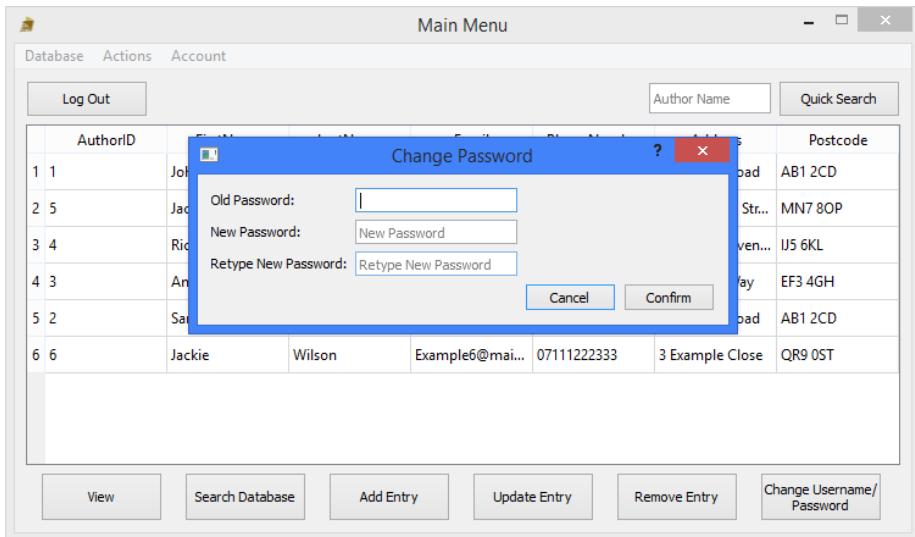


Figure 3.17: Test 1.18

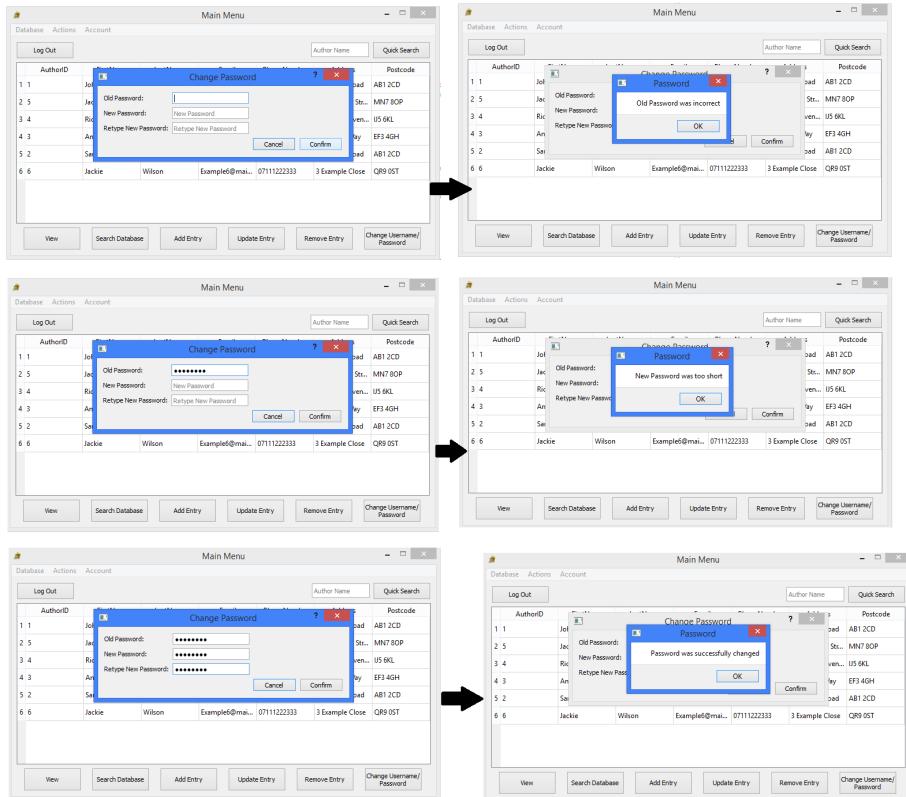


Figure 3.18: Test 1.19

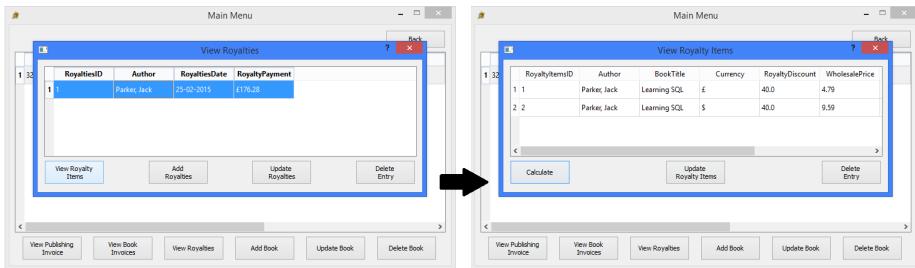


Figure 3.19: Test 1.20

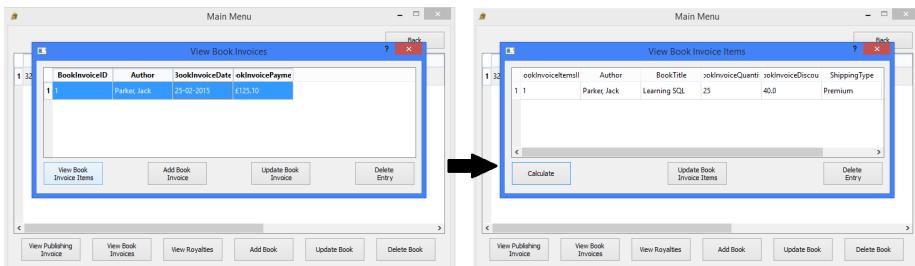


Figure 3.20: Test 1.21

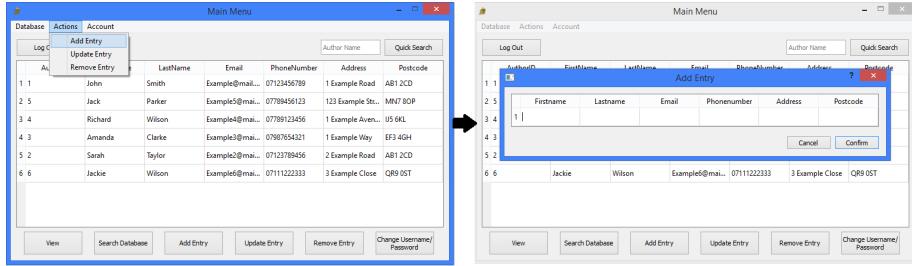


Figure 3.21: Test 1.23

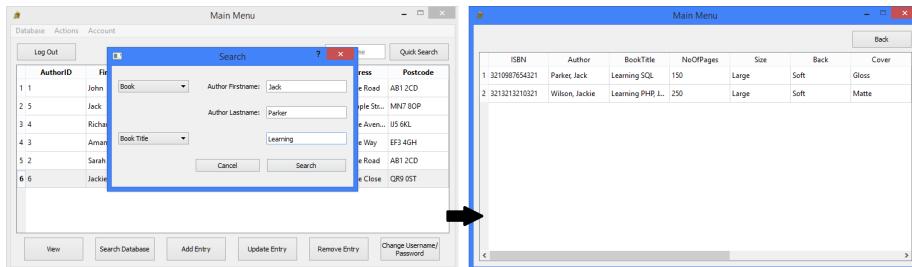


Figure 3.22: Test 1.24

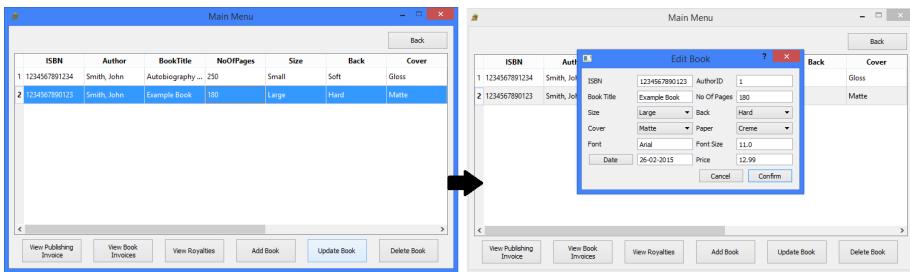


Figure 3.23: Test 1.27

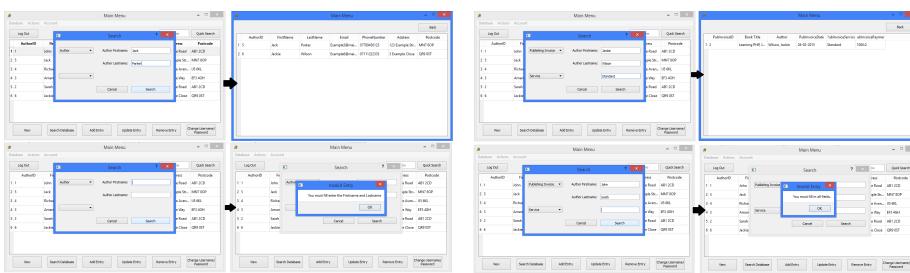


Figure 3.24: Test 2.1

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

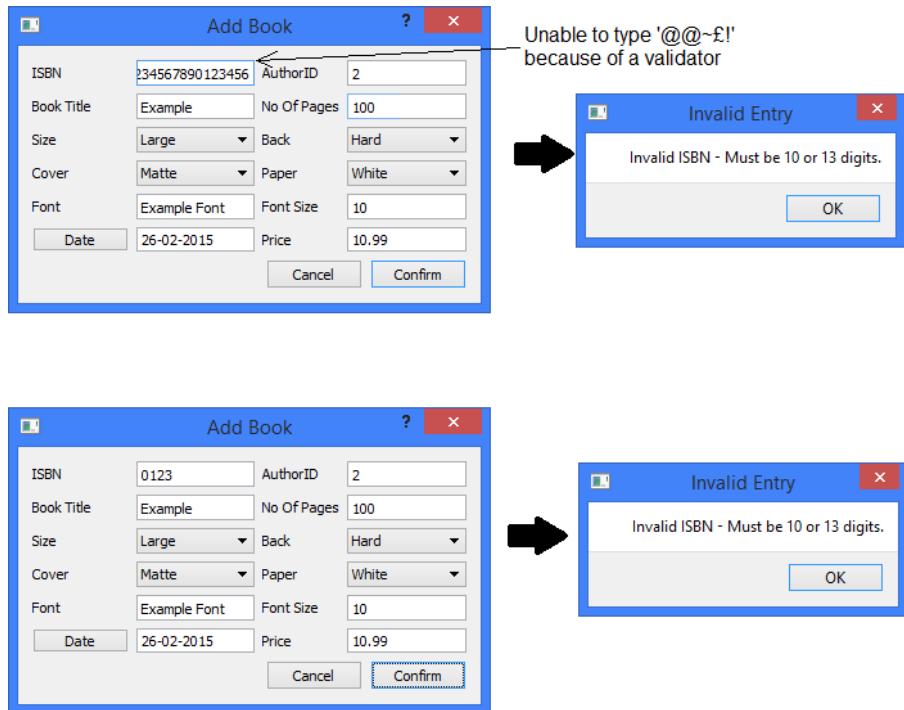


Figure 3.26: Test 2.7

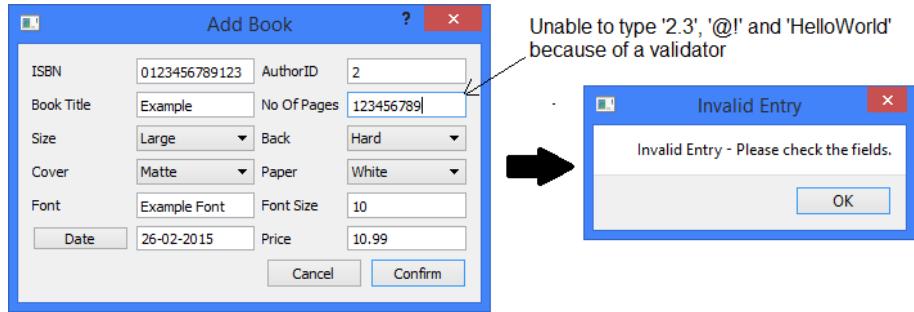


Figure 3.27: Test 2.8

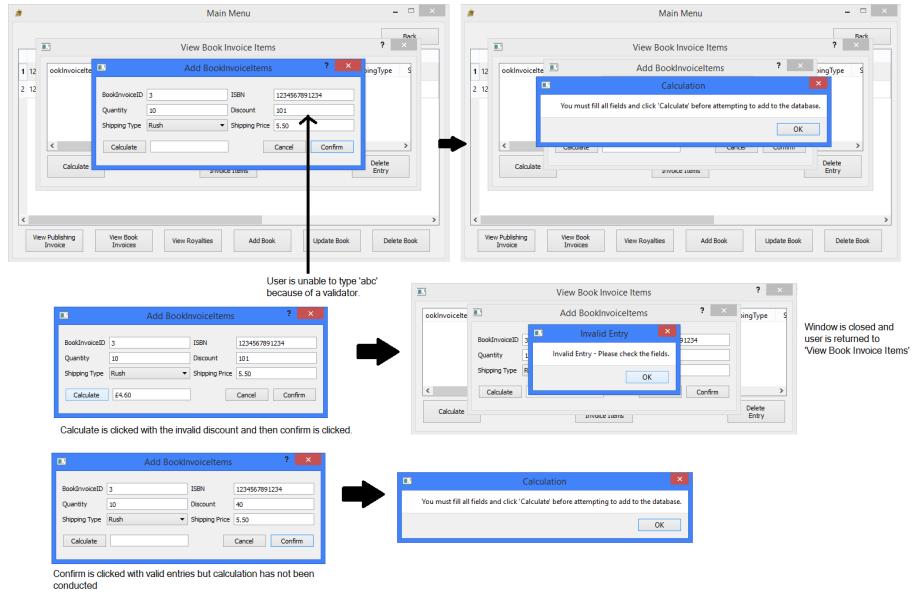


Figure 3.28: Test 2.9

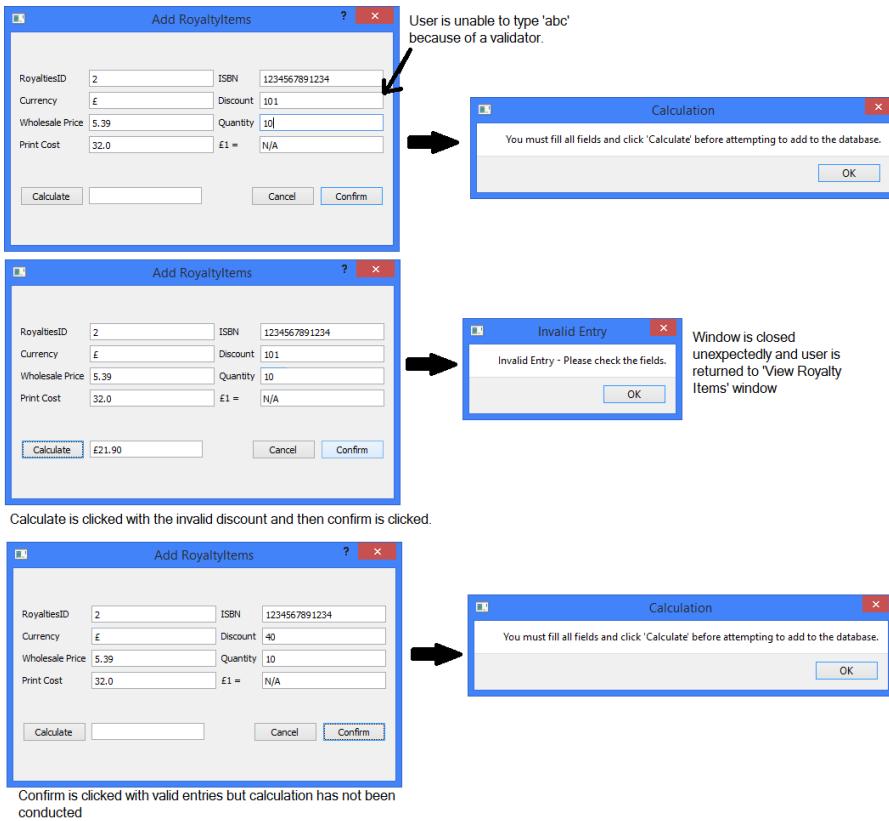


Figure 3.29: Test 2.10

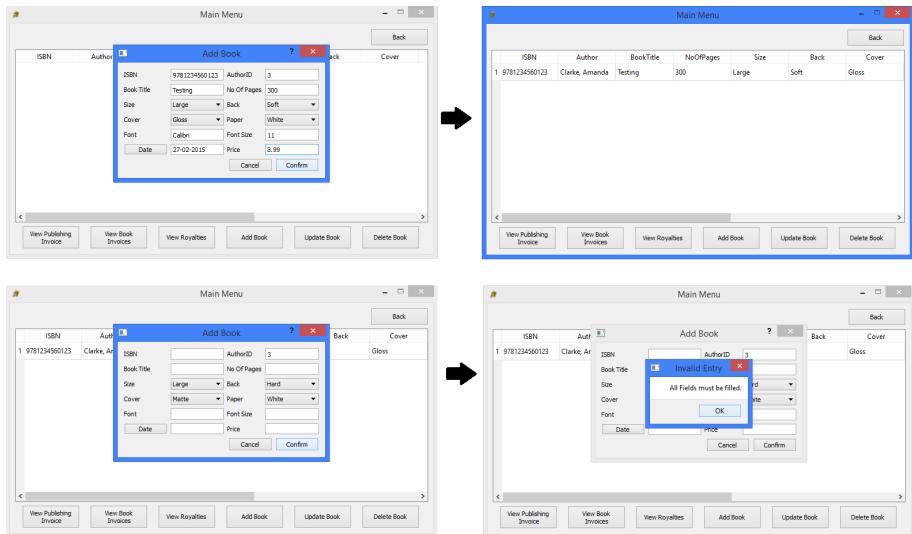


Figure 3.30: Test 3.1

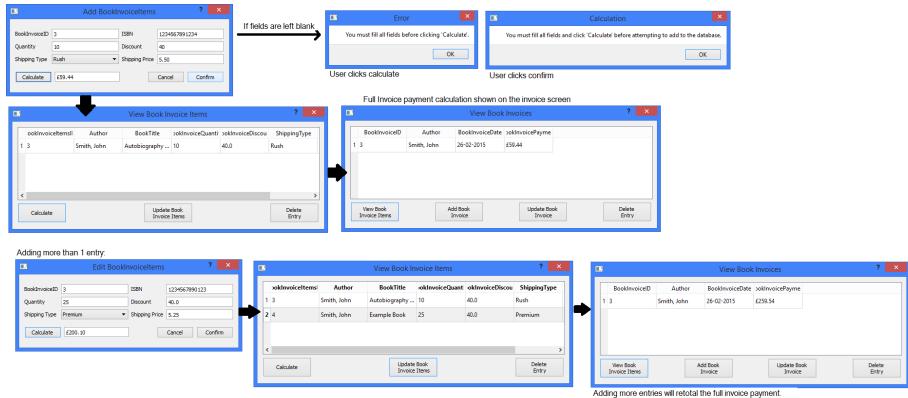


Figure 3.31: Test 3.2

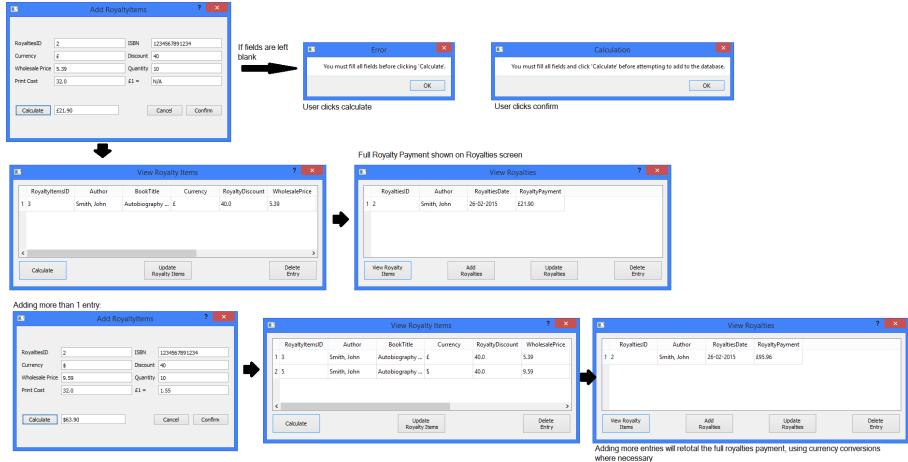


Figure 3.32: Test 3.3

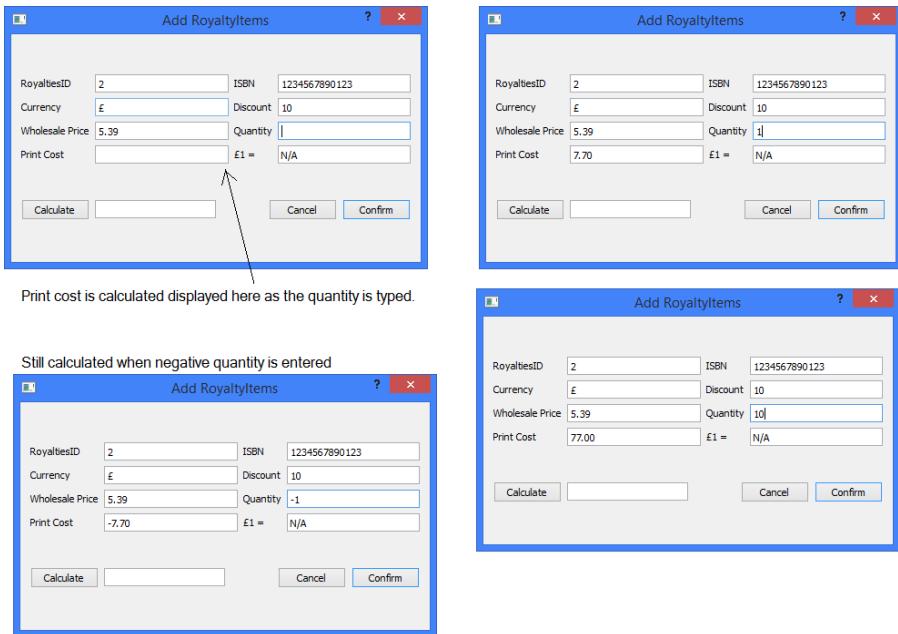


Figure 3.33: Test 3.4

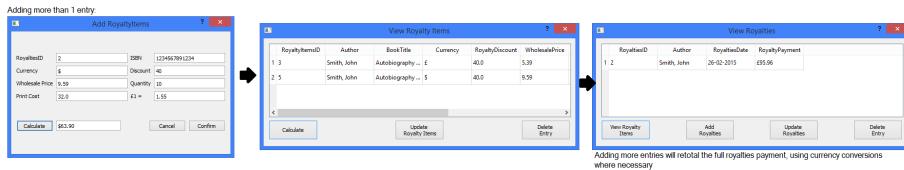
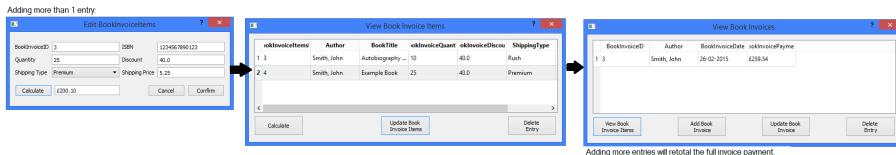


Figure 3.34: Test 3.5



175

Figure 3.35: Test 3.6

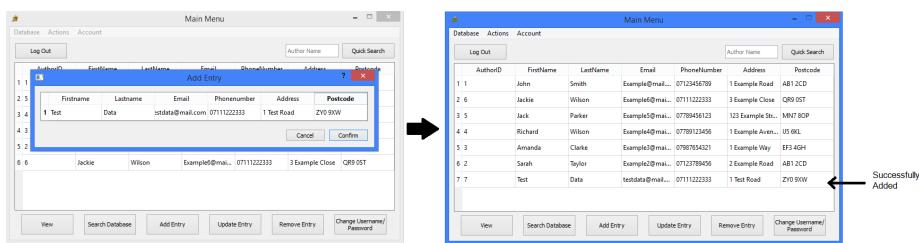


Figure 3.36: Test 4.1

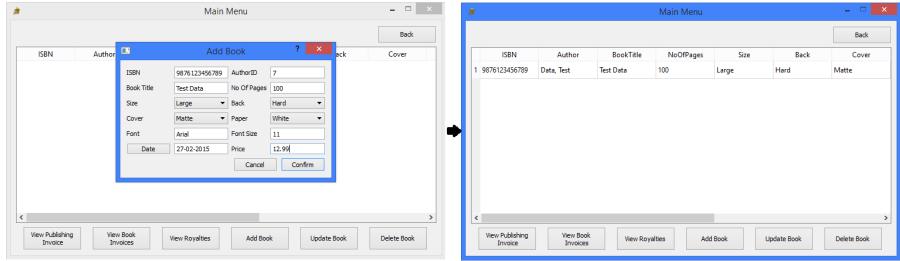


Figure 3.37: Test 4.2

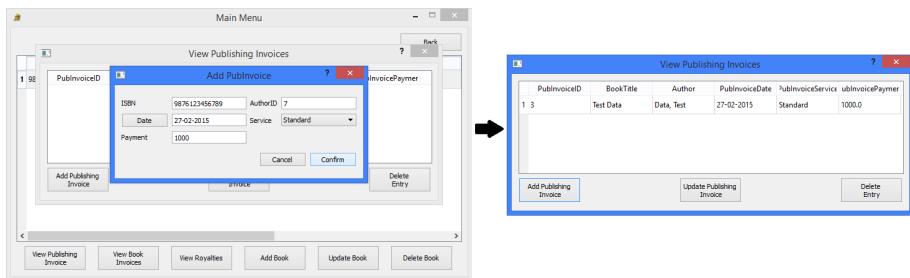


Figure 3.38: Test 4.3

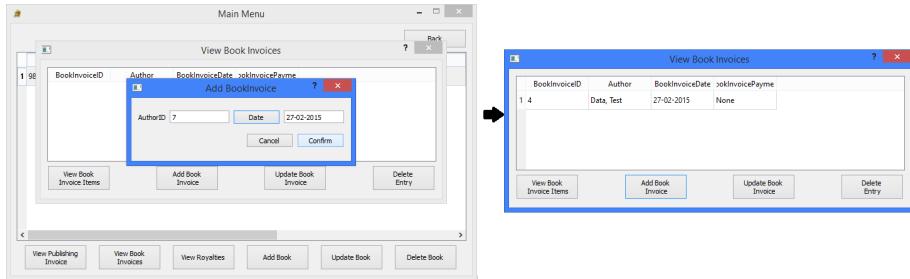


Figure 3.39: Test 4.4

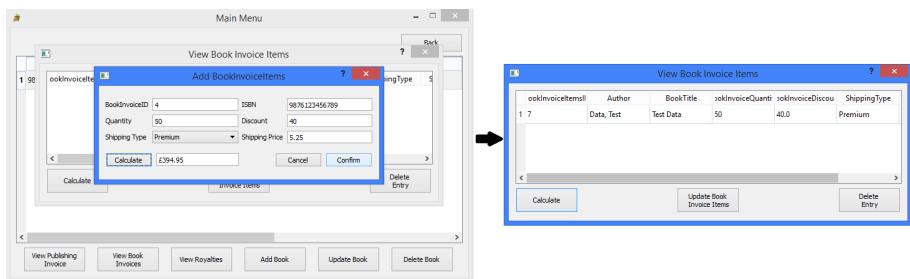


Figure 3.40: Test 4.5

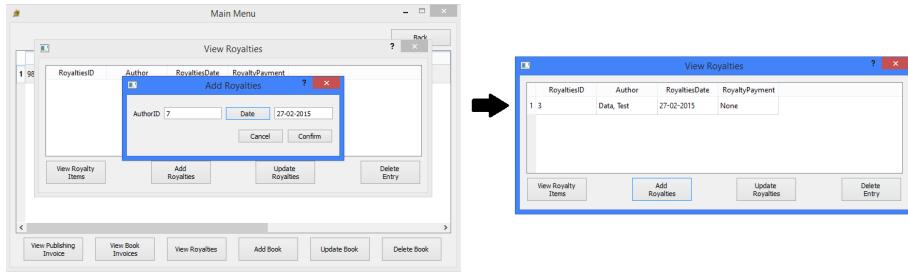


Figure 3.41: Test 4.6

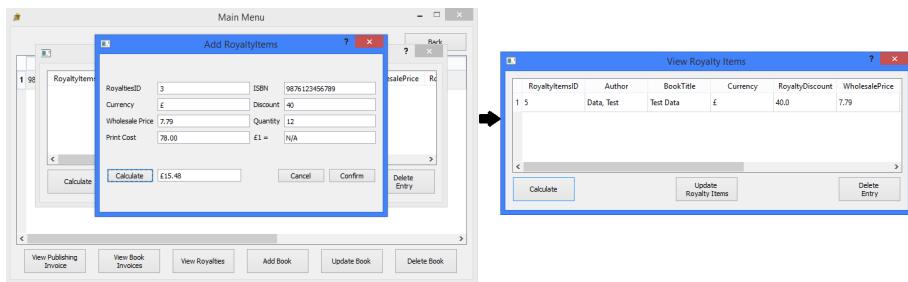


Figure 3.42: Test 4.7

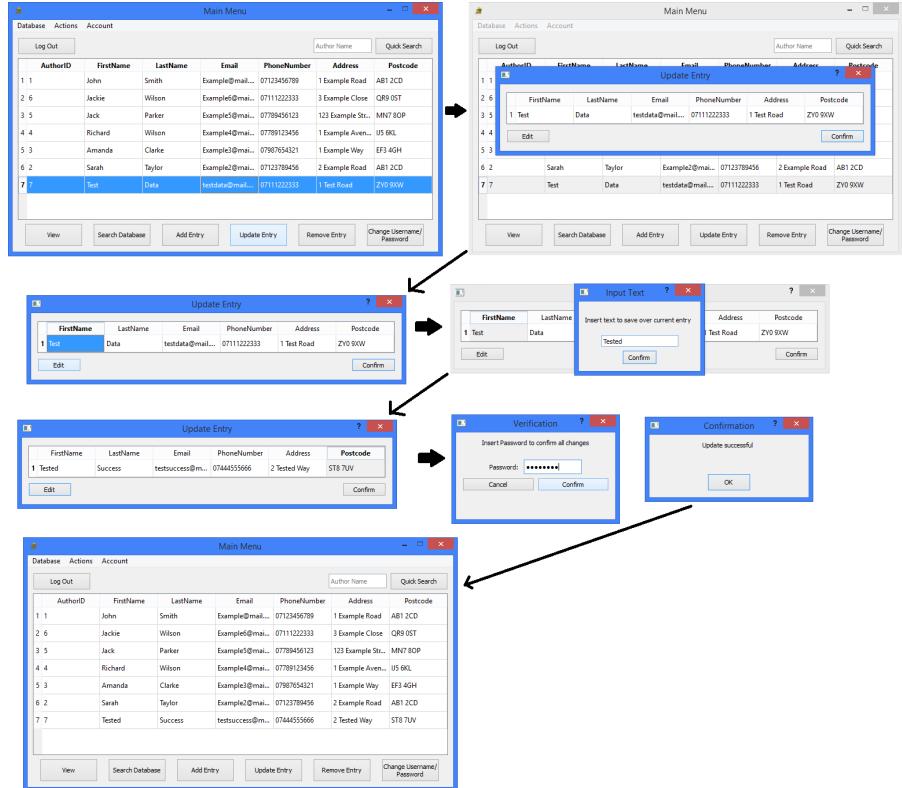


Figure 3.43: Test 4.8

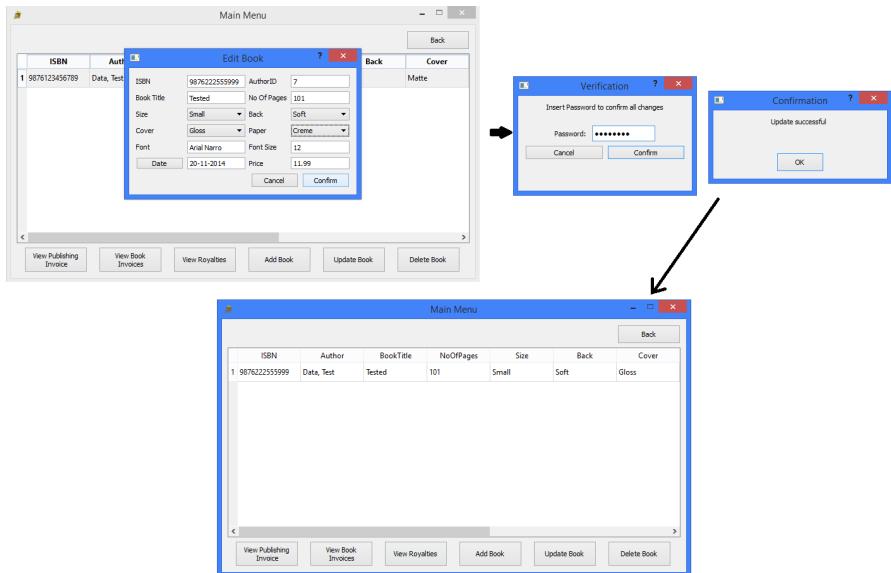


Figure 3.44: Test 4.9

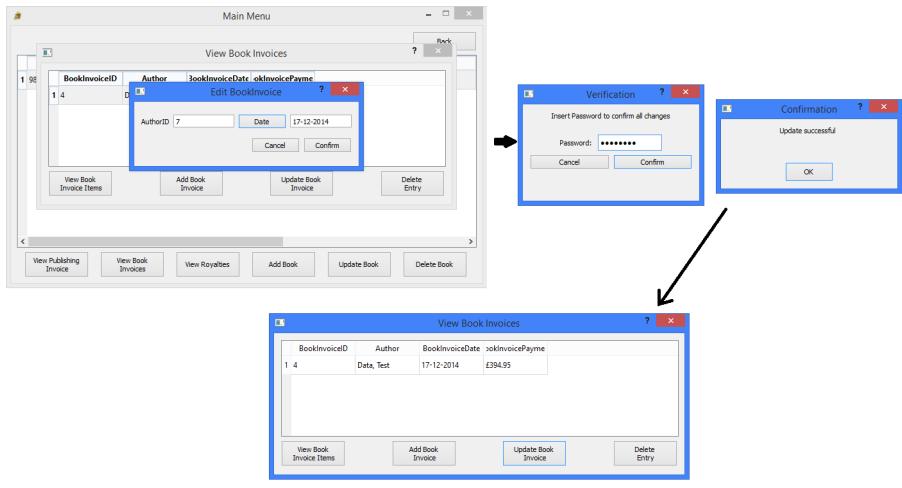


Figure 3.45: Test 4.10

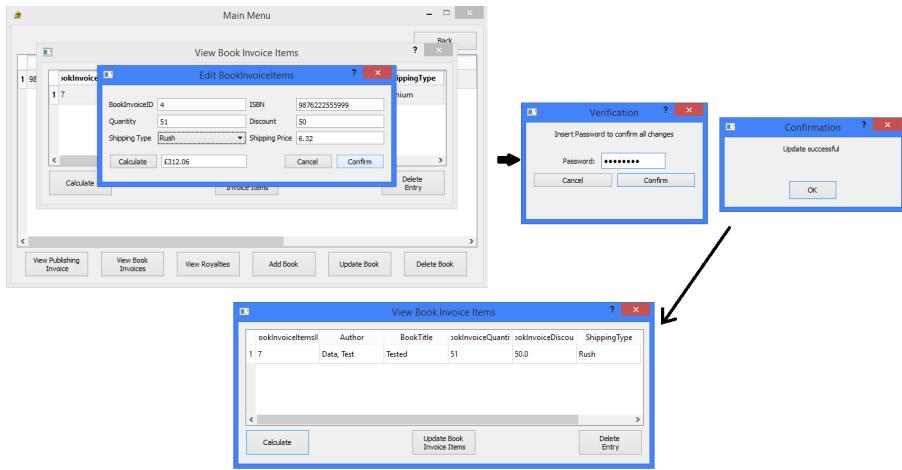


Figure 3.46: Test 4.11

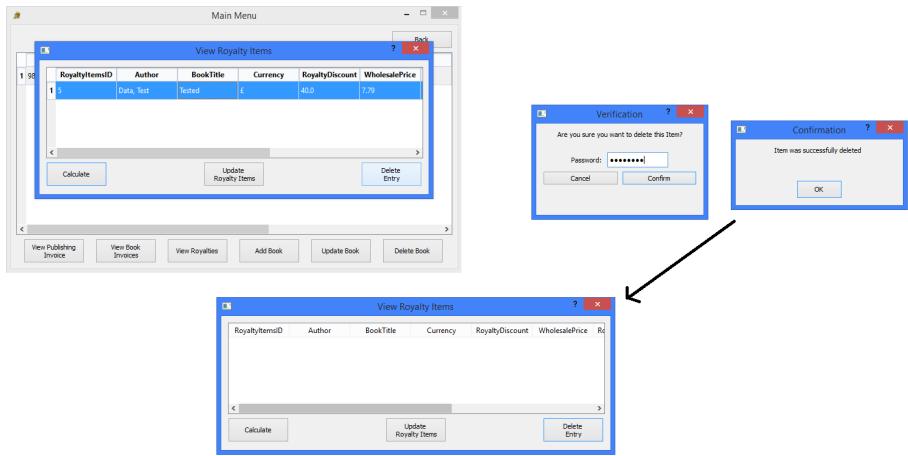


Figure 3.47: Test 4.12

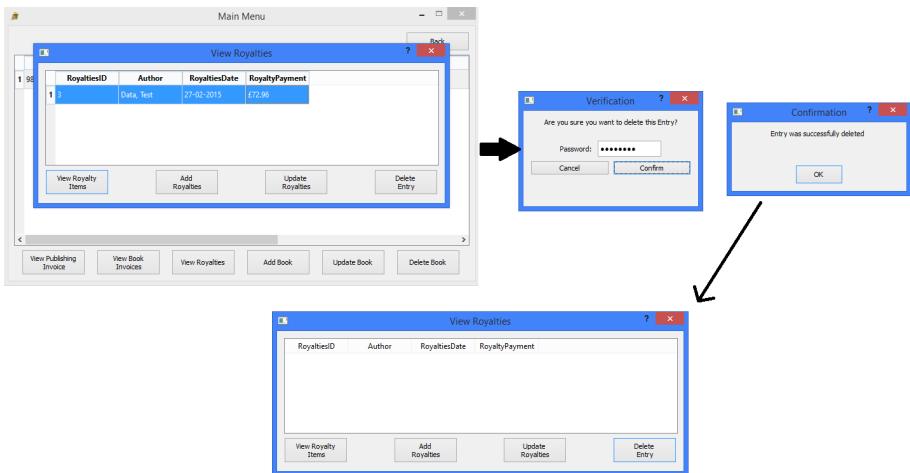


Figure 3.48: Test 4.13

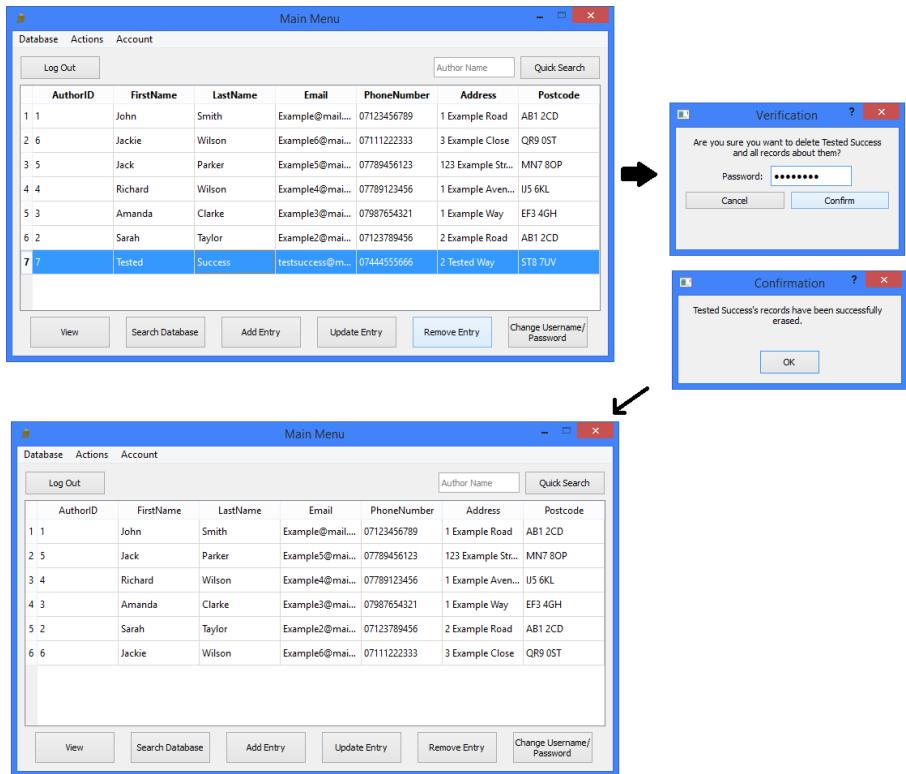


Figure 3.49: Test 4.14

3.4 Evaluation

3.4.1 Approach to Testing

I have chosen to use a range of testing strategies. For instance, series 1 of 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:

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

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

3.4.3 Strengths of Testing

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

3.4.4 Weaknesses of Testing

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

3.4.5 Reliability of Application

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

3.4.6 Robustness of Application

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

Chapter 4

System Maintenance

4.1 Environment

4.1.1 Software

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

- Python 3.4
- IDLE
- PyQt4
- SQLite 3

4.1.2 Usage Explanation

Python 3.4:

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

IDLE:

- Software is included with python
- Designed for python scripting

PyQt4:

- Designed especially for implementing graphical interface

SQLite 3:

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

4.1.3 Features Used

Python 3.4:

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

IDLE:

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

PyQt4:

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

SQLite 3:

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

Imran Rahman

Candidate No. 30928

Centre No. 22151

4.2 System Overview

4.2.1 System Component

4.3 Code Structure

4.3.1 Particular Code Section

4.4 Variable Listing

4.5 System Evidence

4.5.1 User Interface

4.5.2 ER Diagram

4.5.3 Database Table Views

4.5.4 Database SQL

4.5.5 SQL Queries

4.6 Testing

4.6.1 Summary of Results

4.6.2 Known Issues

4.7 Code Explanations

4.7.1 Difficult Sections

4.7.2 Self-created Algorithms

4.8 Settings

4.9 Acknowledgements

4.10 Code Listing

4.10.1 Module 1

```
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         self.details = ("Username", "Password")
17         if self.details == ("Username", "Password"):
18             self.customer_table()
19             self.book_table()
20             self.pub_invoice_table()
21             self.book_invoice_table()
22             self.book_invoice_items_table()
23             self.royalties_table()
24             self.royalty_items_table()
25             self.MainProgram = MainWindow("Username") #runs main window if
                                                Username/Password haven't been changed before
```

```
26         self.MainProgram.show()
27     else:
28         super().__init__() #runs login screen if they have been changed
29         self.initLoginScreen()
30
31     def initDetails(self):
32         with sqlite3.connect("dbLogin.db") as db:
33             cursor = db.cursor()
34
35             self.sql = "select name from sqlite_master WHERE type='table' and
36                         name='LoginDetails'"
37             cursor.execute(self.sql) #checking whether the login table exists
38             try:
39                 self.Exists = list(cursor.fetchone())[0]
40             except:
41                 self.Exists = False
42
43             self.sql = "create table if not exists LoginDetails (Username text,
44                         Password text)"
45             cursor.execute(self.sql) #creates login table if it doesn't exist
46
47             if self.Exists == False:
48                 self.details = "Username", "Password"
49                 self.sql = "insert into LoginDetails (Username, Password) values (?, ?)"
50                 cursor.execute(self.sql, self.details) #adds default Username and
51                         Password
52             else:
```

```
50             self.details = True
51
52     def initLoginScreen(self):
53         self.setWindowTitle("Login")
54         self.setWindowIcon(QIcon("LoginIcon.png"))
55         self.setFixedSize(300, 190)
56         self.lblLogin = QLabel("Please Log In", self)
57         self.lblLogin.setFont(QFont("Calibri", 14))
58         self.lblLogin.setAlignment(Qt.AlignHCenter)
59         self.btnLogin = QPushButton("Login", self)
60         self.btnLogin.setFixedSize(self.btnLogin.sizeHint())
61         self.lblUsername = QLabel("Username:", self)
62         self.lblUsername.setFont(QFont("Calibri", 10))
63         self.lblPassword = QLabel("Password:", self)
64         self.lblPassword.setFont(QFont("Calibri", 10))
65         self.leUsername = QLineEdit(self)
66         self.leUsername.setPlaceholderText("Username")
67         self.lePassword = QLineEdit(self)
68         self.lePassword.setEchoMode(self.lePassword.Password)
69         self.lePassword.setPlaceholderText("Password")
70         self.lblForgot = QLabel("Help/Forgotten Password?", self)
71         self.Underline = QFont("Calibri", 10)
72         self.Underline.setUnderline(True)
73         self.lblForgot.setFont(self.Underline)
74         self.lblForgot.setAlignment(Qt.AlignHCenter)
75         self.lblForgot.mousePressEvent = self.getEmail
76         self.lblVertical = QVBoxLayout()
77         self.leVertical = QVBoxLayout()
```

```
78     self.horizontalLogin = QHBoxLayout()
79     self.lblVertical.addWidget(self.lblUsername)
80     self.leVertical.addWidget(self.leUsername)
81     self.lblVertical.addWidget(self.lblPassword)
82     self.leVertical.addWidget(self.lePassword)
83     self.horizontalLogin.addStretch(1)
84     self.horizontalLogin.addWidget(self.btnLogin)
85     self.horizontalLogin.addStretch(1)
86     self.vertical = QVBoxLayout()
87     self.vertical.addWidget(self.lblLogin)
88     self.horizontalEntry = QHBoxLayout()
89     self.horizontalEntry.setLayout(self.lblVertical)
90     self.horizontalEntry.setLayout(self.leVertical)
91     self.vertical.addLayout(self.horizontalEntry)
92     self.vertical.addWidget(self.lblForgot)
93     self.vertical.addLayout(self.horizontalLogin)
94     self.vertical.addStretch(1)
95     self.horizontal = QHBoxLayout()
96     self.horizontal.addStretch(1)
97     self.horizontal.addLayout(self.vertical)
98     self.horizontal.addStretch(1)
99     self.CentralWidget = QWidget()
100    self.CentralWidget.setLayout(self.horizontal)
101    self.setCentralWidget(self.CentralWidget)
102    self.btnLogin.clicked.connect(self.Login)
103    self.lblInvalid = None
104
105 def Login(self):
```

```
106     with sqlite3.connect("dbLogin.db") as db:
107         cursor = db.cursor()
108         cursor.execute("select Username from LoginDetails")
109         self.Username = list(cursor.fetchall()) #fetches original username and
110             password
111         cursor.execute("select Password from LoginDetails")
112         self.Password = list(cursor.fetchall())
113         self.Valid = False
114
115         for count in range(0, len(self.Username)):
116             if self.leUsername.text().lower() ==
117                 list(self.Username[count])[0].lower():
118                 if self.lePassword.text() == list(self.Password[count])[0]:
119                     self.Valid = True
120                     self.customer_table() #checking all tables to see if they're
121                         existent
122                     self.book_table()
123                     self.pub_invoice_table()
124                     self.book_invoice_table()
125                     self.book_invoice_items_table()
126                     self.royalties_table()
127                     self.royalty_items_table()
128                     self.hide() #Username and password match, so the user is
129                         logged in
130                     self.MainProgram = MainWindow(list(self.Username[count])[0])
131                     self.MainProgram.show()
132                     break
133
134             else:
```

```
130                     self.Valid = False
131
132             if self.Valid == False:
133                 if self.lblInvalid == None: #Username and password do not match, user
134                     is rejected
135                     self.lblInvalid = QLabel("Invalid Username or Password - Please
136                         try again.", self)
137                     self.lblInvalid.setWordWrap(True)
138                     self.lblInvalid.setAlignment(Qt.AlignHCenter)
139                     self.horizontalInvalid = QBoxLayout()
140                     self.horizontalInvalid.addStretch(1)
141                     self.horizontalInvalid.addWidget(self.lblInvalid)
142                     self.horizontalInvalid.addStretch(1)
143                     self.vertical.addLayout(self.horizontalInvalid)
144
145             def getEmail(self, QMouseEvent):
146                 with sqlite3.connect("dbLogin.db") as db:
147                     cursor = db.cursor()
148                     cursor.execute("select Username from LoginDetails")
149                     self.Username = list(cursor.fetchone())[0]
150                     cursor.execute("select Password from LoginDetails")
151                     self.Password = list(cursor.fetchone())[0]
152
153             if self.Username == "Username" and self.Password == "Password":
154                 self.Msg = QMessageBox()
155                 self.Msg.setWindowTitle("First Time")
```

```
156         self.Msg.setText("This is your first time using this application.\n Your\n        Username is 'Username' and your Password is 'Password'.\n Please change\n        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
160             your Email:')
161         self.Email = self.Email.lower() #email is received from the user
162         if self.Email == self.Username:
163             with sqlite3.connect("dbLogin.db") as db:
164                 cursor = db.cursor()
165                 cursor.execute("select Password from LoginDetails")
166                 self.CurrentPassword = list(cursor.fetchone())[0] #gets password
167                     from database
168                 self.sender = "pp.loginhelp@gmail.com"
169                 self.recipient = [str(self.Email)]
170                 self.server = smtplib.SMTP('smtp.gmail.com', 587)
171                 self.server.ehlo()
172                 self.server.starttls()
173                 self.server.ehlo()
174                 self.server.login("pp.loginhelp@gmail.com", "DB1061NH31P")
175
176                 self.Subject = "Forgotten Login Details"
177                 self.message = "From: {}\\r\\nTo: {}\\r\\nSubject:
178                     {}\\r\\n\\r\\n".format(self.sender, ", ".join(self.recipient),
179                     self.Subject)
180                 self.message += "Your Password is: {}\\nPlease change this upon login
181                     for security reasons.".format(self.CurrentPassword)
```

```
177         self.server.sendmail(self.sender, self.recipient, self.message)
178         self.server.close()
179
180         self.Msg = QMessageBox()
181         self.Msg.setWindowTitle("Email Sent") #email is sent if email is
182             existent in the database, with password details
183         self.Msg.setText("You have been sent an email with the corresponding
184             password details")
185         self.Msg.exec_()
186
187     elif self.Email != self.Username and ok == True:
188         self.Msg = QMessageBox()
189         self.Msg.setWindowTitle("No Match found") #email is not found, so
190             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
```

```
201     self.FindTable = "select name from sqlite_master WHERE type='table' and  
202         name='{}'".format(self.TableName)  
203     cursor.execute(self.FindTable) #checks if the tables exist  
204     try:  
205         self.Exists = list(cursor.fetchone())  
206     except:  
207         self.Exists = False  
208     if self.Exists == False: #creates tables for the database if they don't  
209         already exist  
210         cursor.execute(self.sql)  
211         db.commit()  
212  
213     def customer_table(self):  
214         self.sql = """create table Customer  
215             (AuthorID integer,  
216              FirstName text,  
217              LastName text,  
218              Email text,  
219              PhoneNumber text,  
220              Address text,  
221              Postcode text,  
222              primary key(AuthorID))"""  
223     self.TableName = "Customer"  
224     self.create_table()  
225  
226     def book_table(self):  
227         self.sql = """create table Book
```

```
227     (ISBN text,
228     AuthorID integer,
229     BookTitle text,
230     NoOfPages integer,
231     Size text,
232     Back text,
233     Cover text,
234     Paper text,
235     Font text,
236     FontSize real,
237     DatePublished date,
238     Price real,
239     primary key(ISBN),
240     foreign key(AuthorID) references Customer(AuthorID))"""
202    self.TableName = "Book"
241    self.create_table()
242
243
244    def pub_invoice_table(self):
245        self.sql = """create table PubInvoice
246                    (PubInvoiceID integer,
247                     ISBN text,
248                     AuthorID integer,
249                     PubInvoiceDate date,
250                     PubInvoiceService text,
251                     PubInvoicePayment real,
252                     primary key(PubInvoiceID),
253                     foreign key(AuthorID) references Customer(AuthorID),
254                     foreign key(ISBN) references Book(ISBN))"""

```

```
255     self.TableName = "PubInvoice"
256     self.create_table()
257
258     def book_invoice_table(self):
259         self.sql = """create table BookInvoice
260                     (BookInvoiceID integer,
261                      AuthorID integer,
262                      BookInvoiceDate date,
263                      BookInvoicePayment real,
264                      primary key(BookInvoiceID),
265                      foreign key(AuthorID) references Customer(AuthorID))"""
266         self.TableName = "BookInvoice"
267         self.create_table()
268
269     def book_invoice_items_table(self):
270         self.sql = """create table BookInvoiceItems
271                     (BookInvoiceItemsID integer,
272                      BookInvoiceID integer,
273                      ISBN text,
274                      BookInvoiceQuantity integer,
275                      BookInvoiceDiscount real,
276                      ShippingType text,
277                      ShippingPrice real,
278                      primary key(BookInvoiceItemsID),
279                      foreign key(BookInvoiceID) references BookInvoice(BookInvoiceID),
280                      foreign key(ISBN) references Book(ISBN))"""
281         self.TableName = "BookInvoiceItems"
282         self.create_table()
```

```
283
284     def royalties_table(self):
285         self.sql = """create table Royalties
286             (RoyaltiesID integer,
287              AuthorID integer,
288              RoyaltiesDate date,
289              RoyaltyPayment real,
290              primary key(RoyaltiesID),
291              foreign key(AuthorID) references Customer(AuthorID))"""
292         self.TableName = "Royalties"
293         self.create_table()
294
295     def royalty_items_table(self):
296         self.sql = """create table RoyaltyItems
297             (RoyaltyItemsID integer,
298              RoyaltiesID integer,
299              ISBN text,
300              Currency text,
301              RoyaltyDiscount real,
302              WholesalePrice real,
303              RoyaltyQuantity integer,
304              NetSales real,
305              PrintCost real,
306              ExcRateFromGBP real,
307              primary key(RoyaltyItemsID),
308              foreign key(RoyaltiesID) references Royalties(RoyaltiesID),
309              foreign key(ISBN) references Book(ISBN))"""
310         self.TableName = "RoyaltyItems"
```

```
311         self.create_table()
312
313     def initSplashScreen(self):
314         self.pixmap = QPixmap("PerfectPublishersLtd.png")
315         self.Splashscreen = QSplashScreen(self.pixmap, Qt.WindowStaysOnTopHint)
316         self.Splashscreen.setMask(self.pixmap.mask())
317         self.Splashscreen.show()
318         time.sleep(2)
319         self.Splashscreen.finish(self.Splashscreen)
320
321
322     def main():
323         app = QApplication(sys.argv)
324         launcher = dbLogin()
325         try:
326             launcher.raise_()
327             launcher.show()
328         except RuntimeError:
329             pass
330         app.exec_()
331
332
333
334
335 if __name__ == "__main__":
336     main()
```

4.10.2 Module 2

```
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 initMainMenuButtons 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)
```

209

```
83     self.MenuBar.log_out.triggered.connect(self.LogOut)
84     self.MenuBar.change_password.triggered.connect(self.ChangeUsernameOrPassword)
85     self.ViewWindow.btnExit.clicked.connect(self.Back)
86     self.ViewWindow.btnAddBook.clicked.connect(self.AddItem)
87     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.btnExit.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,
116             PubInvoiceService, PubInvoicePayment from 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,
129             BookInvoiceDiscount, ShippingType, ShippingPrice from BookInvoiceItems"
130         self.AddWindow.setFixedSize(450, 150)
131         self.AddWindow.selectedISBN = self.SelectedISBN
132         self.Editing = False
133         self.AddWindow.btnExit.clicked.connect(self.BookInvoiceItemCalculation)
134
135     elif self.CurrentTable == "RoyaltyItems":
136         self.AddWindow.sql = "select RoyaltiesID, ISBN, Currency, RoyaltyDiscount,
137             WholesalePrice, RoyaltyQuantity, PrintCost, ExcRateFromGBP from
138             RoyaltyItems"
```

```
135         self.AddWindow.setFixedSize(450, 250)
136         self.AddWindow.selectedISBN = self.SelectedISBN
137         self.Editing = False
138         self.AddWindow.btnCalculate.clicked.connect(self.RoyaltyItemCalculation)
139
140         self.AddWindow.selectedID = self.SelectedID
141         self.AddWindow.CalendarWidget = dbCalendarWidget()
142         self.AddWindow.CalendarWidget.Calendar()
143
144         self.AddWindow.initAddItemWindow()
145         try:
146             if self.AddWindow.Valid == True: #inputs must pass validation before being
147                 added
148                 self.AddToDB()
149         except AttributeError:
150             pass #exception of window being closed
151         self.RefreshTables()
152
153     def RecalculateItems(self): #recalculates after changes
154         if self.CurrentTable == "BookInvoiceItems":
155             self.BookInvoiceItemsWindow.CalculateBookInvoiceItems()
156             self.CurrentTable = "BookInvoice"
157             self.RefreshTables()
158             self.CurrentTable = "BookInvoiceItems"
159             self.RefreshTables()
160
161         elif self.CurrentTable == "RoyaltyItems" or self.BookEdited == True:
162             self.RoyaltyItemsWindow.CalculateRoyaltyItems()
```

```
162         self.CurrentTable = "Royalties"
163         self.RefreshTables()
164         self.CurrentTable = "RoyaltyItems"
165         self.RefreshTables()
166
167
168     def AddToDB(self): #Adding Entries to database
169         self.input_data = []
170         if self.CurrentTable == "Book":
171             self.TableValues = "Book (ISBN, AuthorID, BookTitle, NoOfPages, Size,
172                               Back, Cover, Paper, Font, FontSize, DatePublished, Price)"
173             self.Placeholders = "(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)"
174             self.NoOfEntries = 12
175
176         elif self.CurrentTable == "PubInvoice":
177             self.TableValues = "PubInvoice (ISBN, AuthorID, PubInvoiceDate,
178                               PubInvoiceService, PubInvoicePayment)"
179             self.Placeholders = "(?, ?, ?, ?, ?)"
180             self.NoOfEntries = 5
181
182         elif self.CurrentTable == "BookInvoice":
183             self.TableValues = "BookInvoice (AuthorID, BookInvoiceDate)"
184             self.Placeholders = "(?, ?)"
185             self.NoOfEntries = 2
186
187         elif self.CurrentTable == "Royalties":
188             self.TableValues = "Royalties (AuthorID, RoyaltiesDate)"
189             self.Placeholders = "(?, ?)"
```

```
188         self.NoOfEntries = 2
189
190     elif self.CurrentTable == "BookInvoiceItems":
191         self.TableValues = "BookInvoiceItems (BookInvoiceID, ISBN,
192             BookInvoiceQuantity, BookInvoiceDiscount, ShippingType, ShippingPrice)"
193         self.Placeholders = "(?, ?, ?, ?, ?, ?, ?)"
194         self.NoOfEntries = 6
195
196     elif self.CurrentTable == "RoyaltyItems":
197         self.TableValues = "RoyaltyItems (RoyaltiesID, ISBN, Currency,
198             RoyaltyDiscount, WholesalePrice, RoyaltyQuantity, PrintCost,
199                 ExcRateFromGBP, NetSales)"
200         self.Placeholders = "(?, ?, ?, ?, ?, ?, ?, ?, ?, ?)"
201         self.NoOfEntries = 9
202
203     for count in range(0, self.NoOfEntries):
204         try: #putting the input data in a list
205             self.input_data.append(str(self.AddWindow.inputList[count].currentText()))
206         except:
207             if count == 8 and self.CurrentTable == "RoyaltyItems":
208                 self.input_data.append(self.NetSales) #Net sales are calculated,
209                     as opposed to being entered
210             else:
211                 self.input_data.append(self.AddWindow.inputList[count].text())
212
213     with sqlite3.connect("PP.db") as db:
214         cursor = db.cursor()
215         cursor.execute("PRAGMA foreign_keys = ON")
```

```
212         self.sql = "insert into {} values {}".format(self.TableValues,
213             self.Placeholders)
214         cursor.execute(self.sql, self.input_data)
215         db.commit()
216
217         self.RefreshTables()
218
219     try:
220         self.RecalculateItems()
221     except:
222         pass
223
224     def RemoveEntry(self): #removing a customer
225         self.SelectedRow = self.TableWidget.CurrentRow()
226         self.ConfirmDialog = dbConfirmationDialog()
227         self.ConfirmDialog.Username = self.Username
228         self.ConfirmDialog.SelectedAuthorID =
229             QTableWidgetItem(self.TableWidget.item(self.SelectedRow, 0)).text()
230         #getting AuthorID of a row
231         if self.ConfirmDialog.SelectedAuthorID != "":
232             self.Firstname = QTableWidgetItem(self.TableWidget.item(self.SelectedRow,
233                 1)).text()
234             self.Lastname = QTableWidgetItem(self.TableWidget.item(self.SelectedRow,
235                 2)).text()
236             self.ConfirmDialog.Name = "{} {}".format(self.Firstname, self.Lastname)
237             self.ConfirmDialog.Msg = "Are you sure you want to delete {} and all
238             records about them?".format(self.ConfirmDialog.Name)
```

```
233     self.ConfirmDialog.ConfirmedMsg = "{}'s records have been successfully  
234         erased.".format(self.ConfirmDialog.Name)  
235     self.ConfirmDialog.VerifyDlg()  
236  
237     if self.ConfirmDialog.ConfirmedDialog.Accepted == True:  
238         with sqlite3.connect("PP.db") as db:  
239             cursor = db.cursor()  
240             cursor.execute("PRAGMA foreign_keys = ON")  
241             sql = "select ISBN from Book where AuthorID =  
242                 {}".format(self.ConfirmDialog.SelectedAuthorID)  
243             cursor.execute(sql)  
244             self.ISBNDeletion = list(cursor.fetchall())  
245             for count in range(0, len(self.ISBNDeletion)): #removes all  
246                 customer details, starting with foreign keys for referential  
247                 integrity  
248                 sql = "delete from BookInvoiceItems where ISBN =  
249                     {}".format(list(list(self.ISBNDeletion)[count])[0])  
250                 cursor.execute(sql)  
251                 sql = "delete from RoyaltyItems where ISBN =  
252                     {}".format(list(list(self.ISBNDeletion)[count])[0])  
253                 cursor.execute(sql)  
254                 sql = "delete from PubInvoice where AuthorID =  
255                     {}".format(self.ConfirmDialog.SelectedAuthorID)  
256                 cursor.execute(sql)  
257                 sql = "delete from BookInvoice where AuthorID =  
258                     {}".format(self.ConfirmDialog.SelectedAuthorID)  
259                 cursor.execute(sql)
```

```
252             sql = "delete from Royalties where AuthorID =
253                 {}".format(self.ConfirmDialog.SelectedAuthorID)
254             cursor.execute(sql)
255             sql = "delete from Book where AuthorID =
256                 {}".format(self.ConfirmDialog.SelectedAuthorID)
257             cursor.execute(sql)
258             sql = "delete from Customer where AuthorID =
259                 {}".format(self.ConfirmDialog.SelectedAuthorID)
260             cursor.execute(sql)
261             db.commit()
262             self.RefreshTables()
263
264     def RemoveFromDB(self): #removing other entries
265         #getting primary key of the row
266         self.ConfirmDialog = dbConfirmationDialog()
267         self.ConfirmDialog.Username = self.Username
268         self.ConfirmDialog.DeleteMsg = self.CurrentTable
269         self.SelectedAuthorID = self.SelectedID
270
271         if self.CurrentTable == "Book":
272             self.ForeignKeyMsg = "Royalties and Invoices"
273             self.SelectedRow = self.ViewWindow.table.currentRow()
274             self.SelectedID =
275                 QTableWidgetItem(self.ViewWindow.table.item(self.SelectedRow, 0)).text()
276             self.SelectedIDName = "ISBN"
277             self.ConfirmDialog.Msg = "Are you sure you want to delete this book?"
```

```
276         self.ConfirmDialog.ConfirmedMsg = "Book was successfully deleted"
277
278     elif self.CurrentTable == "PubInvoice":
279         self.SelectedRow = self.PubInvoiceWindow.table.currentRow()
280         self.SelectedID =
281             QTableWidgetItem(self.PubInvoiceWindow.table.item(self.SelectedRow,
282                                         0)).text()
283         self.SelectedIDName = "PubInvoiceID"
284         self.ConfirmDialog.Msg = "Are you sure you want to delete this Invoice?"
285         self.ConfirmDialog.ConfirmedMsg = "Invoice was successfully deleted"
286
287     elif self.CurrentTable == "BookInvoice":
288         self.ForeignKeyMsg = "Book Invoice Items"
289         self.SelectedRow = self.BookInvoiceWindow.table.currentRow()
290         self.SelectedID =
291             QTableWidgetItem(self.BookInvoiceWindow.table.item(self.SelectedRow,
292                                         0)).text()
293         self.SelectedIDName = "BookInvoiceID"
294         self.ConfirmDialog.Msg = "Are you sure you want to delete this Invoice?"
295         self.ConfirmDialog.ConfirmedMsg = "Invoice was successfully deleted"
296
297     elif self.CurrentTable == "Royalties":
298         self.ForeignKeyMsg = "Royalty Items"
299         self.SelectedRow = self.RoyaltiesWindow.table.currentRow()
300         self.SelectedID =
301             QTableWidgetItem(self.RoyaltiesWindow.table.item(self.SelectedRow,
302                                         0)).text()
302         self.SelectedIDName = "RoyaltiesID"
```

```
298         self.ConfirmDialog.Msg = "Are you sure you want to delete this Entry?"  
299         self.ConfirmDialog.ConfirmedMsg = "Entry was successfully deleted"  
300  
301     elif self.CurrentTable == "BookInvoiceItems":  
302         self.SelectedRow = self.BookInvoiceItemsWindow.table.currentRow()  
303         self.SelectedID =  
304             QTableWidgetItem(self.BookInvoiceItemsWindow.table.item(self.SelectedRow,  
305                             0)).text()  
306         self.SelectedIDName = "BookInvoiceItemsID"  
307         self.ConfirmDialog.Msg = "Are you sure you want to delete this Item?"  
308         self.ConfirmDialog.ConfirmedMsg = "Item was successfully deleted"  
309  
310  
311     elif self.CurrentTable == "RoyaltyItems":  
312         self.SelectedRow = self.RoyaltyItemsWindow.table.currentRow()  
313         self.SelectedID =  
314             QTableWidgetItem(self.RoyaltyItemsWindow.table.item(self.SelectedRow,  
315                             0)).text()  
316         self.SelectedIDName = "RoyaltyItemsID"  
317         self.ConfirmDialog.Msg = "Are you sure you want to delete this Item?"  
318         self.ConfirmDialog.ConfirmedMsg = "Item was successfully deleted"  
319         self.ConfirmDialog.Prevention = True #deletion may be prevented if integrity  
320             error occurs  
321     if self.SelectedRow != -1:  
322         self.ConfirmDialog.VerifyDlg() #verification  
323     try:  
324         if self.ConfirmDialog.ConfirmedDialog.Accepted == True:  
325             with sqlite3.connect("PP.db") as db:  
326                 cursor = db.cursor()
```

```
321         cursor.execute("PRAGMA foreign_keys = ON")
322         sql = "delete from {} where {} =
323             ' {}'.format(self.CurrentTable, self.SelectedIDName,
324                         self.SelectedID)
325         cursor.execute(sql)
326         db.commit()
327         self.ConfirmDialog.ConfirmedDialog.Confirmed()
328         self.SelectedID = self.SelectedAuthorID
329         self.RefreshTables()
330     except sqlite3.IntegrityError:
331         self.Msg = QMessageBox()
332         self.Msg.setWindowTitle("Error")
333         self.Msg.setText("You must delete all the {}"
334                         " first.".format(self.ForeignKeyMsg))
335         self.Msg.exec_() #informs user that selected entry cannot be deleted
336         and what must be done for it to be deleted
337
338     try:
339         self.RecalculateItems()
340     except:
341         pass
342
343 def ViewCustomer(self): #displaying customer data
344     self.SelectedRow = self.TableWidget.currentRow()
```

```
344     self.SelectedID = QTableWidgetItem(self.TableWidget.item(self.SelectedRow,
345                                         0)).text()
346     self.Firstname = QTableWidgetItem(self.TableWidget.item(self.SelectedRow,
347                                         1)).text()
348     self.Lastname = QTableWidgetItem(self.TableWidget.item(self.SelectedRow,
349                                         2)).text()
350
351     if self.SelectedID != "":
352         self.SelectedAuthorID = self.SelectedID
353         self.CurrentTable = "Book"
354         self.RefreshTables()
355         self.StackedLayout.setCurrentIndex(1)
356         self.MenuBar.setVisible(False)
357
358     def ViewPubInvoice(self): #initialising the view publishing invoice window
359         self.CurrentTable = "PubInvoice"
360         self.ViewWindow.SelectedRow = self.ViewWindow.table.currentRow()
361         self.SelectedISBN =
362             QTableWidgetItem(self.ViewWindow.table.item(self.ViewWindow.SelectedRow,
363                                         0)).text()
364         self.SelectedID =
365             QTableWidgetItem(self.TableWidget.item(self.TableWidget.currentRow(),
366                                         0)).text()
367         self.SelectedAuthorID = self.SelectedID
368         if self.SelectedISBN != "":
369             self.PubInvoiceWindow = dbRoyaltiesAndInvoices()
```

```
365         self.PubInvoiceWindow.PubInvoiceButtons()
366         self.PubInvoiceWindow.btnAddPubInvoice.clicked.connect(self.AddItem)
367         self.PubInvoiceWindow.btnUpdatePubInvoice.clicked.connect(self.UpdateEntry)
368         self.PubInvoiceWindow.btnDeleteEntry.clicked.connect(self.RemoveFromDB)
369         self.PubInvoiceWindow.table = dbTableWidget()
370         self.RefreshTables()
371         self.PubInvoiceWindow.table.setFixedSize(620, 150)
372         self.PubInvoiceWindow.PubInvoice()
373         self.CurrentTable = "Book"
374
375
376
377     def ViewBookInvoice(self): #initialising the view book invoice window
378         self.CurrentTable = "BookInvoice"
379         self.ViewWindow.SelectedRow = self.ViewWindow.table.currentRow()
380         self.SelectedISBN =
381             QTableWidgetItem(self.ViewWindow.table.item(self.ViewWindow.SelectedRow,
382                                         0)).text()
383         self.SelectedAuthorID =
384             QTableWidgetItem(self.TableWidget.item(self.TableWidget.currentRow(),
385                                         0)).text()
386
387         if self.SelectedISBN != "":
388             self.BookInvoiceWindow = dbRoyaltiesAndInvoices()
389             self.BookInvoiceWindow.BookInvoiceButtons()
390             self.BookInvoiceWindow.btnViewBookInvoiceItems.clicked.connect(self.ViewBookInvoiceItems)
391             self.BookInvoiceWindow.btnAddBookInvoice.clicked.connect(self.AddItem)
392             self.BookInvoiceWindow.btnUpdateBookInvoice.clicked.connect(self.UpdateEntry)
```

```
389         self.BookInvoiceWindow.btnDeleteEntry.clicked.connect(self.RemoveFromDB)
390         self.BookInvoiceWindow.table = dbTableWidget()
391         self.RefreshTables()
392         self.BookInvoiceWindow.table.setFixedSize(620, 150)
393         self.BookInvoiceWindow.BookInvoice()
394         self.CurrentTable = "Book"
395
396
397
398     def ViewBookInvoiceItems(self): #initialising the view book invoice items window
399         self.CurrentTable = "BookInvoiceItems"
400         self.BookInvoiceWindow.SelectedRow = self.BookInvoiceWindow.table.currentRow()
401         self.holder = self.SelectedAuthorID
402         self.SelectedAuthorID = self.SelectedID
403         self.BookInvoiceWindow.selectedISBN = self.SelectedISBN
404         self.SelectedID =
405             QTableWidgetItem(self.BookInvoiceWindow.table.item(self.BookInvoiceWindow.SelectedRow,
406             0)).text()
407
408     if self.SelectedID != "":
409         self.BookInvoiceItemsWindow = dbItems()
410         self.BookInvoiceItemsWindow.selectedID = self.SelectedID
411         self.BookInvoiceItemsWindow.selectedISBN = self.SelectedISBN
412         self.BookInvoiceItemsWindow.BookInvoiceItemsButtons()
413         self.BookInvoiceItemsWindow.btnCalculate.clicked.connect(self.AddItem)
414         self.BookInvoiceItemsWindow.btnUpdateBookInvoiceItems.clicked.connect(self.UpdateEntry)
415         self.BookInvoiceItemsWindow.btnDeleteEntry.clicked.connect(self.RemoveFromDB)
416         self.BookInvoiceItemsWindow.table = dbTableWidget()
```

```
415         self.RefreshTables()
416         self.BookInvoiceItemsWindow.table.setFixedSize(620, 150)
417         self.BookInvoiceItemsWindow.BookInvoiceItems()
418         self.SelectedID = self.SelectedAuthorID
419         self.SelectedAuthorID = self.holder
420         self.CurrentTable = "BookInvoice"
421         self.RefreshTables()
422
423
424
425     def ViewRoyalties(self): #initialising the view royalties window
426         self.CurrentTable = "Royalties"
427         self.ViewWindow.SelectedRow = self.ViewWindow.table.currentRow()
428         self.SelectedISBN =
429             QTableWidgetItem(self.ViewWindow.table.item(self.ViewWindow.SelectedRow,
430                 0).text())
430         self.SelectedAuthorID =
431             QTableWidgetItem(self.TableWidget.item(self.TableWidget.currentRow(),
432                 0).text())
433         if self.SelectedISBN != "":
434             self.RoyaltiesWindow = dbRoyaltiesAndInvoices()
435             self.RoyaltiesWindow.RoyaltiesButtons()
436             self.RoyaltiesWindow.btnViewRoyaltyItems.clicked.connect(self.ViewRoyaltyItems)
437             self.RoyaltiesWindow.btnAddRoyalties.clicked.connect(self.AddItem)
438             self.RoyaltiesWindow.btnUpdateRoyalties.clicked.connect(self.UpdateEntry)
439             self.RoyaltiesWindow.btnDeleteEntry.clicked.connect(self.RemoveFromDB)
440             self.RoyaltiesWindow.table = dbTableWidget()
441             self.RefreshTables()
```

```
439         self.RoyaltiesWindow.table.setFixedSize(620, 150)
440         self.RoyaltiesWindow.Royalties()
441     self.CurrentTable = "Book"
442
443     def ViewRoyaltyItems(self): #initialising the view royalty items window
444         self.CurrentTable = "RoyaltyItems"
445         self.RoyaltiesWindow.SelectedRow = self.RoyaltiesWindow.table.currentRow()
446         self.holder = self.SelectedAuthorID
447         self.SelectedAuthorID = self.SelectedID
448         self.RoyaltiesWindow.selectedISBN = self.SelectedISBN
449         self.SelectedID =
450             QTableWidgetItem(self.RoyaltiesWindow.table.item(self.RoyaltiesWindow.SelectedRow
451 0)).text()
452
453     if self.SelectedID != "":
454         self.RoyaltyItemsWindow = dbItems()
455         self.RoyaltyItemsWindow.selectedID = self.SelectedID
456         self.RoyaltyItemsWindow.selectedISBN = self.SelectedISBN
457         self.RoyaltyItemsWindow.RoyaltyItemsButtons()
458         self.RoyaltyItemsWindow.btnCalculate.clicked.connect(self.AddItem)
459         self.RoyaltyItemsWindow.btnUpdateRoyaltyItems.clicked.connect(self.UpdateEntry)
460         self.RoyaltyItemsWindow.btnDeleteEntry.clicked.connect(self.RemoveFromDB)
461         self.RoyaltyItemsWindow.table = dbTableWidget()
462         self.RefreshTables()
463         self.RoyaltyItemsWindow.table.setFixedSize(620, 150)
464         self.RoyaltyItemsWindow.RoyaltiesItems()
465     self.SelectedID = self.SelectedAuthorID
466     self.SelectedAuthorID = self.holder
```

```
465     self.CurrentTable = "Royalties"
466     self.RefreshTables()
467
468
469
470     def BookInvoiceItemCalculation(self): #calculating the bookinvoicepayment
471         try:
472             if self.Editing == False:
473                 self.Quantity = int(self.AddWindow.inputList[2].text())
474                 self.Discount = float(self.AddWindow.inputList[3].text()) / 100
475                 self.ShippingPrice = float(self.AddWindow.inputList[5].text())
476                 self.ISBN = self.AddWindow.inputList[1].text()
477             elif self.Editing == True:
478                 self.Quantity = int(self.EditWindow.inputList[2].text())
479                 self.Discount = float(self.EditWindow.inputList[3].text()) / 100
480                 self.ShippingPrice = float(self.EditWindow.inputList[5].text())
481                 self.ISBN = self.EditWindow.inputList[1].text()
482
483             with sqlite3.connect("PP.db") as db: #fetching data from db
484                 cursor = db.cursor()
485                 cursor.execute("select Price from Book where ISBN =
486                             {}".format(self.ISBN))
487                 self.Price = list(cursor.fetchone())[0]
488                 db.commit()
489
490                 self.BookInvoiceItemPayment = (self.Quantity * self.Price)
491                 self.Discount = self.BookInvoiceItemPayment * self.Discount
492                 self.BookInvoiceItemPayment -= self.Discount
```

```
492     self.BookInvoiceItemPayment += self.ShippingPrice
493     self.BookInvoiceItemPayment =
494         "f{0:.2f}.".format(self.BookInvoiceItemPayment)
495     if self.Editing == False:
496         self.AddWindow.btnConfirm.clicked.connect(self.AddWindow.accept)
497         self.AddWindow.qleCalculation.setText(self.BookInvoiceItemPayment)
498     elif self.Editing == True:
499         if self.EditWindow.Calculated == False:
500             self.EditWindow.Calculated = True #prevents duplicate connections
501             self.EditWindow.btnConfirm.clicked.connect(self.VerifyUpdate)
502             self.EditWindow.qleCalculation.setText(self.BookInvoiceItemPayment)
503     except:
504         self.Msg = QMessageBox()
505         self.Msg.setWindowTitle("Error")
506         self.Msg.setText("You must fill all fields before clicking 'Calculate'.")
507         self.Msg.exec_()
508
509     def RoyaltyItemCalculation(self): #calculating the royalty payment
510         try:
511             if self.Editing == False:
512                 self.Currency = self.AddWindow.inputList[2].text()
513                 self.WholesalePrice = float(self.AddWindow.inputList[4].text())
514                 self.Quantity = int(self.AddWindow.inputList[5].text())
515                 self.PrintCost = float(self.AddWindow.inputList[6].text())
516             elif self.Editing == True:
517                 self.Currency = self.EditWindow.inputList[2].text()
518                 self.WholesalePrice = float(self.EditWindow.inputList[4].text())
519                 self.Quantity = int(self.EditWindow.inputList[5].text())
```

```
519         self.PrintCost = float(self.EditWindow.inputList[6].text())
520
521         self.NetSales = self.WholesalePrice * self.Quantity
522         self.RoyaltyItemPayment = self.NetSales - self.PrintCost
523         self.RoyaltyItemPayment = "{0:.2f}".format(self.RoyaltyItemPayment)
524         if self.Editing == False:
525             self.AddWindow.btnExit.clicked.connect(self.AddWindow.accept)
526             self.AddWindow.qleCalculation.setText("{}{}".format(self.Currency,
527                                         self.RoyaltyItemPayment))
528             self.AddWindow.NetSales = self.NetSales
529         elif self.Editing == True:
530             if self.EditWindow.Calculated == False:
531                 self.EditWindow.Calculated = True #prevents duplicate connections
532                 self.EditWindow.btnExit.clicked.connect(self.VerifyUpdate)
533                 self.EditWindow.qleCalculation.setText("{}{}".format(self.Currency,
534                                         self.RoyaltyItemPayment))
535                 self.EditWindow.NetSales = self.NetSales
536         except:
537             self.Msg = QMessageBox()
538             self.Msg.setWindowTitle("Error")
539             self.Msg.setText("You must fill all fields before clicking 'Calculate'.")
540             self.Msg.exec_()
541
542     def UpdateCustomerEntry(self): #updating customer entries only
543         self.SelectedRow = self.TableWidget.CurrentRow()
544         self.SelectedAuthorID =
545             QTableWidgetItem(self.TableWidget.item(self.SelectedRow, 0)).text()
```

```
544
545     if self.SelectedAuthorID != "":
546         self.UpdateEntryWindow = dbUpdateEntryWindow()
547         self.UpdateEntryWindow.setFixedSize(640, 115)
548         self.UpdateEntryWindow.selectedID = self.SelectedAuthorID
549         self.UpdateEntryWindow.table = dbTableWidget()
550         self.selectText = "Firstname, Lastname, Email, Phonenumber, Address,
551                         Postcode"
552         self.UpdateEntryWindow.table.sql = "select {} from Customer where AuthorID
553                                         = {}".format(self.selectText, self.SelectedAuthorID)
554         self.SelectedID = self.SelectedAuthorID
555         self.UpdateEntryWindow.table.initTable()
556         self.UpdateEntryWindow.table.setFixedSize(617, 55)
557         self.UpdateEntryWindow.Verify = dbConfirmationDialog()
558         self.UpdateEntryWindow.Verify.Username = self.Username
559         self.UpdateEntryWindow.initConfirmBtn()
560         self.UpdateEntryWindow.btnConfirm.clicked.connect(self.VerifyCustomerUpdate)
561         self.UpdateEntryWindow.initUpdateEntryWindowDlg()
562         self.TableWidget.sql = "select * from Customer"
563         self.RefreshTables()
564
565     def UpdateEntry(self): #getting the update input
566         self.Selection = False
567         self.EditWindow = dbAddItemWindow() #uses the add window to init same
568             interface but fill boxes with data
569         self.EditWindow.setFixedSize(360, 200)
```

```
569     self.EditWindow.AddType = self.CurrentTable
570     self.EditWindow.AnswerButtons()
571     self.EditWindow.ReadyToVerify = False
572     self.EditWindow.originalItemList = []
573
574     if self.CurrentTable == "Book":
575         self.EditWindow.sql = "select * from Book"
576         self.ViewWindow.SelectedRow = self.ViewWindow.table.currentRow()
577         if self.ViewWindow.SelectedRow != -1:
578             self.Selection = True
579             with sqlite3.connect("PP.db") as db:
580                 cursor = db.cursor() #fetching data from database for the edit
581                 window
582                 sql = "select * from Book where ISBN =
583                     {}".format(self.ViewWindow.table.item(self.ViewWindow.SelectedRow,
584                         0).text())
585                 cursor.execute(sql)
586                 self.EditWindow.originalItemList = list(cursor.fetchone())
587
588             elif self.CurrentTable == "PubInvoice":
589                 self.EditWindow.setFixedSize(400,150)
590                 self.EditWindow.selectedISBN = self.SelectedISBN
591                 self.EditWindow.sql = "select ISBN, AuthorID, PubInvoiceDate,
592                     PubInvoiceService, PubInvoicePayment from PubInvoice"
593                 self.PubInvoiceWindow.SelectedRow =
594                     self.PubInvoiceWindow.table.currentRow()
595                 if self.PubInvoiceWindow.SelectedRow != -1:
596                     self.Selection = True
```

```
592     with sqlite3.connect("PP.db") as db:
593         cursor = db.cursor() #fetching data from database for the edit
594             window
595         sql = "select ISBN, AuthorID, PubInvoiceDate, PubInvoiceService,
596             PubInvoicePayment from PubInvoice where PubInvoiceID =
597             {}".format(self.PubInvoiceWindow.table.item(self.PubInvoiceWindow.SelectedRow,
598                 0).text())
599         cursor.execute(sql)
600         self.EditWindow.originalItemList = list(cursor.fetchone())
601
602 elif self.CurrentTable == "BookInvoice":
603     self.EditWindow.setFixedSize(350, 100)
604     self.EditWindow.selectedID = self.SelectedID
605     self.EditWindow.sql = "select AuthorID, BookInvoiceDate from BookInvoice"
606     self.BookInvoiceWindow.SelectedRow =
607         self.BookInvoiceWindow.table.currentRow()
608 if self.BookInvoiceWindow.SelectedRow != -1:
609     self.Selection = True
610     with sqlite3.connect("PP.db") as db:
611         cursor = db.cursor() #fetching data from database for the edit
612             window
613         sql = "select AuthorID, BookInvoiceDate from BookInvoice where
614             BookInvoiceID =
615             {}".format(self.BookInvoiceWindow.table.item(self.BookInvoiceWindow.SelectedRow,
616                 0).text())
617         cursor.execute(sql)
618         self.EditWindow.originalItemList = list(cursor.fetchone())
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

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

```
633         self.Selection = True
634         with sqlite3.connect("PP.db") as db:
635             cursor = db.cursor() #fetching data from database for the edit
636             window
637             sql = "select BookInvoiceID, ISBN, BookInvoiceQuantity,
638                   BookInvoiceDiscount, ShippingType, ShippingPrice from
639                   BookInvoiceItems where BookInvoiceItemsID =
640                   {}".format(self.BookInvoiceItemsWindow.table.item(self.BookInvoiceItemsWindow.Select
641                   0).text())
642             cursor.execute(sql)
643             self.EditWindow.originalItemList = list(cursor.fetchone())
644
645 elif self.CurrentTable == "RoyaltyItems":
646     self.EditWindow.setFixedSize(450, 250)
647     self.EditWindow.selectedID = self.SelectedID
648     self.Editing = True
649     self.EditWindow.btnCalculate.clicked.connect(self.RoyaltyItemCalculation)
650     self.EditWindow.selectedISBN = self.SelectedISBN
651     self.EditWindow.sql = "select RoyaltiesID, ISBN, Currency,
652                           RoyaltyDiscount, WholesalePrice, RoyaltyQuantity, PrintCost,
653                           ExcRateFromGBP from RoyaltyItems"
654     self.RoyaltyItemsWindow.SelectedRow =
655         self.RoyaltyItemsWindow.table.currentRow()
656     if self.RoyaltyItemsWindow.SelectedRow != -1:
657         self.Selection = True
658         with sqlite3.connect("PP.db") as db:
659             cursor = db.cursor() #fetching data from database for the edit
660             window
```

```
652         sql = "select RoyaltiesID, ISBN, Currency, RoyaltyDiscount,
653             WholesalePrice, RoyaltyQuantity, PrintCost, ExcRateFromGBP from
654             RoyaltyItems where RoyaltyItemsID =
655                 {}".format(self.RoyaltyItemsWindow.table.item(self.RoyaltyItemsWindow.SelectedRow,
656                     0).text())
656             cursor.execute(sql)
657             self.EditWindow.originalItemList = list(cursor.fetchone())
658
659             self.EditWindow.Editing = True
660
661             if self.Selection == True: #initialising the edit window
662                 self.EditWindow.btnExit.clicked.connect(self.EditWindow.Validate)
663                 if self.CurrentTable in ["RoyaltyItems", "BookInvoiceItems"]:
664                     self.EditWindow.btnExit.clicked.connect(self.EditWindow.CheckCalculated)
665                 else:
666                     self.EditWindow.btnExit.clicked.connect(self.VerifyUpdate)
667                 self.EditWindow.AddType = self.CurrentTable
668                 self.EditWindow.selectedID = self.SelectedID
669
670
671
672             self.EditWindow.CalendarWidget = dbCalendarWidget()
673             self.EditWindow.CalendarWidget.Calendar()
674             self.EditWindow.initAddItemWindow()
675
676
677             def VerifyCustomerUpdate(self):
678                 self.UpdateEntryWindow.Verify = dbConfirmationDialog()
679                 #new instance for customer verification
```

```
676
677
678
679     def VerifyUpdate(self): #verification dialog
680         if self.EditWindow.ReadyToVerify == True:
681             self.EditWindow.Verify = dbConfirmationDialog()
682             self.EditWindow.Verify.Msg = "Insert Password to confirm all changes"
683             self.EditWindow.Verify.ConfirmedMsg = "Update successful"
684             self.EditWindow.VerifyDlg()
685             if self.EditWindow.Verify.ConfirmedDialog.Accepted == True:
686                 self.UpdateChanges()
687                 self.EditWindow.accept()
688
689
690
234 691     def UpdateChanges(self): #committing changes
692         self.UpdateList = []
693
694         with sqlite3.connect("PP.db") as db:
695             cursor = db.cursor()
696             if self.CurrentTable == "Book":
697                 self.NoOfEntries = 12
698                 cursor.execute("select * from Book")
699                 self.ID = "ISBN"
700                 self.SelectedISBN =
701                     QTableWidgetItem(self.ViewWindow.table.item(self.ViewWindow.SelectedRow,
702                                         0)).text()
702                 self.SelectedID = self.SelectedISBN
```

```
702         self.BookEdited = True
703
704     elif self.CurrentTable == "PubInvoice":
705         cursor.execute("select ISBN, AuthorID, PubInvoiceDate,
706                         PubInvoiceService, PubInvoicePayment from PubInvoice")
707         self.NoOfEntries = 5
708         self.ID = "PubInvoiceID"
709         self.OldID = self.SelectedID
710         self.SelectedID =
711             QTableWidgetItem(self.PubInvoiceWindow.table.item(self.PubInvoiceWindow.SelectedRow,
712                                         0)).text()
713
714     elif self.CurrentTable == "BookInvoice":
715         cursor.execute("select AuthorID, BookInvoiceDate from BookInvoice")
716         self.NoOfEntries = 2
717         self.ID = "BookInvoiceID"
718         self.OldID = self.SelectedID
719         self.SelectedID =
720             QTableWidgetItem(self.BookInvoiceWindow.table.item(self.BookInvoiceWindow.SelectedRow,
721                                         0)).text()
722
723     elif self.CurrentTable == "Royalties":
724         cursor.execute("select AuthorID, RoyaltiesDate from Royalties")
725         self.NoOfEntries = 2
726         self.ID = "RoyaltiesID"
727         self.OldID = self.SelectedID
728         self.SelectedID =
729             QTableWidgetItem(self.RoyaltiesWindow.table.item(self.RoyaltiesWindow.SelectedRow,
```

```
    0)).text()

724
725     elif self.CurrentTable == "BookInvoiceItems":
726         cursor.execute("select BookInvoiceID, ISBN, BookInvoiceQuantity,
727                         BookInvoiceDiscount, ShippingType, ShippingPrice from
728                         BookInvoiceItems")
729         self.NoOfEntries = 6
730         self.ID = "BookInvoiceItemsID"
731         self.OldID = self.SelectedID
732         self.SelectedID =
733             QTableWidgetItem(self.BookInvoiceItemsWindow.table.item(self.BookInvoiceItemsWindow.SelectedRow,
734                                         0)).text()

735
736     elif self.CurrentTable == "RoyaltyItems":
737         cursor.execute("select RoyaltiesID, ISBN, Currency, RoyaltyDiscount,
738                         WholesalePrice, RoyaltyQuantity, PrintCost, ExcRateFromGBP from
739                         RoyaltyItems")
740         self.NoOfEntries = 6
741         self.ID = "RoyaltyItemsID"
742         self.OldID = self.SelectedID
743         self.SelectedID =
744             QTableWidgetItem(self.RoyaltyItemsWindow.table.item(self.RoyaltyItemsWindow.SelectedRow,
745                                         0)).text()

746
747     for count in range(0, self.NoOfEntries): #creating the update string for sql
748         try:
749             self.UpdateList.append(str(self.EditWindow.inputList[count].currentText()))
750         except:
```

```
743         if count == 8 and self.CurrentTable == "RoyaltyItems":
744             self.UpdateList.append(str(self.NetSales))
745         else:
746             self.UpdateList.append(self.EditWindow.inputList[count].text())
747
748     self.Update = ""
749
750     self.Headers = list(cursor.description)
751     for count in range(0, len(self.UpdateList)):
752         self.Update += "{} = '{}'.format(list(self.Headers[count])[0],
753                                         self.UpdateList[count])
754         if count != len(self.UpdateList) - 1:
755             self.Update += ", "
756
757     with sqlite3.connect("PP.db") as db: #update
758         cursor = db.cursor()
759         if self.CurrentTable == "Book": #ISBN is primary key which may be changed,
760             so all foreign keys will change first for it
761         try:
762             sql = "update PubInvoice set ISBN = {0} where ISBN =
763                 {1}".format(self.UpdateList[0], self.SelectedID)
764             cursor.execute(sql)
765             sql = "update BookInvoiceItems set ISBN = {0} where ISBN =
766                 {1}".format(self.UpdateList[0], self.SelectedID)
767             cursor.execute(sql)
768             sql = "update RoyaltyItems set ISBN = {0} where ISBN =
769                 {1}".format(self.UpdateList[0], self.SelectedID)
770             cursor.execute(sql)
```

```
766             except:
767                 pass # error if no entry is there for
768                     pubinvoice/bookinvoiceitems/royaltyitems
769             sql = "update {} set {} where {} = {}".format(self.CurrentTable,
770                     self.Update, self.ID, self.SelectedID)
771             cursor.execute(sql)
772             db.commit()
773
774         if self.CurrentTable in ["Royalties", "BookInvoice", "PubInvoice",
775             "BookInvoiceItems", "RoyaltyItems"]:
776             self.SelectedID = self.OldID
777
778         self.RefreshTables()
779
780     try:
781         self.RecalculateItems()
782         self.BookEdited = False
783
784     except:
785         pass
786
787
788     def RefreshTables(self): #refreshing tables
789         if self.CurrentTable == "Customer":
790             self.TableWidget.sql = "select * from Customer"
791             self.TableWidget.initTable()
792
793         if self.CurrentTable == "Book":
```

```
791     self.ViewWindow.table.sql = "select * from Book where AuthorID =  
792         {}".format(self.SelectedAuthorID)  
793     self.ViewWindow.table.initTable()  
794     with sqlite3.connect("PP.db") as db:  
795         cursor = db.cursor()  
796         cursor.execute("select Firstname, Lastname from Customer where  
797             AuthorID = {}".format(self.SelectedAuthorID))  
798         self.Name = list(cursor.fetchone())  
799         self.Name = "{} , {}".format(self.Name[1], self.Name[0])  
800         for count in range(0, self.TableWidget.rowCount()+1):  
801             self.ViewWindow.table.setItem(count, 1,  
802                 QTableWidgetItem(self.Name))  
803             self.ViewWindow.table.setHorizontalHeaderItem(1,  
804                 QTableWidgetItem("Author"))  
805             db.commit()  
806 elif self.CurrentTable == "PubInvoice":  
807     self.PubInvoiceWindow.table.sql = "select * from PubInvoice where ISBN =  
808         {}".format(self.SelectedISBN)  
809     self.PubInvoiceWindow.table.initTable()  
810     with sqlite3.connect("PP.db") as db:  
811         cursor = db.cursor()  
812         cursor.execute("select BookTitle from Book where ISBN =  
813             {}".format(self.SelectedISBN))  
814         self.Title = "{}".format(list(cursor.fetchone())[0])  
815         for count in range(0, self.TableWidget.rowCount()+1):  
816             self.PubInvoiceWindow.table.setItem(count, 1,  
817                 QTableWidgetItem(self.Title))
```

```
811         self.PubInvoiceWindow.table.setHorizontalHeaderItem(1,
812             QTableWidgetItem("BookTitle"))
813         cursor.execute("select Firstname, Lastname from Customer where
814             AuthorID = {}".format(self.SelectedAuthorID))
815         self.Name = list(cursor.fetchone())
816         self.Name = "{}, {}".format(self.Name[1], self.Name[0])
817         for count in range(0, self.TableWidget.rowCount()+1):
818             self.PubInvoiceWindow.table.setItem(count, 2,
819                 QTableWidgetItem(self.Name))
820         self.PubInvoiceWindow.table.setHorizontalHeaderItem(2,
821             QTableWidgetItem("Author"))
822         db.commit()
823
824
825 elif self.CurrentTable == "BookInvoice":
826     self.BookInvoiceWindow.table.sql = "select * from BookInvoice where
827         AuthorID = {}".format(self.SelectedAuthorID)
828     self.BookInvoiceWindow.table.initTable()
829     with sqlite3.connect("PP.db") as db:
830         cursor = db.cursor()
831         cursor.execute("select Firstname, Lastname from Customer where
832             AuthorID = {}".format(self.SelectedAuthorID))
833         self.Name = list(cursor.fetchone())
834         self.Name = "{}, {}".format(self.Name[1], self.Name[0])
835         for count in range(0, self.TableWidget.rowCount()+1):
836             self.BookInvoiceWindow.table.setItem(count, 1,
837                 QTableWidgetItem(self.Name))
838         self.BookInvoiceWindow.table.setHorizontalHeaderItem(1,
839             QTableWidgetItem("Author"))
```

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

```
853     cursor.execute(sql)
854     self.IDList = list(cursor.fetchall())
855     for count in range(0, len(self.IDList)):
856         self.temp = list(self.IDList[count])[0]
857         self.ID.append(self.temp)
858
859     for count in range(0, len(self.ID)):
860         try:
861             cursor.execute("select Book.BookTitle, BookInvoiceItems.ISBN,
862                             Firstname, Lastname, BookInvoiceItems.BookInvoiceID from
863                             Book, BookInvoiceItems, Customer, BookInvoice where
864                             BookInvoiceItems.BookInvoiceID = {} and
865                             BookInvoiceItems.BookInvoiceItemsID = {} and Book.ISBN =
866                             BookInvoiceItems.ISBN and BookInvoice.BookInvoiceID =
867                             BookInvoiceItems.BookInvoiceID and Customer.AuthorID =
868                             BookInvoice.AuthorID".format(self.SelectedID,
869                             self.ID[count]))
870             self.Title = list(cursor.fetchone())
871             self.Name = "{} , {}".format(self.Title[3], self.Title[2])
872             self.Title = self.Title[0]
873             self.BookInvoiceItemsWindow.table.setItem(count, 2,
874                 QTableWidgetItem(str(self.Title)))
875             self.BookInvoiceItemsWindow.table.setItem(count, 1,
876                 QTableWidgetItem(str(self.Name)))
877         except:
878             pass
879     self.BookInvoiceItemsWindow.table.setHorizontalHeaderItem(2,
880     QTableWidgetItem("BookTitle"))
```

```
870         self.BookInvoiceItemsWindow.table.setHorizontalHeaderItem(1,
871                         QTableWidgetItem("Author"))
872         db.commit()
873
874     elif self.CurrentTable == "RoyaltyItems":
875         self.RoyaltyItemsWindow.table.sql = "select * from RoyaltyItems where
876             RoyaltiesID = {}".format(self.SelectedID)
877         self.RoyaltyItemsWindow.table.initTable()
878         with sqlite3.connect("PP.db") as db:
879             cursor = db.cursor()
880             self.ID = []
881             sql = "select RoyaltyItemsID from RoyaltyItems where RoyaltiesID =
882                 {}".format(self.SelectedID)
883             cursor.execute(sql)
884             self.IDList = list(cursor.fetchall())
885             for count in range(0, len(self.IDList)):
886                 self.temp = list(self.IDList[count])[0]
887                 self.ID.append(self.temp)
888
889             for count in range(0, len(self.ID)):
890                 try:
891                     cursor.execute("select Book.BookTitle, RoyaltyItems.ISBN,
892                         Firstname, Lastname, RoyaltyItems.RoyaltiesID from Book,
893                         RoyaltyItems, Customer, Royalties where
894                         RoyaltyItems.RoyaltiesID = {} and
895                         RoyaltyItems.RoyaltyItemsID = {} and Book.ISBN =
896                         RoyaltyItems.ISBN and Royalties.RoyaltiesID =
897                         RoyaltyItems.RoyaltiesID and Customer.AuthorID =
```

```
889             Royalties.AuthorID".format(self.SelectedID, self.ID[count]))
890         self.Title = list(cursor.fetchone())
891         self.Name = "{}, {}".format(self.Title[3], self.Title[2])
892         self.Title = self.Title[0]
893         self.RoyaltyItemsWindow.table.setItem(count, 2,
894             QTableWidgetItem(str(self.Title)))
895         self.RoyaltyItemsWindow.table.setItem(count, 1,
896             QTableWidgetItem(str(self.Name)))
897     except:
898         pass
899     self.RoyaltyItemsWindow.table.setHorizontalHeaderItem(2,
900         QTableWidgetItem("BookTitle"))
901     self.RoyaltyItemsWindow.table.setHorizontalHeaderItem(1,
902         QTableWidgetItem("Author"))
903     db.commit()
904
905
906
907
908     def Back(self): #going back from the view window to the main menu
909         self.ViewWindow.table.selectedID = None
910         self.CurrentTable = "Customer"
911         self.StackedLayout.setCurrentIndex(0)
912         self.MenuBar.setVisible(True)
913
914
915
916
917     def QuickSearch(self): #quick search from main menu
918         self.QSText = self.MainMenuButtons.leQuickSearch.text()
919         self.QSText = self.QSText.split(' ')
920         with sqlite3.connect("PP.db") as db:
```

```
912     cursor = db.cursor()
913
914     if len(self.QSText) == 1:
915         self.TableWidget.sql = "select * from Customer where Firstname like
916             '{0}%' or Lastname like '{0}%'.format(self.QSText[0])
917     elif len(self.QSText) == 0:
918         self.TableWidget.sql = "select * from Customer"
919     else:
920         for count in range(1, len(self.QSText)):
921             if count != 1:
922                 self.QSText[1] += " {}".format(self.QSText[count])
923     self.TableWidget.sql = "select * from Customer where Firstname like
924             '{0}%' and Lastname like '{1}%'.format(self.QSText[0],
925                 self.QSText[1])
926     self.TableWidget.initTable()
927
928
929
930
931
932
933
934
935     if self.SearchDatabase.Valid == True:
936         if self.SearchDatabase.Table != None:
```

```
937     try:
938         if self.SearchDatabase.Table == "Book":
939             for count in range(0, len(self.SearchDatabase.Results) +1):
940                 try:#using the results to fetch the correct data
941                     if count == 0:
942                         self.SearchTable.sql = "select * from {} where
943                             ISBN = '{}'.format(self.SearchDatabase.Table,
944                             list(self.SearchDatabase.Results[count])[2])
945                     else:
946                         self.SearchTable.sql += " or ISBN =
947                             '{}'".format(list(self.SearchDatabase.Results[count])[2])
948             self.SearchTable.initTable()
949         except IndexError:
950             self.SearchTable.setHorizontalHeaderItem(1,
951                 QTableWidgetItem("Author"))
952             with sqlite3.connect("PP.db") as db:
953                 cursor = db.cursor() #fetching firstname and
954                     lastname using foreign key
955                 for count2 in range(0,
956                     self.SearchTable.rowCount()):
957                         cursor.execute("select Firstname, Lastname,
958                             Book.AuthorID, Customer.AuthorID from
959                             Customer, Book where Book.ISBN = {} and
960                             Customer.AuthorID =
961                             Book.AuthorID".format(list(self.SearchDatabase.Results[count2])[2]))
962                         self.Name = list(cursor.fetchone())
963                         self.Name = "{} , {}".format(self.Name[1],
964                             self.Name[0])
```

```
954                     self.SearchTable.setItem(count2, 1,
955                                         QTableWidgetItem(self.Name))
956             db.commit()
957
958         if self.SearchDatabase.Table in ["RoyaltyItems",
959                                         "BookInvoiceItems"]:
960             index = 4
961         else:
962             index = 1
963
964         if self.SearchDatabase.Table == "Customer":
965             for count in range(0, len(self.SearchDatabase.Results)+1):
966                 try: #using the results to fetch the correct data
967                     if count == 0:
968                         self.SearchTable.sql = "select * from Customer
969                         where AuthorID =
970                             '{}'.format(list(self.SearchDatabase.Results[count])[0])
971
972                     else:
973                         self.SearchTable.sql += " or AuthorID =
974                             '{}'.format(list(self.SearchDatabase.Results[count])[0])
975             self.SearchTable.initTable()

except IndexError:
    self.SearchTable.initTable()
```



```
992
993
994
995
996
997
249 998
999
1000
1001
1002
1003
1004
    where Customer.AuthorID = {1} and
    {0}.AuthorID =
    Customer.AuthorID".format(self.SearchDatabase.Table,
    list(self.SearchDatabase.Results[count2])[0])
    self.Name = list(cursor.fetchone())
    self.Name = "{} , {}".format(self.Name[1],
        self.Name[0])
    self.SearchTable.setItem(count2, 1,
        QTableWidgetItem(self.Name))

elif self.SearchDatabase.Table in ["RoyaltyItems",
    "BookInvoiceItems"]:
    self.SearchTable.setHorizontalHeaderItem(1,
        QTableWidgetItem("Author"))
    self.SearchTable.setHorizontalHeaderItem(2,
        QTableWidgetItem("Book Title"))
if self.SearchDatabase.Table == "RoyaltyItems":
    self.IDType = "Royalties"
else:
    self.IDType = "BookInvoice"
for count2 in range(0,
    self.SearchTable.rowCount()):
    cursor.execute("select BookTitle ,
        Book.ISBN, {0}.ISBN, {1}.AuthorID ,
        Firstname, LastName from Book, {0},
        {1}, Customer where Book.ISBN = {2} and
        {0}.ISBN = Book.ISBN and {1}.{1}ID =
        {0}.{1}ID and Customer.AuthorID =
```

```
1005                                         {1}.AuthorID".format(self.SearchDatabase.Table),
1006                                         self.IDType,
1007                                         list(self.SearchDatabase.Results[count2])[2])
1008                                         self.Title = cursor.fetchone()
1009                                         self.Name = "{} , {}".format(self.Title[5],
1010                                         self.Title[4])
1011                                         self.Title =
1012                                         "{}".format(list(self.Title)[0])
1013                                         self.SearchTable.setItem(count2, 2,
1014                                         QTableWidgetItem(self.Title))
1015                                         self.SearchTable.setItem(count2, 1,
1016                                         QTableWidgetItem(self.Name))
1017
1018 elif self.SearchDatabase.Table == "PubInvoice":
1019     self.SearchTable.setHorizontalHeaderItem(1,
1020         QTableWidgetItem("Book Title"))
1021     self.SearchTable.setHorizontalHeaderItem(2,
1022         QTableWidgetItem("Author"))
1023
1024 for count2 in range(0,
1025     self.SearchTable.rowCount()):
1026     cursor.execute("select Firstname ,
1027                     Lastname, BookTitle, {0}.AuthorID,
1028                     Customer.AuthorID, Book.ISBN, {0}.ISBN
1029                     from Customer, {0}, Book where
1030                     Customer.AuthorID = {1} and
1031                     {0}.AuthorID = Customer.AuthorID and
1032                     Book.AuthorID = Customer.AuthorID and
1033                     {0}.ISBN =
1034                     Book.ISBN".format(self.SearchDatabase.Table,
```

```
1015                                         list(self.SearchDatabase.Results[count2])[0]
1016                                         self.Name = list(cursor.fetchone())
1017                                         self.Title = "{}".format(self.Name[2])
1018                                         self.Name = "{} , {}".format(self.Name[1] ,
1019                                                               self.Name[0])
1020                                         self.SearchTable.setItem(count2 , 2 ,
1021                                                               QTableWidgetItem(self.Name))
1022                                         self.SearchTable.setItem(count2 , 1 ,
1023                                                               QTableWidgetItem(self.Title))
1024
1025                                         db.commit()
1026
1027                                         self.StackedLayout.setCurrentIndex(2)
1028                                         self.MenuBar.setVisible(False)
1029
1030                                         except IndexError:
1031                                         self.Msg = QMessageBox()
1032                                         self.Msg.setWindowTitle("No Results Found")
1033                                         self.Msg.setText("No data matches your search")
1034                                         self.Msg.exec_()
1035
1036
1037                                         def keyReleaseEvent(self , QKeyEvent):
1038                                         if self.MainMenuButtons.leQuickSearch.text() == "":
```

```
1039         self.TableWidget.sql = "select * from Customer"
1040         self.TableWidget.initTable()
1041
1042     def ChangeUsernameOrPassword(self):
1043
1044         self.ChangeWhat = dbUsernameOrPassword()
1045         self.ChangeWhat.Selection = None
1046         self.ChangeWhat.ChangeSelection()
1047         if self.ChangeWhat.Selection == "Username":
1048             self.UsernameChange = dbChangeUsername()
1049             self.UsernameChange.Username = self.Username
1050             self.UsernameChange.initChangeUsernameScreen()
1051
1052         elif self.ChangeWhat.Selection == "Password":
1053             self.PasswordChange = dbChangePassword()
1054             self.PasswordChange.Username = self.Username
1055             self.PasswordChange.initChangePasswordScreen()
1056
1057     def LogOut(self):
1058         self.hide()
1059         subprocess.call("LoginDB.py", shell=True)
```

4.10.3 Module 3

```
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.btnExit = QPushButton("Log Out", self)
18    self.btnExit.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.btnExit)
29    self.horizontalTop.addStretch(1)
30    self.horizontalTop.addWidget(self.leQuickSearch)
31    self.horizontalTop.addWidget(self.btnQuickSearch)
```

```
32         self.btnExit = QPushButton("View", self)
33         self.btnExit.setFixedSize(100, 40)
34
35         self.btnSearchdb = QPushButton("Search Database", self)
36         self.btnSearchdb.setFixedSize(100, 40)
37
38         self.btnAddEntry = QPushButton("Add Entry", self)
39         self.btnAddEntry.setFixedSize(100, 40)
40
41         self.btnUpdateEntry = QPushButton("Update Entry", self)
42         self.btnUpdateEntry.setFixedSize(100, 40)
43
44         self.btnRemoveEntry = QPushButton("Remove Entry", self)
45         self.btnRemoveEntry.setFixedSize(100, 40)
46
47         self.btnChangePassword = QPushButton("Change Username/\nPassword", self)
48         self.btnChangePassword.setFixedSize(100, 40)
49
50         self.horizontalBottom = QHBoxLayout()
51
52         self.horizontalBottom.addWidget(self.btnExit)
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.btnChangePassword)
```

4.10.4 Module 4

```
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
26         self.account_menu = self.addMenu("Account")
```

```
27         self.account_menu.addAction(self.change_password)
28         self.account_menu.addAction(self.log_out)
```

4.10.5 Module 5

```
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
```

256

```
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

```
257 1 from PyQt4.QtCore import *
2 2 from PyQt4.QtGui import *
3 3 import sqlite3
4 4 import sys
5 5 import re
6
7 7 class dbAddEntryWindow(QDialog):
8 8     """add entry window dialog"""
9
10 10     def __init__(self):
11 11         super().__init__()
12
13 13     def initAddEntryWindow(self):
14 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
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\.\w+\.\w{3,4}\$")))
32     self.inputList[3].setValidator(QRegExpValidator(QRegExp("^\((\+44\s?\d{4})|(\(?0\d{4}\)\?)\s?\d{3}\)\s?\d{2}"))
33     self.inputList[4].setValidator(QRegExpValidator(QRegExp("[a-zA-Z \d\-\.\.]+")))
34     self.inputList[5].setValidator(QRegExpValidator(QRegExp("^\w\.\w+\@\w\.\w+\.\w{3,4}\$")))
35     #2, 3 & 5 from www.regexlib.com
36     self.CustomerHeaders = ["Firstname", "Lastname", "Email", "Phonenumber",
37                             "Address", "Postcode"]
38     self.AddEntryTable.setHorizontalHeaderLabels(self.CustomerHeaders)
39     self.btnExit = QPushButton("Confirm", self) #buttons
40     self.btnCancel = QPushButton("Cancel", self)
41     self.horizontal = QHBoxLayout()
```

```
41     self.horizontal.addStretch(1)
42     self.horizontal.addWidget(self.btnCancel)
43     self.horizontal.addWidget(self.btnConfirm)
44
45     self.vertical = QVBoxLayout() #vbox layout
46     self.vertical.addWidget(self.AddEntryTable)
47     self.vertical.addStretch(1)
48     self.vertical.addLayout(self.horizontal)
49     self.setLayout(self.vertical)
50
51     self.btnCancel.clicked.connect(self.reject) #reject on clicking cancel
52     self.btnConfirm.clicked.connect(self.AddEntryTodb) #call function after
53         clicking confirm
54     self.exec_()
55
56 def AddEntryTodb(self):
57     #fetching inputs from table
58     self.Valid = True
59     self.Message = "All Fields must be filled."
60     self.Firstname = self.inputList[0].text()
61     self.Lastname = self.inputList[1].text()
62     self.Email = self.inputList[2].text()
63     self.Phonenumber = self.inputList[3].text()
64     self.Address = self.inputList[4].text()
65     self.Postcode = self.inputList[5].text()
66
67     self.input_data = (self.Firstname, self.Lastname, self.Email,
68                       self.Phonenumber, self.Address, self.Postcode)
```

```
67     for count in range(0, len(list(self.input_data))):
68         if list(self.input_data)[count].replace(" ", "") == "":
69             self.Valid = False
70     try:
71         float(self.input_data[4])
72         self.Valid = False
73         self.Message = "Invalid Address"
74     except:
75         pass
76
77     if self.Valid == True:
78         with sqlite3.connect("PP.db") as db:
79             cursor = db.cursor()
80             cursor.execute("PRAGMA foreign_keys = ON")
81             sql = "insert into Customer (FirstName, LastName, Email, PhoneNumber,
82                 Address, Postcode) values (?, ?, ?, ?, ?, ?, ?)"
83             cursor.execute(sql, self.input_data)
84             db.commit()
85             self.accept()
86     else:
87         self.Msg = QMessageBox()
88         self.Msg.setWindowTitle("Invalid Entry")
89         self.Msg.setText(self.Message)
90         self.Msg.exec_()
```

4.10.7 Module 7

```
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.btnExit = QPushButton("Edit", self)
19 #        self.btnConfirm = QPushButton("Confirm", self)
20         self.horizontal = QHBoxLayout()
21         self.horizontal.addWidget(self.btnExit)
22         self.horizontal.addStretch(1)
23         self.horizontal.addWidget(self.btnConfirm)
24         self.vertical = QVBoxLayout()
25         self.vertical.addWidget(self.table)
26         self.vertical.setLayout(self.horizontal)
27         self.setLayout(self.vertical)
28         self.btnExit.clicked.connect(self.Edit)
```

261

```
29         self.btnExit.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
35         window
36         self.btnExit = QPushButton("Confirm", self)
37
38     def Verification(self):
39
40         self.Verify.Msg = "Insert Password to confirm all changes"
41         self.Verify.ConfirmedMsg = "Update successful"
42         self.Verify.VerifyDlg()
43         if self.Verify.ConfirmedDialog.Accepted == True:
44             self.UpdateChanges()
45             self.accept()
46
47     def Edit(self):
48         self.SelectedItem = self.table.currentItem()
49         self.SelectedRow = self.table.currentRow()
50         self.SelectedColumn = self.table.currentColumn()
51         if self.SelectedItem != None:
52             self.EditDlg = dbUpdateEntryWindow()
53             self.EditDlg.setModal(True)
54             self.EditDlg.setWindowTitle("Input Text")
55             self.EditDlg.setFixedSize(210, 120)
56             self.EditDlg.lbl = QLabel("Insert text to save over current entry", self)
```



```
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 = '{}', Phononenumber =
88             '{}', Address = '{}', Postcode = '{}'".format(UL[0], UL[1], UL[2], UL[3],
89             UL[4], UL[5])
90         with sqlite3.connect("PP.db") as db:
91             cursor = db.cursor()
92             cursor.execute("PRAGMA foreign_keys = ON")
93             sql = "update {} set {} where {} = {}".format(self.TableName, self.Update,
94                 self.ID, self.selectedID)
95             cursor.execute(sql)
96             db.commit()
```

264

4.10.8 Module 8

```
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()
```

4.10.9 Module 9

```
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.Editing == 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()
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,
52         11]:
53         if count == 0:
54             self.input = QLineEdit(self)
55             self.input.setValidator(QRegExpValidator(QRegExp("[\w]+")))
56         elif count == 1: #exceptions for book
57             self.input = QLineEdit(self) #new line edit
58             self.input.setReadOnly(True)
59             self.input.setText(self.selectedID)
60         elif count in [3, 9, 11]:
61             self.input = QLineEdit(self)
62             if count == 3:
63                 self.input.setValidator(QIntValidator())
64             else:
65                 self.input.setValidator(QDoubleValidator())
66         elif count in [4, 5, 6, 7]: #exceptions for book where combobox is
67             needed
68             self.input = QComboBox(self) #new combo box
69
70         elif count == 10:
71             self.input = QLineEdit(self)
72             self.input.setReadOnly(True)
73
74     elif self.AddType == "PubInvoice" and count in [0, 1, 2, 3, 4]:
```

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

```
128         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
166             to list for further reference
167             self.gridLayout.addWidget(self.inputList[count], *self.coordinate)
168
169             count += 1
272
170     else: #adding qlabels with the line edits
171         self.DateEntry = False
172         if self.AddType == "Book" and count == 10: #adding date button instead
173             of qlabel
174                 self.DateEntry = True
175         elif self.AddType == "PubInvoice" and count == 2: #data button instead
176             of qlabel
177                 self.DateEntry = True
178         elif self.AddType == "BookInvoice" and count == 1:
179                 self.DateEntry = True
180         elif self.AddType == "Royalties" and count == 1:
181                 self.DateEntry = True
```

```
181
182     else:
183         if str(self.columnHeader) == "PubInvoiceService":
184             self.QLabel = QLabel("Service", self)
185         elif str(self.columnHeader) in ["PubInvoicePayment",
186                                       "BookInvoicePayment", "RoyaltyPayment"]:
187             self.QLabel = QLabel("Payment", self)
188         elif str(self.columnHeader) in ["BookInvoiceDiscount",
189                                       "RoyaltyDiscount"]:
190             self.QLabel = QLabel("Discount", self)
191         elif str(self.columnHeader) in ["BookInvoiceQuantity",
192                                       "RoyaltyQuantity"]:
193             self.QLabel = QLabel("Quantity", self)
194         elif str(self.columnHeader) == "ExcRateFromGBP":
195             self.QLabel = QLabel("f1 = ", self)
196         elif str(self.columnHeader)[-2:] in ["ID", "BN"]:
197             self.QLabel = QLabel(str(self.columnHeader))
198
199     else:
200         self.Label = str(self.columnHeader)
201         self.LabelLength = len(self.Label)
202         self.CamelCase = False
203         for count2 in range(1, self.LabelLength):
204             if self.CamelCase == True:
205                 pass
206             else:
207                 self.CamelCase = True
208             if self.Label[count2].isupper() == True:
```

```
205                     self.String1 = re.sub('(.)([A-Z][a-z]+)', r'\1 \2',
206                                         self.Label[0:count2])
207                     self.String2 = re.sub('(.)([A-Z][a-z]+)', r'\1 \2',
208                                         self.Label[count2:self.LabelLength])
209                     self.tempLabel = "{} {}".format(self.String1,
210                                         self.String2)
211                     self.CamelCase = True
212
213             if self.CamelCase == True:
214                 self.Label = self.tempLabel
215
216             self.QLabel = QLabel(str(self.Label), self)
217             self.QLabelList.append(self.QLabel)
218             self.gridLayout.addWidget(self.QLabelList[count], *self.coordinate)
219
220     if self.DateEntry == True:
221         self.btnDate = QPushButton("Date", self)
222         self.QLabelList.append(self.btnDate)
223         self.gridLayout.addWidget(self.QLabelList[count], *self.coordinate)
224         self.btnDate.clicked.connect(self.CalendarWidget.DisplayCalendar)
225         self.CalendarWidget.btnSelect.clicked.connect(self.getDate)
226
227     if self.AddType == "Book":
228         self.inputList[4].addItem("Large")
229         self.inputList[4].addItem("Small")
230         self.inputList[5].addItem("Hard")
231         self.inputList[5].addItem("Soft")
232         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 =
256                         self.inputList[count].findText(str(self.originalItemList[count]))
257                     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 =
265                     self.inputList[count].findText(str(self.originalItemList[count]))
266                 self.inputList[count].setCurrentIndex(self.originalIndex)
267
268         elif self.AddType in ["BookInvoice", "Royalties"]:
269
270             for count in range(0, 2):
271                 self.inputList[count].setText(str(self.originalItemList[count]))
272         elif self.AddType == "BookInvoiceItems":
273             for count in range(0, 6):
274                 try:
275                     self.inputList[count].setText(str(self.originalItemList[count]))
276                 except:
277                     self.originalIndex =
278                         self.inputList[count].findText(str(self.originalItemList[count]))
279                         self.inputList[count].setCurrentIndex(self.originalIndex)
280         elif self.AddType == "RoyaltyItems":
281             for count in range(0, 8):
282                 self.inputList[count].setText(str(self.originalItemList[count]))
283
284         self.horizontal = QBoxLayout()
```

```
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.setLayout(self.gridLayout)
296             self.vertical.setLayout(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
309                             attempting to add to the database.")
310             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) *
342             int(self.inputList[5].text())
343         self.inputList[6].setText("{0:.2f}".format(self.PrintCost))
344     elif self.inputList[5].text() == "":
345         self.inputList[6].clear()
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
377             len(self.input_data[count]) != 10: #isbn must be 10 or 13 digits
378             self.Valid = False
379             self.ErrorMessage = "Invalid ISBN - Must be 10 or 13 digits."
380             break
381     elif self qlabelList[count].text() == "No Of Pages":
382         if int(self.input_data[count]) > 2000:
383             self.Valid = False
384             self.ErrorMessage = "Invalid Entry - Please check the fields."
385             break
386     elif self qlabelList[count].text() == "Discount": #%'s must be between
387         0 and 100
388         if float(self.input_data[count]) > 100 or
389             float(self.input_data[count]) < 0:
390             self.Valid = False
391             self.ErrorMessage = "Invalid Entry - Please check the fields."
392             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
```

281

4.10.10 Module 10

```
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     self.horizontalTop = QHBoxLayout()
27     self.horizontalTop.addStretch(1)
28     self.horizontalTop.addWidget(self.lblInstruction)
29     self.horizontalTop.addStretch(1)

30     self.horizontalMid1 = QHBoxLayout()
31     self.horizontalMid1.addStretch(1)
32     self.horizontalMid1.addWidget(calendar)
33     self.horizontalMid1.addStretch(1)

34     self.horizontalMid2 = QHBoxLayout()
35     self.horizontalMid2.addStretch(1)
36     self.horizontalMid2.addWidget(self.qle)
37     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.addLayout(self.horizontalTop)
49     self.vertical.addLayout(self.horizontalMid1)
50     self.vertical.addLayout(self.horizontalMid2)
51     self.vertical.addLayout(self.horizontalBottom)
52     self.setLayout(self.vertical)
53
54
283    def DisplayCalendar(self):
55        self.setModal(True)
56        self.btnSelect.clicked.connect(self.accept)
57        self.btnCancel.clicked.connect(self.reject)
58        self.exec_()
59
60
61    def date(self, date):
62        self.qle.setText(date.toString("dd-MM-yyyy"))
```

4.10.11 Module 11

```
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 = QBoxLayout()
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
81                                     try again.", self)
82                     self.lblInvalid.setWordWrap(True)
83                     self.lblInvalid.setAlignment(Qt.AlignHCenter)
84                     self.horizontalInvalid = QHBoxLayout()
85                     self.horizontalInvalid.addStretch(1)
86                     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

```
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",
35                                                 self)
36     self.btnDeleteEntry = QPushButton("Delete \n Entry", self)
37
38 def RoyaltiesItems(self):
39     self.setWindowTitle("View Royalty Items")
40     self.setModal(True)
41     self.setFixedSize(640, 220)
42     self.vertical = QVBoxLayout(self)
43     self.vertical.addWidget(self.table)
44     self.btnCalculate.setFixedSize(100, 40)
45     self.btnUpdateRoyaltyItems.setFixedSize(100, 40)
```

```
45     self.btnExit.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.btnExit)
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.btnExit = 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
72             = {}".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 =
83           {}".format(self.selectedID)
84     cursor.execute(sql)
85     self.ISBNs = list(cursor.fetchall())
86
87     for count2 in range(0, len(self.ISBNs)):
88         if self.currentID != 0:
89             for count in range(0, self.currentID +1):
90                 self.Empty = False
91                 Selection = "BookInvoiceItems.BookInvoiceQuantity ,
92                               BookInvoiceItems.BookInvoiceDiscount ,
93                               BookInvoiceItems.ShippingPrice , Book.Price"
94                 Tables = "BookInvoiceItems , Book"
95                 sql = "select {} from {} where BookInvoiceItems.BookInvoiceID
96                       = {} and BookInvoiceItems.BookInvoiceItemsID = {} and
97                       BookInvoiceItems.ISBN = {} and Book.ISBN =
98                       BookInvoiceItems.ISBN".format(Selection, Tables,
99                     self.selectedID, count+1, list(self.ISBNs[count2])[0])
```

```
93         cursor.execute(sql)
94     try:
95         self.SelectionList = list(cursor.fetchone())
96     except:
97         self.Empty = True
98
99     if self.Empty != True:
100         self.Quantity = self.SelectionList[0]
101         self.Discount = self.SelectionList[1] / 100
102         self.ShippingPrice = self.SelectionList[2]
103         self.Price = self.SelectionList[3]
104
105         self.TempPayment = (self.Quantity * self.Price)
106         self.Discount *= self.TempPayment
107         self.TempPayment -= self.Discount
108         self.TempPayment += self.ShippingPrice
109
110         self.BookInvoicePayment += self.TempPayment
111
112
113 elif self.currentID == 0:
114     self.BookInvoicePayment = None
115     sql = "update BookInvoice set BookInvoicePayment = '{}' where
116         BookInvoiceID = {}".format(self.BookInvoicePayment,
117             self.selectedID)
118     cursor.execute(sql)
119     db.commit()
```

```
119     if self.currentID != 0:
120         self.BookInvoicePayment = "{0:.2f}".format(self.BookInvoicePayment)
121         sql = "update BookInvoice set BookInvoicePayment = '{}' where
122             BookInvoiceID = {}".format(self.BookInvoicePayment, self.selectedID)
123         cursor.execute(sql)
124         db.commit()
125
126     def CalculateRoyaltyItems(self):
127         with sqlite3.connect("PP.db") as db:
128             cursor = db.cursor()
129             cursor.execute("PRAGMA foreign_keys = ON")
130             self.Empty = False
131             self.RoyaltyPayment = 0
132             self.Currency = ""
133             self.IDList = []
134             sql = "select RoyaltyItemsID from RoyaltyItems where RoyaltiesID =
135                 {}".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
144                     iteration limit
145                     self.currentID = self.IDList[count]
```

```
144     sql = "select ISBN from RoyaltyItems where RoyaltiesID =
145         {}".format(self.selectedID)
146     cursor.execute(sql)
147     self.ISBNs = list(cursor.fetchall())
148
149     for count2 in range(0, len(self.ISBNs)):
150         if self.currentID != 0:
151             for count in range(0, self.currentID+1):
152                 self.Empty = False
153                 Selection = "RoyaltyItems.Currency, RoyaltyItems.NetSales,
154                     RoyaltyItems.ExcRateFromGBP, RoyaltyItems.RoyaltyQuantity,
155                     Book.NoOfPages, Book.Size, Book.Cover, Book.Back"
156                 Tables = "RoyaltyItems, Book"
157                 sql = "select {} from {} where RoyaltyItems.RoyaltiesID = {}
158                     and RoyaltyItems.RoyaltyItemsID = {} and RoyaltyItems.ISBN
159                     = {} and Book.ISBN = RoyaltyItems.ISBN".format(Selection,
160                         Tables, self.selectedID, count+1,
161                         list(self.ISBNs[count2])[0])
162                 cursor.execute(sql)
163
164                 try:
165                     self.SelectionList = list(cursor.fetchone())
166                 except:
167                     self.Empty = True
168
169                 if self.Empty != True:
170                     self.Currency = self.SelectionList[0]
171                     self.NetSales = float(self.SelectionList[1])
```

```
165         self.Quantity = self.SelectionList[3]
166         self.NoOfPages = int(self.SelectionList[4])
167         self.Size = self.SelectionList[5]
168         self.Cover = self.SelectionList[6]
169         self.Back = self.SelectionList[7]
170
171     if self.Size == "Large":
172         self.PagePrice = 0.015 * self.NoOfPages
173         if self.Back == "Hard":
174             self.CoverPrice = 5
175         elif self.Back == "Soft":
176             self.CoverPrice = 1
177     elif self.Size == "Small":
178         self.PagePrice = 0.01 * self.NoOfPages
179         if self.Back == "Hard":
180             self.CoverPrice = 4
181         elif self.Back == "Soft":
182             self.CoverPrice = 0.7
183
184     self.PrintCost = (self.PagePrice + self.CoverPrice) *
185             self.Quantity
186
187     self.TempPayment= self.NetSales - self.PrintCost
188     if self.Currency != "£":
189         self.ExcRateFromGBP = float(self.SelectionList[2])
190         self.TempPayment /= self.ExcRateFromGBP
191         self.RoyaltyPayment += self.TempPayment
192
193 elif self.currentID == 0:
```

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

296

4.10.13 Module 13

```
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.setLayout(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.btnViewBookInvoiceItems.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.btnViewBookInvoiceItems)
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.btnViewBookInvoiceItems = 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

```
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)
```

300

```
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.btnDate, 2, 2, Qt.AlignHCenter)
37     self.gridLayout.addWidget(self.leSearch, 2, 3)
38     self.gridLayout.addWidget(self.btnCancel, 3, 2)
39     self.gridLayout.addWidget(self.btnSearch, 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()
51     self.leSearch.hide()
52     self.CalendarWidget.btnExit.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.btnSearch.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.btnDate.show()
88
89     elif self.cbTable.currentText() == "Royalty Items":
90         self.cbCategory.addItem("Quantity")
91         self.cbCategory.addItem("Discount")
92         self.cbCategory.addItem("Currency")
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"
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(" ", "")  
120         == "" or self.Category == "" or 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
```

```
142
143     if self.Table not in ["BookInvoiceItems", "RoyaltyItems"]:
144         if self.Table in ["PubInvoice", "Royalties", "BookInvoice"]:
145             self.sql = "select Customer.AuthorID, {0}ID from Customer, {0}
146                 where (Customer.Firstname like '{1}%' or Customer.Lastname like
147                     '{2}%') and {0}.{3} like '{4}%' and {0}.AuthorID =
148                         Customer.AuthorID".format(self.Table, self.Firstname,
149                                         self.Lastname, self.Category, self.Search)
150
151     elif self.Table == "Book":
152         self.sql = "select Customer.AuthorID, Book.AuthorID, Book.ISBN
153             from Customer, {0} where (Customer.Firstname like '{1}%' or
154                 Customer.Lastname like '{2}%' and {0}.{3} like '{4}%' and
155                     {0}.AuthorID = Customer.AuthorID".format(self.Table,
156                                         self.Firstname, self.Lastname, self.Category, self.Search)
157
158 else:
159     self.sql = "select Customer.AuthorID, Book.AuthorID, Book.ISBN,
160         {0}.ISBN, {0}ID from Customer, Book, {0} where (Customer.Firstname
161             like '{1}%' or Customer.Lastname like '{2}%' and {0}.{3} like
162                 '{4}%' and Customer.AuthorID = Book.AuthorID and Book.ISBN =
163                     {0}.ISBN".format(self.Table, self.Firstname, self.Lastname,
164                                     self.Category, self.Search)
165
166
167 else:
168     if self.Firstname.replace(" ", "") == "" or self.Lastname.replace(" ", "")
169         == "":
170         self.Msg = QMessageBox()
171         self.Msg.setWindowTitle("Invalid Entry")
172         self.Msg.setText("You must fill enter the Firstname and Lastname")
```

```
156         self.Msg.exec_()
157         self.Valid = False
158
159         self.Table = "Customer" #Author table is referred to as 'Customer'
160         self.sql = "select AuthorID from Customer where Firstname like '{0}%' or
161                     Lastname like '{1}%'{format(self.Firstname, self.Lastname)
162         if self.Valid == True:
163             with sqlite3.connect("PP.db") as db:
164                 cursor = db.cursor()
165                 cursor.execute(self.sql)
166                 self.Results = list(cursor.fetchall())
167                 self.accept()
```

306

4.10.15 Module 15

```
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.btnExitFixedSize(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

```
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.btnExit = QPushButton("Change Password", self)
```

```
18     self.btnCancel = QPushButton("Cancel", self)
19     self.btnCancel.clicked.connect(self.reject)
20     self.btnUsername.clicked.connect(self.UsernameSelected)
21     self.btnPassword.clicked.connect(self.PasswordSelected)
22     self.gridLayout = QGridLayout()
23     self.gridLayout.addWidget(self.btnUsername, 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()
```

4.10.17 Module 17

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

309

Imran Rahman

Centre No. 30928

Centre No. 22151

```
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.addLayout(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.leNewUsername.text() == self.leRetype.text():
48
49         if len(self.leNewUsername.text()) < 5:
50             self.Msg = QMessageBox()
51             self.Msg.setWindowTitle("Username")
52             self.Msg.setText("New Username was too short")
53             self.Msg.exec_()
54     else:
55
56         with sqlite3.connect("dbLogin.db") as db:
57             cursor = db.cursor()
58             cursor.execute("select Username from LoginDetails")
59             self.OldUsername = list(cursor.fetchone())[0]
60             if self.leOldUsername.text() == self.OldUsername:
61                 self.NewUsername = self.leNewUsername.text()
```

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

4.10.18 Module 18

```
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.addLayout(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
55     else:
56         with sqlite3.connect("dbLogin.db") as db:
57             cursor = db.cursor()
58             cursor.execute("select Password from LoginDetails")
59             self.OldPassword = list(cursor.fetchone())[0]
60             if self.leOldPassword.text() == self.OldPassword:
61                 self.Password = self.leNewPassword.text()
62                 cursor.execute("update LoginDetails set Password = '{}' where"
63                               "Username = '{}'".format(self.Password, self.Username))
```

```
62         self.Msg = QMessageBox()
63         self.Msg.setWindowTitle("Password")
64         self.Msg.setText("Password was successfully changed")
65         self.Msg.exec_()
66         self.accept()
67
68     else:
69         self.Msg = QMessageBox()
70         self.Msg.setWindowTitle("Password")
71         self.Msg.setText("Old Password was incorrect")
72         self.Msg.exec_()
73
74 else:
75     self.Msg = QMessageBox()
76     self.Msg.setWindowTitle("Password")
77     self.Msg.setText("New Passwords did not match")
    self.Msg.exec_()
```


Chapter 5

User Manual

5.1 Introduction

5.2 Installation

5.2.1 Prerequisite Installation

Installing Python

Installing PyQt

Etc.

5.2.2 System Installation

5.2.3 Running the System

5.3 Tutorial

5.3.1 Introduction

5.3.2 Assumptions

5.3.3 Tutorial Questions

Question 1

Question 2

316

5.3.4 Saving

5.3.5 Limitations

5.4 Error Recovery

Chapter 6

Evaluation

6.1 Customer Requirements

6.1.1 Objective Evaluation

6.2 Effectiveness

6.2.1 Objective Evaluation

6.3 Learnability

6.4 Usability

6.5 Maintainability

6.6 Suggestions for Improvement

6.7 End User Evidence

6.7.1 Questionnaires

6.7.2 Graphs

6.7.3 Written Statements