

COMP 4

Tommy Tham

March 6, 2015

# Contents

<b>1</b>	<b>Analysis</b>	<b>5</b>
1.1	Introduction . . . . .	5
1.1.1	Client Identification . . . . .	5
1.1.2	Define the current system . . . . .	5
1.1.3	Describe the problems . . . . .	6
1.1.4	Section appendix . . . . .	6
1.2	Investigation . . . . .	7
1.2.1	The current system . . . . .	7
1.2.2	The proposed system . . . . .	16
1.3	Objectives . . . . .	20
1.3.1	General Objectives . . . . .	20
1.3.2	Specific Objectives . . . . .	21
1.3.3	Core Objectives . . . . .	22
1.3.4	Other Objectives . . . . .	22
1.4	ER Diagrams and Descriptions . . . . .	23
1.4.1	ER Diagram . . . . .	23
1.4.2	Entity Descriptions . . . . .	24
1.5	Object Analysis . . . . .	24
1.5.1	Object Listing . . . . .	24
1.5.2	Relationship diagrams . . . . .	25
1.5.3	Class definitions . . . . .	26
1.6	Other Abstractions and Graphs . . . . .	27
1.7	Constraints . . . . .	27
1.7.1	Hardware . . . . .	27
1.7.2	Software . . . . .	27
1.7.3	Time . . . . .	27
1.7.4	User Knowledge . . . . .	27
1.7.5	Access restrictions . . . . .	28
1.8	Limitations . . . . .	28
1.8.1	Areas which will not be included in computerisation . . . . .	28
1.8.2	Areas considered for future computerisation . . . . .	28
1.9	Solutions . . . . .	29
1.9.1	Alternative solutions . . . . .	29

1.9.2	Justification of chosen solution . . . . .	29
<b>2</b>	<b>Design</b>	<b>30</b>
2.1	Overall System Design . . . . .	30
2.1.1	Short description of the main parts of the system . . . . .	30
2.1.2	System flowcharts showing an overview of the complete system . . . . .	34
2.2	User Interface Designs . . . . .	41
2.3	Hardware Specification . . . . .	51
2.4	Program Structure . . . . .	53
2.4.1	Top-down design structure charts . . . . .	53
2.4.2	Algorithms in pseudo-code for each data transformation process . . . . .	56
2.4.3	Object Diagrams . . . . .	59
2.4.4	Class Definitions . . . . .	61
2.5	Prototyping . . . . .	62
2.6	Definition of Data Requirements . . . . .	62
2.6.1	Identification of all data input items . . . . .	62
2.6.2	Identification of all data output items . . . . .	63
2.6.3	Explanation of how data output items are generated . . . . .	65
2.6.4	Data Dictionary . . . . .	65
2.6.5	Identification of appropriate storage media . . . . .	67
2.7	Database Design . . . . .	69
2.7.1	Normalisation . . . . .	69
2.7.2	SQL Queries . . . . .	72
2.8	Security and Integrity of the System and Data . . . . .	73
2.8.1	Security and Integrity of Data . . . . .	73
2.8.2	System Security . . . . .	74
2.9	Validation . . . . .	75
2.10	Testing . . . . .	75
2.10.1	Outline Plan . . . . .	76
2.10.2	Detailed Plan . . . . .	76
<b>3</b>	<b>Testing</b>	<b>86</b>
3.1	Test Plan . . . . .	86
3.1.1	Original Outline Plan . . . . .	87
3.1.2	Changes to Outline Plan . . . . .	87
3.1.3	Original Detailed Plan . . . . .	87
3.1.4	Changes to Detailed Plan . . . . .	101
3.2	Test Data . . . . .	113
3.2.1	Original Test Data . . . . .	113
3.2.2	Changes to Test Data . . . . .	114
3.3	Annotated Samples . . . . .	115
3.3.1	Actual Results . . . . .	115
3.3.2	Evidence . . . . .	117
3.4	Evaluation . . . . .	128

3.4.1	Approach to Testing . . . . .	128
3.4.2	Problems Encountered . . . . .	128
3.4.3	Strengths of Testing . . . . .	128
3.4.4	Weaknesses of Testing . . . . .	128
3.4.5	Reliability of Application . . . . .	128
3.4.6	Robustness of Application . . . . .	129
<b>4</b>	<b>System Maintenance</b>	<b>130</b>
4.1	Environment . . . . .	130
4.1.1	Software . . . . .	130
4.1.2	Usage Explanation . . . . .	130
4.1.3	Features Used . . . . .	132
4.2	System Overview . . . . .	132
4.2.1	System Component . . . . .	132
4.3	Code Structure . . . . .	135
4.3.1	Particular Code Section . . . . .	135
4.3.2	Displaying a table . . . . .	135
4.3.3	Switching central widgets . . . . .	136
4.4	Variable Listing . . . . .	136
4.5	System Evidence . . . . .	139
4.5.1	User Interface . . . . .	139
4.5.2	ER Diagram . . . . .	162
4.5.3	Database Table Views . . . . .	163
4.5.4	Database SQL . . . . .	169
4.5.5	SQL Queries . . . . .	170
4.6	Testing . . . . .	175
4.6.1	Summary of Results . . . . .	175
4.6.2	Known Issues . . . . .	175
4.7	Code Explanations . . . . .	175
4.7.1	Difficult Sections . . . . .	175
4.7.2	Select function (section 4.10.4) . . . . .	176
4.7.3	Creating the combo box (section 4.10.4) . . . . .	177
4.7.4	Self-created Algorithms . . . . .	179
4.8	Settings . . . . .	183
4.9	Acknowledgements . . . . .	183
4.10	Code Listing . . . . .	184
4.10.1	add_booking.py . . . . .	185
4.10.2	add_item_to_menu.py . . . . .	191
4.10.3	add_item_to_order.py . . . . .	195
4.10.4	assign_table_customer.py . . . . .	200
4.10.5	cascade_style_sheet.py . . . . .	208
4.10.6	delete_booking.py . . . . .	212
4.10.7	delete_item_off_menu.py . . . . .	215
4.10.8	delete_item_off_order.py . . . . .	219
4.10.9	main_window.py . . . . .	225
4.10.10	manage_booking.py . . . . .	245

4.10.11	manage_order.py . . . . .	247
4.10.12	new_create_tables_cli.py . . . . .	255
4.10.13	print_invoice.py . . . . .	260
4.10.14	radio_button_widget_class.py . . . . .	266
4.10.15	search_order.py . . . . .	268
4.10.16	table_display.py . . . . .	272
4.10.17	update_booking.py . . . . .	275
4.10.18	update_item_price.py . . . . .	282
<b>5</b>	<b>User Manual</b>	<b>285</b>
5.1	Introduction . . . . .	286
5.2	Installation . . . . .	286
5.2.1	Prerequisite Installation . . . . .	286
5.2.2	System Installation . . . . .	286
5.2.3	Running the System . . . . .	286
5.3	Tutorial . . . . .	286
5.3.1	Introduction . . . . .	286
5.3.2	Assumptions . . . . .	286
5.3.3	Tutorial Questions . . . . .	286
5.3.4	Saving . . . . .	286
5.3.5	Limitations . . . . .	286
5.4	Error Recovery . . . . .	286
5.4.1	Error 1 . . . . .	286
5.4.2	Error 2 . . . . .	286
5.5	System Recovery . . . . .	286
5.5.1	Backing-up Data . . . . .	286
5.5.2	Restoring Data . . . . .	286
<b>6</b>	<b>Evaluation</b>	<b>287</b>
6.1	Customer Requirements . . . . .	288
6.1.1	Objective Evaluation . . . . .	288
6.2	Effectiveness . . . . .	288
6.2.1	Objective Evaluation . . . . .	288
6.3	Learnability . . . . .	288
6.4	Usability . . . . .	288
6.5	Maintainability . . . . .	288
6.6	Suggestions for Improvement . . . . .	288
6.7	End User Evidence . . . . .	288
6.7.1	Questionnaires . . . . .	288
6.7.2	Graphs . . . . .	288
6.7.3	Written Statements . . . . .	288

# Chapter 1

## Analysis

### 1.1 Introduction

#### 1.1.1 Client Identification

My client is Linh Tham, the owner of Linh's Restaurant. Linh's Restaurant is a family run Chinese restaurant that is situated in small village called Fordham in Cambridgeshire. Linh works 'outside' usually on her own where she carries out many roles such as taking phone calls for orders/bookings, serving customers and calculating the bills. Outside is referred as the place where the the serving takes place. When times are busy, relatives come and help out at Linh's Restaurant.

Linh would like a more computerized system to be more efficient as manually transferring order details to the invoice book can be time consuming. In addition, when it is busy, using time more efficiently is definitely going to give a better service to the customers and would make it less stressful for Linh. Linh has basic knowledge on how to use a computer such as surfing the internet, checking emails and streaming videos.

#### 1.1.2 Define the current system

Customers come in and get seated according to the number of people. Menus are given and the customers are asked what they would like to drink and what dishes they would like if they are ready. The dishes and drinks are recorded on separate papers. The top copy of the ordering pad, where the dish order is recorded, is given to the chefs where they cook the dishes and one is kept for outside. Once the dishes are served, the customers are checked upon to see if there are any problems occasionally, and once the customers are satisfied and

finish with their meal, they ask for the bill. The recorded order is then copied on to an invoice where the price is calculated for their meal. One invoice is kept and one is given to the customers. The meal is then paid and the customers leave.

The current system is paper based. A diary is used for bookings in which customers can book a table over the phone. A name, number of people on the table, a date and time are recorded in the diary. Orders are taken upfront which is recorded on an ordering pad where one copy is handed over to the kitchen and one is kept to refer to and to transfer order details onto an invoice form.

### **1.1.3 Describe the problems**

When times are busy there could be confusion between on what has been ordered by what table. Also, having to rewrite the order into the invoice book takes time, this is a problem. Furthermore, any inexperienced workers will have to keep referring to the menu when taking orders or calculating the total bill to check if the dish is on the menu and the prices for each dish. This can also lead to a problem where the total price calculated is wrong. Additionally, recorded orders can go missing but that would only happen if the recorded order drops on the floor and no one realises it, this is something that is very unlikely to happen.

### **1.1.4 Section appendix**

#### **Interview with Linh Tham**

##### **What is the current system?**

LT : I ask the customers what they would like to eat and drink and record it on an ordering pad, I then take top copy to the kitchen and keep the second copy for myself so I can refer to who ordered what. Once the customers has finished eating and ready to pay, I transfer all the details from the ordering pad on to the invoice form such as the drinks and dishes with the prices of each. I give a copy of the invoice to the customer and keep one for myself.

##### **How are the second copies of the order and invoice created?**

LT : Because of how thin the paper is on the ordering pad , writing things down marks down what I write on the second copy. However, the second copies are hard to read because the ink from the pen isn't exactly transferred.

##### **What are the problems with the current way of doing things?**

LT : Doing it manually is very time consuming as it takes one person just to rewrite everything on the invoice book. If that one person would be able to finish quicker, that person could help out which would benefit us.

**What data or information is recorded in the current system?**

LT : Food items, drinks, total price and the date of an order.

**What are the benefits of the current system?**

LT : As the system is paper based, any power cuts or weather issues, will not affect how we run the restaurant.

**What should the new system be able to do?**

LT : Having a way to look at what tables have ordered what, like a simulator, this will help the restaurant staff to keep track of tables and will reduce confusion. Storing sit down orders would be helpful also.

Having a way to look at what tables have ordered what, like a simulator, this will help the restaurant staff to keep track of tables and will reduce confusion. Storing sit down orders would be helpful also.

**Would you like to store phone call orders?**

LT: No, I would only like to store sit down orders.

**How long would you like to store the information?**

LT: I would like to store the information for 3 months.

## **1.2 Investigation**

### **1.2.1 The current system**

#### **Data sources and destinations**

There are two main data sources in the current system, the menu and the customer. The menu contains foods and drinks the customer can choose from, the restaurant staff takes the order and then writes down the drinks and dishes onto separate ordering pads without prices. The details of the order is copied onto the invoice including prices and the date once the customer is finished. Each dish ordered is recorded on the invoice however, each drink ordered isn't and so the total price of drinks ordered is recorded instead and referred as 'Drinks' on the invoice form. Additionally, the number of people on the table isn't recorded on the invoice.



Source	Data	Example Data	Destination
Menu	Drink and dishes with prices	Orange Juice £0.70 Special fried rice £3.70	Customer
Customer	Drink	Bottled water	Restaurant staff
Customer	Dish	Wonton soup	Restaurant staff
Restaurant staff	Drink ordered by customer, table number	Bottled water Sprite Table No. 3	Ordering pad 1
Restaurant staff	Dish ordered by customer, date of order, number of people, table number	Wonton soup Special fried rice 30/9/14 Covers 2 Table No. 3	Ordering pad 2
Ordering pad 1	Total price of drinks - each drink is not specified on invoice, table number	(£0.60+£0.70) Total £1.30 Table No. 3	Invoice pad
Ordering pad 2	Dishes ordered by customer including price of each dish, table number	Wonton soup £1.80 Special fried rice £3.70 Table No. 3	Invoice pad
Restaurant staff	Total price of order	Total price £6.8	Invoice pad
Invoice pad	Copy of invoice	Wonton soup £1.70 Special fried rice £3.70 Drinks £1.30 Total price £6.8 Date 30/9/14 Table No. 3	Customer

**Algorithms**

---

**Algorithm 1** Taking an order

---

```
1: OrderTaken  $\leftarrow$  false
2:
3: WHILE notOrderTaken
4:   IF Customer ready to order THEN
5:     Order  $\leftarrow$  USERINPUT
6:     OrderTaken  $\leftarrow$  true
7:   ELSE
8:     Wait
9:   ENDIF
10: ENDWHILE
```

---

---

**Algorithm 2** Generating invoice

---

```
1: InvoiceGenerated  $\leftarrow$  false
2:
3: WHILE notInvoiceGenerated
4:   IF Customer has finished ordering THEN
5:     Copy order details from order pad onto invoice pad
6:     Get prices of each dish and drink ordered from menu
7:     Copy prices onto invoice pad
8:     Calculate total price
9:     Add date
10:    InvoiceGenerated  $\leftarrow$  true
11:   ELSE
12:     Wait for customer to ask for the bill
13:   ENDIF
14: ENDWHILE
```

---

---

**Algorithm 3** Payment

---

```
1: Payment  $\leftarrow$  false
2:
3: WHILE notPayment
4:   IF Customer ask for bill THEN
5:     Give invoice
6:     Payment  $\leftarrow$  USERINPUT
7:     Payment  $\leftarrow$  true
8:   ELSE
9:     Wait
10:   ENDIF
11: ENDWHILE
```

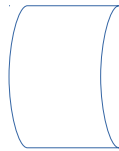
---

**Data flow diagram**Key

Data source/destination



Process



Data store

Figure 1.1: Data flow key

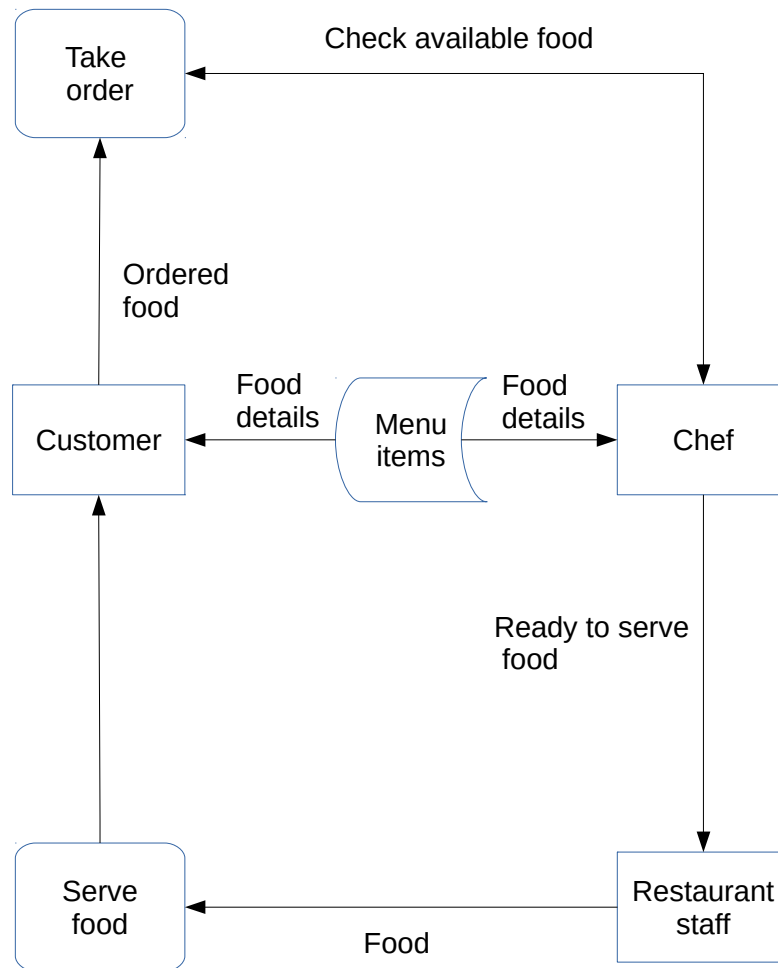


Figure 1.2: Data flow diagram of placing an order

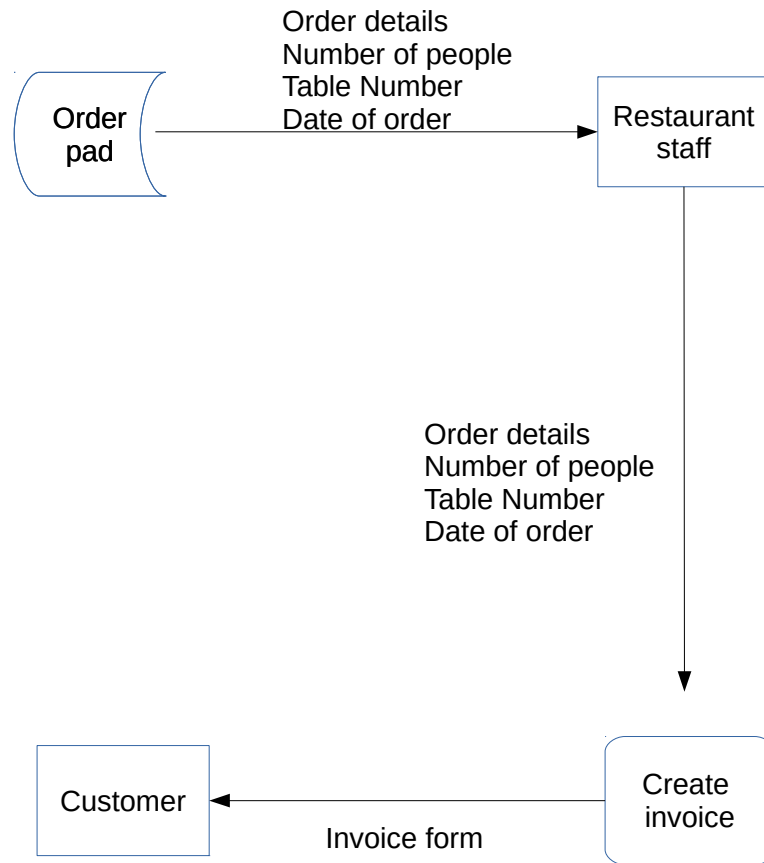


Figure 1.3: Data flow diagram of generating an invoice

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

Drinks are recorded separately from dishes as shown below. The number at the top represents what table number this order is from.

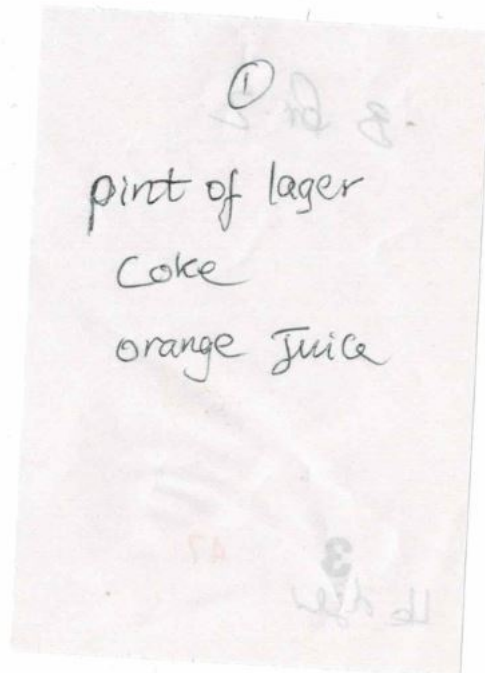


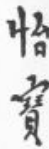
Figure 1.4: Writing down drinks ordered on the drink pad

Below is an example of what the ordering pad looks like when a customer's order has been taken. It provides information about the order such as the table number, how many people is seated, dishes ordered and the date the order has taken place. Two copies of this is made, one is taken to the chefs and one is kept for the waitors. This is an input form.

TABLE No. ①	COVERS 4
spare Ribs	
Butterfly prawn	
Sesame prawn	
$\frac{1}{4}$ Duck	
kung po chicken	
chicken chowmen	
Egg Ric	
chilli beef	
Sweet sour Pork	
DATE 30/9/14	INITIALS
2	27

Figure 1.5: Getting an order from a customer

A picture of an invoice is shown below, the information has been transferred from the ordering pad, as shown above, to the invoice pad. An invoice is created once a table has finished eating and ready to pay. Only the date, description, prices and total price is put on the invoice. This is an output which is given to the customers and another copy of the invoice is kept.



**Links**  
CHINESE  
RESTAURANT  
48A CARTER STREET  
FORDHAM - ELY  
CAMBRIDGESHIRE  
TEL: (01638) 721117

Date 30/9/14 VAT No. 599 4464 72

ITEM	DESCRIPTION	TOTAL
	spare ribs	6.50
	Butterfly prawns	7.10
	Sesame prawn	4.50
	1/2 duck	8.50
	kung po chicken	5.90
	chicken chow mein	6.10
	chilli beef	5.90
	Sweet Sour Pork	5.90
	Egg Rice	3.10
	Drinks	6.90
	Sweets	5.80
	Bottle of wine	7.60
	① TOTAL	73.8

All meal rates are inclusive of VAT  
There is no Service Charge

INVOICE NO.

Figure 1.6: Creating invoice



### 1.2.2 The proposed system

#### Data sources and destinations

In the proposed system, getting an order from the customer is still the same via using restaurant staff and an order pad. The only change in the propose system is transferring the order details onto an invoice.

Source	Data	Example Data	Destination
Menu	Dish and drink	Spare ribs, orange juice	Customer
Customer	Drink ordered	Orange juice	Restaurant staff
Customer	Dish ordered	Wonton soup	Restaurant staff
Restaurant staff	Drink ordered by customer, table number	Orange juice Table No. 1	Ordering pad
Restaurant staff	Dish ordered by customer, table number, number of people, date of order	Wonton soup, Table No. 1, Covers 1 04/09/14	Ordering pad
Proposed system software	Invoice form	04/09/14 Wontop soup £ 1.80 Drinks £0.7 Total price £2.50	Customer

The new part of the system's data sources and destinations is shown below.

Entering the food item onto the software should automatically retrieve its price from the menu database. After a customer has finished with their meal, the simulator saves the Table status ( drinks, dishes, table number and date) to the order history database and creates an invoice form.

Source	Data	Data type	Destination
Restaurant staff	Dish	String	Computer - Table status
Restaurant staff	Drink	String	Computer - Table status
Restaurant staff	TableNumber	Integer	Computer - Table status
Restaurant staff	NumberOfPeople	Integer	Computer - Table status
Restaurant staff	DateOfOrder	Date	Computer - Table status
Computer - Table status	OrderID	Integer	Database - Order records
Computer - Table status	Dish	String	Database - Order records
Computer - Table status	Drink	String	Database - Order records
Computer - Table status	TableNumber	Integer	Database - Order records
Computer - Table status	DateOfOrder	Date	Database - Order records
Computer - Table status	TotalDrinkPrice	Float	Database - Order records
Computer - Table status	TotalPrice (TotalDrinkPrice + each dish)	Float	Database - Order records
Computer - Table status	InvoiceForm	string	InvoiceFolder

**Data flow diagram**

The data flow diagram of placing an order will be the same due to no changes to the way of placing and processing the order.

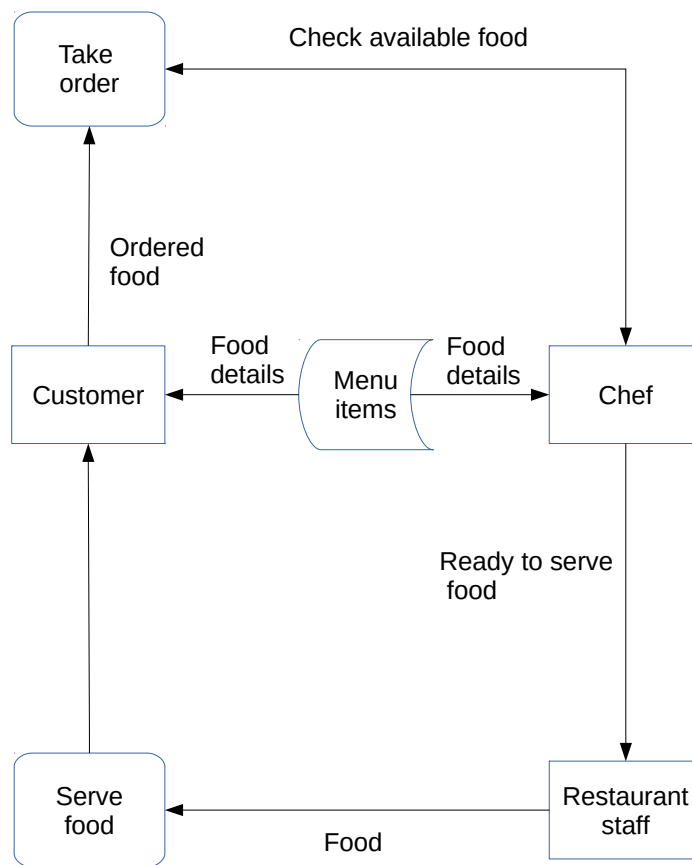


Figure 1.7: A data flow diagram of the proposed system - placing and processing the order

The proposed system will make the restaurant staff input data into the system which will be shown on the application if the user checks what table has ordered what. In addition the inputted data saved in a database once the customer has finished with their meal. Also, invoices will be created through this application.

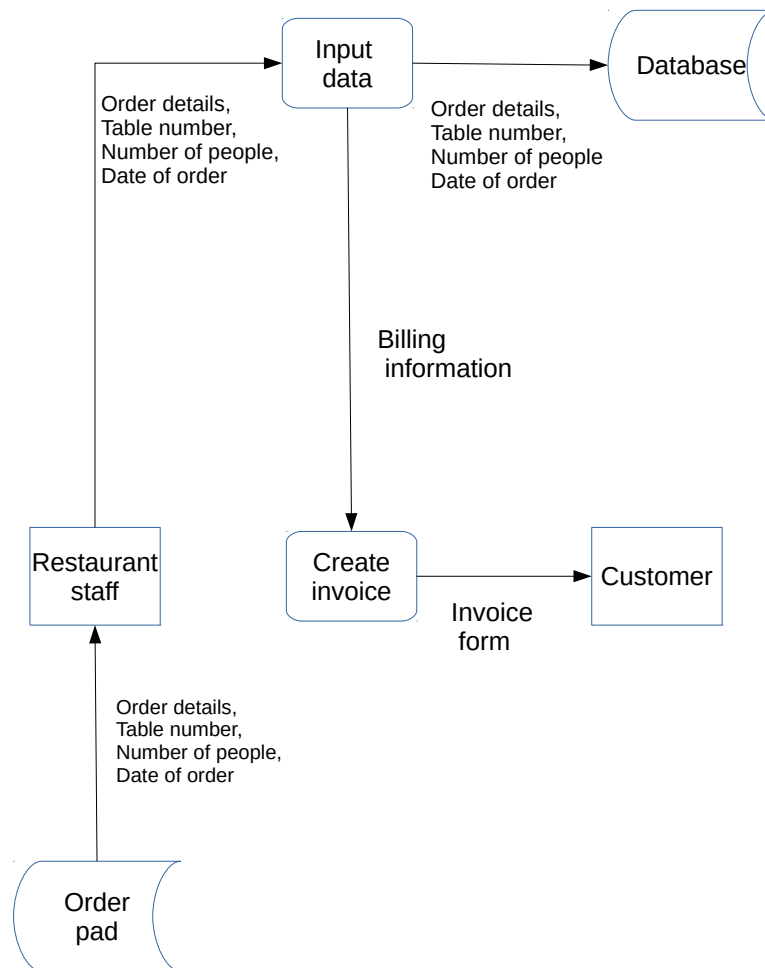


Figure 1.8: Data flow diagram proposed system

**Data dictionary**

Name	Data Type	Length	Validation	Example Data
TableNumber	Integer	1 - 16	Range	13
NumberOfPeople	Integer	1 - 20	Range	4
MenuItem	String	1 - 20 Characters	Length	Spare ribs
ItemQuantity	Integer	1 - 10	Range	4
ItemPrice	Float	0 - 20	Range	3.2
TotalPrice	Float	0 - 500	Range	54.4
DateOfOrder	Date	4 - 6	Format	16/11/14
InvoiceCreated	Boolean		Presence Check	

**Volumetrics**

As an ascii character is 1 byte, there will be 35 bytes for one sitdown order.  $35 \times 30$  (approximately the max sit down orders per day) = 1050 bytes is stored per day. Linh's restaurant is open 6 days a week so  $1050 \times 6 = 6300$  bytes and Linh has stated that she would like to store the information for 3 months so  $6300 \times 13.2$  (weeks) = 83160 bytes will need to be stored.

81360 bytes is equivalent to 79.45 kilobytes ( $81360/1024$ ). 79.45 kb would be needed to store 3 months of information. The software it self will contain pictures which will increase the size by roughly 2MB. Therefore the total space required would be 6MB if the application itself took 4MB without any images ( 2MB + 4MB).

**1.3 Objectives****1.3.1 General Objectives**

- Create a restaurant simulator to track orders
- Simple and clear GUI for user-friendly experience.
- Having the ability to easily modify orders.
- Create a digital invoice after table has finished their meal.
- Storing orders.

### 1.3.2 Specific Objectives

#### Simple and clear GUI

- Having a very simple birds eye view image of the restaurant which is made out of shapes to ease the understanding of where each table is.
- Label table with their corresponding number.
- Table shapes will be big so it won't be hard to click on them but not so big that 16 tables can fit on the GUI.
- Clicking on table will bring up a window which shows the status such as the date and food items ordered with noticeable order modification options.

#### Order alterations

- Have clear Add, Delete and Create invoice buttons.
- When user chooses the add option, have an input box appear where user can type in an ID for a dish/drink or the actual name of the dish/drink.
- Make the input search function not case sensitive.
- When user wants to delete a food item off the list, have clear red X boxes appear next to the name. When red X boxes are clicked on and with confirmation, the item gets deleted.
- Have an up arrow or bottom arrow button just in case a customer orders another food item which is already on the list. The up arrow would increase the quantity of the item by 1 and the down arrow would decrease the item by 1 .
- Clicking on create invoice button will clear the information on the table status and save the digital invoice in a folder.

#### Track orders

- Drinks and dishes will be seperated by columns.
- Clicking on a table will bring up a small window with the list of food items that the table has ordered, formatted like the invoice form shown on page 15. This also includes the date and table number.

#### Invoice creation

- Automatically creating a digital invoice when a customer has finished.
- Calculate total price
- The digital invoice will look very similar to the invoice on page 15.
- Invoice will contain the items ordered, prices of each and total price.

- Have the option to print out invoice.

Storing orders

- When using the clear information button, the information is stored in the database.
- Filtering database for user if searching specific information.
- Have an option to view database.

### 1.3.3 Core Objectives

- Have a working simulator that will have the restaurant layout
- Having clickable tables that will bring up a window showing a digital invoice
- The digital invoice will show the current status such as items ordered, date of order and number of people on the table.
- Application must be able to modify orders
- Application must be able to generate an invoice after table has finished with their meal

### 1.3.4 Other Objectives

- Print invoice function
- Store order data in a database
- Database search functions such as sort and filtering.

## 1.4 ER Diagrams and Descriptions

### 1.4.1 ER Diagram

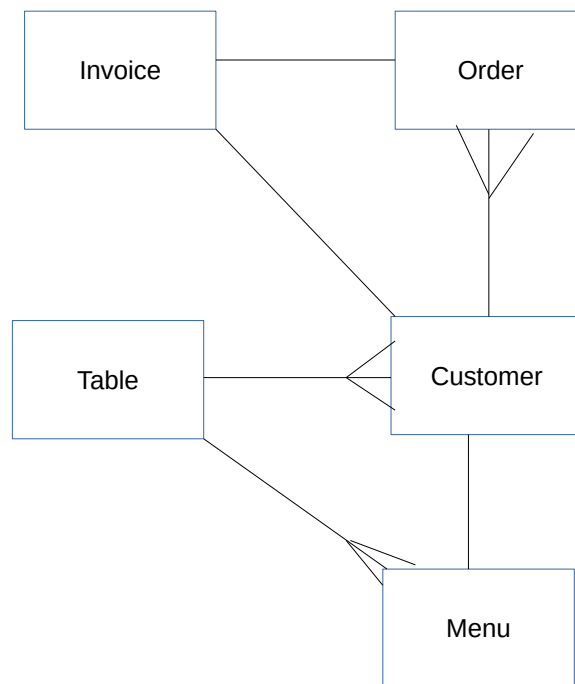


Figure 1.9: E-R Diagram



### 1.4.2 Entity Descriptions

Customer(CustomerID, *TableID*, *OrderID*, NumberOfPeople, Invoice, Date)

Order(OrderID, *CustomerID*, *TableID*, *MenuID*, DishOrdered, DrinkOrdered, Quantity)

Table(TableID, *OrderID*, *CustomerID*, TableNumber)

Menu(MenuID, Dishes, Drinks, DishPrice, DrinkPrice)

Invoice(InvoiceID, *CustomerID*, *OrderID*, TotalDrinkPrice, TotalPrice)

## 1.5 Object Analysis

### 1.5.1 Object Listing

- Customer
- RestaurantStaff
- Dish
- Drink
- Invoice
- Menu

### 1.5.2 Relationship diagrams

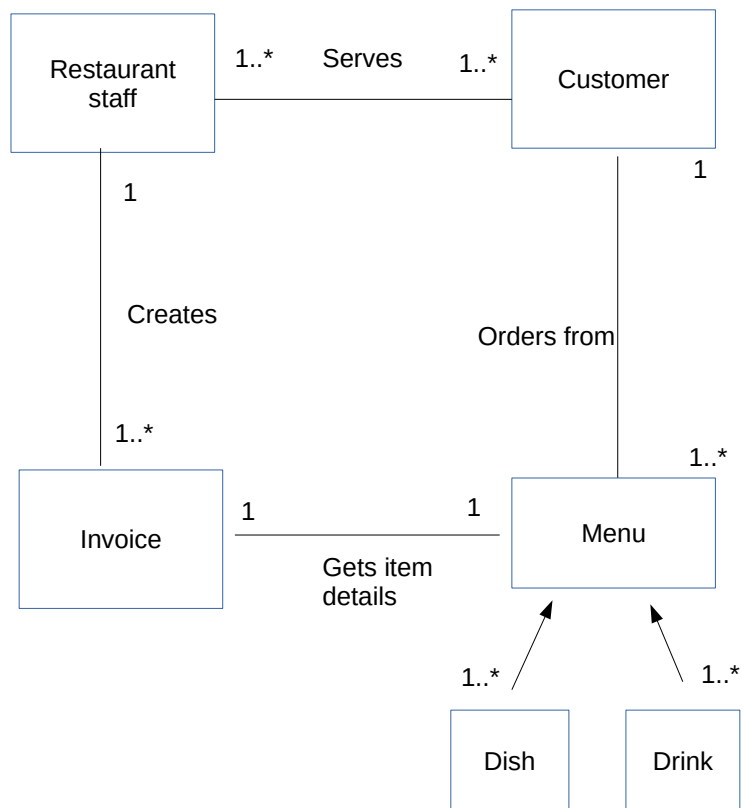


Figure 1.10: Relationship diagram

### 1.5.3 Class definitions

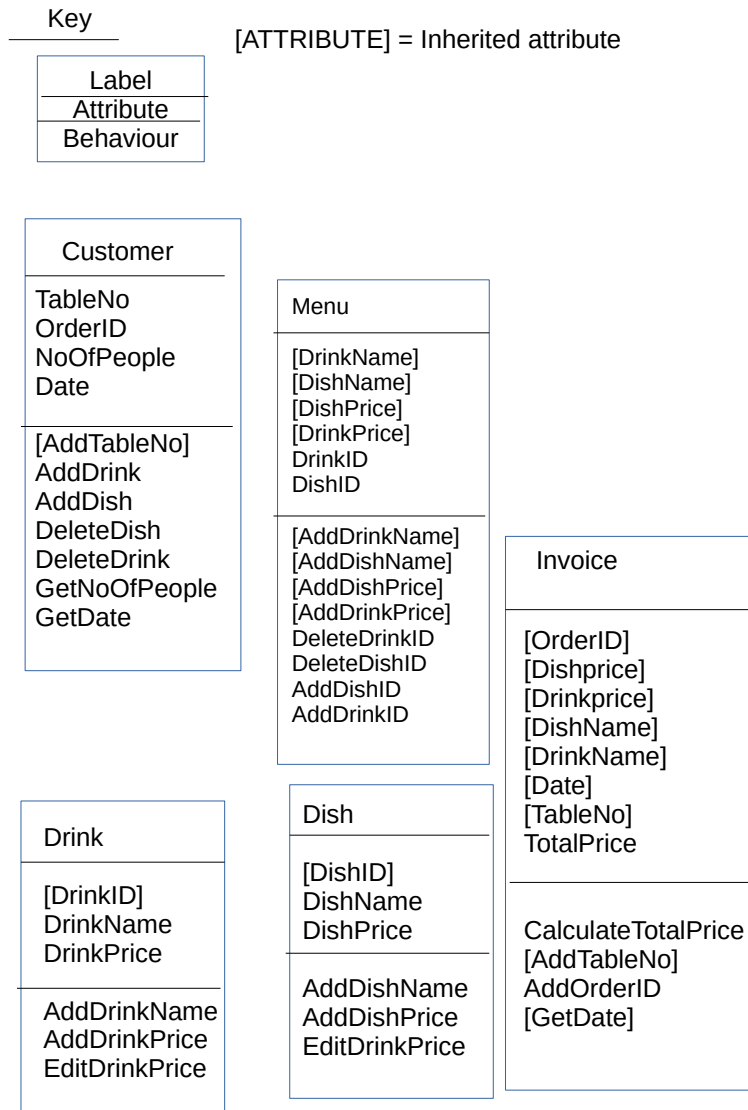


Figure 1.11: Class diagram

## 1.6 Other Abstractions and Graphs

### 1.7 Constraints

#### 1.7.1 Hardware

The current computer specifications is as follows:

- 19" Display
- AMD FX(fm) - 6300 six-core CPU 3.50Hz
- 8GB RAM
- NViDiA GeForce 9600 GT 1GB
- Windows 8.1 64 bit

There shouldn't be any constraints apart from the fact that the new system will have to be designed to fit the 19" screen. Also the position of where the computer will be placed in the restaurant is a limitation.

#### 1.7.2 Software

The current computer uses Windows 8.1 and Linh would prefer it to stay that way as she is familiar with the operating system. This is not a problem as the proposed system will run fine on Windows 8.1. Apart from that, Linh has not stated what software can or cannot be used.

#### 1.7.3 Time

Linh has not set me a deadline for the new system and is in no rush for it to get done. Therefore the deadline will be Friday 27th March 2015 which is the coursework deadline set by my teacher.

#### 1.7.4 User Knowledge

Linh has basic knowledge on how to use a computer such as being able to check emails and simple web surfing. Basic knowledge will not constrain the project as one of the objectives is for the software to be simple and clear.

### **1.7.5 Access restrictions**

All working staff should be able to use this software due to the nature on how the business is run. All waiting staff should be able to carry out the same roles such taking an order, serving and creating an invoice form. However, customers should not be able to access this application at all which could be considered as a constraint. A simple enter password-to-access mechanic could be used as a solution to this.

## **1.8 Limitations**

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

The method of taking orders will not be computerised as it more convenient to just take orders by pad. Using an ordering pad is useful as it is small, light and easy to carry around. Also, the payment system ( receiving money and giving back change) will not be computerised as there are no problems with the current payment system. More problems will likely be created if it was to be computerised such as giving back the correct amount of change and registering the amount of money received.

### **1.8.2 Areas considered for future computerisation**

Tracking bookings for tables can be a feature for later as it could be helpful if the book of table bookings goes missing or if theres no more space to write down bookings. In addition, Linh has not stated that she wanted take aways to be computerised. This could be an additional feature in the future to the program where it creates invoices for take aways.

## 1.9 Solutions

### 1.9.1 Alternative solutions

Solution	Advantages	Disadvantages
Python Desktop Application with a GUI	The design can be changed according to client needs. Not complicated to use. Very low cost. User-friendly and problems with current system will be fixed. Extra features can be implemented.	Application will take up noticeable computer storage. Will take a long time to create GUI application. If there's a power cut then system will not be useable
Touch screen self-order system	Customer has more freedom. Less work for restaurant staff. Problems with current system will be gone.	More hardware and software needed - can be very costly. Technical issues will be hard to fix.
Getting someone to do invoices only	Will solve the main problem with current system. No need for a computer.	Will be hard to find someone who will only do invoices. If business isn't busy then invoice person will be almost useless. Could be more costly in long run.
Redesign current manual system	No cost or very low cost as no computer/software will be needed. Current manual system is simple.	May not be able to fix problems. Will take some time to figure out how to fix problem.

### 1.9.2 Justification of chosen solution

I have chosen Python Desktop Application with a GUI as the solution because of many reasons. One reason is that the touch screen solution will be very costly and customers would have to queue up to use the machine if it gets busy. This will affect how the business will run as many customers do not like to wait. Also, hiring out someone to do invoices will not be efficient as money will be wasted if business is not busy. Furthermore redesigning the current system will take time as Linh would need to figure how fix the problems in the system and also this will most likely not fix most problems with the current system. Therefore I choice Python Desktop Application because it would not need any further hardware, this will not negatively affect customers experience at the restaurant in any way and due to Python being very flexible, the program can always be changed to Linh's wants.

# Chapter 2

## Design

### 2.1 Overall System Design

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

- Restaurant Simulator
  - Core Elements of System
  - General User Interface
  - Adding Item
  - Deleting Item
  - Saving Order Information
  - Managing Bookings
  - Managing Item Menu

##### Core Elements of System

The system will be designed to make it easier to track information about the restaurant for the restaurant staff, information will be displayed on the application. Information tracked down includes order information such as what has been ordered by each table and the information about that table like the number of people, date and time arrived. In addition, booking times will be displayed at the main screen. As well as displaying key information, the system will have features to add/delete/edit information. For example, adding items to an order, deleting irrelevant bookings and editing booking times. The core elements of the system will be based on managing orders and bookings.

##### General User Interface

- Only staff will be able to access this application, so a box will be the first thing that prompts up when the applicaiton is opened. This box will require staff members to enter a password which they have created.
- After entering the correct password, the application will display the layout of restaurant in a birds eye view way. The layout will contain shapes which represent each table, each shape will have the number of table on it.
- Clicking on table will bring up a box with a layout like an invoice such as the one on page 15. This screen will contain the table's status such as what they ordered, date, time, table number, number of people and total price. The main box in the middle will be split in half where the left half will contain the dishes ordered and the right will contain the drinks ordered. At the bottom will be contain the editing features where there will be an Add, Delete and Finish buttons. In addition, there will be a back arrow at the top and once this is clicked, it will return to the main interface with the restaurant layout and save the order information.

#### Adding Item

- The managing order box that pops up when clicked on a table, will have an 'Add' button at the bottom. This button will have the feature to add a menu item to the order.
- When the 'Add' button is clicked, a box will pop up where the user enters the name or item ID and if name or ID is entered correctly, the item will appear on the table status. The menu will be displayed to aid the user.

#### Deleting Item

- The managing order box will have a 'Delete' button located at the bottom. This button will have the feature to delete a menu item off the order
- When the 'Delete' button is clicked, red boxes with an X will appear next to each item ordered. If the red button is clicked, the item will disappear off the order.

#### Saving Order Information

- A 'Finish' button will be located along with the 'Add' and 'Delete' buttons.
- The 'Finish' button will be used once a table has finished eating/ordering. It will save all of the current information about this particular order.
- Information will be the ordered menu items, table number, date, time, number of people and the total price.

#### Managing Bookings

- Any table bookings will be displayed on the main screen
- A button labelled "Bookings" will be at the main screen. A box will appear that will be used to manage bookings.



- Adding and deleting bookings will be available through this box that is used to manage bookings.

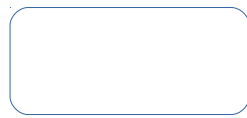
#### Managing Item Menu

- There will be an option to add an item to the menu or delete an item off the menu. This will be accessed at the menu bar.
- The menu bar will have a drop down box containing "Add Item" "Delete Item"
- Adding an item requires the user to input the information required.
- Deleting an item will be done by the user entering either the name of the item or the ID. The menu will be displayed to aid the user.



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

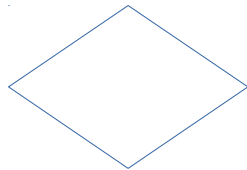
#### Key



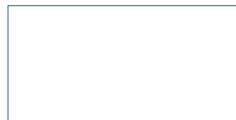
Start/Stop



Input/Output



Decision



Process



Storage

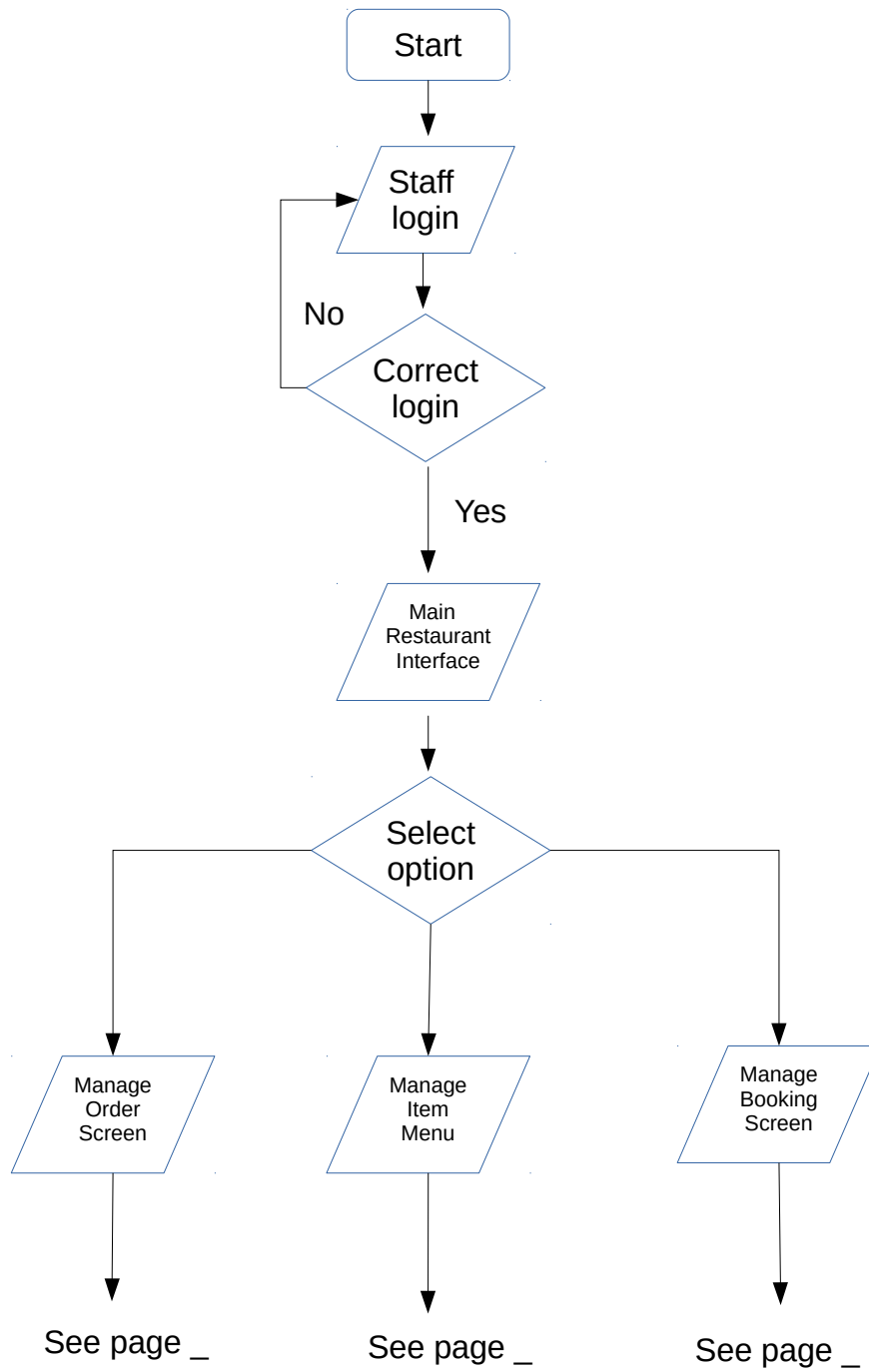


Figure 2.2: Flow chart of system

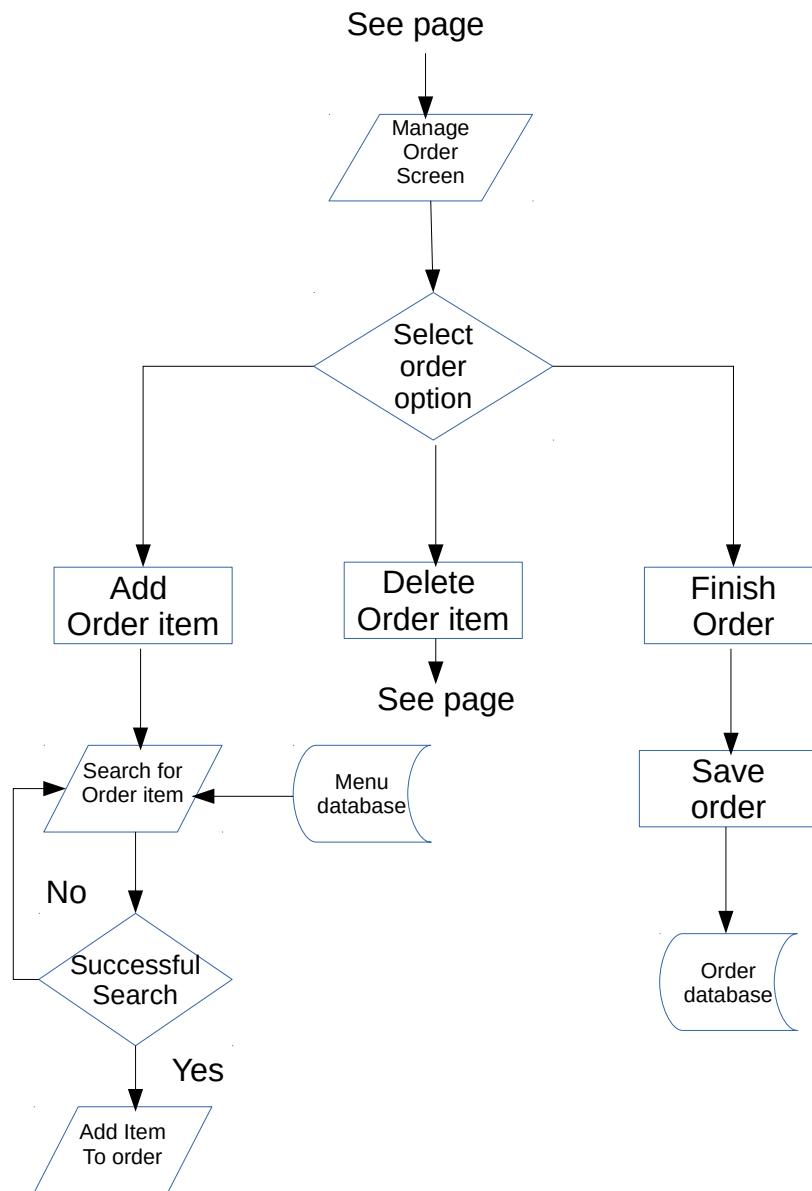


Figure 2.3: Flow chart of order

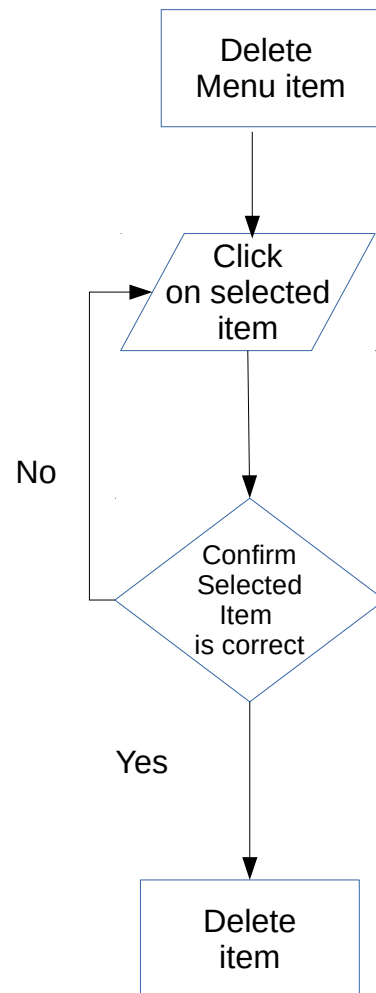


Figure 2.4: Flow chart of deleting an item of an order

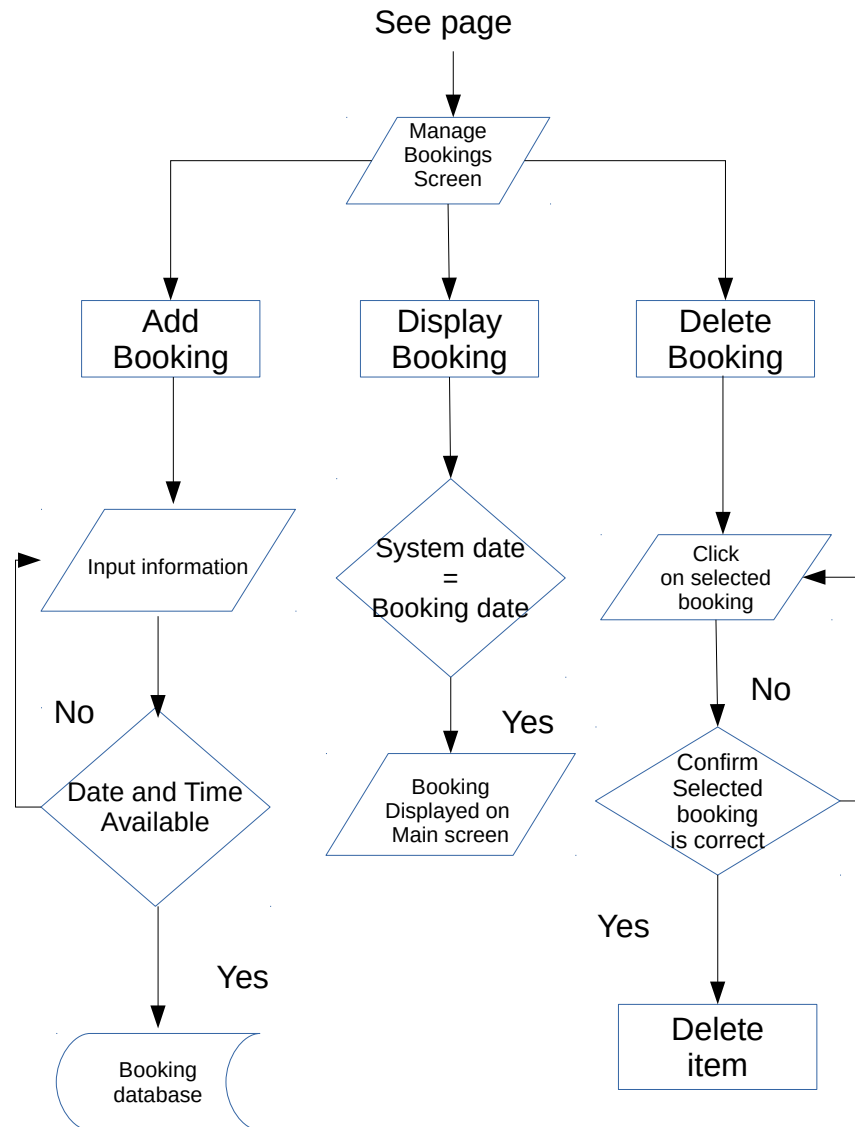


Figure 2.5: Flow chart of bookings

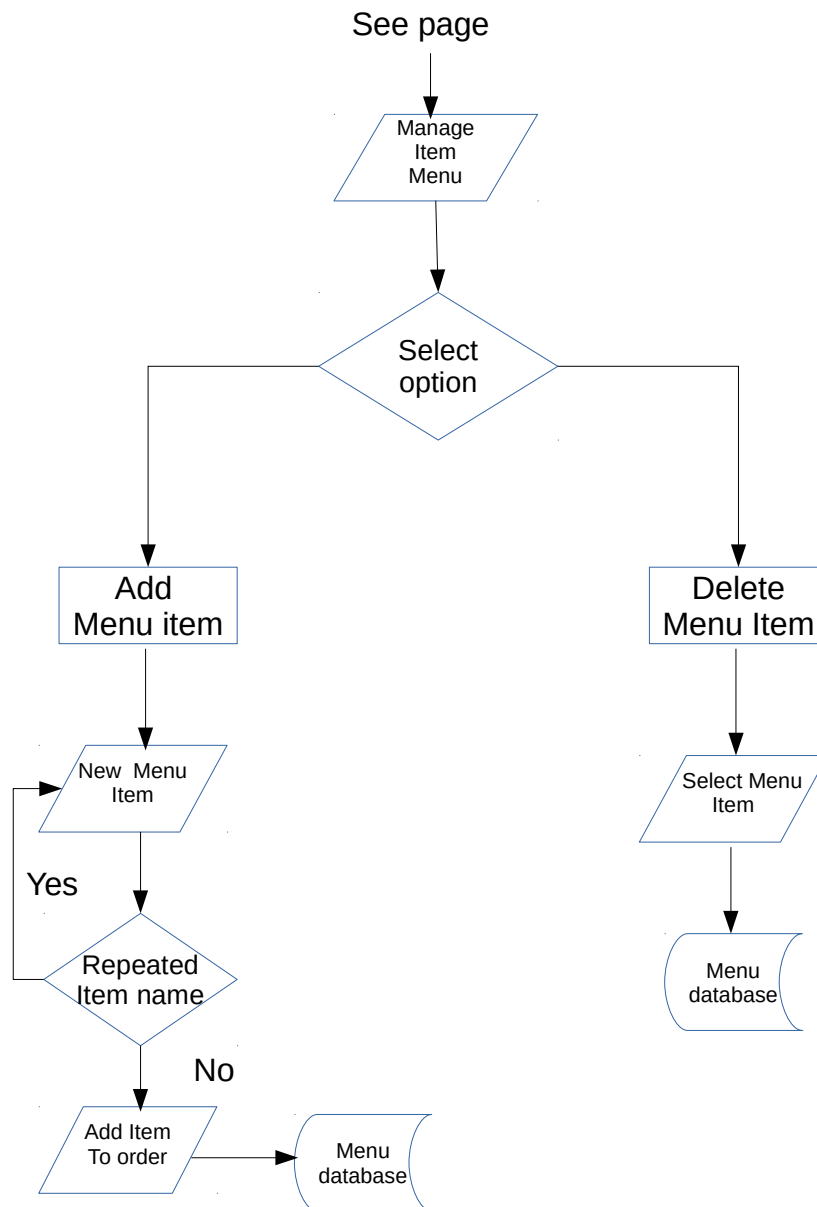


Figure 2.6: Flow chart of adding an item to the menu





## 2.2 User Interface Designs



**Restaurant Simulator**

Please enter the password

This box asking for the password will pop up once the program is opened. The purpose of this is to only allow staff to access this program.



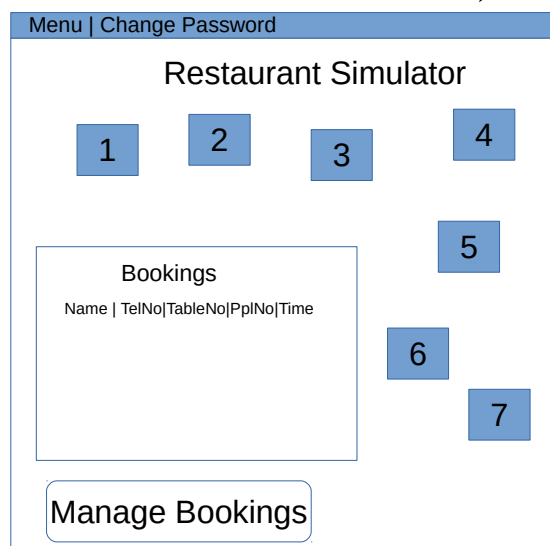
**Restaurant Simulator**

Please enter the password

You have entered the wrong password.  
Please try again.

If the user enters the wrong password then it will inform the user at the bottom left of the box

The password prompt box will disappear once the correct password has been entered. The main program will continue to run.



Menu | Change Password

**Restaurant Simulator**

1 2 3 4

5

6

7

**Bookings**  
Name | TelNo|TableNo|PplNo|Time

Manage Bookings

Figure 2.7: Password Prompt

The diagram shows a horizontal menu bar with two items: 'Menu' and 'Change Password'. Below the 'Menu' item, a dropdown menu is displayed, containing two options: 'Add Item' and 'Delete Item'. An arrow points from the 'Add Item' option to the 'Add item to menu' dialog box.

This is the menu bar that will be on the window.

Clicking on Menu will bring down a drop down box containing 'Add Item' and 'Delete Item'.

This is the box that will pop up once clicked on 'Add Item'.

The user will input information and once completed, the item will be added to the database.

Clicking Cancel will close the box.

The dialog box is titled 'Add item to menu'. It contains three input fields: 'Item Name', 'Item Price', and 'Item Type'. At the bottom, there are two buttons: 'Add Item' and 'Cancel'.

The diagram shows a horizontal menu bar with two items: 'Menu' and 'Change Password'. Below the 'Menu' item, a dropdown menu is displayed, containing two options: 'Add Item' and 'Delete Item'. An arrow points from the 'Delete Item' option to the 'Delete Item off the menu' dialog box.

Clicking on 'Delete Item' will bring up this window. The menu database will be displayed for the user to see which item(s) they would like to delete.

The dialog box is titled 'Delete Item off the menu'. It contains a table with the following header: 'ID | Menu Item | Price | Type'. Below the table, it says 'Delete item by entering:'. At the bottom, there are three buttons: 'Item ID', 'Item Name', and 'Cancel'.

Carried on over the next page

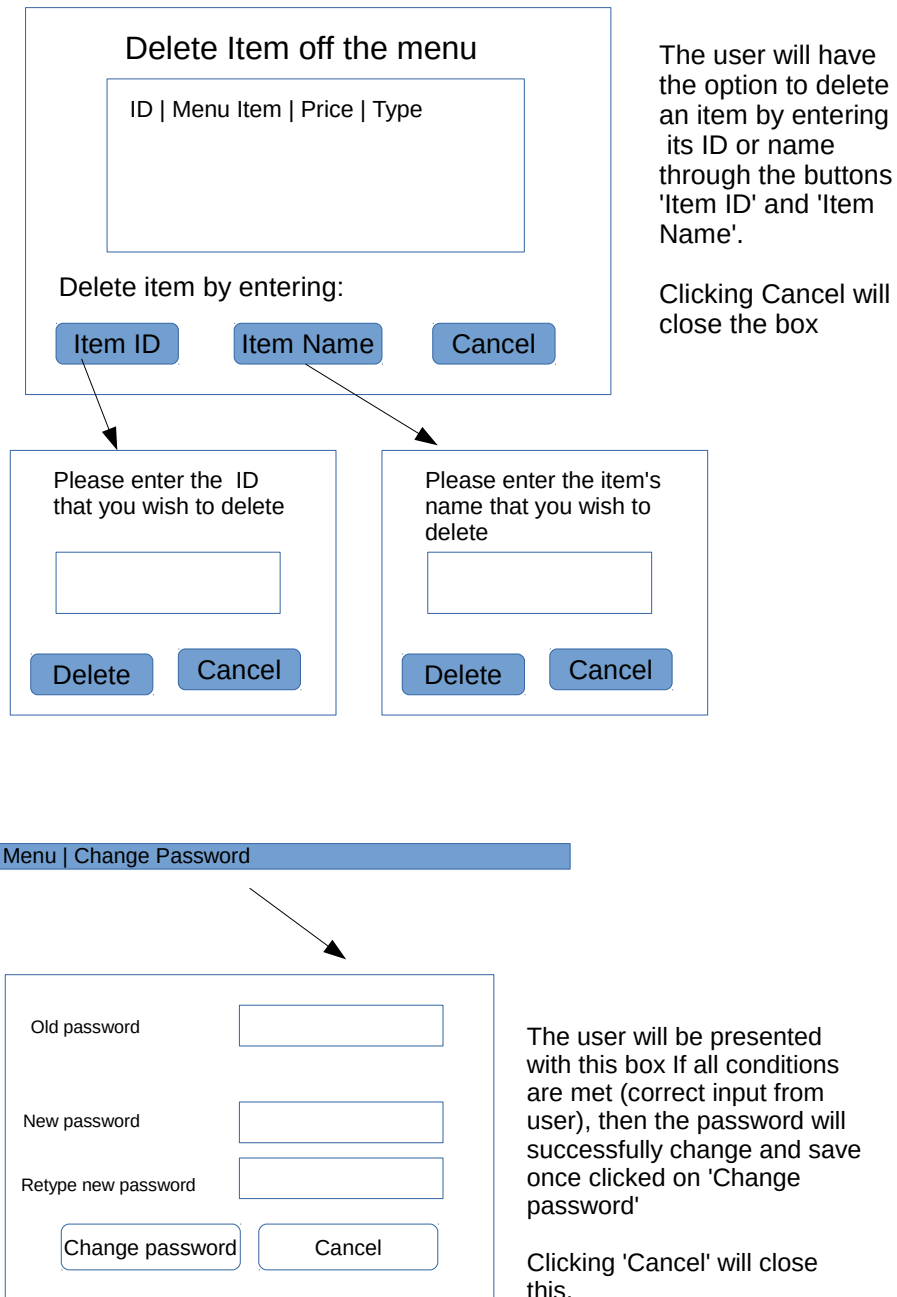
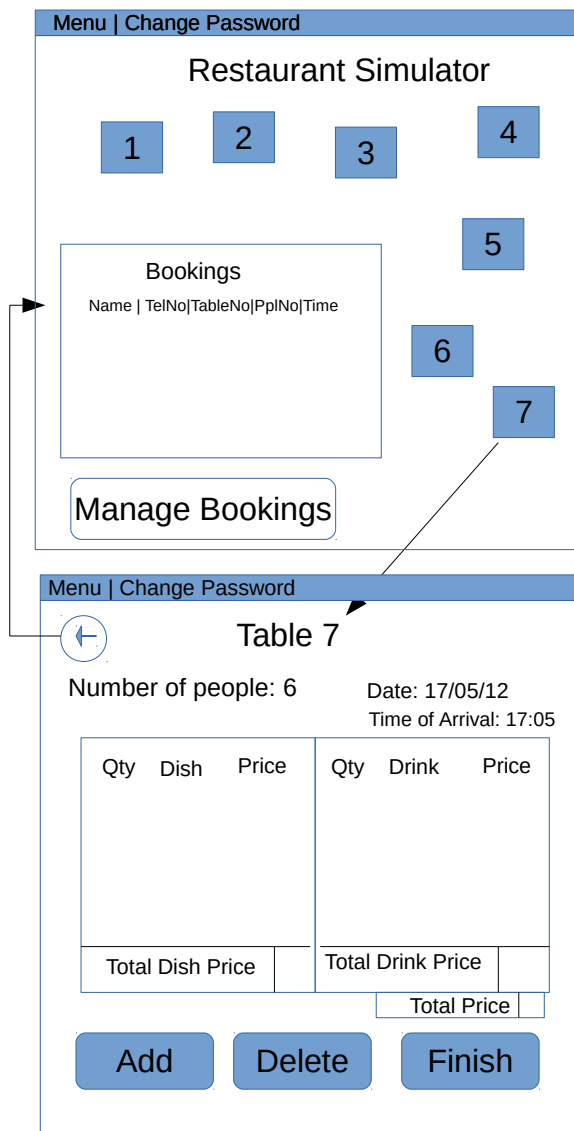


Figure 2.9: Explaining Menu Bar



This is the main screen. Squares with numbers will represent the tables in the restaurant, the number represents the table number. There will be 16 tables and the squares will be large so it will be easy to click on.

Bookings will be shown inside the left box and the button 'Manage Bookings' will be used to add/delete bookings.

An order can be checked by clicking on the respective table.

This is the order screen that is displayed once clicked on a table. The order information such as the items ordered, number of people and prices will be displayed according to the table number.

The 'Add' button will be used to add menu items to the order, the item added will be displayed in the appropriate box.

The 'Delete' button will be used to delete any items of the order.

Selecting 'Finish' will result in the information to be saved into the database and clear any information on the order screen apart from the table number.

The back arrow at the top left will save the current information about the order and return the user to the main screen

However, if a table hasn't been occupied then the 'Assign customers to table' box will appear once clicked on the unoccupied table.

User inputs the number of people that will be on the table

The user will not input the time of arrival or date of arrival as these will be from the system time and date.

Selecting 'Create' will transfer all the details from the 'Assign customer' box such as number of people, date and time of arrival to the order screen.

Selecting 'Cancel' will close the 'Assign customer' box.

Clicking on Add will display a window which shows the menu database so the user can see what items are on the menu.

The user will have the option to add an item by entering its ID or name through the buttons 'Item ID' and 'Item Name'.

Clicking Cancel will close the box

Please enter the item's name that you wish to add

Add

Cancel

### Add Item to order

ID | Menu Item | Price | Type

Add item by entering:

Item ID

Item Name

Cancel

Please enter the ID that you wish to add

14

Add

Cancel

Please enter the item's name that you wish to add

Coke

Add

Cancel

Menu | Change Password

### Table 7

Number of people: 6      Date: 17/05/12  
Time of Arrival: 17:05

Qty	Dish	Price	Qty	Drink	Price
1	Spare ribs	4.20	1	Coke	0.70
Total Dish Price		4.20	Total Drink Price		0.70
				Total Price	

Add

Delete

Finish

ID 14 has been added to the order which is spare ribs.

Using the Item name 'Coke', the item has been added to the order.

Adding items to the order obtains the prices of these items and displays it. It also calculates the totals for dishes and drinks thus far.

Figure 2.13: Add Item



Menu | Change Password

**Table 7**

Number of people: 6      Date: 17/05/12  
Time of Arrival: 17:05

Qty	Dish	Price	Qty	Drink	Price
1	Spare ribs	4.20			
Total Dish Price		4.20	Total Drink Price		
				Total Price	

**Add   Delete   Finish**

Menu | Change Password

**Table 7**

Number of people: 6      Date: 17/05/12  
Time of Arrival: 17:05

Qty	Dish	Price	Qty	Drink	Price
1	Spare ribs	4.20 <b>X</b>			
Total Dish Price		4.20	Total Drink Price		
				Total Price	

**Add   Delete   Finish**

Are you sure you want to delete Spare ribs off the order?

**Yes   No**

Selecting 'Delete' will change the colour of the button to make the user aware that it is in the 'Delete' mode.

In 'Delete' mode, red boxes with an 'x' will appear next to each order item. Clicking on it will bring up a confirmation box. If the quantity is more than 1 then it will reduce the quantity by 1.

After selecting 'Yes', on the confirmation box, the item chosen will be removed off the order. Clicking 'Yes' or 'No' on the confirmation box will make the user go out of the Delete mode.

So if the user wanted to delete another item, the user would have to click the 'Delete' button again.

On the other hand, if the user accidentally clicked on the 'Delete' button, the user could just click on the 'Delete' button again to get out of the Delete mode.

Figure 2.14: Delete Item

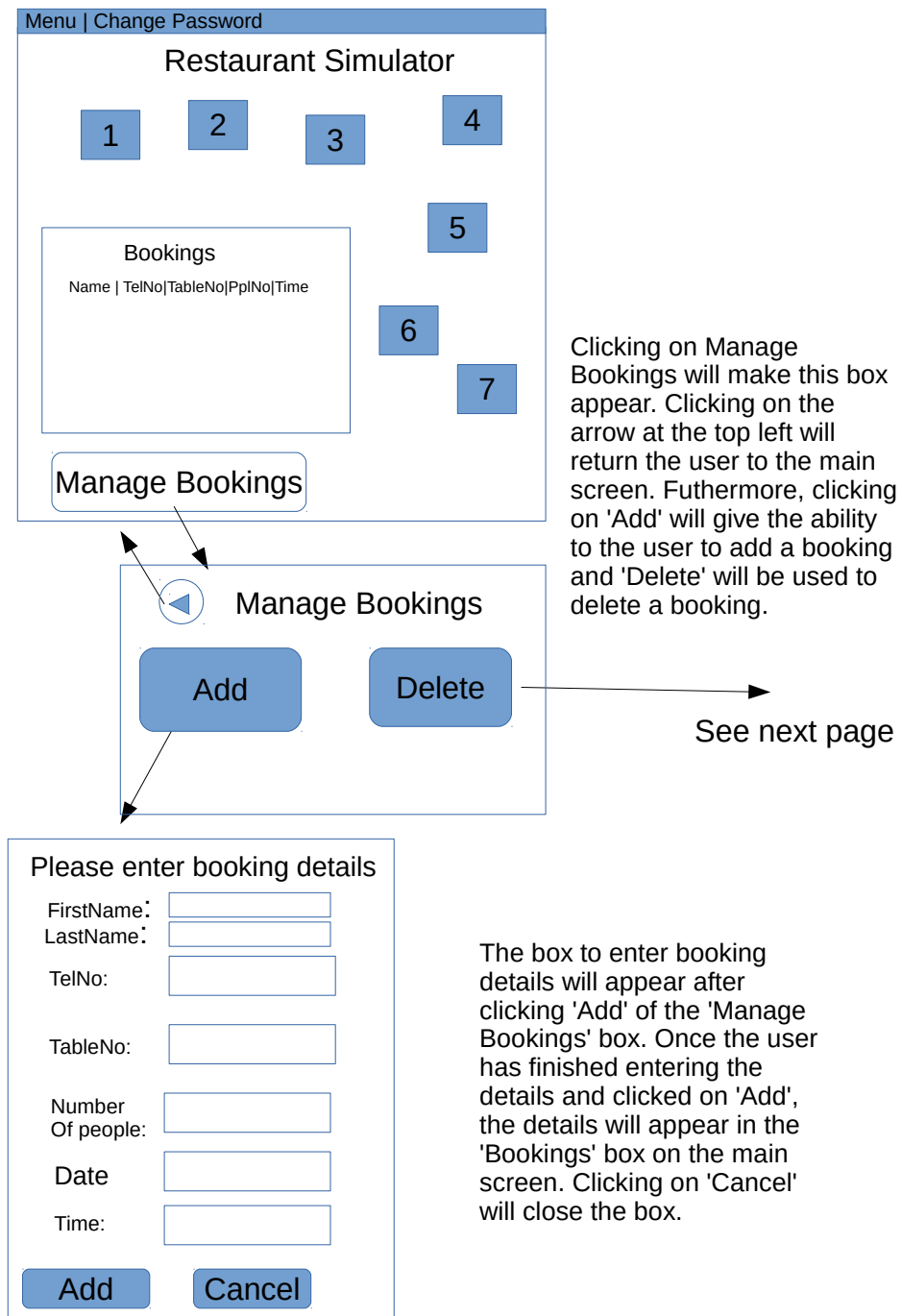
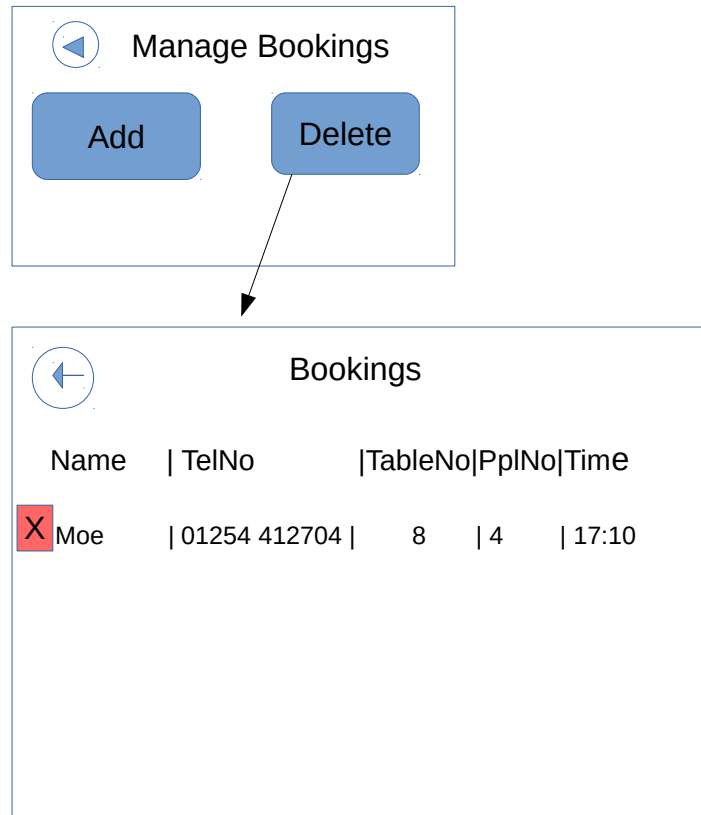


Figure 2.15: Add Booking



Selecting 'Delete' will put the user in delete mode where red boxes will appear. The Bookings box from the main screen will appear but in a larger view. Clicking the arrow at the top left will return the user back to the main screen.

Just like deleting an item from an order, clicking on Delete will make red boxes appear for each booking. Clicking on the red boxes will delete the booking of the list.

## 2.3 Hardware Specification

Keyboard and mouse are essential as the keyboard will be used to input information and the mouse will be used to navigate. The program would need to fit a 19" screen, this is important because one of my client's main requirements is to be able to track information and so having a large window fitting the screen will make it easier to look at. A processer with 1GHz will be perfectly suitable for this program to run smoothly and since the user has AMD FX(fm) - 6300 six-core CPU 3.50GHz, the program shouldnt run without any problems. In addition, not much RAM would be needed to run this program, 1GB would be more than enough and since the user has 8GB RAM the program shouldn't experience any further hardware based problems.



## 2.4 Program Structure

### 2.4.1 Top-down design structure charts

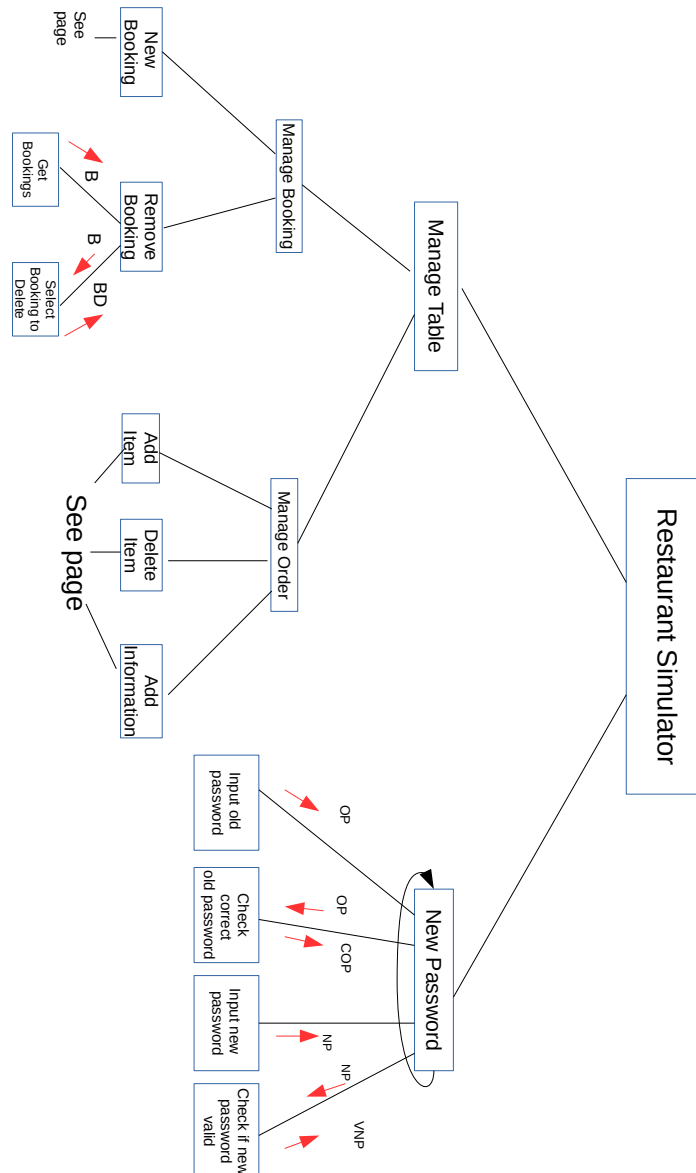


Figure 2.17: Main structure

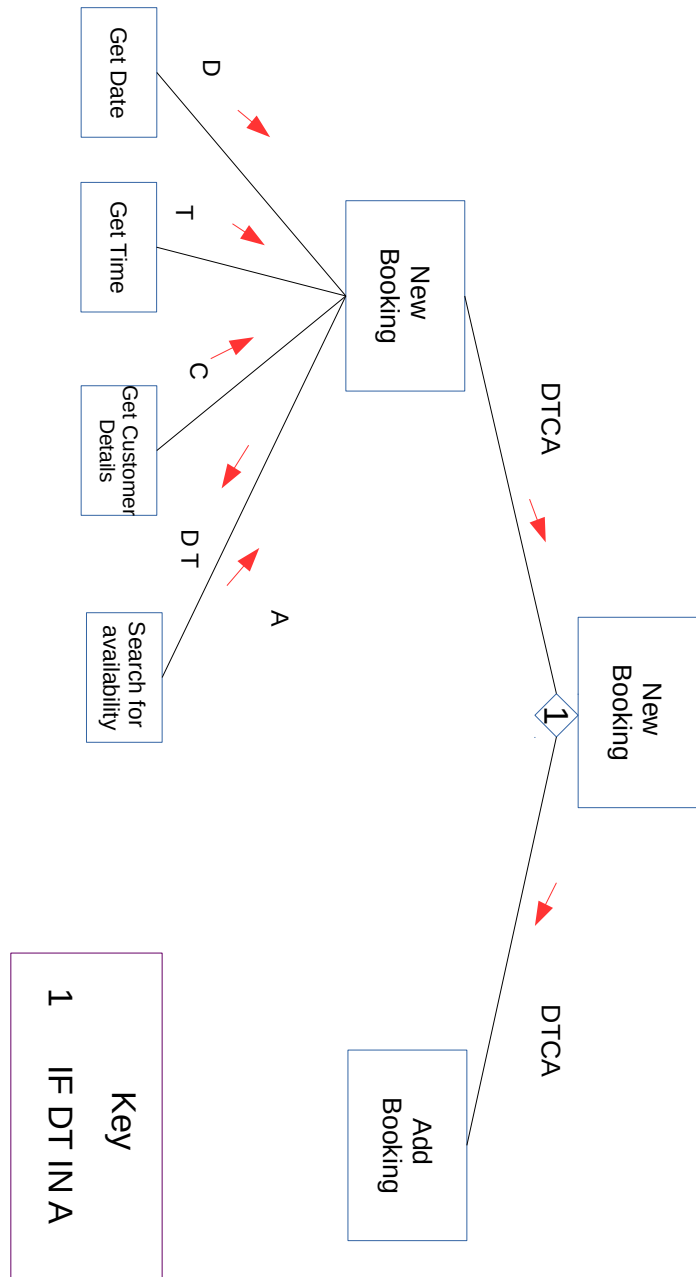
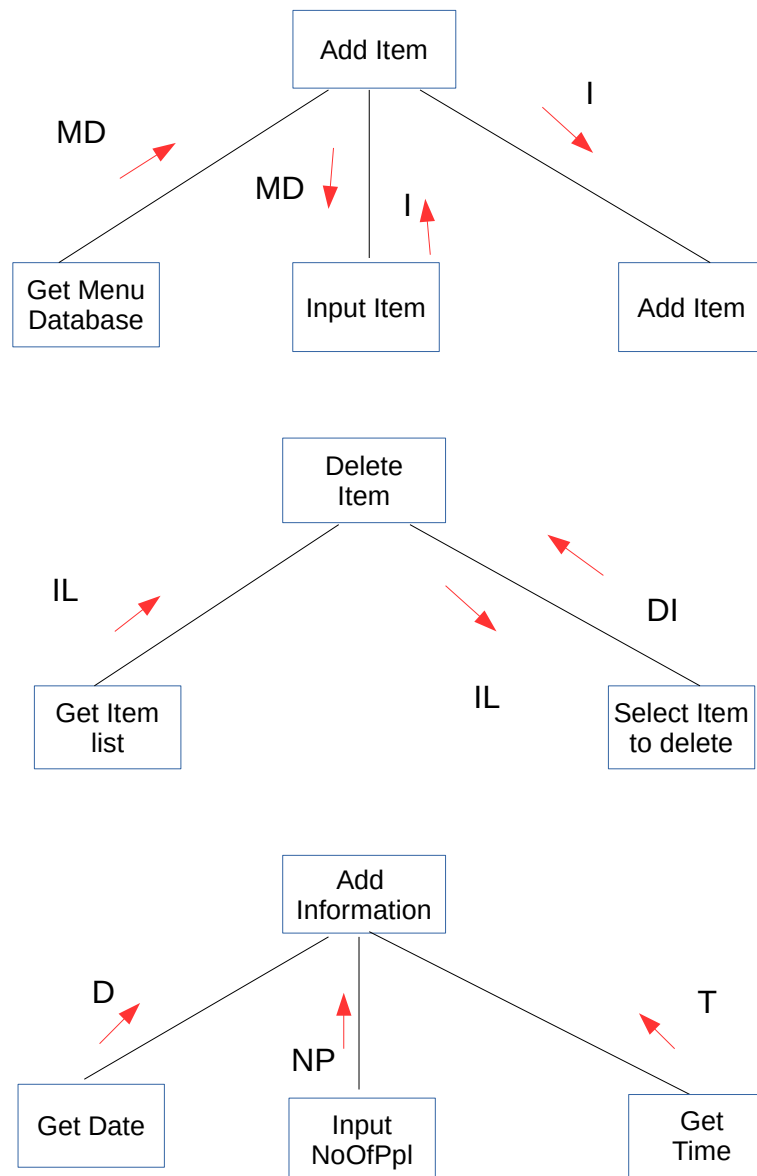


Figure 2.18: Add Booking Structure





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

---

**Algorithm 4** Password change
 

---

```

1: OldPassword  $\leftarrow$  CurrentPassword
2: ValidNewPassword  $\leftarrow$  False
3:
4: OUTPUT "Please enter the old password"
5: UserCurrentPassword  $\leftarrow$  USERINPUT
6:
7: IF UserCurrentPassword = OldPassword THEN
8:   WHILE notValidNewPassword
9:     OUTPUT "Please enter a new password(Must be longer than 4 characters)"
10:    NewPassword  $\leftarrow$  USERINPUT
11:    OUTPUT "Please re – enter the new password"
12:    ReEnteredNewPassword  $\leftarrow$  USERINPUT
13:    IF len(NewPassword) > 4 AND NewPassword =
        ReEnteredNewPassword THEN
14:      CurrentPassword  $\leftarrow$  NewPassword
15:      ValidNewPassword  $\leftarrow$  True
16:    ELSE
17:      OUTPUT Please try again.
18:    ENDIF
19:  ENDWHILE
20:
21: ELSE
22:   OUTPUT You have entered the wrong password.
23: ENDIF

```

---



---

**Algorithm 5** Adding an item to an order(MenuID database will need to be retrieved)
 

---

```

1:
2: OUTPUT "Please enter a menuID"
3: GetMenuID  $\leftarrow$  USERINPUT
4: IF GetMenuID in MenuID Database THEN
5:   ItemAdded  $\leftarrow$  (MenuIDDatabase , MenuItems
        OrderList.insert(ItemAdded)
6: ELSE
7:   OUTPUT You have entered an invalid menuID
8: ENDIF

```

---

---

**Algorithm 6** Calculating prices

---

```
1:  $TotalPrice \leftarrow 0$ 
2:  $OrderLength \leftarrow Length(OrderedItems)$ 
3:
4: FOR  $OrderedItems.Price \leftarrow 1$  TO  $OrderLength$ 
5:    $TotalPrice \leftarrow TotalPrice + OrderedItems.Price$ 
6: ENDFOR
```

---



### 2.4.3 Object Diagrams

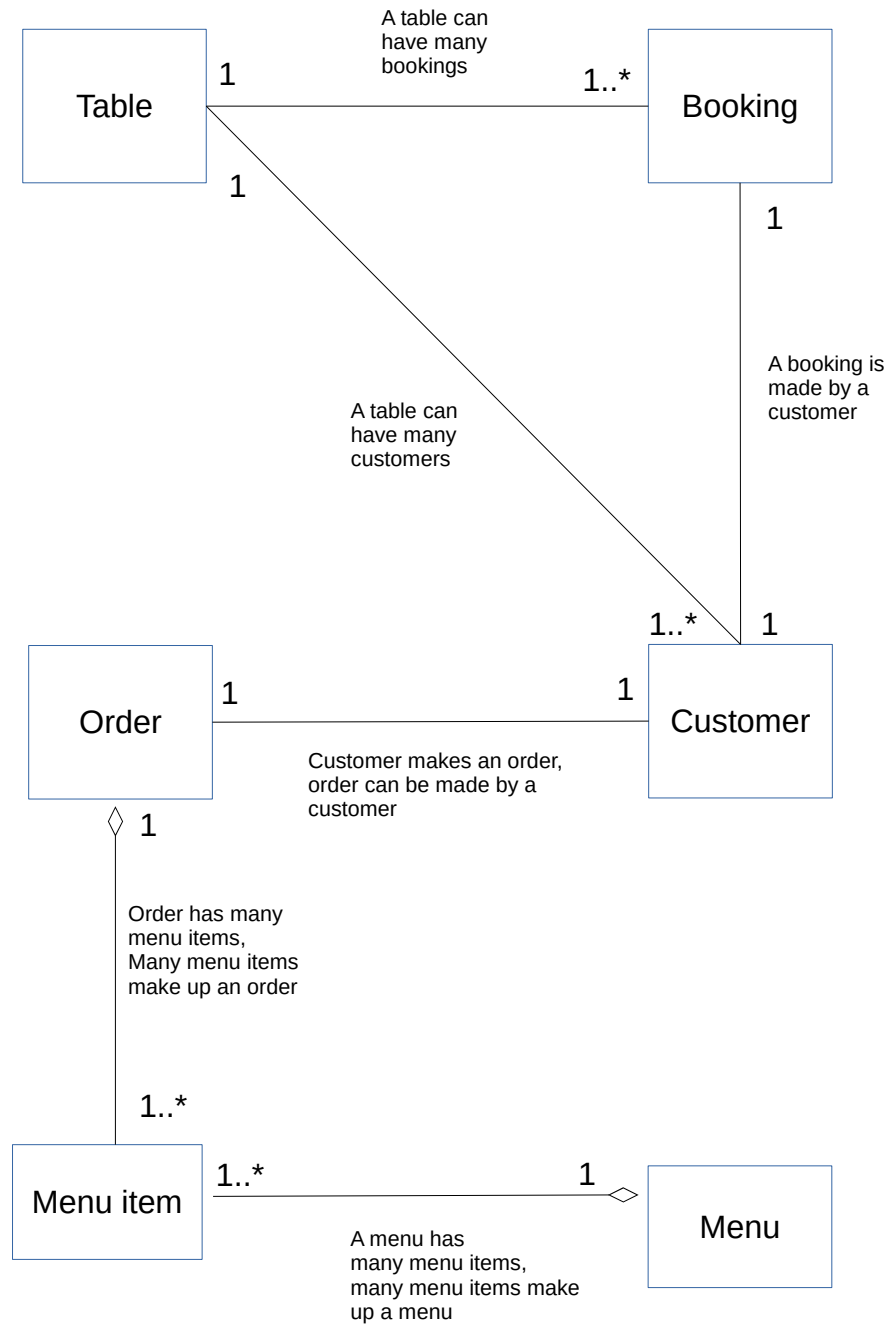


Figure 2.20: Object Diagram



### 2.4.4 Class Definitions

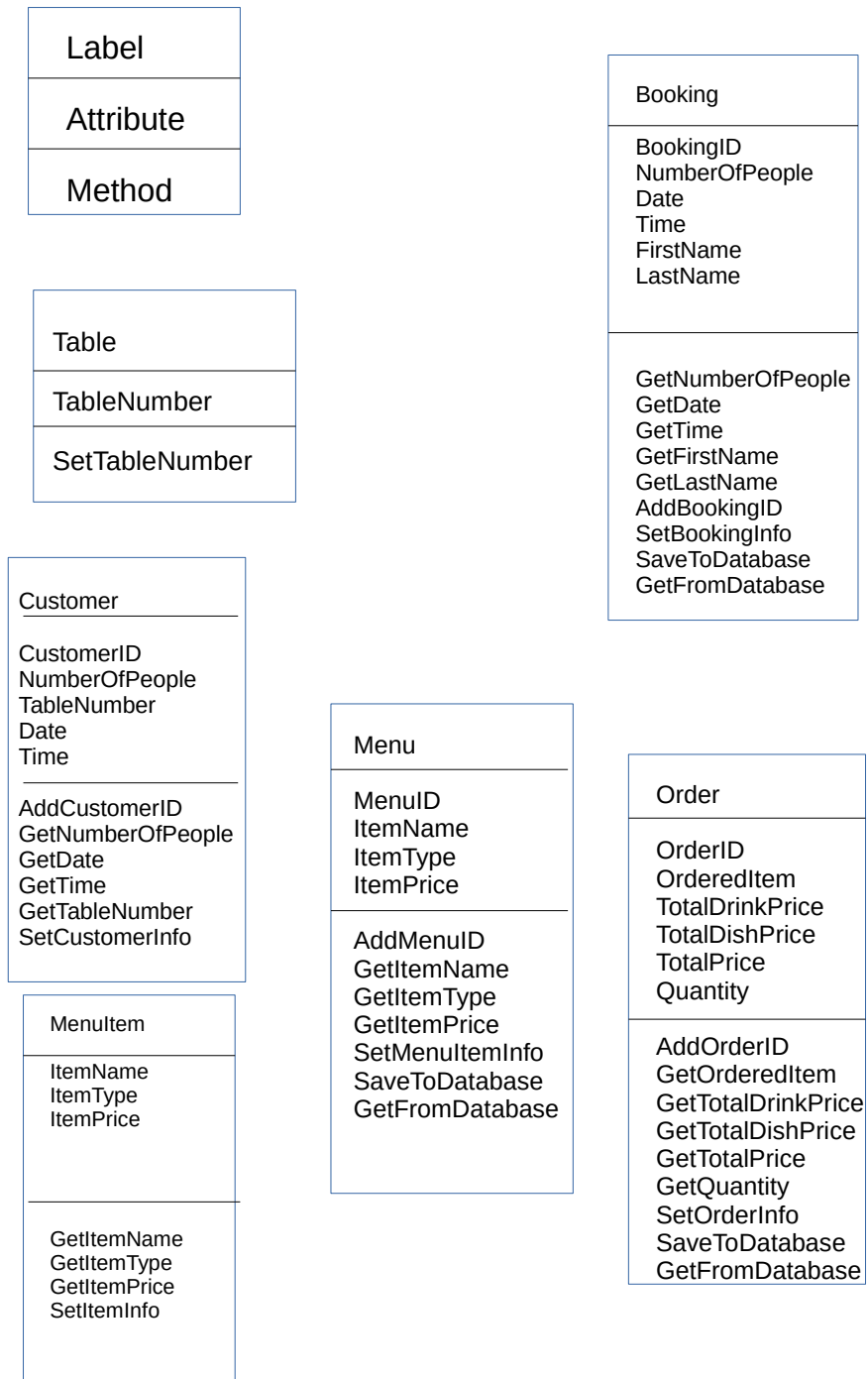


Figure 2.21: Class Definitions

## 2.5 Prototyping

There are many parts of the system that I would like to prototype due to my limited knowledge of them or its complexity.

I will try to prototype:

- The graphical user interface as this would probably one of the most difficult parts of the system I have to create due to not having a lot of experience in the area.
- Linking tables to the correct current customer order through GUI. By linking, I want it to display all the correct information such as what they ordered.
- The order screen where I would have to function the ability to add items to an order using the database as the source for the items. In addition, displaying the items in a simple and clear layout such as the one on page 46. Also, functioning both Delete and Finish would be parts of the program that I am going to prototype.
- The linking to the database and have the ability to manipulate different records through the GUI. I am not sure how to display tables from the database either and so I will attempt this. I want to display tables because it would help the user to track information such as displaying bookings where the booking date matches the system date.

## 2.6 Definition of Data Requirements

### 2.6.1 Identification of all data input items

- Password - used to access program
- Booking name
- Booking telephone number
- Booking time
- Booking date
- Booking table number
- Number of people
- Order menu item - menu item ID from database
- Menu item - adding item to menu
- Menu item type

- Menu item price

### 2.6.2 Identification of all data output items

#### Output to order screen

- Dish price
- Drink price
- Total dish price
- Total drink price
- Total price
- Ordered items
- Date of order
- Time of order
- Number of people
- Table number
- Quantity of ordered item

#### Output to booking screen

- Booking name
- Booking telephone number
- Booking time
- Booking date
- Booking table number
- Booking number of people

#### Output to database

- Total dish price
- Total drink price
- Total price
- Ordered items
- Quantity of ordered item



- Date of order
- Time of order
- Number of people
- Table number
- Booking name
- Booking telephone number
- Booking time
- Booking date
- Booking table number
- Booking number of people
- Quantity of ordered item
- Menu item
- Menu item price

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

Output	How the output is generated
Dish price	Retrieved from the menu database
Drink price	Retrieved from menu database
Total dish price	Calculated by adding up the dish prices
Total drink price	Calculated by adding up the drink prices
Total price	Calculated by adding together total dish price and total drink price
Ordered items	A member of staff inputs information
Quantity of ordered item	A member of staff inputs information
Date of order	Taken from system time
Time of order	Taken from system time
Number of people	A member of staff inputs information
Table number	Predefined by program
Booking name	A member of staff inputs information
Booking telephone number	A member of staff inputs information
Booking time	A member of staff inputs information
Booking date	A member of staff inputs information
Booking table number	A member of staff inputs information
Number of people	A member of staff inputs information
Menu item	A member of staff inputs information when adding a new item to the menu
Menu item price	A member of staff inputs information when adding a new item to the menu
Menu item type	A member of staff inputs information when adding a new item to the menu

### 2.6.4 Data Dictionary

#### Data dictionary

Name	Data Type	Length	Validation	Example Data	Comment
TableNumber	Integer	2 Characters	Range	13	Max range will be 16
Number Of People	Integer	2 Characters	Not empty and must be a number	4	Number of people sitting on a table

MenuID	Integer	3 Characters	Range(Not out of range of number of menuIDs)	52	Unique ID to identify an item from the menu
MenuItem	String	1 - 20 Characters	Not empty	Spare ribs	Item description
ItemType	Boolean		Presence Check		If false then type is drink, true is dish
ItemQuantity	Integer	2 Characters	Not empty and must be a number	4	
ItemPrice	Float	4 Characters	Not empty and must be a number	11.20	
Total DrinkPrice	Float	5 Characters	Must be a number	42.35	Added from price of drinks ordered
Total Dish-Price	Float	5 Characters	Must be a number	75.63	Added from price of dishes ordered
TotalPrice	Float	5 Characters	Must be a number	154.43	Total price of an order calculated by adding total dishprice and total drinkprice
DateOfOrder	String	4 - 6	Format	16/11/14	
TimeOfOrder	String	4 Characters	Format	07:32	
CustomerID	Integer	2bytes	Number, not used before	0412	Unique ID for someone who sits down and makes an order
OrderID	Integer	2 bytes	Number, not used before	0315	Unique ID for an order
OrderedItem	String	0-20 Characters	Item from menu	Egg fried rice	Item ordered by customer
FirstName	String	2-20 Characters	Not empty or contain numbers	Moe	Used for booking

LastName	String	2-20 Char-acters	Not empty or contain numbers	Moon	Used for booking
Telephone Number	String	11 Char-acters	Not empty and 11 numbers	03125 425634	Used for booking
BookingID	Integer	2bytes	Number, not used before	1232	Unique ID when someone makes booking

### 2.6.5 Identification of appropriate storage media

A hard drive would be preferable to store information due to the large capacity size for the database and the speed to transfer data. The only data that will be stored will be stored in the database which will hold old data for 2 years as data older than 2 years will be deleted. The application itself shouldn't be more than 20mb and the database shouldn't take the majority of a hard drive as a lot of hard drives will be more than 100gb, the database shouldn't be 1gb. In addition, a USB flash drive would be a much preferred option to back-up data. A USB flash drive is portable and the capacity size is large enough to store the data from the database. Also, they are immune to mechanical shock, magnetic fields, scratches and dust which makes them suitable for backing-up data - data will not corrupt easily. Almost all computers supports USB in this current time and may still be for many more years as USBs keep getting developed and improved.

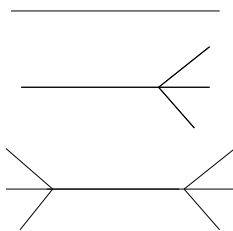


## 2.7 Database Design

### 2.7.1 Normalisation

#### ER Diagrams

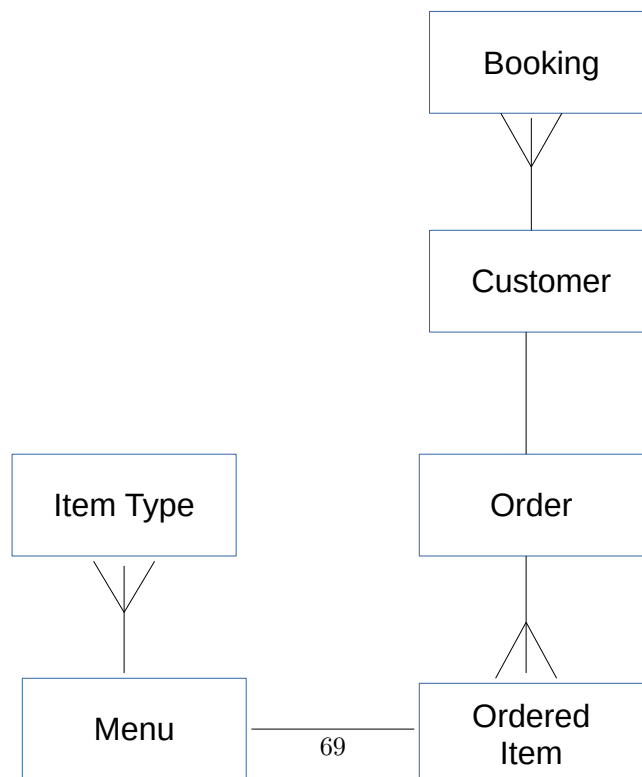
Key



One to one relationship

One to many relationship

Many to many relationship



**Entity Descriptions**

Customer(CustomerID, *BookingID*, *OrderID*, Date, Time, NoOfPpl, TableNumber)  
 Booking(BookingID, FirstName, LastName, TelephoneNo, BookingDate, BookingTime)  
 Menu(MenuID, *Type*, MenuItem, ItemPrice)  
 ItemType(Type, TypeDescription)  
 Order(OrderID, TotalDrinkPrice, TotalDishPrice, TotalPrice)  
 OrderedItem(OrderItemID, *OrderID*, *MenuID*, Quantity)

**Key**

\* Primary Key

- Foreign Key

UNF
CustomerID
Date
Time
NoOfPpl
TableNumber
MenuID
MenuItem
Type
TypeDescription
ItemPrice
OrderID
TotalDrinkPrice
TotalDishPrice
TotalPrice
OrderItemID
Quantity
BookingID
FirstName
LastName
TelephoneNo
BookingDate
BookingTime

**1NF**

Repeating	Non-Repeating
*OrderID	*CustomerID
*CustomerID	Date
MenuID	Time
MenuItem	NoOfPpl
Type	TableNumber
TypeDescription	BookingID
ItemPrice	FirstName
OrderItemID	LastName
Quantity	TelephoneNo
TotalDrinkPrice	BookingDate
TotalDishPrice	BookingTime
TotalPrice	

**2NF**

Repeating	Non-Repeating
*OrderID	*CustomerID
*CustomerID	Date
	Time
*OrderID	NoOfPpl
TotalDrinkPrice	TableNumber
TotalDishPrice	
TotalPrice	BookingID
OrderItemID	FirstName
Quantity	LastName
	TelephoneNo
MenuID	BookingDate
MenuItem	BookingTime
Type	
TypeDescription	
ItemPrice	

**3NF**

<b>*CustomerID</b>
-BookingID
-OrderID
Date
Time
NoOfPpl
TableNumber

<b>*BookingID</b>
FirstName
LastName
TelephoneNo
BookingDate
BookingTime



<b>*MenuID</b>
-Type
MenuItem
ItemPrice

<b>*Type</b>
TypeDescription

<b>*OrderID</b>
TotalDrinkPrice
TotalDishPrice
TotalPrice

<b>*OrderItemID</b>
-OrderID
-MenuID
Quantity

### 2.7.2 SQL Queries

The following SQL Queries will be formatted using Python.

---

```

1  create table  Menu(
2      MenuID integer,
3      MenuItem text,
4      ItemPrice real,
5      ItemTypeID integer,
6      primary key(MenuID)
7      foreign key(ItemTypeID) references
          ItemType(ItemTypeID))

```

---

This creates a table called Menu with the attributes *MenuItem*, *ItemPrice*. The primary key is *MenuID* and the foreign key is *ItemTypeID*

---

```

1  insert into OrderItem
2  where OrderID = ?, MenuID = ? and Quantity = ?

```

---

This inserts a new OrderItem record with the attributes *OrderID*, *MenuID* and *Quantity*

---

```

1  select *
2  from Booking
3  where BookingDate = TodaysDate

```

---

This will return all of the records from the *Booking* table that has the booking date matched with the present system date. The parameter *Today'sDate* holds the system date at that current time.

---

```
1 delete from Booking
2 where BookingID = ?
```

---

This will delete a booking from the *Booking* table with the ID of *BookingID*

---

```
1 select *
2 from OrderedItems
3 where OrderID = ?
```

---

This will return all ordered items from an order.

---

```
1 update ItemPrice
2 from Menu
3 where MenuItem = ?
```

---

This will update the price of an item from the menu with the item name of what the user chooses.

## 2.8 Security and Integrity of the System and Data

### 2.8.1 Security and Integrity of Data

To ensure that certain data is accurate such as prices of items, I will implement referential integrity to various tables in my database. Adding referential integrity would mean, if I perform a certain action to a record in a table which is also used in different table, the records in both tables will be both affected by this action. So if I updated a price of an item from the Menu table, this would also update the price of the item in a previous order.

This program will store personal information about customers such as the customer's name and telephone number and so according to the data protection act, the information must not be kept longer than necessary. Information that is 2 years old will be deleted automatically, this will be done through the start up of the application. The application will compare the records of the customers booking dates and the system dates, if there is a difference of 2 years,

then the application will delete the records off the database. The information entered must also be accurate and so there will be many validations to make sure information is as accurate as possible.

### **2.8.2 System Security**

I will implement a simple yet effective security feature where a password would need to be inputted by the user to access the program. The user would have to enter the correct password when accessing any data on the system, this will prevent unauthorised access to data. Unauthorised access is also supported by the Computer Misuse Act 1990 which covers:

- unauthorised access to computer material
- unauthorised access to computer material with criminal intent
- unauthorised modification of computer material

## 2.9 Validation

Item	Example	Validation / Verification applied	Comments
OrderedItem	Wonton soup	Presence check Lookup check	To check that this item exists in menu database
Telephone Number	01325 419603	Presence check Length check Number check	To make sure that a number has been entered which is 11 characters long
FirstName	Rudolph	Presence check	To make sure that a name has been entered
LastName	Moln	Presence check	To make sure that a name has been entered
TableNumber	4	Look up check	Make sure that a non-existing number is not created
MenuID	63	Lookup check	Make sure that a non-existing menuid is not created
MenuItem	Crispy duck	Presence check Lookup check	Check that there aren't repeating menu items
TotalPrice	42.1	Float check	Must be calculated from TotalDrinkPrice and TotalDishPrice
Total Drink Price	1.6	Float check Look up check	Must be calculated from the correct order and drink category
Total Dish Price	40.5	Float check Look up check	Must be calculated from the correct order and dish category
Number Of People	9	Range check	Must be a number but not an unrealistic number like 100 or 0

## 2.10 Testing

### 2.10.1 Outline Plan

Test Series	Purpose of Test Series	Testing Strategy	Strategy Rationale
1	Test the flow of control between the user interfaces	Top-down testing	
2	Test validation of data input is detected	Bottom-up testing	Each component will be tested once it is developed
3	Test information input is stored in the correct place	Black box testing	Each component will be tested once it is developed
4	Test algorithms to make sure that the output is correct	White box testing	Each component will be tested once it is developed
5	Test that the system fulfils the specification	Acceptance testing	Each component will be tested once it is developed
6	Test database has referential integrity	Integration testing	Each component will be tested once it is developed

### 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

1.01	Test 'Change password' button functions correctly	Should direct user to change password interface	Click Change password button	Normal	Change password interface should be displayed		
1.02	Test Cancel button functions correctly on change password interface	Should redirect user to login screen	Click Cancel button on change password interface	Normal	Change password interface should close		
1.03	Test interactive table functions correctly	Should direct user to the order details from the table selected	Click on occupied table	Normal	Table information screen should be displayed		
1.04	Test unoccupied table functions correctly	Should direct user to 'add details to table' interface	Click on unoccupied table	Normal	'Add details to table' interface should be displayed		
1.05	Test Table information screen, add button functions correctly	Should direct user to add item interface	Click Add on table information screen	Normal	Add item interface should be displayed		

1.06	Test table information screen, delete function correctly	Should change colour of delete button and red box will appear to indicate deletion for items	Click Delete button	Normal	Delete button should change colour and red boxes should appear next to each order item		
1.07	Test 'Change password' button functions correctly	Should direct user to change password interface	Click Change password button	Normal	Change password interface should be displayed		
1.08	Test back arrow button functions correctly on table information screen	Should direct user to main screen	Click back arrow button	Normal	User redirected back to main screen should be displayed		
1.09	Test 'Manage Bookings' button functions correctly on main screen	Should direct user to Manage Bookings interface	Click Manage Bookings	Normal	Manage Bookings interface should be displayed		

1.10	Test Add button functions correctly on Manage Bookings interface	Should direct user to create booking interface	Click Add button	Normal	Create booking interface should be displayed		
1.11	Test Cancel button functions correctly on create booking interface	Should redirect user to Manage Bookings interface	Click Cancel button	Normal	User should be redirected to Manage Bookings interface		
1.12	Test back arrow on manage bookings interface functions correctly	Should redirect user to main screen	Click Change back arrow button	Normal	Main screen should be displayed		
1.13	Test Delete button on Manage Bookings screen	Should direct user to bookings display interface	Click Delete button	Normal	Bookings display should be displayed		
1.14	Test back arrow button functions correctly on bookings display screen	Should redirect user to Manage Bookings interface	Click back arrow button	Normal	User should be redirected to Manage Bookings interface		
2.01	Verify password entered	The field cannot be left blank	(Nothing), Treem	Erroneous, Normal	Error, Accepted		



2.02	Verify new password entered at change password screen	The field cannot be left blank	(Nothing), PineTree	Erroneous, Normal	Error, Accepted		
2.03	Verify retype new password entered at change password screen	The field cannot be left blank	(Nothing), PineTree	Erroneous, Normal	Error, Accepted		
2.04	Verify old password entered at change password screen	The field cannot be left blank	(Nothing), Treem	Erroneous, Normal	Error, Accepted		
2.05	Verify Number of people entered at 'add details table'	The field cannot be left blank	(Nothing), 3, pigs	Erroneous, Normal, Erroneous	Error, Accepted, Error		
2.06	Verify MenuID entered at 'add item to order' interface	The field cannot be left blank	(Nothing), 3, 9552	Erroneous, Normal, Erroneous	Error, Accepted, Error		
2.07	Verify First Name entered at 'enter booking details' interface	The field cannot be left blank	(Nothing), Milly, 63	Erroneous, Normal, Erroneous	Error, Accepted, Error		

2.08	Verify Last Name entered at 'enter booking details' interface	The field cannot be left blank	(Nothing), Milk, 2	Erroneous, Normal, Erroneous	Error, Accepted, Error		
2.09	Verify Telephone Number entered at 'enter booking details' interface	The field cannot be left blank	(Nothing), 01523859372, 014829, 01589258295289	Erroneous, Normal, Erroneous, Erroneous	Error, Accepted, Error, Error		
2.10	Verify Table Number entered at 'enter booking details' interface	The field cannot be left blank	(Nothing), 7, Hey	Erroneous, Normal, Erroneous	Error, Accepted, Error		
2.11	Verify Number Of People entered at 'enter booking details' interface	The field cannot be left blank	(Nothing), 3, Lisa	Erroneous, Normal, Erroneous	Error, Accepted, Error		
2.12	Verify Date entered at 'enter booking details' interface	The field cannot be left blank	(Nothing), 06/05/18, Homer, 032/63/153	Erroneous, Normal, Erroneous, Erroneous	Error, Accepted, Error, Error		

2.13	Verify Time entered at 'enter booking details' interface	The field cannot be left blank	(Nothing),18:12, Bart, 53:62	Erroneous, Normal, Erroneous, Erroneous	Error, Accepted, Error, Error		
3.01	Verify all table details entered are added to relevant database tables	Information should be added to the correct fields in customer, order and orderitem tables. If necessary reservation table	customer information, order information, orderitem information, if necessary reservation table	Normal	Added to customer, order and orderitem table. If necessary reservation table		
3.02	Verify that all details entered at 'enter booking details' interface are added to the reservation database	All of the information should be added to the correct field in the reservations table	Reservation informationl	Normal	Added to the reservation table		

4.01	Verify password changed	Password should not successfully change if length is not bigger than 4 and old password does not match input old password	-Try changing password with incorrect input and length of 2 new password, -Try changing password with new password having length of 2, - Try changing password with correct input and correct length	Error, Error, Accepted			
4.02	Verify add item function works correctly	Entering MenuID will return information based on that ID	Enter ID	Normal	Return all information based on the ID		

4.03	Verify Total price calculation functions correctly	Adds up all items prices together to get a total	Enter items to order	Normal	Calculates the total price based on items entered		
4.04	Check bookings displayed on correct day	Should display all bookings that match with system date	Create a range of bookings that have different dates	Normal	Displays correct bookings		
5.01	Verify program fulfills the specification	Run through the program, testing all aspects to make sure the meet the objectives in the specification	Enter information in all places required input	Normal	Program fulfills specification		
6.01	Verify menu item name updates in case an item is mistakenly spelt	Check the item name is updated in all records that it appears in	Update name of a menu item (Wate to Water)	Normal	Wate should change to Water		

6.02	Verify menu item price updates in case an item is mistakenly priced	Check the price of the item is updated in all records the item appears in	Update price of a menu item (0.060) to (0.60)	Normal	Price should change to 0.60		
------	---------------------------------------------------------------------	---------------------------------------------------------------------------	-----------------------------------------------	--------	-----------------------------	--	--

## Chapter 3

# Testing

### 3.1 Test Plan

### 3.1.1 Original Outline Plan

Test Series	Purpose of Test Series	Testing Strategy	Strategy Rationale
1	Test the flow of control between the user interfaces	Top-down testing	
2	Test validation of data input is detected	Bottom-up testing	Each component will be tested once it is developed
3	Test information input is stored in the correct place	Black box testing	Each component will be tested once it is developed
4	Test algorithms to make sure that the output is correct	White box testing	Each component will be tested once it is developed
5	Test that the system fulfils the specification	Acceptance testing	Each component will be tested once it is developed
6	Test database has referential integrity	Integration testing	Each component will be tested once it is developed

### 3.1.2 Changes to Outline Plan

Test Series	Purpose of Test Series	Testing Strategy	Strategy Rationale
4	Test algorithms and SQL statements to make sure that the output is correct	White box testing	Each component will be tested once it is developed

### 3.1.3 Original Detailed Plan

The original details plan below looks different than the one in the Design section as I have formatted the plan below so that each test data has its own row.



I have not tested the rows that are in grey due to changes in my program.

Test Series	Purpose of Test	Test Description	Test Data	Test Data Type (Normal/ Erroneous/ Boundary)	Expected Result	Actual Result	Evidence
1.01	Test 'Change password' button functions correctly	Should direct user to change password interface	Click Change password button	Normal	Change password interface should be displayed		
*1.02	Test Cancel button functions correctly on change password interface	Should redirect user to login screen	Click Cancel button on change password interface	Normal	Change password interfact should close		
*1.03	Test interactive table functions correctly	Should direct user to the order details from the table selected	Click on occupied table	Normal	Table information screen should be displayed		
*1.04	Test unoccupied table functions correctly	Should direct user to 'add details to table' interface	Click on unoccupied table	Normal	'Add details to table' interface should be displayed	Add details to table displayed - expected	

*1.05	Test Table information screen, add button functions correctly	Should direct user to add item interface	Click Add on table information screen	Normal	Add item interface should be displayed	Add item interface displayed - expected	
*1.06	Test table information screen, delete function correctly	Should change colour of delete button and red box will appear to indicate deletion for items	Click Delete button	Normal	Delete button should change colour and red boxes should appear next to each order item		
*1.07	Test 'Change password' button functions correctly	Should direct user to change password interface	Click Change password button	Normal	Change password interface should be displayed		
*1.08	Test back arrow button functions correctly on table information screen	Should direct user to main screen	Click back arrow button	Normal	User redirected back to main screen should be displayed		

1.09	Test 'Manage Bookings' button functions correctly on main screen	Should direct user to Manage Bookings interface	Click Manage Bookings	Normal	Manage Bookings interface should be displayed	Manage Bookings interface displayed - expected	3.8 on page 125
1.10	Test Add Booking button functions correctly on Manage Bookings interface	Should direct user to create booking interface	Click Add Booking button	Normal	Create booking interface should be displayed	Create booking interface displayed - expected	3.2 on page 118
*1.11	Test Cancel button functions correctly on create booking interface	Should redirect user to Manage Bookings interface	Click Cancel button	Normal	User should be redirected to Manage Bookings interface		
*1.12	Test back arrow on manage bookings interface functions correctly	Should redirect user to main screen	Click Change back arrow button	Normal	Main screen should be displayed		
1.13	Test Delete Booking button on Manage Bookings screen	Should direct user to delete bookings display interface	Click Delete button	Normal	Delete bookings display should be displayed	Delete bookings layout displayed - expected	3.3 on page 119

*1.14	Test back arrow button functions correctly on bookings display screen	Should redirect user to Manage Bookings interface	Click back arrow button	Normal	User should be redirected to Manage Bookings interface		
* 2.01	Verify password entered	The field cannot be left blank	(Nothing), Treem	Erroneous, Normal	Error, Accepted		
*2.01.01	Verify new password entered at change password screen	The field cannot be left blank	(Nothing), PineTree	Erroneous, Normal	Error, Accepted		
*2.02	Verify retype new password entered at change password screen	The field cannot be left blank	(Nothing), PineTree	Erroneous, Normal	Error, Accepted		
*2.03	Verify old password entered at change password screen	The field cannot be left blank	(Nothing), Treem	Erroneous, Normal	Error, Accepted		
2.04	Verify Number of people entered at 'assign customer to table'	User inputs nothing	(Nothing)	Erroneous	Error	Nothing - expected	3.4 on page 121

2.04.01	Verify Number of people entered at 'assign customer to table'	User value inputs	3	Normal	Accepted	Accept input - expected	
2.04.02	Verify Number of people entered at 'assign customer to table'	User value inputs	pigs	Erroneous	Error	Can only enter numbers - expected due to regular expression	
2.05	Verify ItemID entered at 'add item to order' interface	User vnothing inputs	(Nothing)	Erroneous	Error/ nothing	No changes - expected	
2.05.01	Verify ItemID entered at 'add item to order' interface	User value inputs	3	Normal	Accepted	Accepted	
2.05.02	Verify ItemID entered at 'add item to order' interface	User value inputs	9552	Erroneous	Error	Only allowed to input 3 digits	

2.06	Verify First Name entered at 'enter booking details' interface	User inputs nothing	(Nothing)	Erroneous	Error	Add booking did not proceed after clicking add booking - expected	
2.06.01	Verify First Name entered at 'enter booking details' interface	User inputs name	Milly	Normal	Accepted	Milly was accepted and booking proceeded - expected	
2.06.02	Verify First Name entered at 'enter booking details' interface	User inputs name	63	Erroneous	Error	Could not enter numbers - expected due to regular expression	
2.07	Verify Last Name entered at 'enter booking details' interface	User inputs name	(Nothing)	Erroneous	Error	Add booking did not proceed after clicking add booking	
2.07.01	Verify Last Name entered at 'enter booking details' interface	User inputs name	Milk	Normal	Accepted	Milk was accepted and booking proceeded - expected	

2.07.02	Verify Last Name entered at 'enter booking details' interface	User inputs name	2	Erroneous	Error	Could not enter numbers - expected due to regular expression	
2.08	Verify Telephone Number entered at 'enter booking details' interface	User inputs nothing	(Nothing)	Erroneous	Error	Add booking did not proceed - expected	
2.08.01	Verify Telephone Number entered at 'enter booking details' interface	User inputs number	01523 859372	Normal	Accepted	Add booking did proceed - expected	
2.08.02	Verify Telephone Number entered at 'enter booking details' interface	User inputs number	014829	Boundary	Error	Add booking did not proceed - expected	
*2.09	Verify Table Number entered at 'enter booking details' interface	User inputs number	(Nothing)	Erroneous	Error		

*2.09.01	Verify Table Number entered at 'enter booking details' interface	User inputs number	7	Normal	Accepted		
*2.09.02	Verify Table Number entered at 'enter booking details' interface	User inputs number	Hey	Erroneous	Error		
2.10	Verify Number Of People entered at 'enter booking details' interface	User inputs nothing	(Nothing)	Erroneous	Error	Add booking did not proceed - expected	
2.10.01	Verify Number Of People entered at 'enter booking details' interface	User inputs number	3	Normal	Accepted	Add booking proceeded - expected	
2.10.02	Verify Number Of People entered at 'enter booking details' interface	User inputs number	Lisa	Erroneous	Error	Could not enter letters - expected due to regular expression	



*2.11	Verify Date entered at 'enter booking details' interface	User inputs date	(Nothing	Erroneous	Error		
*2.11.01	Verify Date entered at 'enter booking details' interface	User inputs date	06/05/13	Normal	Accepted		
*2.11.02	Verify Date entered at 'enter booking details' interface	User inputs date	Homer	Erroneous	Error		
*2.11.03	Verify Date entered at 'enter booking details' interface	User inputs date	032/63/153	Erroneous	Error		
*2.12	Verify Time entered at 'enter booking details' interface	User inputs time	(Nothing)	Erroneous	Error		
2.12.01	Verify Time entered at 'enter booking details' interface	User inputs time	18:12	Normal	Accepted	Add booking proceeded - expected	

*2.12.02	Verify Time entered at 'enter booking details' interface	User inputs time	Bart	Erroneous	Error		
*2.12.03	Verify Time entered at 'enter booking details' interface	User inputs time	53:62	Erroneous	Error		
*3.01	Verify all table details entered are added to relevant database tables	Information should be added to the correct fields in customer, order and orderitem tables. If necessary reservation table	customer information, order information, orderitem information, if necessary reservation table	Normal	Added to customer, order and orderitem table. If necessary reservation table		
3.02	Verify that all details entered at 'enter booking details' interface are added to the booking database	All of the information should be added to the correct field in the booking table	Booking information	Normal	Relevant details added to booking table	All details have been added to relevant database tables - expected	

*4.01	Verify password changed	Password should not successfully change if length is not bigger than 4 and old password does not match input old password	Try changing password with incorrect input and length of 2 new password,	Error			
*4.01.01	Verify password changed	Password should not successfully change if length is not bigger than 4 and old password does not match input old password	Try changing password with new password having length of 2	Error			
*4.01.02	Verify password changed	Password should not successfully change if length is not bigger than 4 and old password does not match input old password	Try changing password with correct input and correct length	Accepted			

*4.02	Verify add item function works correctly	Entering MenuID will return information based on that ID	Enter ID	Normal	Return all information based on the ID		
4.03	Verify Total price calculation functions correctly	Adds up all items prices together to get a total	Enter items to order	Normal	Calculates the total price based on items entered	Total doesn't update - unexpected	
4.04	Check bookings displayed on correct day	Should display all bookings that match with system date	Create a range of bookings that have different dates	Normal	Displays correct bookings		

5.01	Verify program fulfills the specification	Run through the program, testing all aspects to make sure the meet the objectives in the specification	Enter information in all places required input	Normal	Program fulfills specification	Can run through program without any problems, some minor objectives were not met such as having clickable tables (I have radio buttons instead).	
*6.01	Verify menu item name updates in case an item is mistakenly spelt	Check the item name is updated in all records that it appears in	Update name of a menu item (Wate to Water)	Normal	Wate should change to Water		
*6.02	Verify menu item price updates in case an item is mistakenly priced	Check the price of the item is updated in all records the item appears in	Update price of a menu item (0.060) to (0.60)	Normal	Price should change to 0.60		

I have removed some tests under 2.09, 2.11 and 2.12 due to changes in my program which made it impossible to have the wrong input in terms of erroneous and boundary inputs. For example, test 2.09 was to verify the table number inputted was valid, I have made my program so now the user can only select tables which exist through a combo box. As for tests 2.11 and 2.12, the user is forced into using the correct times/dates format. I set the minimum date for QDateEdit to be the system date which would mean the user will not be able to input boundary data (make a booking for yesterday).

### 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
1.15	Test Add Item on 'Item Menu' menu bar	Check if Add Item layout is displayed after clicking on Add Item	Click on Add Item	Normal	Add Item layout displayed	Add Item layout displayed - expected	
1.16	Test Delete Item on 'Item Menu' menu bar	Check if Delete Item layout is displayed after clicking on Delete Item	Click on Delete Item	Normal	Delete Item layout displayed	Delete Item layout is displayed	Example

1.17	Test Update Item Price on 'Item Menu' menu bar	Check if Update Item Price layout is displayed after clicking on Update Item Price	Click on Update Item Price	Normal	Update Item Price layout displayed	Update Item Price layout displayed - expected	
1.18	Test Add Booking on 'Bookings' menu bar	Check if Add Booking layout is displayed after clicking on Add Booking	Click on Add Booking	Normal	Add Booking layout displayed	Add Booking layout displayed - expected	
1.19	Test Delete Booking on 'Bookings' menu bar	Check if Delete Booking layout is displayed after clicking on Delete Booking	Click on Delete Booking	Normal	Delete Booking layout displayed	Delete Booking layout displayed - expected	
1.20	Test Update Booking on 'Bookings' menu bar	Check if Update Booking layout is displayed after clicking on Update Booking	Click on Update Booking	Normal	Update Booking layout displayed	Update Booking layout displayed - expected	
1.21	Test 'Search Order' on tool bar	Check if Search Order layout is displayed after clicking on Search Order	Click on Search Order	Normal	Search Order layout displayed	Search Order layout displayed - expected	

1.22	Test 'View Bookings' on tool bar	Check if View Bookings layout is displayed after clicking on View Bookings	Click on View Bookings	Normal	View Bookings layout displayed	View Bookings layout displayed - expected	
1.23	Test 'View Customers' on tool bar	Check if View Customers layout is displayed after clicking on View Customers	Click on View Customers	Normal	View Customers layout displayed	View Customers layout displayed - expected	
1.24	Test 'View Dishes' on tool bar	Check View Dishes layout is displayed after clicking on View Dishes	Click on View Dishes	Normal	View Dishes layout displayed	View Dishes layout displayed - expected	
1.25	Test 'View Drinks' on tool bar	Check if View Drinks layout is displayed after clicking on View Drinks	Click on View Drinks	Normal	View Drinks layout displayed	View Drinks layout displayed - expected	
1.26	Test 'Main Screen' on tool bar	Check if Main Screen is displayed after clicking on 'Main Screen'	Click on Main Screen from Search Order layout	Normal	Main Screen layout displayed	Main Screen layout displayed - expected	



1.27	Test table radio buttons on main screen	Check if dialog box shows after clicking on Select Table	Choose an unoccupied table and click Select Table	Normal	Assign customer dialog box shows	Assign customer dialog box is shown - expected	
1.28	Test Assign customer layout	Check if manage order box shows after clicking on Create (after filling in required field)	Click on Create	Normal	Relevant manage order dialog box shows	Relevant manage order dialog box is shown - expected	
1.28.01	Test Assign customer layout	Check if manage order box shows after clicking on Select	Click on Select	Normal	Relevant manage order dialog box shows	Relevant manage order dialog box is shown	
1.29	Test Add button on manage order box	Check if Add Item To Order box shows after clicking on Add	Click on Add	Normal	Add Item To Order dialog box shows	Add Item To Order dialog box is shown - expected	
1.30	Test Delete button on manage order box	Check Delete Item Off Order box shows after clicking on Delete	Click on Delete	Normal	Delete Item Off Order dialog box shows	Delete Item Off Order dialog box is shown	

1.31	Test Finish button on manage order box	Check if Manage Order box closes after clicking on Finish	Click on Finish	Normal	Manage Order box closes	Manage Order box closes - expected	
1.32	Test Invoice Preview button on manage order box	Check if the preview of the invoice box shows after clicking on Invoice Preview	Click on Invoice Preview	Normal	Invoice preview shows	Invoice shown - expected	3.7 on page 124
1.33	Test Print Invoice button on manage order box	Check if print option appear after clicking Print Invoice	Click on Print Invoice	Normal	Print options appear	Print options appeared - expected	
2.04.03	Verify Number of people entered at 'assign customer to table'	Check if user input is valid after clicking Create	0	Boundary	Input not accepted	Input was not accepted - expected	
2.13	Verify Item Name input at Add Item to Menu	Check if user input is valid after clicking Add Item assuming all other fields are filled with normal data	Rice	Normal	Accepted	Item was successfully added	

2.13.01	Verify Item Name input at Add Item to Menu	Check if user input is valid after clicking Add Item assuming all other fields are filled with normal data	(Nothing)	Erroneus	Error	Item add unsuccessful - expected	
2.14	Verify ItemID input at Update Item Price	Check if user input is valid after clicking Update Item assuming all other fields are filled with normal data	7	Normal	Accepted	Price was successfully updated	
2.14.01	Verify ItemID input at Update Item Price	Check if user input is valid after clicking Update Item assuming all other fields are filled with normal data	0	Boundary	Error	Nothing happened - expected	
2.15	Verify Number Of People at Update Booking	Check if user input is valid after clicking Update Number Of People with a booking that exists	5	Normal	Accepted	Booking updated - expected	

2.15.01	Verify Number Of People at Update Booking	Check if user input is valid after clicking Update Number Of People with a booking that exists	0	Boundary	No changes will be made	Booking did not update - expected	
2.16	Verify Item Name at Delete Item Off Menu	Check if user input is valid after clicking Delete Item for Item Name	Nothing	Erroneous	Error(no changes will be made)	The process of deleting an item did not happen - expected	
2.16.01	Verify Item Name at Delete Item Off Menu	Check if user input is valid after clicking Delete Item for Item Name	Steak	Normal	Steak will be deleted	Steak was successfully deleted (removed from the displayed table widget) - expected	
2.17	Verify Item ID at Delete Item Off Menu	Check if user input is valid after clicking Delete Item for Item Name	10(Steak which i've added again)	Normal	Record with Item ID 10 deleted	Item ID 10 was deleted (removed from the displayed table) - expected	

2.17.01	Verify Item ID at Delete Item Off Menu	Check is user input is valid after clicking Delete Item for Item Name	(Nothing)	Erroneous	Error(no changes will be made)	The process of deleting an item did not happen - expected	
2.17.02	Verify Item ID at Delete Item Off Menu	Check is user input is valid after clicking Delete Item for Item Name	645(There was not an item with itemID 645)	Boundary	Error(no changes will be made)	The process of deleting an item did not happen - expected	
2.18	Verify Booking ID at Delete Booking	Check is user input is valid	(Nothing )	Erroneous	Error(no changes will be made)	The process of deleting a booking did not happen - expected	
2.18.01	Verify Booking ID at Delete Booking	Check is user input is valid	888	Boundary	Error(no changes will be made)	There was not an error but the process of deleting a booking did not happen	

2.18.02	Verify Booking ID at Delete Booking	Check is user input is valid	18(Booking which ive added for testing purposes)	Normal	Booking deleted	Booking removed(Booking was removed from the displayed boking table) - expected	
3.03	Check all details from the creation of a booking at Assign Customer to table is transferred	Check all correct details are transferred over to the Manage Order box	Click on Create after filling in fields correctly	Normal	All relevent details displayed	All relevent details transferred	3.5 on page 122
3.04	Check all details from the selecting of a booking at Assign Customer to table is transferred	Check all correct details are transferred over to the Manage Order box	Click on Select after selecting a booking from combo box	Normal	All relevent details displayed	All relevent details transferred 3.8 on page 125	
3.05	Check all details from an order is transferred to invoice	Compare the order details to invoice details to make sure they are the same	Click on Invoice Preview	Normal	All relevent details displayed	All relevent details transferred	3.7 on page 124

3.06	Booking added is stored in correct table	After successfully adding a booking, the booking should be displayed in table above	Add Booking	Normal	All relevant details displayed	All relevant details displayed in table above	
3.07	Customer added is stored in correct table	After successfully adding a booking, a customer record should be added to Customer table	Add Booking	Normal	All relevant details displayed	All relevant details added to table	
3.08	Item added to menu is stored in correct table	After successfully adding an item, an item record should be appear on Items table	Add Booking	Normal	All relevant details displayed	New item appeared	
4.03.01	Verify Total price calculation algorithm is correct	Check that an algorithm adds up all items prices together to get a total	Check invoice preview to see total price	Normal	Calculates the total price based on items ordered	Correct total price - expected	

4.05	Check if the 'Increasing the quantity of an ordered item' algorithm works	A customer can order x more of an item - the quantity should therefore increase by x amount	Add an item initially then add 10 more if it	Normal	Should expect quantity to be 11	Quantity increased to 11 - expected	3.6 on page 123
4.06	Check if only drinks are displayed on View Drinks tool bar	After clicking View Drinks, a table should appear with only drinks in it	Click on View Drinks	Normal	Only items with Item-TypeID 2 appear	Drinks were only displayed - expected	
4.07	Check if only dishes are displayed on View Dishes tool bar	After clicking View Dishes, a table should appear with only dishes in it	Click on View Dishes	Normal	Only items with Item-TypeID 1 appear	Dishes were only displayed	
4.08	Check search order function	Leave the booking field blank and click Search Order	(Nothing)	Erroneous	Error /empty table appear	Empty table appeared	
4.08.01	Check search order function	Enter a booking ID that doesn't exist and click Search Order	93	Boundary	Error / empty table appear	Empty table appeared	



4.08.02	Check search order function	Enter a booking ID that exists and click Search Order	11	Normal	A populated table appear	Correct table appeared	3.9 on page 126
6.03	Ensure order gets deleted when booking gets deleted	After deleting a booking, the order should be deleted with it.	Delete a booking which has booking items	Normal	Order should be deleted	Order(Booking items) has been deleted	3.10 on page 127
6.04	Check if an item that does not exist is added to an order	Adding an item that does not exist from the manage order box	Add Item ID 933	Boundary	Nothing should happen as there isnt an item with an ID of 993	No items were added	

## 3.2 Test Data

### 3.2.1 Original Test Data

Test Number	Test Data	Justification for choice of test data
2.04.01	3	Program must allow the correct input
2.04.01	pigs	Program must not allow the wrong input, regular expression shouldn't allow letters to be inputted
2.05	(Nothing)	User could accidentally try to proceed without entering anything which shouldn't be accepted by validation
2.05.01	3	Program must allow the correct input
2.05.02	9552	Program must not allow an erroneous input
2.06	(Nothing)	User could accidentally try to proceed without entering anything which shouldn't be accepted by validation
2.06.01	3	Program must allow the correct input
2.06.02	63	Program must not allow the wrong input, regular expression shouldn't allow numbers to be inputted
2.07	(Nothing)	User could accidentally try to proceed without entering anything which shouldn't be accepted by validation
2.07.01	Milk	Program must allow the correct input
2.07.02	2	Program must not allow the wrong input, regular expression shouldn't allow numbers to be inputted
2.08	(Nothing)	User could accidentally try to proceed without entering anything which shouldn't be accepted by validation
2.08.01	0152385972	Program must allow the correct input (11 numbers)
2.08.02	014829	Program must not allow an invalid number (not 11 numbers)
2.10	(Nothing)	User could accidentally try to proceed without entering anything which shouldn't be accepted by validation
2.10.01	3	Program must allow the correct input
2.10.02	2	Program must not allow the wrong input, regular expression shouldn't allow letters to be inputted

### 3.2.2 Changes to Test Data

Some of the new test's test data will be included in the table below.

Test Number	Test Data	Justification for choice of test data
2.04.03	0	Program should not allow 0 as it would not make sense if there was a booking for 0 people
2.13	Rice	Program must allow the correct input
2.13.01	(Nothing)	User could accidentally try to proceed without entering anything which shouldn't be accepted by validation
2.14	7	Program must allow the correct input
2.14.01	0	Program must not allow 0 because it will not exist in the database
2.15	5	Program must allow the correct input
2.15.01	0	Program should not allow 0 as it would not make sense if there was a booking for 0 people
2.16	(Nothing)	User could accidentally try to proceed without entering anything which shouldn't be accepted by validation
2.16.01	Steak	Program must allow the correct input
2.17	10(Steak)	Program must allow the correct input
2.17.01	(Nothing)	User could accidentally try to proceed without entering anything which shouldn't be accepted by validation
2.17.02	645	Should allow the correct input but not do anything/error stating input is incorrect
2.18	(Nothing)	User could accidentally try to proceed without entering anything which shouldn't be accepted by validation
2.18.01	888	Should allow 888 as it is a valid input but not do anything as there is not a booking with an id of 888 in the current database used for testing
2.18.02	18	Should allow the correct input but not do anything/error stating input is incorrect
4.05	Have an item with a quantity of 1 then add 10 more of it	I have chosen 1 as the initial quantity and the additional 10 because it will be very clear if the adding quantity algorithm works.

### **3.3 Annotated Samples**

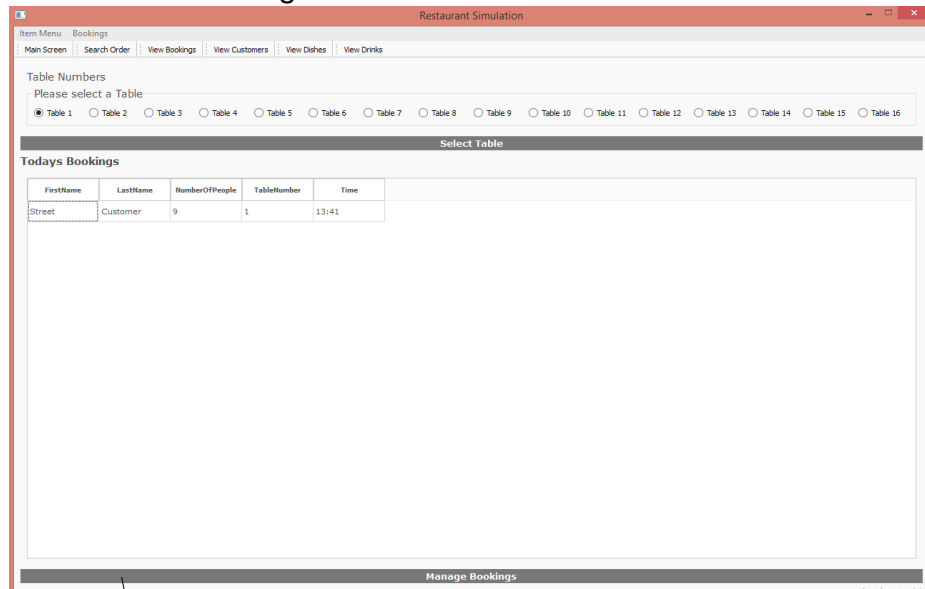
#### **3.3.1 Actual Results**

The actual results for the tests can be found on the detailed plans, there is a seperate column for it named 'Actual Results'.



### 3.3.2 Evidence

Below is an image of the main screen



Clicking on Manage Bookings will present you with the Manage Bookings layout as shown below

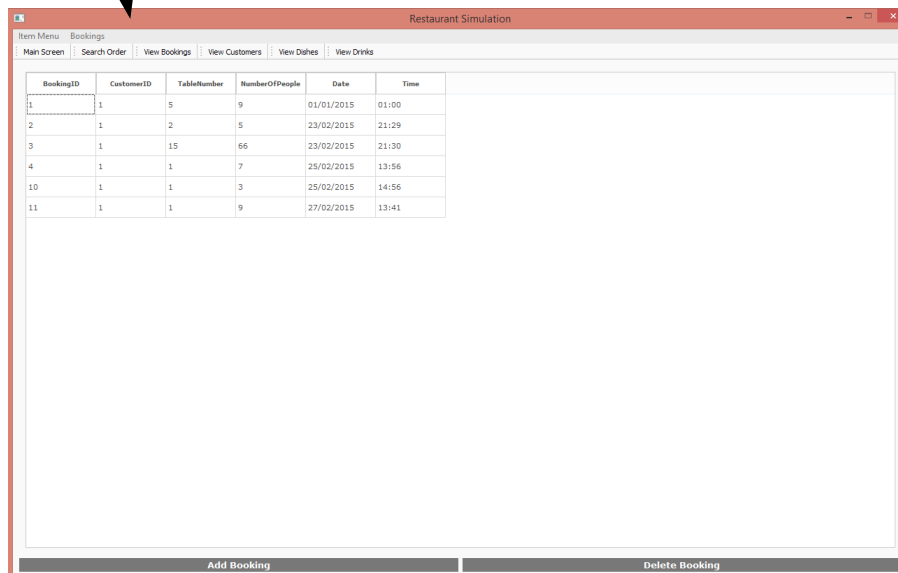


Figure 3.1: Manage Order layout



Clicking on Add Booking will present the Add Booking layout as shown below.

BookingID	CustomerID	TableNumber	NumberOfPeople	Date	Time
1	1	5	9	01/01/2015	01:00
2	1	2	5	23/02/2015	21:29
3	1	15	66	23/02/2015	21:30
4	1	1	7	25/02/2015	13:56
10	1	1	3	25/02/2015	14:56
11	1	1	9	27/02/2015	13:41

First Name:

Last Name:

Date:

Time:

Number Of People:

Telephone Number:

Table Number:

**Add Booking**

Figure 3.2: Add Booking



Clicking on Delete Booking will present the Delete Booking layout as shown below.

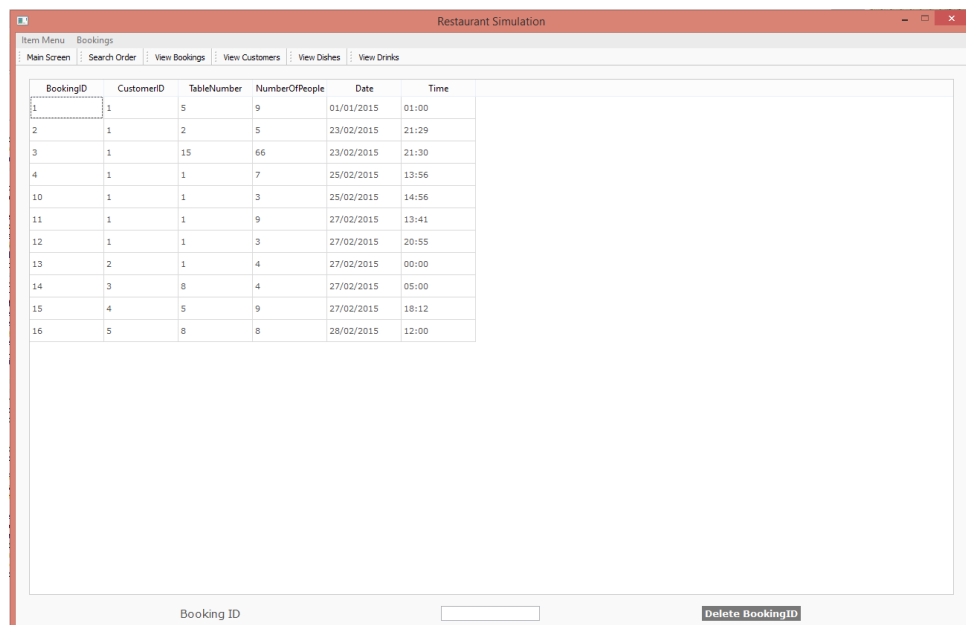
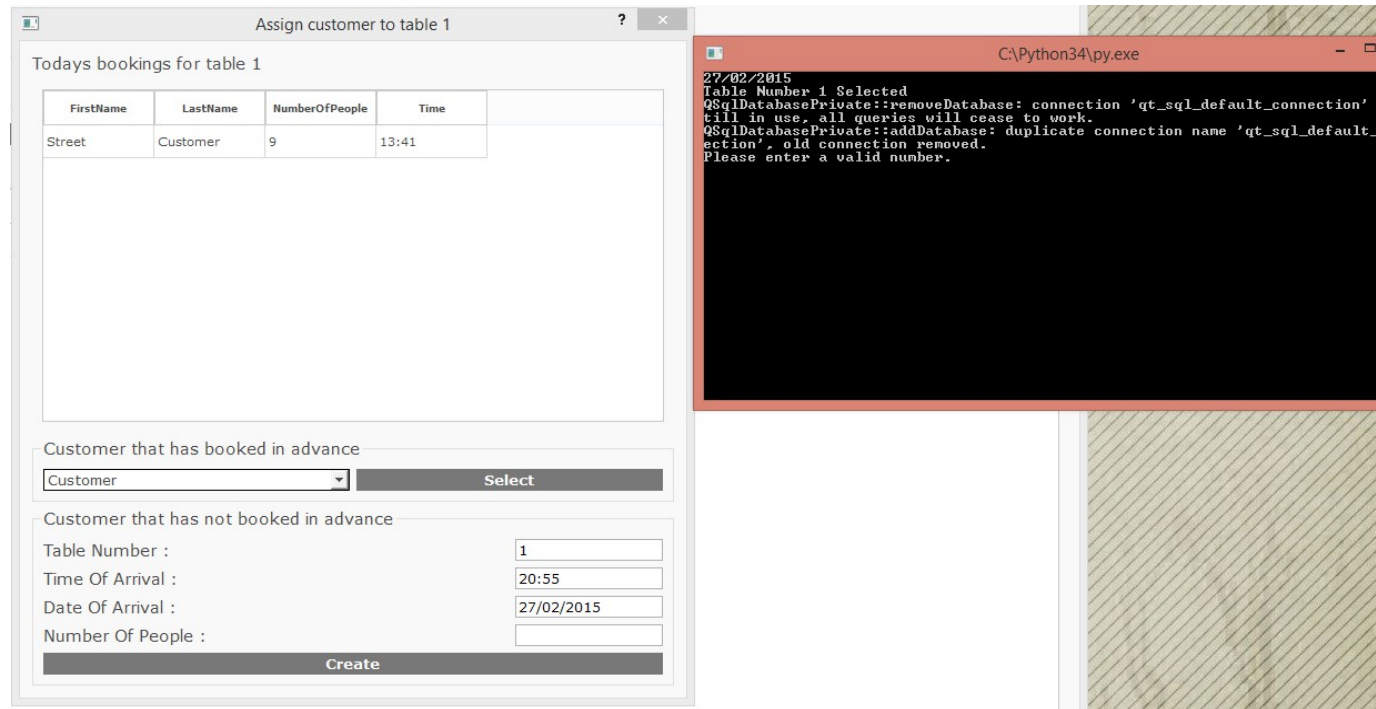


Figure 3.3: Delete Booking layout



120



Leaving the number of people field empty and clicking on create gave a print statement Of 'Please enter a valid number'. The booking was not created.

Figure 3.4: Assign customer validation

The image shows two software windows. The left window, titled "Assign customer to table 1", displays "Todays bookings for table 1" with a table containing one row: Street, Customer, 9, 13:41. Below this, there are sections for "Customer that has booked in advance" (with a dropdown menu showing "Customer" and a "Select" button) and "Customer that has not booked in advance" (with input fields for Table Number: 1, Time Of Arrival: 20:55, Date Of Arrival: 27/02/2015, and Number Of People: 3, followed by a "Create" button). The right window, titled "Manage Order", shows "Booking Information" (Table: 1, Date: 27/02/2015, Time: 20:55, Number of people: 3) and "Items Ordered" (Dishes and Drinks sections, each with a table for Quantity, ItemName, and ItemPrice). At the bottom, it shows "Total Price: 0" and buttons for Add, Delete, Finish, Invoice Preview, and Print Invoice. An arrow points from the "Create" button in the left window to the "Manage Order" window.

All relevant details such as table number, time of arrival, date of arrival and the number of people transferred after clicking Create.

Figure 3.5: Check all details transferred

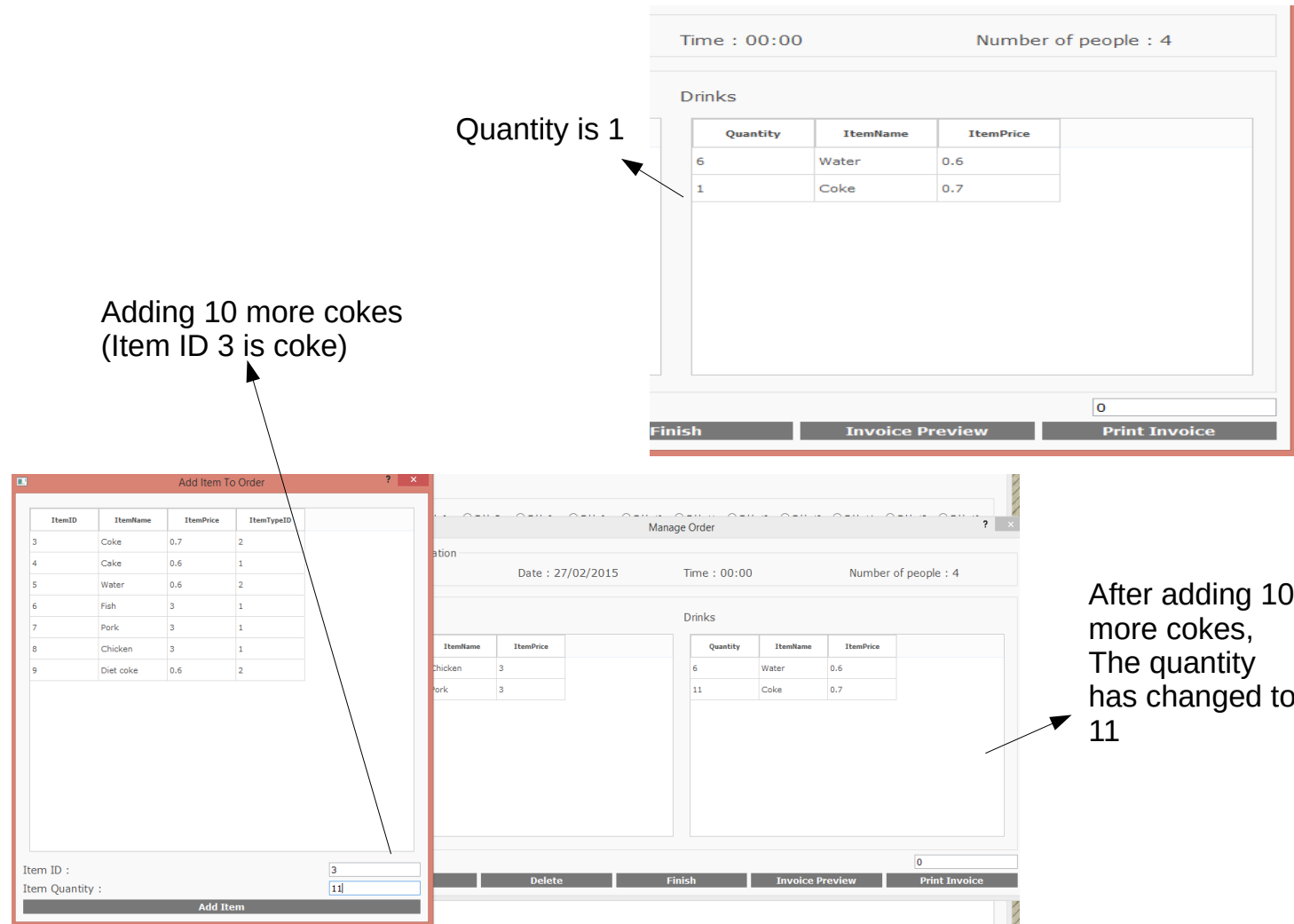


Figure 3.6: Quantity check

**Manage Order**

**Booking Information**  
 Table : 1      Date : 27/02/2015      Time : 00:00      Number of people : 4

**Items Ordered**

Dishes			Drinks		
Quantity	ItemName	ItemPrice	Quantity	ItemName	ItemPrice
3	Chicken	3	6	Water	0.6
3	Pork	3	1	Coke	0.7

Total Price :

**Linhs Chinese Restaurant**

48A CARTER STREET  
 FORDHAM - ELY  
 CAMBRIDGESHIRE  
 TEL: (01638) 721117

Date: 27/02/2015  
 Time: 00:00  
 Table Number: 1  
 Booking No. 13

Quantity	Item	Price (£)
3	Chicken	9.0
3	Pork	9.0
6	Water	3.5999999999999996
1	Coke	0.7

**Total Price : £22.3**

*All meal rates are inclusive of VAT  
 There is no Service Charge*

1

Clicking on Invoice Preview has made the preview pop up.  
 The details have been copied from the manage order to the invoice as shown

Figure 3.7: Invoice check

Assign customer to table 1

Today's bookings for table 1

FirstName	LastName	NumberOfPeople	Time
Street	Customer	9	13:41
Street	Customer	3	20:55
Milly	Milk	4	00:00

Customer that has booked in advance

Milk Select

Customer that has not booked in advance

Table Number :

Time Of Arrival :

Date Of Arrival :

Number Of People :

Create

Manage Order

Booking Information

Table : 1      Date : 27/02/2015      Time : 00:00      Number of people : 4

Items Ordered

Dishes

Quantity	ItemName	ItemPrice

Drinks

Quantity	ItemName	ItemPrice

Total Price :

Add Delete Finish Invoice Preview Print Invoice

Selecting the customer Milly Milk and pressing select has passed the details of the booking to the manage order box

Figure 3.8: Check the select function works ( passes the correct details of the selected customer

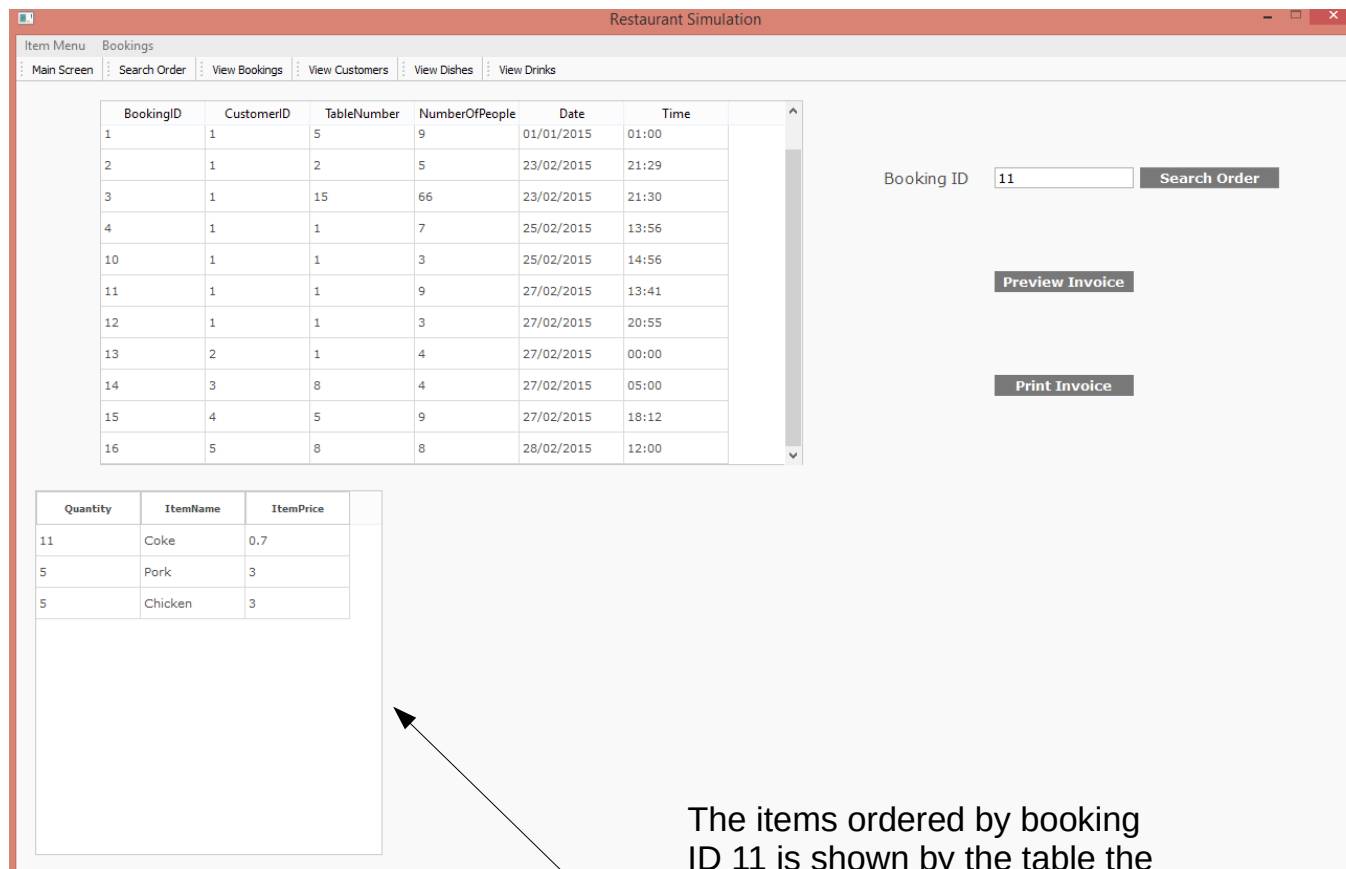


Figure 3.9: Checking the search function

Restaurant Simulation

Item Menu Bookings

Main Screen Search Order View Bookings View Customers View Dishes View Drinks

BookingID	CustomerID	TableNumber	NumberOfPeople	Date	Time
1	1	5	9	01/01/2015	01:00
2	1	2	5	23/02/2015	21:29
3	1	15	66	23/02/2015	21:30
4	1	1	7	25/02/2015	13:56
10	1	1	3	25/02/2015	14:56
12	1	1	3	27/02/2015	20:55
13	2	1	4	27/02/2015	00:00
14	3	8	4	27/02/2015	05:00
15	4	5	9	27/02/2015	18:12
16	5	8	8	28/02/2015	12:00

Booking ID

I have deleted booking ID 11 as you can see, booking id 11 is missing from the top table

Quantity	ItemName	ItemPrice
----------	----------	-----------

After deleting booking ID 11, I tested if the order would still be in the database but as you can see the table is empty.

Figure 3.10: Checking if the order is deleted after deleting booking ID 11



## **3.4 Evaluation**

### **3.4.1 Approach to Testing**

I made sure I tested my program thoroughly by going through different types of testing strategies. Going through different testing strategies made me cover most, if not all areas of my program such as testing the flow of control, validation, algorithms and the outputs. I chose this approach to ensure my whether my program was usable or not.

### **3.4.2 Problems Encountered**

I encountered a problem on test 4.03. The total price did not refresh after adding an item, it would only refresh after closing the manage order box and selecting the table again. However, the algorithm to calculate the total price was correct

### **3.4.3 Strengths of Testing**

The strengths of the approach I took were that I was able to find out whether my program was usable or not. Users of my system will undoubtedly make mistakes when inputting information and so I have tested different test data to ensure the system was still usable. In addition, checking the flow of control proved the navigation of the system to be effective.

### **3.4.4 Weaknesses of Testing**

When I tested the sql statements, I relied on the tables displayed on my application to tell me whether the sql statements worked without problems. For example, I deleted a booking on test 6.03 and checked to see if there was an order attached to the booking. The order did not display after searching for the booking but it may have still been in the database - which is what I didn't check. Also, the application has 16 radio buttons that represent the tables in the restaurant - I did not test all of them which would of meant that there could have been a fault with the application within the radio buttons that I did not test.

### **3.4.5 Reliability of Application**

I believe that the application is reliable. I have tested the input validation which worked as the system did not proceed if there was an invalid input which would mean there couldn't be any faulty data in the database. However, it would be

up to the User to input the correct data such as correctly spelling an item when adding an item to the menu or inputting the right ID when deleting a record.

### **3.4.6 Robustness of Application**

After testing my system without closing it, I believe that application is robust as I did not experience any crashes or any problems that made me not be able use the application normally. Testing the validation did not affect the application in anyway, same goes for the execution of the sql queries.

## Chapter 4

# System Maintenance

### 4.1 Environment

#### 4.1.1 Software

- Python 3
- IDLE
- PyQt
- SQLite 3
- SQLite Database Browser

#### 4.1.2 Usage Explanation

The table below includes all the listed software from the previous section with the explanation as to why I used them. The listed software can be downloaded for free which made the creation of my system an easier approach. Also, this would mean that my client wouldn't need to purchase anything.

Software	Usage Explanation
Python 3	I used Python because it was the only programming language I was familiar with as I learnt how to use it during my time at sixth form.
IDLE	I used IDLE to write the Python scripts and since I am the most familiar with this software , it made implementation of the client application easier.
PyQt	PyQt included everything needed to create the graphical user interface for my system and there was a lot of information on PyQt accessible which helped me create the graphical interface more effectively.
SQLite3	Although this came with python, I have used this to create the database and manipulate the database as it was easy for me to understand on how to do so and was very effective in doing so.
SQLite Inspector	This software helped me make the system's code relating the database a much more easier process because it allowed me to observe whether the database queries executed correctly or not.

### 4.1.3 Features Used

Software	Features Used
Python 3	I took advantage of the ability to import modules to structure my code clearer.
IDLE	There are countless number of features that IDLE has to offer which helped me create the application but I will mention a few of them. The syntax highlighter made it easier to understand the code that I was writing which is important when a system is complex(helps track what you are doing). Also it prevented me from making more errors. For example, I was able to spot out straight away if i misspelt a keyword such as print or while, etc. The 'Go to File/Line' feature when an error occurs, helped me tremendously as it was the main factor of helping me debug my program. Being able to run the system allowed me to test the system
PyQt	PyQt has many features that allowed me to create graphical user interface (GUI) for my system. The core components that I used to create the GUI were main windows, dialog boxes and widgets.
SQLite3	I was able to create the database for the system through SQLite3. I used most of core features that was available to me through SQLite3 such as being able to ADD, DELETE,UPDATE to/from the database. Enforcing referential integrity was useful as it helped the database to become consistent.
SQLite Inspector	I mainly used 'Browse Data' to check whether I have added, deleted or updated a record successfully. I also used 'Execute SQL' for the SELECT statements in the application as it allowed me to see whether the SQL query was correct or not.

## 4.2 System Overview

### 4.2.1 System Component

#### Graphical User Interface (GUI)

Having a graphical user interface for the system makes it a lot more usable, giving the user a much more user friendly experience. Including a GUI makes it easier for the user to navigate around the system.

### **Manage Item Menu**

The item menu can be managed at any time through the menu bar 'Item Menu'. The user can add/delete/update an item.

To add an item to the menu, the user must select 'Add Item'. By selecting 'Add Item', the user will be presented with a layout that consists of a table widget displaying all the records of the menu and the fields which will be used to input information for the new item.

Deleting an item off the menu can also be found under the 'Item Menu' menu by selecting 'Delete Item'. The user will be presented with a layout that contains the same table widget that displays all records of the menu and has either the choose to delete an item by inputting the item name or the item ID.

The user also has the option to update an item's price. To do this, the user must select 'Update Item Price' where the user will be presented with a layout that contains the same the item menu table widget as the add and delete item layout. The user would have to input the ID of the item and the new price then click on the 'Update Item' button to update an item's price.

### **Manage Bookings**

The user will be able to add/delete and update bookings. To do this, the user could either selection these options through "Bookings" on the menu bar or click on the 'Manage Bookings' button at the bottom of the main screen.

Clicking on the button will switch the central widget to the manage bookings widget where the user will be presented with the Bookings table widget where all of the booking records will be displayed and below the widget are the buttons "Add Booking" and "Delete Booking". Clicking on Add Booking will then present the user with the same table widget and the required fields which the user would have to successfully fill to add a booking. As for the "Delete Booking" button, the user will be presented with the same Bookings table widget and a input field for the user to delete a booking by inputting a booking id and pressing "Delete BookingID".

The "Bookings" menu bar also has 3 options; "Add Bookings", "Delete Booking" and "Update Booking". The add/delete booking options are the same as described in the paragraph above. As for the "Update Booking" option, the user will be presented with the usual Bookings table widget and the input fields to update the booking.

**Manage Sit-In Orders**

To manage an order, the user must select the table and if not already, assign a customer to the table ( A dialog box will pop up telling the user to assign a customer to the selected table). After assigning a customer to the table, the table will be known as 'occupied' which would allow the user to select that table without assigning a customer to that table everytime. So now that the table is occupied, there will be a manage order box where the booking details will be displayed on a row at the top of the box. The dishes and drinks ordered will be split into two table widget, the dishes ordered will be displayed on the left and the drinks on the right.

The user has all the necessary options on the manage order box such as "Add", "Delete", "Finish", "Invoice Preview" and "Print Invoice". The "Add" button is for adding items to the order, the "Delete" button will be used to delete items off the order, the "Invoice Preview" will show the user what the invoice would look like for the order, the "Print Invoice" will print the invoice and the "Finish" button will set the status of the table as unoccupied, clearing the booking details for that table and so the user would have to assign a customer to that table when selecting the table from the main screen.

## 4.3 Code Structure

### 4.3.1 Particular Code Section

#### 4.3.2 Displaying a table

---

```
1      def show_results(self, query):
2          self.display_results_layout()
3          if not self.model or not
4              isinstance(self.model, QSqlQueryModel):
5              self.model = QSqlQueryModel()
6          self.model.setQuery(query)
7          self.results_table.setModel(self.model)
8          self.results_table.show()
9
10     def show_table(self, tableName):
11         self.display_results_layout()
12         if not self.model or not
13             isinstance(self.model, QSqlTableModel):
14             self.model = QSqlTableModel()
15         self.model.setTable(tableName)
16         self.model.select()
17         self.results_table.setModel(self.model)
18         self.results_table.show()
```

---

Above are the `show_results` and `show_table` functions which is part of the `table_display.py` module. I needed to display tables on many widgets and so I created a module based around the `show_results` and `show_table` so that I all had to do was import this module and call the functions whenever I needed to display a table.



### 4.3.3 Switching central widgets

---

```

1      def update_item_connect(self):
2          self.update_item = UpdateItemPrice()
3          self.setCentralWidget(self.update_item)
4
5      def add_booking_connect(self):
6          self.add_booking = AddBookingWindow()
7          self.setCentralWidget(self.add_booking)
8
9      def delete_booking_connect(self):
10         self.delete_booking = DeleteBookingWindow()
11         self.setCentralWidget(self.delete_booking)

```

---

The functions shown above are used to switch the central widgets. I have created this in such a way so that I could just call the function whenever there was a click connection. Also, this was helpful because since there is a menu bar, the menu bar is used as an alternative to switch the central widget and so I could just call the connect functions above.

## 4.4 Variable Listing

Variable Name	Purpose	Line numbers	Section
regExp	Holds the regular expression to only allow the user to enter letters for name inputs	31	4.10.1
regex	Holds the regular expression to only allow the user to input digits	42	4.10.1
regex2	Holds the regular expression to only allow the user to input digits	48	4.10.1
each	Stepper variable used in the for loop	57, 58	4.10.1
self.maximumdate	Used to set maximum date for QDateEdit	63	4.10.1
self.minimumdate	Used to set minimum date for QDateEdit	64	4.10.1
FirstName	Used to store user input	106, 119 ,121	4.10.1
LastName	Used to store user input	107, 119, 121	4.10.1
TeleNumber	Used to store user input	108, 119, 121	4.10.1

NumberOfPeople	Used to store user input	110, 119, 135	4.10.1
TableNumber	Holds the index of the table number combo box	113, 129, 135	4.10.1
BookingDate	Used to store user input	114, 135	4.10.1
BookingTime	Used to store user input	115, 135	4.10.1
customer	Holds the variables First-Name, LastName and TeleNumber to create a record for Customers	121	4.10.1
db	Stores the path of the database	123, 129, 137	4.10.1
booking	Holds the variables customerid, TableNumber, NumberOfPeople, BookingDate and BookingTime to create a record for Bookings	135	4.10.1
regexpp	Holds the regular expression to only allow a maximum of 20 letters	35	4.10.2
ItemName	Used to store user input	72	4.10.2
ItemPrice	Used to store user input	73	4.10.2
ItemType	Holds the selected index of the item type combo box	73, 74, 76, 78, 79	4.10.2
MenuItem	Holds the variables ItemName, ItemPrice and ItemType to create a record for Items	79	4.10.2
self.bookingDetails	Holds booking details such as bookingid, customerid, booking date, booking time, table number and number of people	16	4.10.3
bookingID	Holds booking id	57	4.10.3
self.ItemID	Used to store user input	58	4.10.3
Quantity	Used to store user input	59	4.10.3
addedAlready	Holds a boolean value to indicate whether an item has been added to the order already (Used to increase quantity)	61, 66, 83, 97, 128, 130	4.10.3
newQuantity	Used to calculate the new quantity after adding/deleting an item that's already been added to an order	72	4.10.3

updateOrder	Holds the new quantity and item id to update the record	73	4.10.3
itemsOrdered	Holds an array of ordered items for a particular booking	98, 111	4.10.3
css	Holds the cascade style sheet code for the application	19	4.10.4
Date	Holds the system date	120	4.10.4
Time	Holds the system time	121	4.10.4
self.CustomerList	Holds an array of customer id's that has a booking on a certain table and date	152, 166, 176	4.10.4
CustomerLastName	Holds an array of last names from the customer ids in self.CustomerList	167, 184, 188	4.10.4
OneQuantity	Holds a boolean value to indicate whether the quantity of an ordered item is one or not	82, 84, 112, 124, 117	4.10.8
self.TableOne Occupied	Holds a boolean value to indicate whether table one is occupied or not	34, 227	4.10.9
self.TableTwo Occupied	Holds a boolean value to indicate whether table two is occupied or not	35, 241	4.10.9
self.TableThree Occupied	Holds a boolean value to indicate whether table three is occupied or not	36, 255	4.10.9
self.TableFour Occupied	Holds a boolean value to indicate whether table four is occupied or not	37, 269	4.10.9
self.TableFive Occupied	Holds a boolean value to indicate whether table five is occupied or not	38, 283	4.10.9
self.TableSix Occupied	Holds a boolean value to indicate whether table six is occupied or not	39, 297	4.10.9
self.TableSeven Occupied	Holds a boolean value to indicate whether table seven is occupied or not	40, 312	4.10.9
self.TableEight Occupied	Holds a boolean value to indicate whether table eight is occupied or not	41, 326	4.10.9
self.TableNine Occupied	Holds a boolean value to indicate whether table nine is occupied or not	42, 340	4.10.9

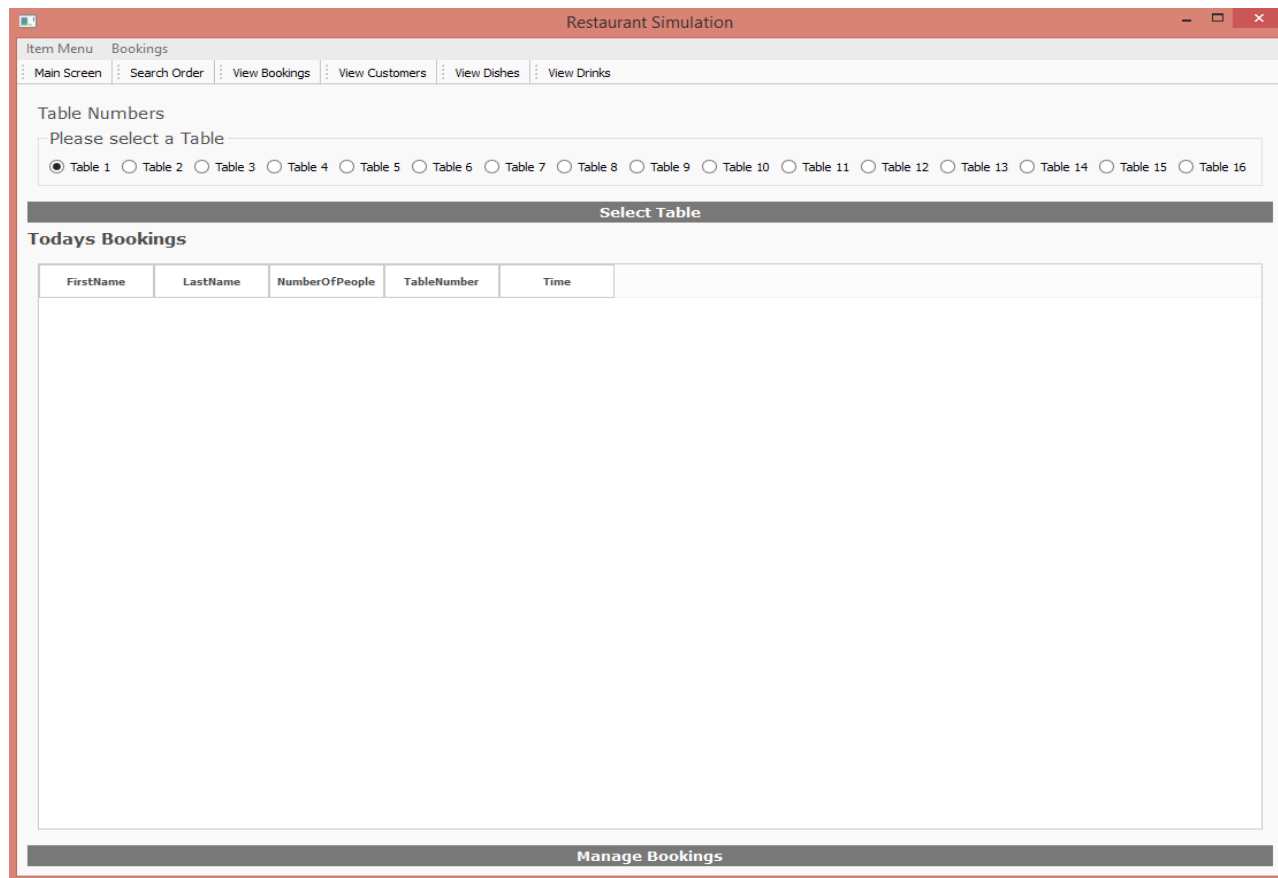
self.TableTen Occupied	Holds a boolean value to indicate whether table ten is occupied or not	43, 354	4.10.9
self.TableEleven Occupied	Holds a boolean value to indicate whether table eleven is occupied or not	44, 368	4.10.9
self.TableTwelve Occupied	Holds a boolean value to indicate whether table twelve is occupied or not	45, 382	4.10.9
self.TableThirteen Occupied	Holds a boolean value to indicate whether table thirteen is occupied or not	46, 396	4.10.9
self.TableFourteen Occupied	Holds a boolean value to indicate whether table fourteen is occupied or not	47, 410	4.10.9
self.TableFifteen Occupied	Holds a boolean value to indicate whether table fifteen is occupied or not	48, 424	4.10.9
self.TableSixteen Occupied	Holds a boolean value to indicate whether table sixteen is occupied or not	49, 438	4.10.9
tableList	Used to store the number of tables for the radio button widget	154,156	4.10.9
Todays Date	Holds the system date	171	4.10.9
self.Finished	Holds a boolean value to indicate whether the table/booking has finished	24, 188	4.10.11
self.TotalPrice	Used to store the total price of all ordered items	149, 181	4.10.11
self.price	Holds an array of all the prices of all ordered items	150, 160, 162	4.10.11
self.quantity	Holds an array of quantities of all ordered items	151, 173, 174, 175	4.10.11

## 4.5 System Evidence

### 4.5.1 User Interface

140

## Main Screen



The screenshot shows a web application titled "Restaurant Simulation". It features a top navigation bar with "Item Menu" and "Bookings" tabs. Below this is a secondary navigation bar with links: "Main Screen", "Search Order", "View Bookings", "View Customers", "View Dishes", and "View Drinks". The main content area is divided into three sections: "Table Numbers" with a "Please select a Table" prompt and radio buttons for Tables 1 through 16 (Table 1 is selected); a "Select Table" button; and "Todays Bookings" which contains a table with headers "FirstName", "LastName", "NumberOfPeople", "TableNumber", and "Time". The table body is empty. At the bottom is a "Manage Bookings" button.

Restaurant Simulation

Item Menu Bookings

... Main Screen ... Search Order ... View Bookings ... View Customers ... View Dishes ... View Drinks

Table Numbers

Please select a Table

☒ Table 1 ☐ Table 2 ☐ Table 3 ☐ Table 4 ☐ Table 5 ☐ Table 6 ☐ Table 7 ☐ Table 8 ☐ Table 9 ☐ Table 10 ☐ Table 11 ☐ Table 12 ☐ Table 13 ☐ Table 14 ☐ Table 15 ☐ Table 16

Select Table

Todays Bookings

FirstName	LastName	NumberOfPeople	TableNumber	Time
-----------	----------	----------------	-------------	------

Manage Bookings

Figure 4.1:

Restaurant Simulation

Item Menu Bookings

Main Screen Search Order View Bookings View Customers View Dishes View Drinks

BookingID	CustomerID	TableNumber	NumberOfPeople	Date	Time
1	1	5	3	02/01/2015	03:00
2	1	2	5	23/02/2015	21:29
3	1	15	66	23/02/2015	21:30
4	1	1	7	25/02/2015	13:56
10	1	1	3	25/02/2015	14:56
12	1	1	3	27/02/2015	20:55
13	2	1	4	27/02/2015	00:00
14	3	8	4	27/02/2015	05:00
15	4	5	9	27/02/2015	18:12
16	5	8	8	28/02/2015	12:00
17	1	1	7	28/02/2015	21:44
18	1	1	5	03/03/2015	19:42

Add Booking Delete Booking

Manage Bookings that was accessed from the main screen

Figure 4.2:

Restaurant Simulation

Item Menu Bookings

Main Screen Search Order View Bookings View Customers View Dishes View Drinks

BookingID	CustomerID	TableNumber	NumberOfPeople	Date	Time
1	1	5	3	02/01/2015	03:00
2	1	2	5	23/02/2015	21:29
3	1	15	66	23/02/2015	21:30
4	1	1	7	25/02/2015	13:56
10	1	1	3	25/02/2015	14:56
12	1	1	3	27/02/2015	20:55
13	2	1	4	27/02/2015	00:00
14	3	8	4	27/02/2015	05:00
15	4	5	9	27/02/2015	18:12
16	5	8	8	28/02/2015	12:00
17	1	1	7	28/02/2015	21:44
18	1	1	5	03/03/2015	19:42

First Name:

Last Name:

Date:

Time:

Number Of People:

Telephone Number:

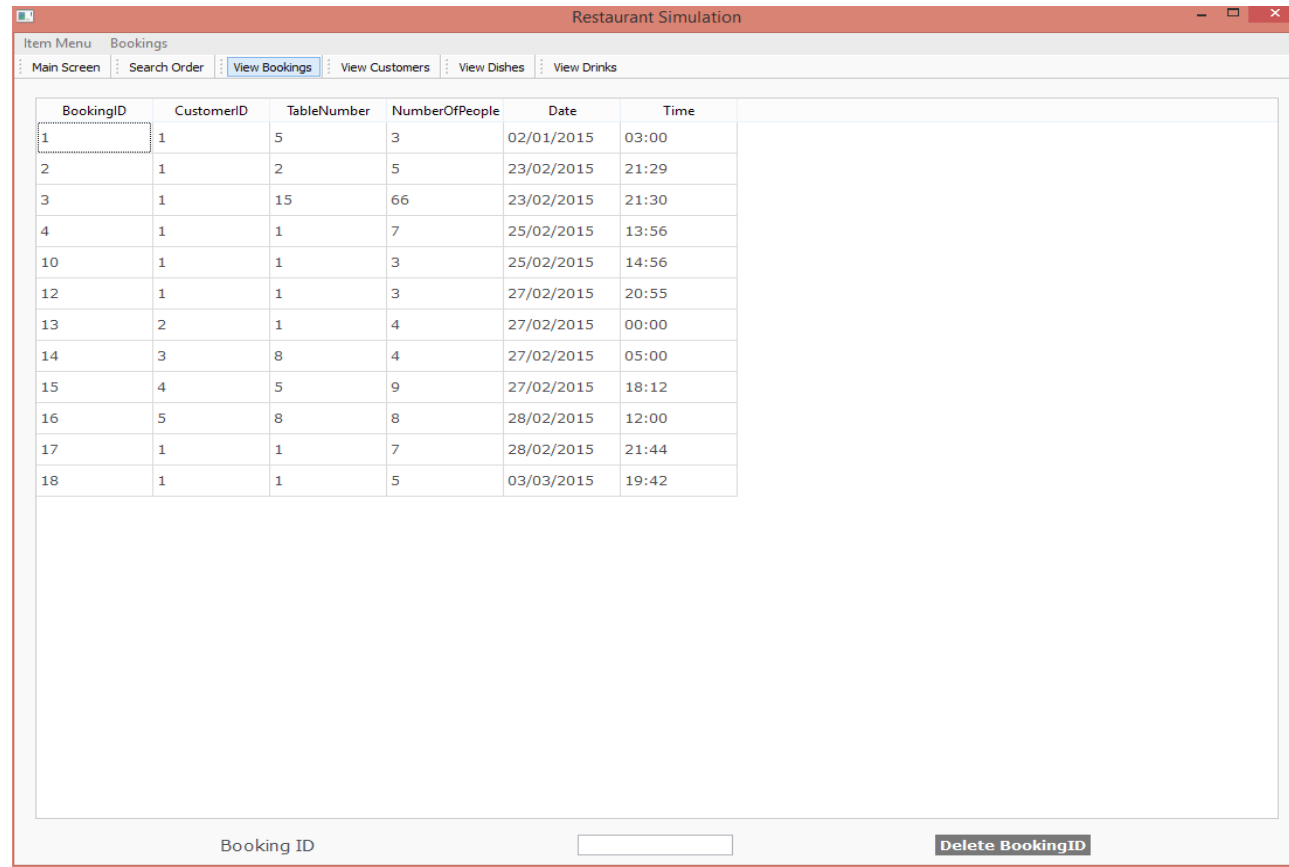
Table Number:

Add Booking

Add Booking screen which can be accessed from Bookings on the menu bar or the Manage Bookings screen

Figure 4.3:





Delete Booking screen which can be accessed from Bookings on the menu bar or the Manage Bookings screen

Figure 4.4:

Restaurant Simulation

Item Menu Bookings

Main Screen Search Order View Bookings View Customers View Dishes View Drinks

ItemID	ItemName	ItemPrice	ItemTypeID
3	Coke	0.7	2
4	Cake	0.6	1
5	Water	0.6	2
6	Fish	3	1
7	Pork	3	1
8	Chicken	3	1
9	Diet coke	0.6	2
10	Steak	4	1

Item Name :

Item Price :

Item Type :

Add Item

Add Item screen which can be accessed from  
Item Menu on the menu bar

Figure 4.5:

Restaurant Simulation

Item Menu Bookings

Main Screen Search Order View Bookings View Customers View Dishes View Drinks

ItemID	ItemName	ItemPrice	ItemTypeID
3	Coke	0.7	2
4	Cake	0.6	1
5	Water	0.6	2
6	Fish	3	1
7	Pork	3	1
8	Chicken	3	1
9	Diet coke	0.6	2
10	Steak	4	1

Item Name :  **Delete Item**

Item ID :  **Delete Item**

Delete Item screen which can be accessed from  
Item Menu on the menu bar

Figure 4.6:

ItemID	ItemName	ItemPrice	ItemTypeID
3	Coke	0.7	2
4	Cake	0.6	1
5	Water	0.6	2
6	Fish	3	1
7	Pork	3	1
8	Chicken	3	1
9	Diet coke	0.6	2
10	Steak	4	1

Item ID :

New Item Price :

**Update Item**

Update Item Price screen which can be accessed from  
Item Menu on the menu bar

Figure 4.7:

Restaurant Simulation

Item Menu Bookings

Main Screen Search Order View Bookings View Customers View Dishes View Drinks

BookingID	CustomerID	TableNumber	NumberOfPeople	Date	Time
1	1	5	3	02/01/2015	03:00
2	1	2	5	23/02/2015	21:29
3	1	15	66	23/02/2015	21:30
4	1	1	7	25/02/2015	13:56
10	1	1	3	25/02/2015	14:56
12	1	1	3	27/02/2015	20:55
13	2	1	4	27/02/2015	00:00
14	3	8	4	27/02/2015	05:00
15	4	5	9	27/02/2015	18:12
16	5	8	8	28/02/2015	12:00
17	1	1	7	28/02/2015	21:44
18	1	1	5	03/03/2015	19:42

Booking ID you want to update:

Date:

Time:

Number Of People:

Table Number:

Update Booking screen which can be accessed from  
Item Menu on the menu bar

Figure 4.8:

Assign customer box after clicking on Select Table on main screen
----------------------------------------------------------------------

Figure 4.9:

Restaurant Simulation

Item Menu Bookings

Main Screen Search Order View Bookings View Customers View Dishes View Drinks

Table Numbers

**Manage Order**

Booking Information

Table : 1 Date : 05/03/2015 Time : 19:45 Number of people : 5

Items Ordered

Dishes

Quantity	ItemName	ItemPrice

Drinks

Quantity	ItemName	ItemPrice

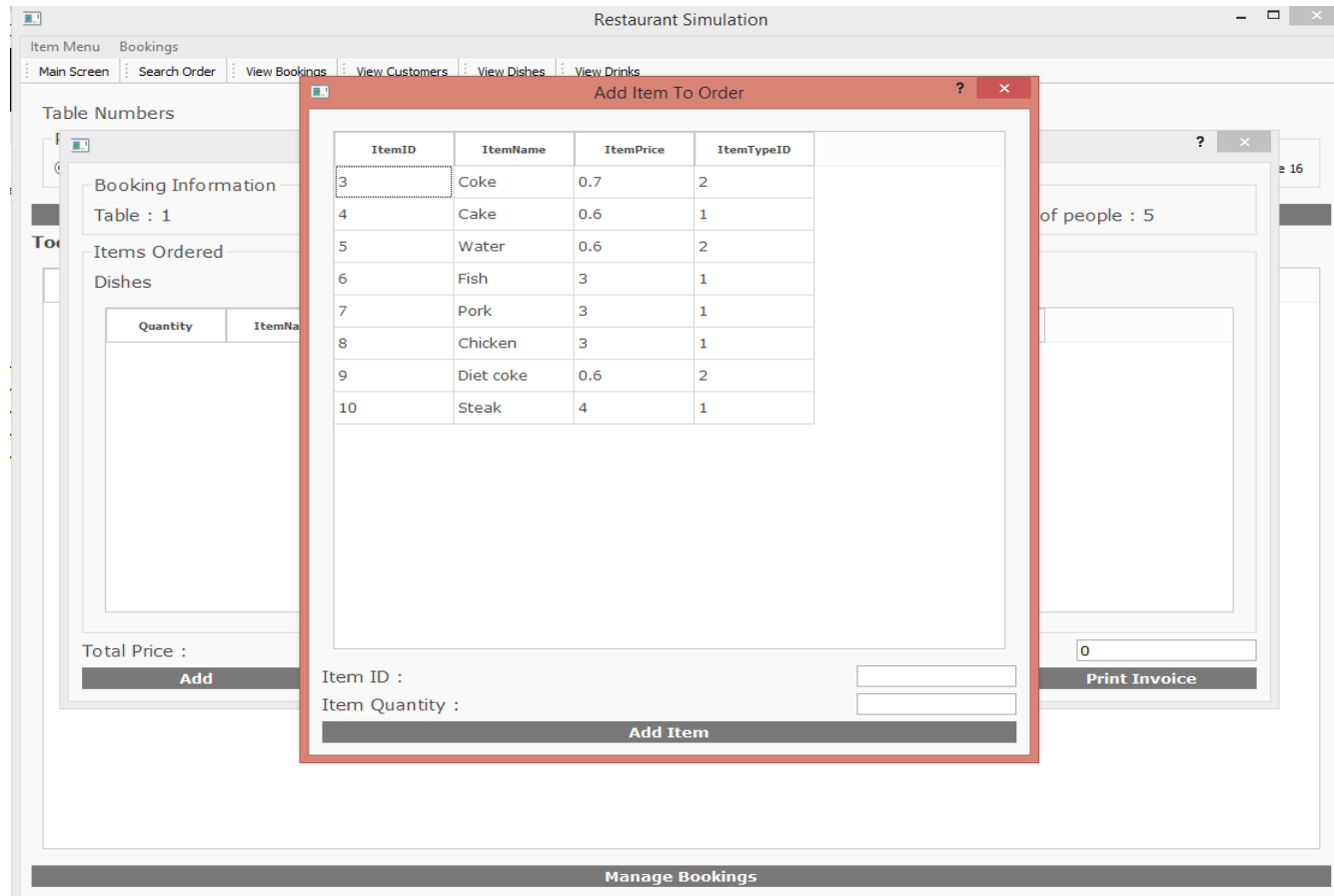
Total Price : 0

Add Delete Finish Invoice Preview Print Invoice

Manage Bookings

Manage Order box after clicking on  
Create from the assign customer box

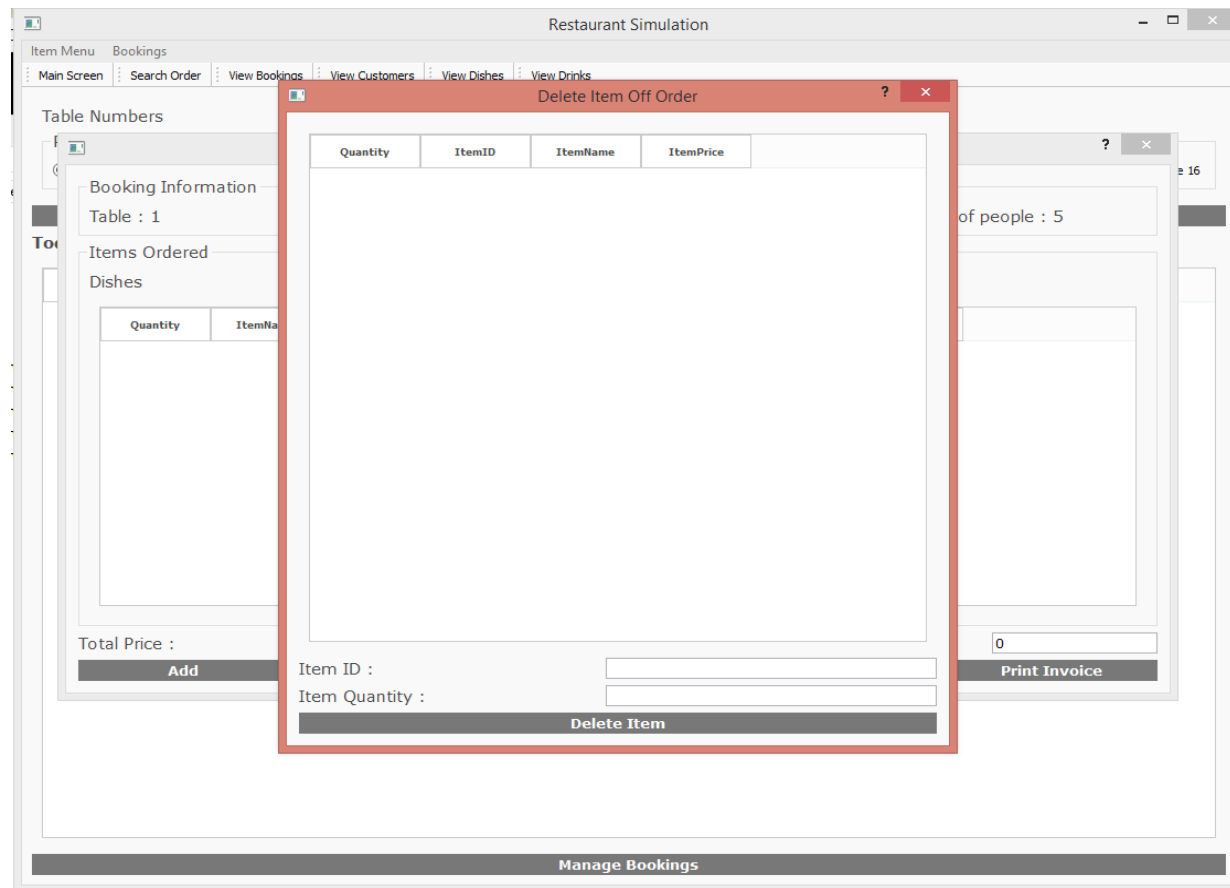
Figure 4.10:



Add Item To Order box after clicking on  
Add from the Manage Order box

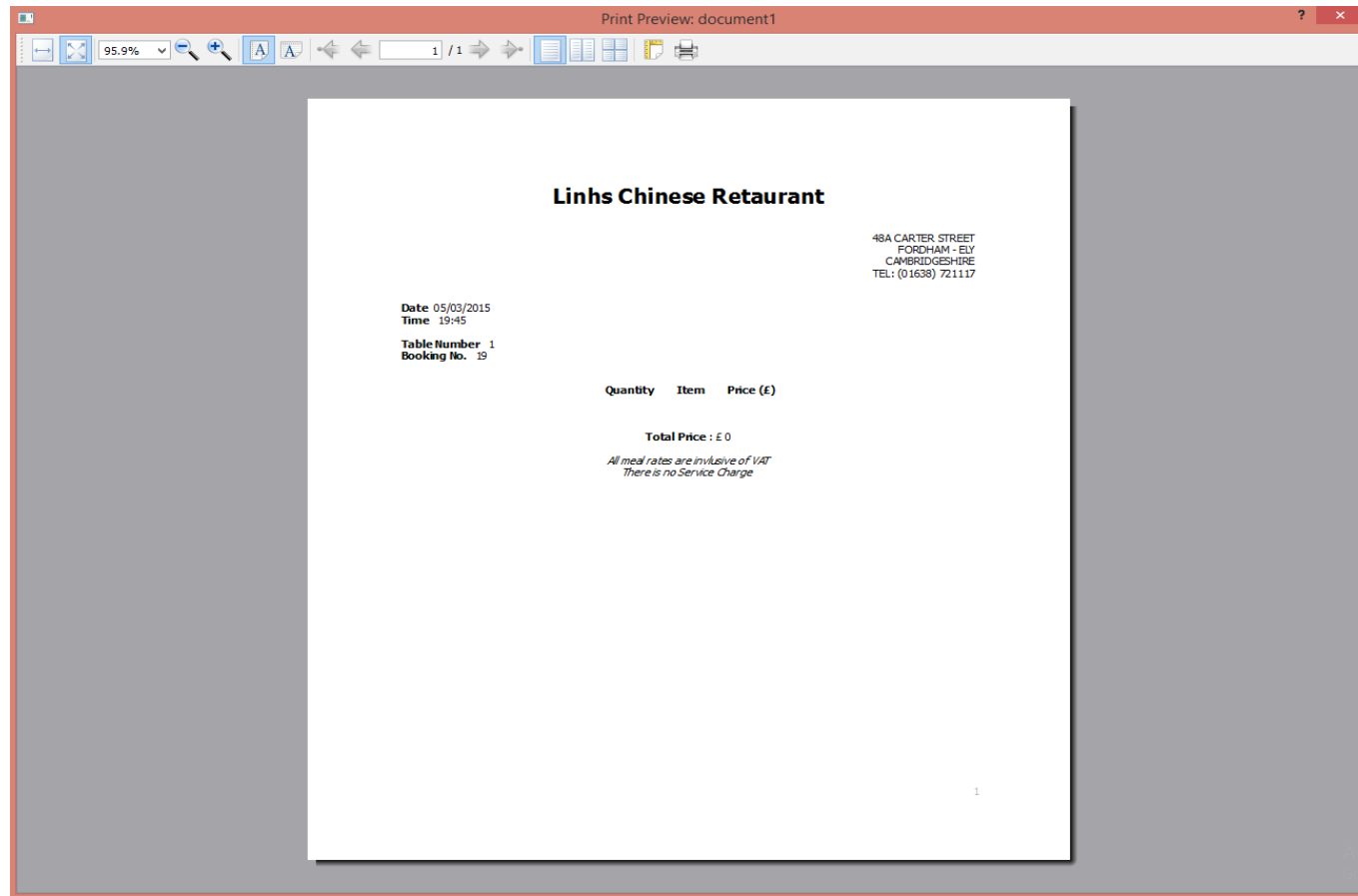
Figure 4.11:





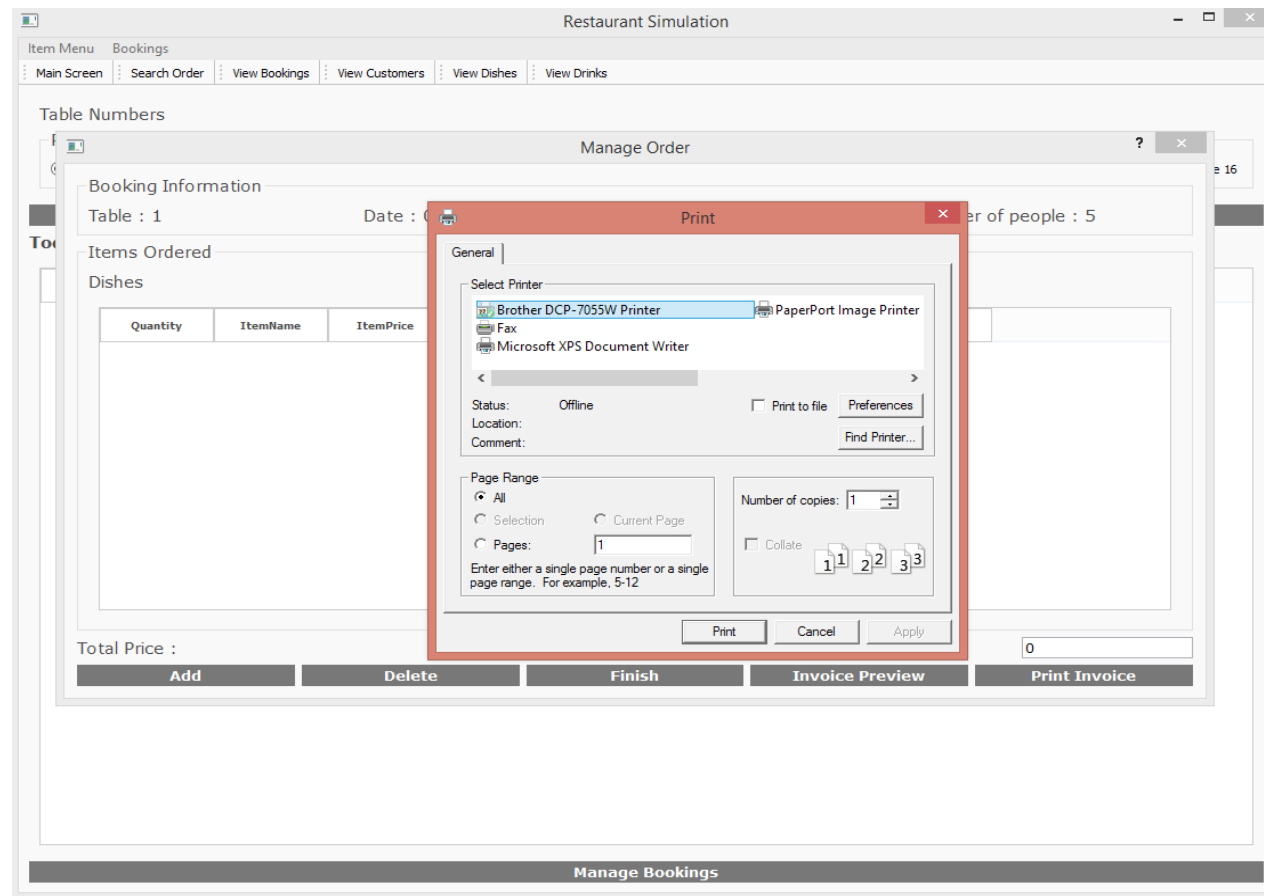
Delete Item To Order box after clicking on Delete from the Manage Order box (The items ordered will be shown in the table)

Figure 4.12:



The preview of the invoice relating to the order after clicking on Invoice Preview from the Manage Order box

Figure 4.13:



Print options before printing after clicking on  
Print Invoice from the Manage Order box

Figure 4.14:

**Restaurant Simulation**

Item Menu Bookings

Main Screen Search Order View Bookings View Customers View Dishes View Drinks

**Table Numbers**

**Manage Order**

Booking Information

Table : 1 Date : 05/03/2015 Time : 19:45 Number of people : 5

Items Ordered

Dishes

Quantity	ItemName	ItemPrice

Drinks

Quantity	ItemName	ItemPrice

Total Price : 0

Add Delete Finish Invoice Preview Print Invoice

**Manage Bookings**

(Before clicking on the Select button)

Figure 4.15:

Restaurant Simulation

Item Menu Bookings

Main Screen Search Order View Bookings View Customers View Dishes View Drinks

Table Numbers

**Manage Order**

Booking Information

Table : 1 Date : 05/03/2015 Time : 19:45 Number of people : 5

Items Ordered

Dishes

Quantity	ItemName	ItemPrice

Drinks

Quantity	ItemName	ItemPrice

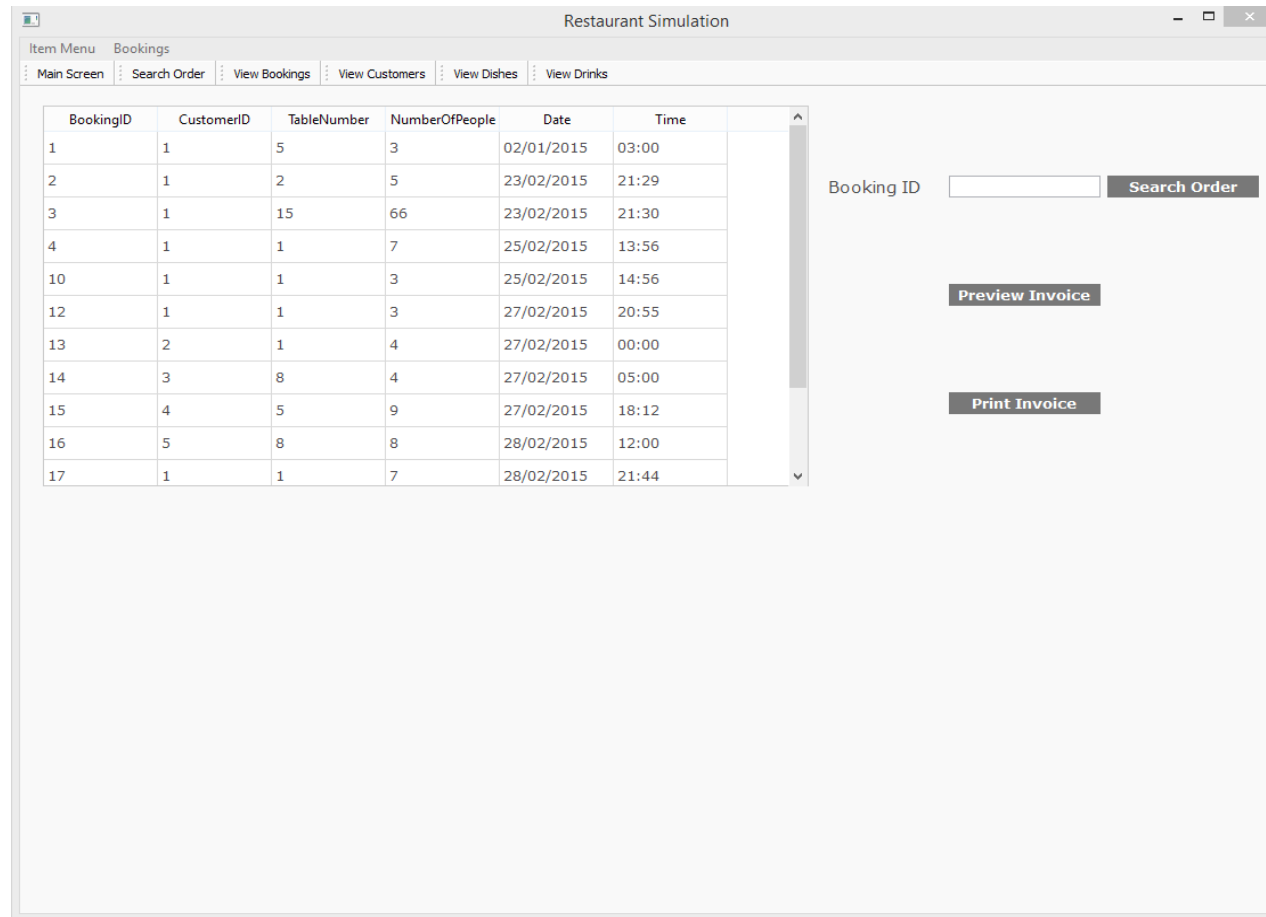
Total Price : 0

Add Delete Finish Invoice Preview Print Invoice

Manage Bookings

(After clicking on the Select button)

Figure 4.16:



Search Order screen which can be found on the Tool Bar

Figure 4.17:

Restaurant Simulation

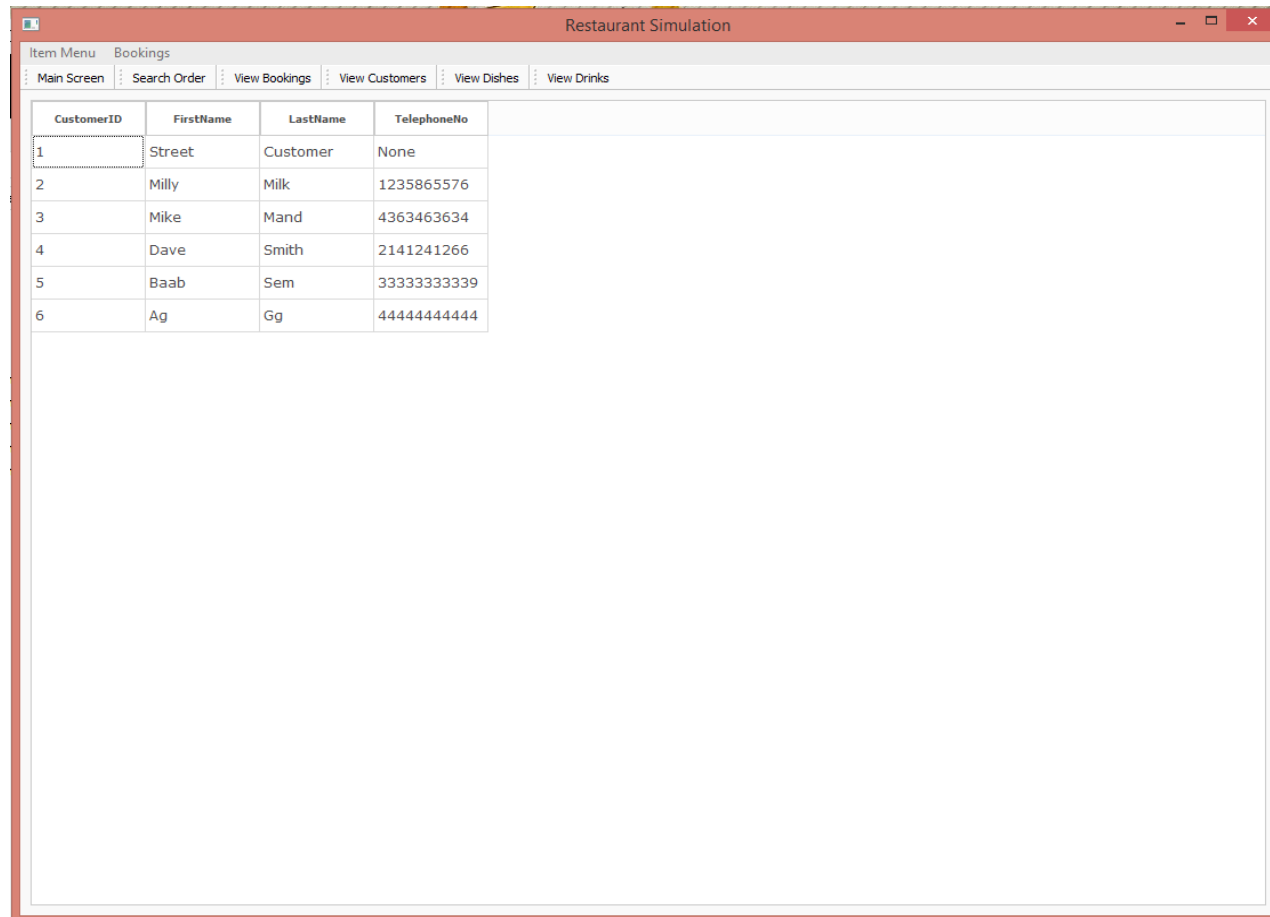
Item Menu Bookings

Main Screen Search Order View Bookings View Customers View Dishes View Drinks

BookingID	CustomerID	FirstName	LastName	NumberOfPeople	TableNumber	Time	Date	TelephoneNo
1	1	Street	Customer	3	5	03:00	02/01/2015	None
18	1	Street	Customer	5	1	19:42	03/03/2015	None
19	1	Street	Customer	5	1	19:45	05/03/2015	None
2	1	Street	Customer	5	2	21:29	23/02/2015	None
3	1	Street	Customer	66	15	21:30	23/02/2015	None
4	1	Street	Customer	7	1	13:56	25/02/2015	None
10	1	Street	Customer	3	1	14:56	25/02/2015	None
13	2	Milly	Milk	4	1	00:00	27/02/2015	1235865576
14	3	Mike	Mand	4	8	05:00	27/02/2015	4363463634
15	4	Dave	Smith	9	5	18:12	27/02/2015	2141241266
12	1	Street	Customer	3	1	20:55	27/02/2015	None
16	5	Baab	Sem	8	8	12:00	28/02/2015	3333333339
17	1	Street	Customer	7	1	21:44	28/02/2015	None

View Bookings screen which can be found on the Tool Bar

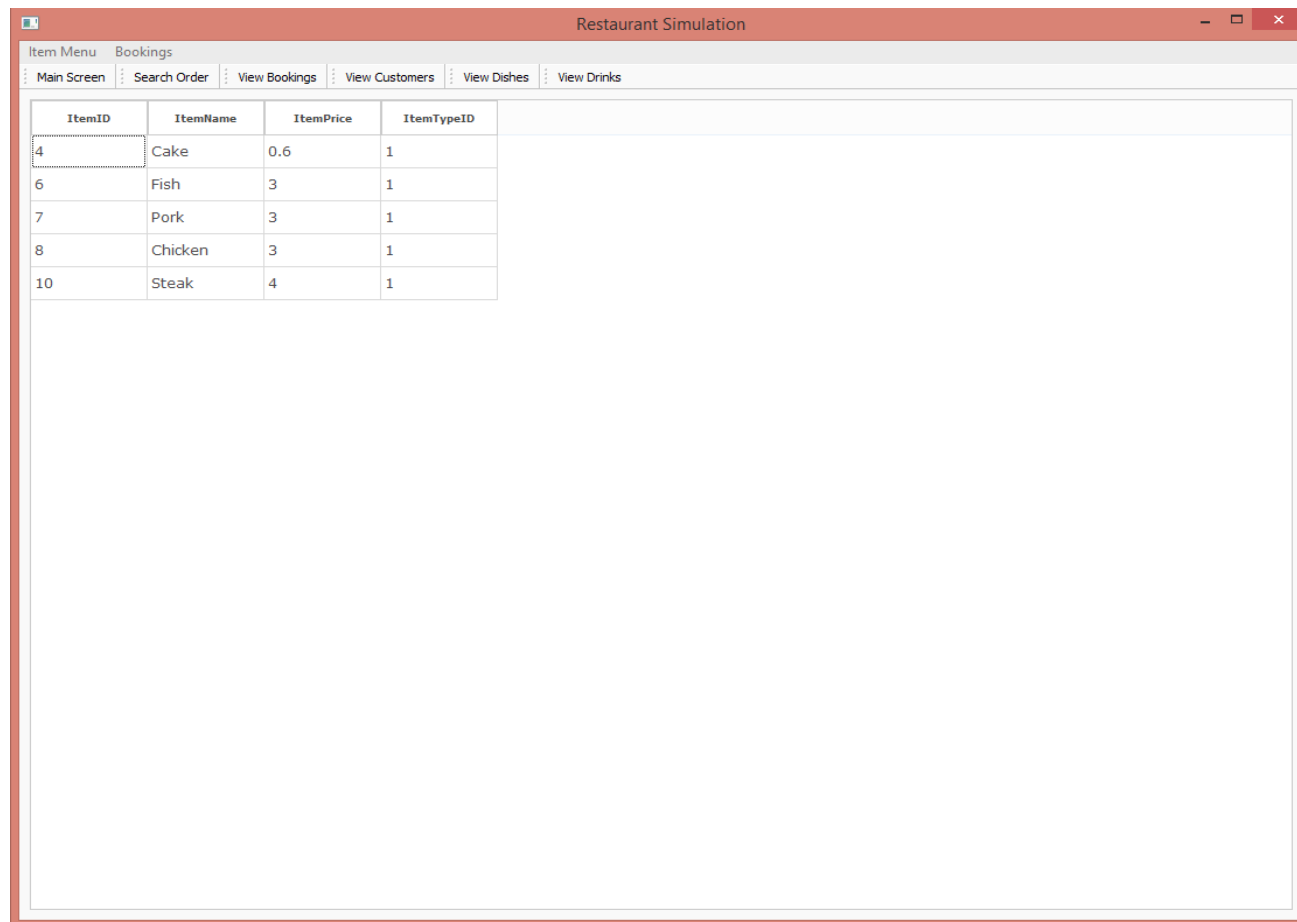
Figure 4.18:



View Customers screen which can be found on the Tool Bar

Figure 4.19:





View Dishes screen which can be found on the Tool Bar

Figure 4.20:

ItemID	ItemName	ItemPrice	ItemTypeID
3	Coke	0.7	2
5	Water	0.6	2
9	Diet coke	0.6	2

View Dishes screen which can be found on the Tool Bar

Figure 4.21:

#### 4.5.2 ER Diagram

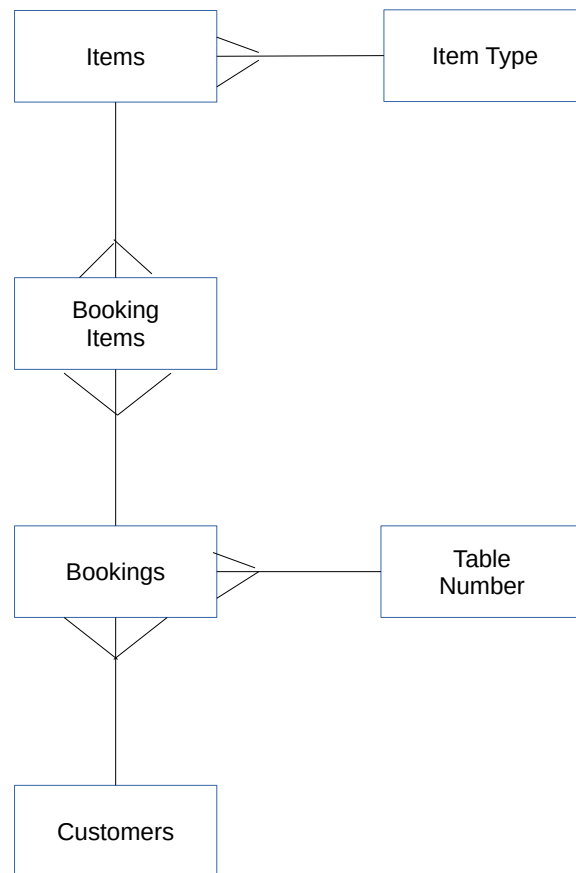


Figure 4.22: ER Diagram of finished database

### 4.5.3 Database Table Views

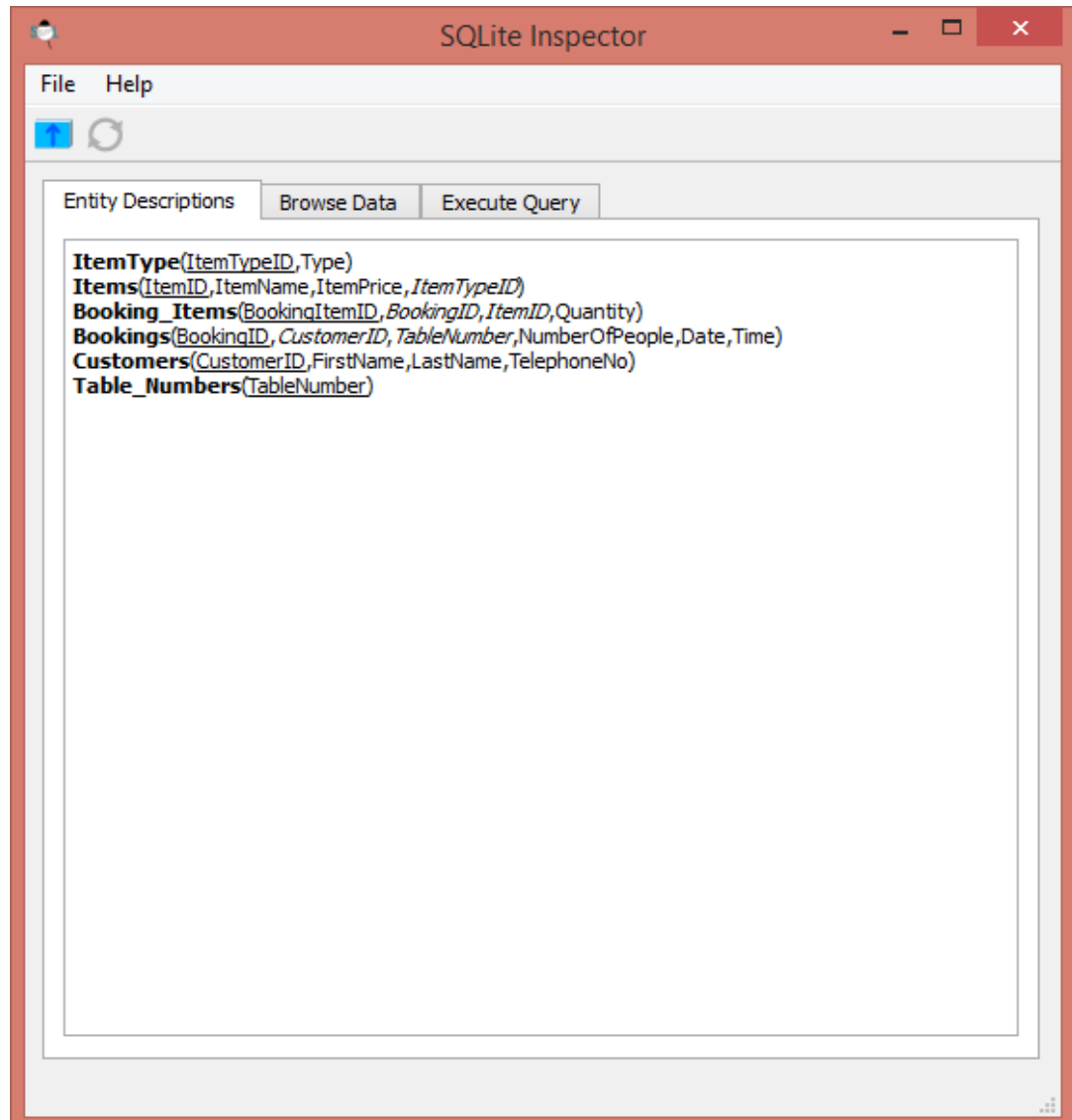
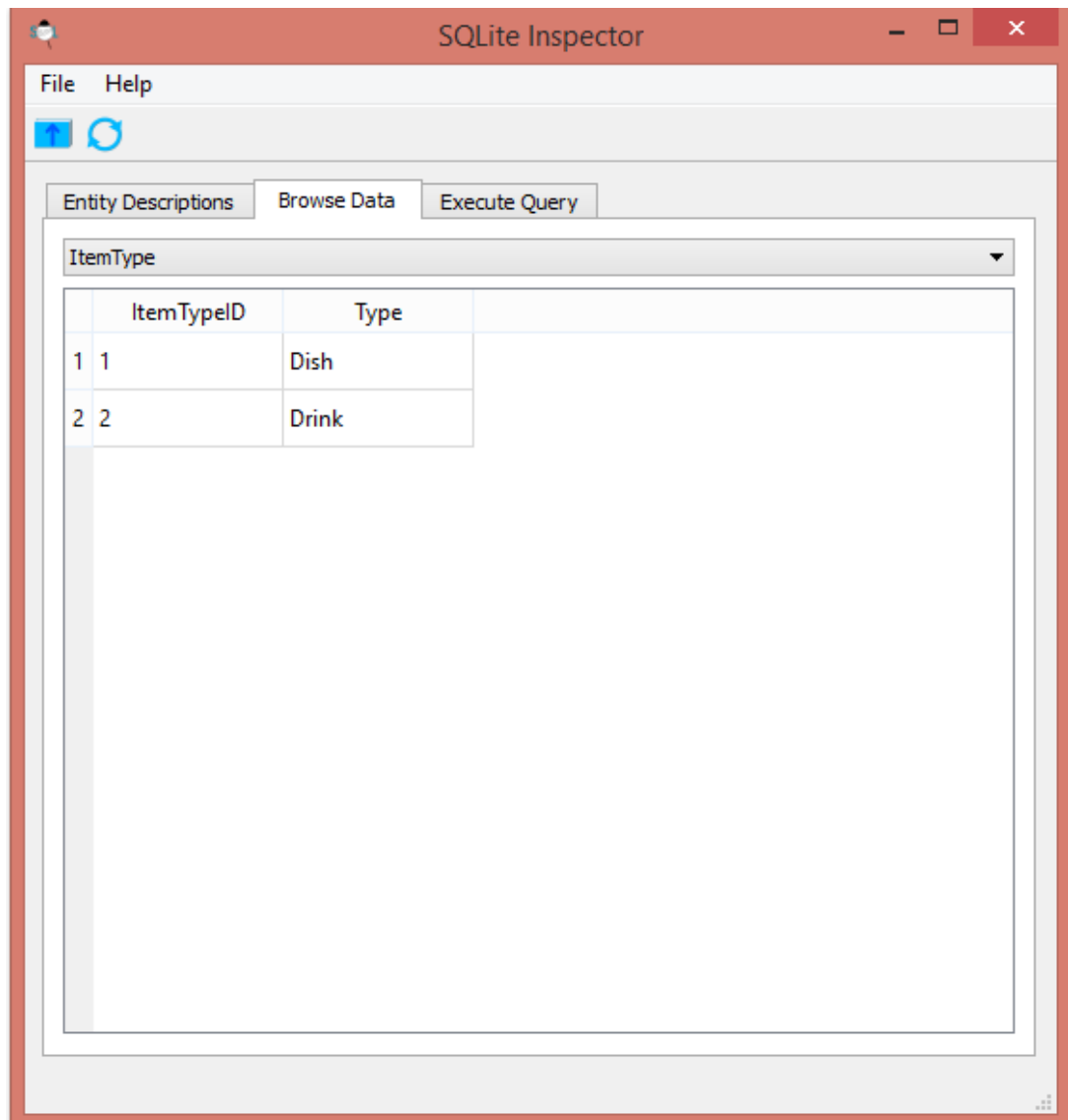


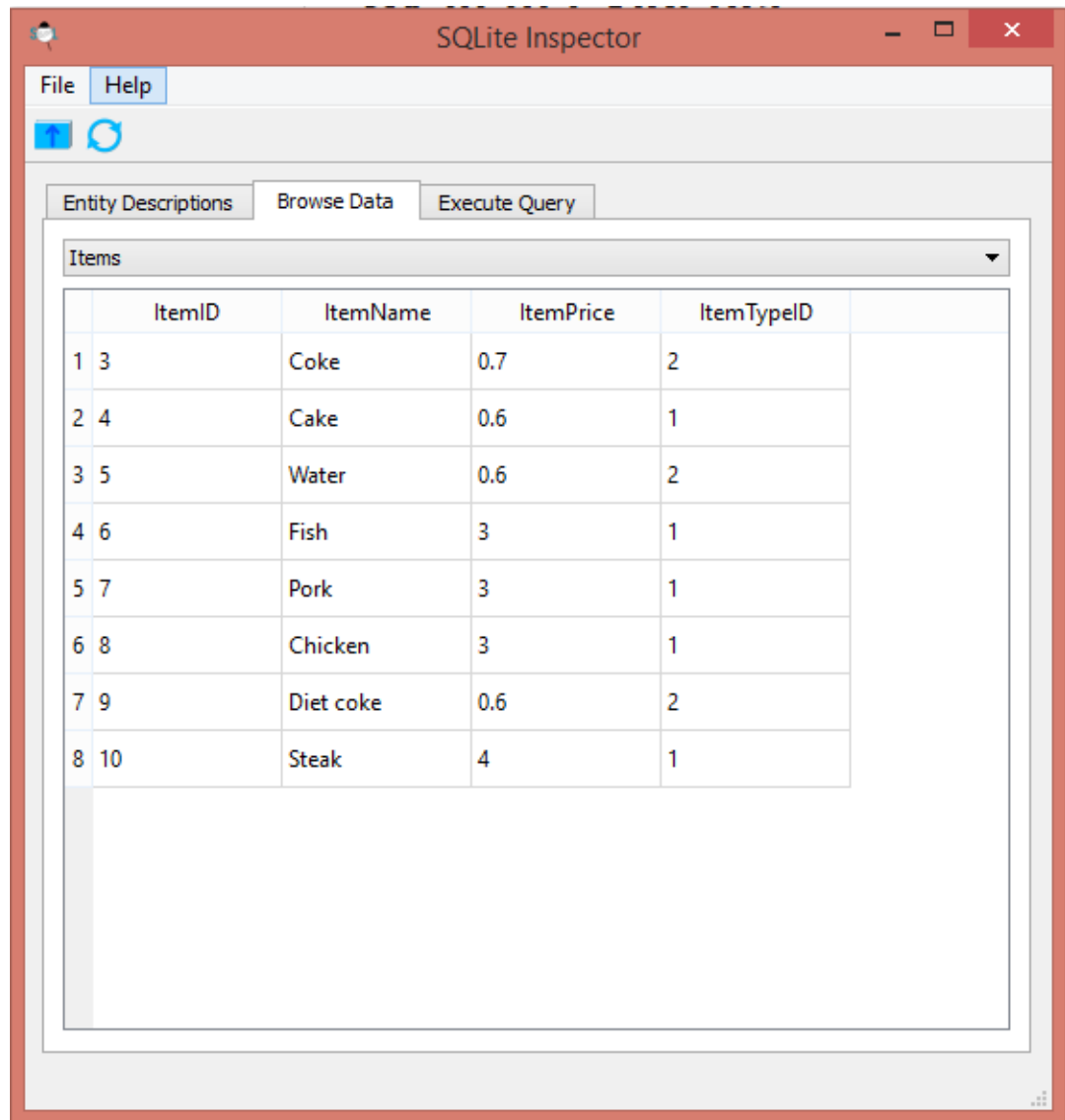
Figure 4.23: ER Diagram of finished database



The screenshot shows the SQLite Inspector application window. The title bar is red and contains the text "SQLite Inspector" along with standard window controls. Below the title bar is a menu bar with "File" and "Help". Under the menu bar are two icons: a blue square with a white upward arrow and a blue circular arrow. Below these are three tabs: "Entity Descriptions", "Browse Data", and "Execute Query". The "Entity Descriptions" tab is selected. Inside this tab, there is a dropdown menu labeled "ItemType" with a downward arrow. Below the dropdown is a table with two columns: "ItemTypeID" and "Type". The table contains two rows of data.

	ItemTypeID	Type
1	1	Dish
2	2	Drink

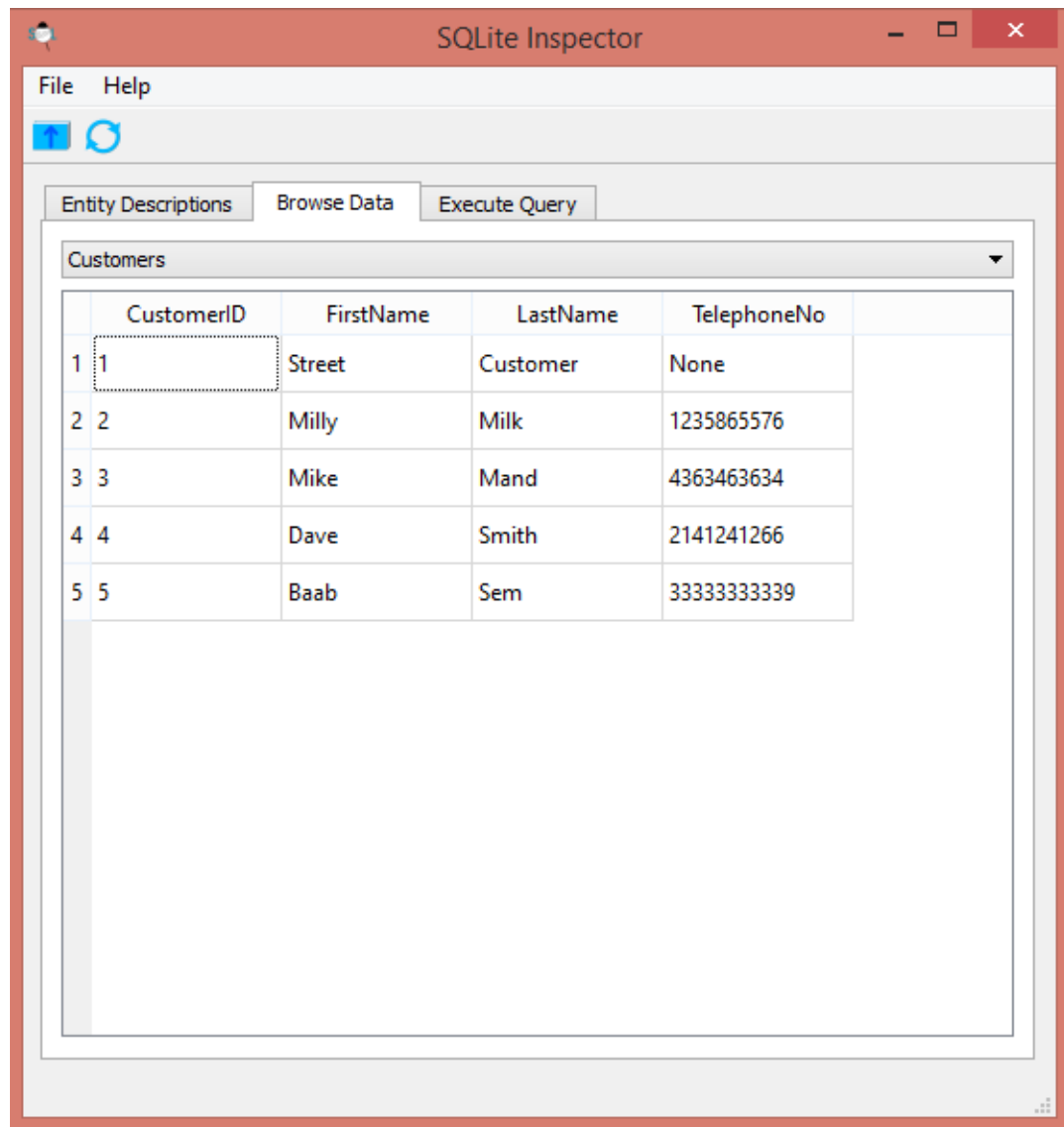
Figure 4.24: Item Type entity



The screenshot shows the SQLite Inspector application window. The title bar is 'SQLite Inspector'. The menu bar has 'File' and 'Help'. Below the menu bar are icons for a folder and a refresh button. The main area has three tabs: 'Entity Descriptions', 'Browse Data', and 'Execute Query'. The 'Browse Data' tab is selected, and a dropdown menu shows 'Items'. Below the dropdown is a table with 5 columns: 'ItemID', 'ItemName', 'ItemPrice', 'ItemTypeID', and an empty column. The table contains 8 rows of data.

	ItemID	ItemName	ItemPrice	ItemTypeID	
1	3	Coke	0.7	2	
2	4	Cake	0.6	1	
3	5	Water	0.6	2	
4	6	Fish	3	1	
5	7	Pork	3	1	
6	8	Chicken	3	1	
7	9	Diet coke	0.6	2	
8	10	Steak	4	1	

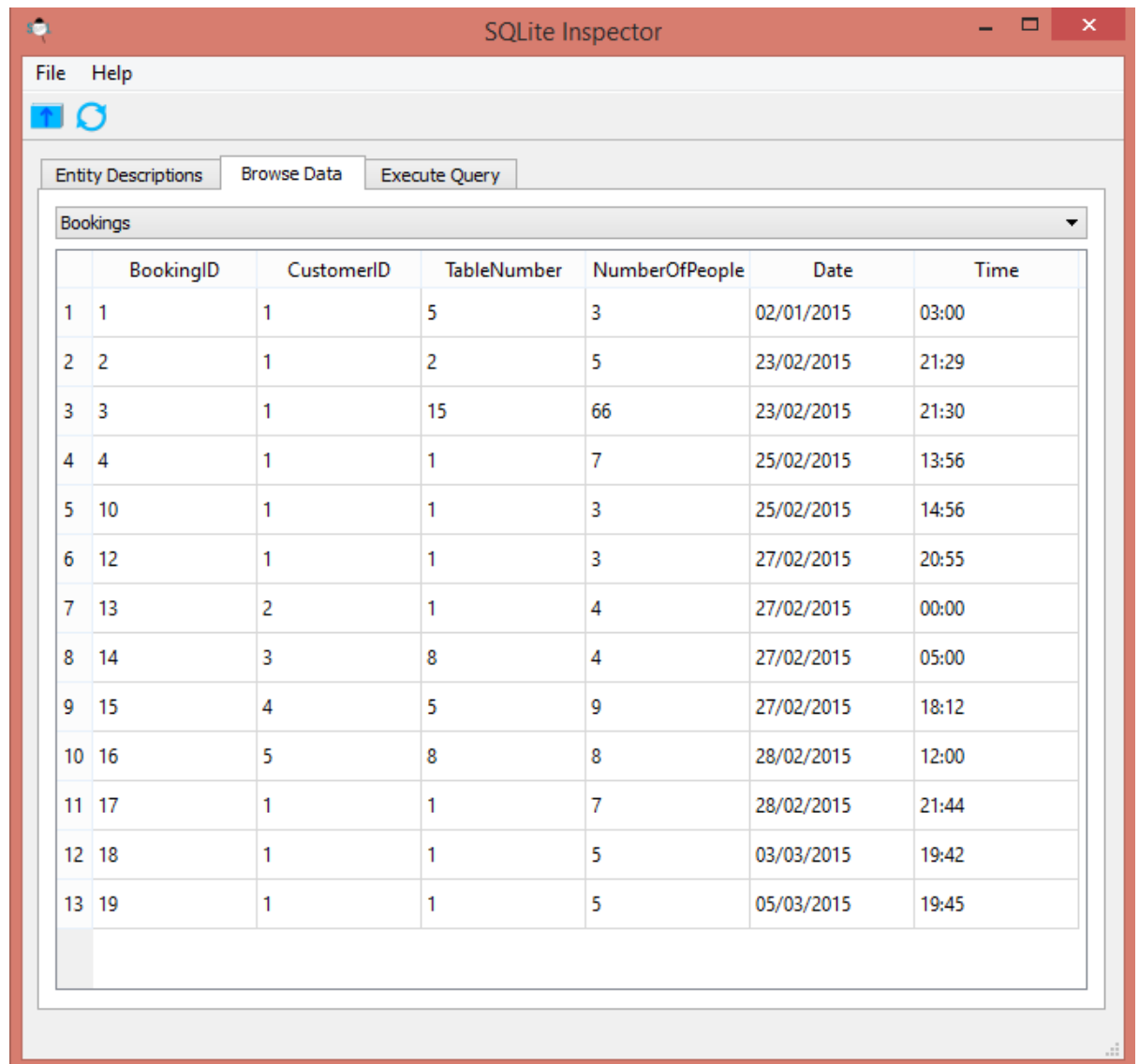
Figure 4.25: Items entity



The screenshot shows the SQLite Inspector application window. The title bar is "SQLite Inspector". The menu bar has "File" and "Help". Below the menu bar are two icons: a blue square with a white upward arrow and a blue circular arrow. The main area has three tabs: "Entity Descriptions", "Browse Data", and "Execute Query". The "Browse Data" tab is selected. Below the tabs is a dropdown menu showing "Customers". Below the dropdown is a table with the following data:

	CustomerID	FirstName	LastName	TelephoneNo
1	1	Street	Customer	None
2	2	Milly	Milk	1235865576
3	3	Mike	Mand	4363463634
4	4	Dave	Smith	2141241266
5	5	Baab	Sem	3333333339

Figure 4.26: Customers entity

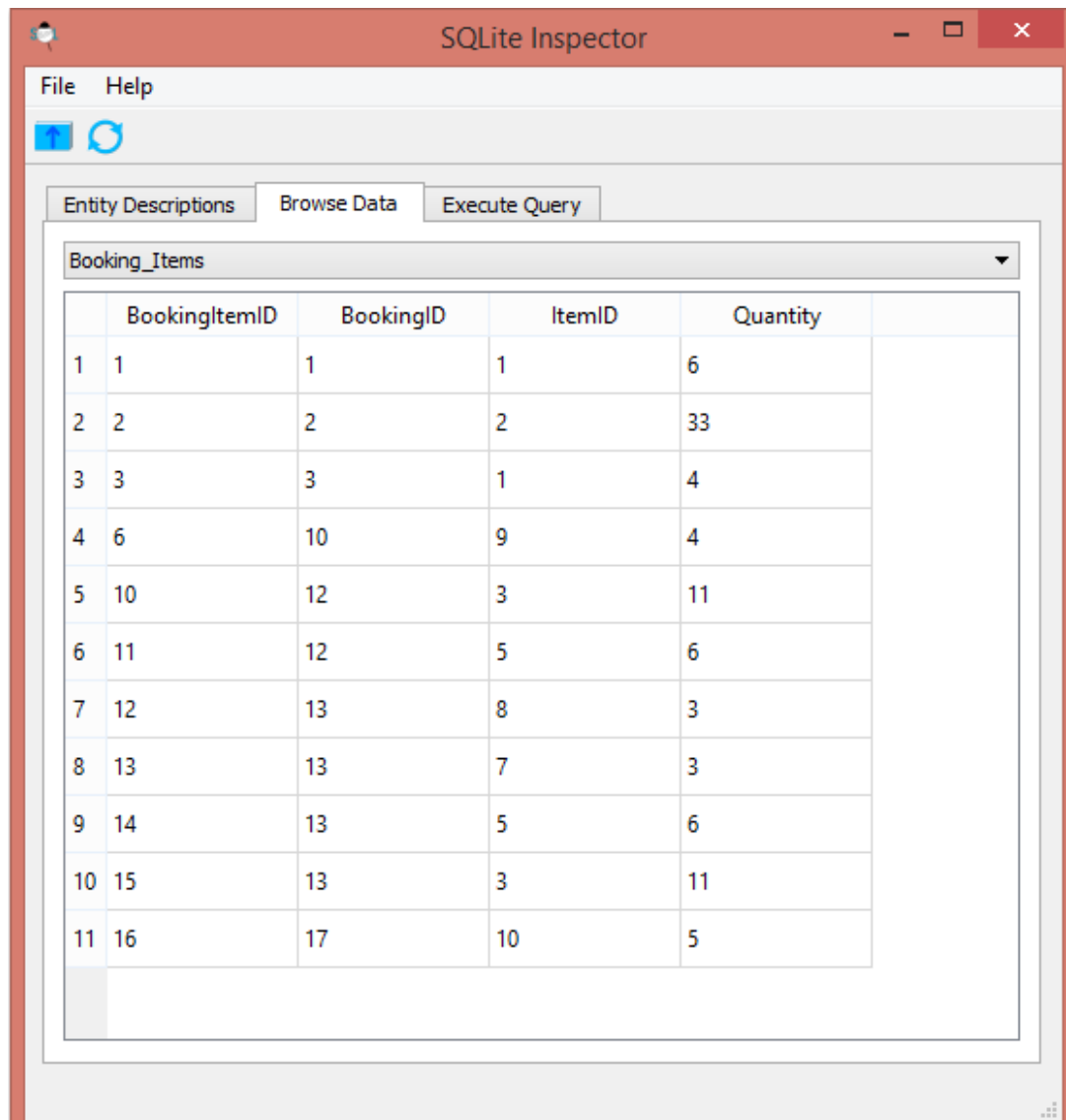


The screenshot shows the SQLite Inspector application window. The title bar is "SQLite Inspector". The menu bar has "File" and "Help". Below the menu bar are icons for a folder, a refresh button, and a search icon. The main area has three tabs: "Entity Descriptions", "Browse Data", and "Execute Query". The "Browse Data" tab is selected, and a dropdown menu shows "Bookings". Below the dropdown is a table with 7 columns: "BookingID", "CustomerID", "TableNumber", "NumberOfPeople", "Date", and "Time". The table contains 13 rows of data.

	BookingID	CustomerID	TableNumber	NumberOfPeople	Date	Time
1	1	1	5	3	02/01/2015	03:00
2	2	1	2	5	23/02/2015	21:29
3	3	1	15	66	23/02/2015	21:30
4	4	1	1	7	25/02/2015	13:56
5	10	1	1	3	25/02/2015	14:56
6	12	1	1	3	27/02/2015	20:55
7	13	2	1	4	27/02/2015	00:00
8	14	3	8	4	27/02/2015	05:00
9	15	4	5	9	27/02/2015	18:12
10	16	5	8	8	28/02/2015	12:00
11	17	1	1	7	28/02/2015	21:44
12	18	1	1	5	03/03/2015	19:42
13	19	1	1	5	05/03/2015	19:45

Figure 4.27: Bookings entity





The screenshot shows the SQLite Inspector application window. The title bar is red and contains the text "SQLite Inspector" along with standard window controls. Below the title bar is a menu bar with "File" and "Help". Under the menu bar is a toolbar with two icons: a blue square with a white arrow pointing up and a blue circular arrow. Below the toolbar are three tabs: "Entity Descriptions", "Browse Data", and "Execute Query". The "Browse Data" tab is selected. Below the tabs is a dropdown menu showing "Booking\_Items". Below the dropdown is a table with 5 columns: an index column, "BookingItemID", "BookingID", "ItemID", and "Quantity". The table contains 11 rows of data.

	BookingItemID	BookingID	ItemID	Quantity
1	1	1	1	6
2	2	2	2	33
3	3	3	1	4
4	6	10	9	4
5	10	12	3	11
6	11	12	5	6
7	12	13	8	3
8	13	13	7	3
9	14	13	5	6
10	15	13	3	11
11	16	17	10	5

Figure 4.28: Booking Items entity

#### 4.5.4 Database SQL

---

```
1 CREATE TABLE ItemType(  
2     ItemTypeID integer,  
3     Type text,  
4     Primary Key(ItemTypeID));
```

---

```
1 CREATE TABLE Items(  
2     ItemID integer,  
3     ItemName text,  
4     ItemPrice real,  
5     ItemTypeID integer,  
6     Primary Key(ItemID),  
7     Foreign Key(ItemTypeID) references  
        ItemType(ItemTypeID));
```

---

```
1 CREATE TABLE Booking_Items(  
2     BookingItemID integer,  
3     BookingID integer,  
4     ItemID integer,  
5     Quantity integer,  
6     Primary Key(BookingItemID)  
7     Foreign Key (BookingID) references  
        Bookings(BookingID) on delete cascade,  
8     Foreign Key(ItemID) references Items(ItemID));
```

---

```
1 CREATE TABLE Bookings(  
2     BookingID integer,  
3     CustomerID integer,  
4     TableNumber integer,  
5     NumberOfPeople integer,  
6     Date text,  
7     Time text,  
8     Primary Key(BookingID),  
9     Foreign Key(CustomerID) references  
        Customers(CustomerID),  
10    Foreign Key(TableNumber) references  
        Table_Numbers(TableNumber));
```

---

```
1 CREATE TABLE Table_Numbers(  
2     TableNumber integer,  
3     Primary Key(TableNumber));
```

---

---

```
1 CREATE TABLE Customers(  
2     CustomerID integer,  
3     FirstName text,  
4     LastName text,  
5     TelephoneNo integer,  
6     Primary key(CustomerID));
```

---

#### 4.5.5 SQL Queries

---

```
1 insert into Customers  
    (FirstName,LastName,TelephoneNo) values (?,?)

1 insert into ItemType (Type) values (<?)

1 insert into Table_Numbers (TableNumber) values (<?)

1 insert into Customers(FirstName,LastName,TelephoneNo)<br/    values (?,?)

1 select CustomerID from Customers where TelephoneNo=?<br/    and FirstName=? and LastName=?

1 insert into  
    Bookings(CustomerID,TableNumber,NumberOfPeople,Date,Time)  
    values (?,?)

1 insert into Items(ItemName,ItemPrice,ItemTypeID)<br/    values (?,?)

1 select Quantity from Booking_Items where ItemID=? and<br/    BookingID = ?

1 update Booking_Items set Quantity=? where ItemID=?

1 insert into Booking_Items(BookingID,ItemID,Quantity)  
    values (?,?)</pre

---


```

---

```
1 SELECT
2 Items.ItemName
3 FROM Items
4 INNER JOIN Booking_Items
5 ON Booking_Items.ItemID = Items.ItemID
6 WHERE Booking_Items.BookingID = ?
```

---

This query fetches all the item names that have been ordered, given the booking id.

---

```
1 SELECT
2 Items.ItemName
3 FROM Items
4 INNER JOIN Booking_Items
5 ON Booking_Items.ItemID = Items.ItemID
6 WHERE Booking_Items.BookingID = ?
7 AND Items.ItemID = ?
```

---

This query fetches the item name of an ordered item, given the booking id and item id.

---

```
1 SELECT
2 Customers.FirstName ,
3 Customers.LastName ,
4 Bookings.NumberOfPeople ,
5 Bookings.Time
6 FROM Customers
7 INNER JOIN Bookings
8 ON Customers.CustomerID = Bookings.CustomerID
9 WHERE Bookings.Date = '{0}'
10 AND Bookings.TableNumber = {1}
11 .format(self.systemdate, TableNumber)
```

---

This query fetches all the bookings that matches the system date and the relevant table number with the customer's first name, last name, number of people and booking time.

---

```
1 select * from Bookings where CustomerID = {?} and
   TableNumber = {1} and NumberOfPeople = {2} and
```

---

```
        Date = '{3}' and Time = '{4}'
        .format(CustomerID,TableNumber,NumberOfPeople,Date,Time))
```

---

```
1 select * from Bookings where CustomerID = {0} and
    TableNumber = {1} and Date =
    '{2}'.format(CustomerID,self.tableNumber,TodaysDate)
```

---

```
1 select CustomerID from Bookings where TableNumber =
    {0} and Date = '{1}'.format(TableNumber,TodaysDate)
```

---

```
1 select LastName from Customers where CustomerID =
    {0}.format(customer))
```

---

```
1 delete from Bookings where BookingID =
    {0}.format(booking)
```

---

```
1 delete from Items where ItemName = ?
```

---

```
1 delete from Items where ItemID = {0}.format(itemID)
```

---

```
1 delete from Booking_Items where BookingID = ? and
    ItemID = ?
```

---

```
1 SELECT
2     Customers.FirstName ,
3     Customers.LastName ,
4     Bookings.NumberOfPeople ,
5     Bookings.TableNumber ,
6     Bookings.Time
7 FROM Customers
8     INNER JOIN Bookings
9     ON Customers.CustomerID = Bookings.CustomerID
10    WHERE Bookings.Date = '{0}'
11    ORDER BY Bookings.Time
12 .format(TodaysDate)
```

---

This query fetches all the bookings that matches the booking date with the system's date with its details including customer details.

---

```
1 SELECT
2     Bookings.BookingID ,
```

```
3  Customers.CustomerID ,
4  Customers.FirstName ,
5  Customers.LastName ,
6  Bookings.NumberOfPeople ,
7  Bookings.TableNumber ,
8  Bookings.Time ,
9  Bookings.Date ,
10 Customers.TelephoneNo
11 FROM Customers
12 INNER JOIN Bookings
13 ON Customers.CustomerID = Bookings.CustomerID
14 ORDER BY Bookings.Date , Bookings.Time
```

---

This query gets all the records from the bookings entity as well as the customer records that matches with the bookings.

---

```
1  SELECT
2  Booking_Items.Quantity ,
3  Items.ItemName ,
4  Items.ItemPrice
5  FROM Items
6  INNER JOIN Booking_Items
7  ON Booking_Items.ItemID = Items.ItemID
8  WHERE Booking_Items.BookingID = {0}
9  AND Items.ItemTypeID = 2
```

---

This query fetches all the ordered drinks , given the booking id.

---

```
1  SELECT
2  Booking_Items.Quantity ,
3  Items.ItemName ,
4  Items.ItemPrice
5  FROM Items
6  INNER JOIN Booking_Items
7  ON Booking_Items.ItemID = Items.ItemID
8  WHERE Booking_Items.BookingID = {0}
9  AND Items.ItemTypeID = 1
```

---

This query fetches all the ordered dishes, given the booking id.

---

```
1 SELECT
2 Items.ItemPrice
3 FROM Items
4 INNER JOIN Booking_Items
5 ON Booking_Items.ItemID = Items.ItemID
6 WHERE Booking_Items.BookingID = ?
```

---

This query fetches the prices of each ordered item

---

```
1 SELECT
2 Booking_Items.Quantity
3 FROM Items
4 INNER JOIN Booking_Items
5 ON Booking_Items.ItemID = Items.ItemID
6 WHERE Booking_Items.BookingID = ?
```

---

This query fetches the quantity of each ordered item

---

```
1 SELECT
2 Booking_Items.Quantity ,
3 Items.ItemName ,
4 Items.ItemPrice
5 FROM Items
6 INNER JOIN Booking_Items
7 ON Booking_Items.ItemID = Items.ItemID
8 WHERE Booking_Items.BookingID = {0}
```

---

This query fetches all the ordered items with its information such as quantity, name and price

---

```
1 update Bookings set Date=? where BookingID=?
```

---

---

```
1 update Bookings set Time=? where BookingID=?
```

---

---

```
1 update Bookings set NumberOfPeople=? where BookingID=?
```

---

---

```
1 update Bookings set TableNumber=? where BookingID=?
```

---

---

```
1 update Items set ItemPrice=? where ItemID=?
```

---

## 4.6 Testing

### 4.6.1 Summary of Results

After completing the series of tests, i believe that my application is reliable and robust as the majority of the results were expected and the application did not crash whilst carrying out these tests ( See page 87 for testing). I used diferrent types of test data such as normal, erroneous and boundery whenever possible to test if my application would crash or give errors - The errors were handled using exceptions and I didn't experience any crashes which is why I believed my application is robust.However, there was a weakness to my testing as I did not test all radio buttons on the main screen.

### 4.6.2 Known Issues

#### Test 4.03 - Manage order total price

There was a problem with the manage order box where the Total Price was suppose to be displayed. The problem was that it was suppose to display the total price after adding an item to the order or deleting an item off the order which it didn't (The widget didn't refresh with the new total price). However, I know that the algorithm for the total price is correct because I used the same algorithm for the invoice (which worked as test 4.03.01 was successful). At this current time, I am not sure how to fix it in terms of coding but I do know that the problem is that the line edit doesn't refresh with the new value.

## 4.7 Code Explanations

### 4.7.1 Difficult Sections



#### 4.7.2 Select function (section 4.10.4)

---

```

1      def select_connect(self):
2          TodaysDate = time.strftime("%d/%m/%Y")
3          customerCurrentIndex =
4              self.customer_combo_box.currentIndex()
5          print("Customer :
6              {0}".format(customerCurrentIndex))
7          CustomerID =
8              self.CustomerList[customerCurrentIndex]
9          print("Customer ID: {0}".format(CustomerID))
10
11         with sqlite3.connect("restaurant.db") as db:
12             cursor = db.cursor()
13             cursor.execute("select * from Bookings
14                 where CustomerID = {0} and TableNumber
15                 = {1} and Date =
16                 '{2}'".format(CustomerID, self.tableNumber, TodaysDate))
17             self.bookingDetails = cursor.fetchone()
18             print(self.bookingDetails)
19
20         self.close()
21
22         return self.bookingDetails

```

---

Above is a function that's connected to the Select button as shown by Figure 4.9 on page 149. This function is used to return the booking details of the selected customer from the combo box beside it. Below is a table explaining code.

Line Number	Explanation
2	Assigns the system's date to TodaysDate
3	Assigns the selected index(customer) of the combo box to customerCurrentIndex
5	Gets the customerID of the selected customer using the list self.CustomerList and assigns it to CustomerID
10	SQL statement to get the booking details of the selected customer
11	The booking details is assigned to self.bookingDetails
14	self.close() closes the assign customer to table box

### 4.7.3 Creating the combo box (section 4.10.4)

---

```

1      def create_combo_box(self, TableNumber):
2          self.CustomerList = []
3          CustomerLastName = []
4          TodaysDate = time.strftime("%d/%m/%Y")
5
6          ## get all customer IDs that are on table _
7          with sqlite3.connect("restaurant.db") as db:
8              cursor = db.cursor()
9              cursor.execute("select CustomerID from
10                             Bookings where TableNumber = {0} and
11                             Date =
12                             '{1}'".format(TableNumber, TodaysDate))
13              customers = cursor.fetchall()
14              for each in customers:
15                  self.CustomerList.append(each[0])
16
17          ## get all last names from previous fetchall
18          for customer in self.CustomerList:
19              with sqlite3.connect("restaurant.db") as
20                  db:
21                  cursor = db.cursor()
22                  cursor.execute("select LastName from
23                                 Customers where CustomerID =
24                                 {0}".format(customer))
25                  customer = cursor.fetchone()
26                  CustomerLastName.append(customer[0])
27
28          #create combo, insert all last names from
29          fetchall
30          self.customer_combo_box = QComboBox(self)
31          for each in CustomerLastName:
32              self.customer_combo_box.addItem(each)

```

---

Above is a function that creates the combo box and populates it with customer last names of bookings that match the system date. The combo box is shown by Figure 4.9 on page 149 but it is not populated in that image. Below is a table explaining the code

Line Number	Explanation
4	Assigns the system's date to TodaysDate
10	SQL statement to get all the customer IDs that have a booking on table (tablenumber) and matches the system date.
11-12	A FOR loop to append the self.CustomerList with the customer ids fetched from line 10
15-18	SQL statement to get all the last names of the customer ids in the self.CustomerList array
20	Appends the CustomerLastName list with the last names from lines 15-18
23-25	Creates the combo box and uses a FOR loop to populate the combobox with the last names

#### 4.7.4 Self-created Algorithms

---

```
1         if TableNumber == 1:
2             if self.TableOneOccupied == False:
3                 self.table1 =
4                     AssignCustomer(TableNumber)
5                     bookingDetails =
6                         self.table1.bookingDetails
7                     self.TableOneOrder =
8                         OrderWindow(bookingDetails)
9                     self.TableOneOccupied = True
10                    if self.TableOneOrder.Finished ==
11                        True:
12                            self.TableOneOccupied = False
13            elif self.TableOneOccupied == True:
14                bookingDetails =
15                    self.table1.bookingDetails
16                self.TableOneOrder =
17                    OrderWindow(bookingDetails)
18                if self.TableOneOrder.Finished ==
19                    True:
20                        self.TableOneOccupied = False
```

---

The above algorithm is used to tell the program if the table is occupied or not. Below is a table explaining the algorithm in more depth.

Line Number	Explanation
1	If the selected table number radio button is 1 then the algorithm will proceed
2	If the table is not occupied then lines 3-8 will proceed
3	The assign customer box will pop up and user would have to create a booking from there or select a booking ( Figure 4.9 on page 149)
4	The booking details is returned if the user has created or selected a booking
5	The assign customer box will close and the manage order box will pop up (Figure 4.10 on page 150)
6	The table is now occupied
7-8	If the user presses the Finish button on the manage order box without closing it then the table will be set as unoccupied
9	This part of the algorithm will only proceed if the user closes the box without pressing Finish
10	The booking details is returned to pass it to the manage order box
11	Manage order box pops up
12-13	If the user presses the Finish button then the table will be set as unoccupied

---

```
1      def checkExistingItem(self):
2          addedAlready = False
3          itemsOrdered = []
4          item = ""
5
6          with sqlite3.connect("restaurant.db") as db:
7              cursor = db.cursor()
8              cursor.execute("""SELECT
9                              Items.ItemName
10                             FROM Items
11                             INNER JOIN Booking_Items
12                             ON Booking_Items.ItemID =
13                                Items.ItemID
14                             WHERE
15                                Booking_Items.BookingID
16                                = ?
17                                """,(self.bookingDetails[0],))
18              items = cursor.fetchall()
19              for each in items:
20                  itemsOrdered.append(each[0])
21
22          try:
23              with sqlite3.connect("restaurant.db") as
24                  db:
25                  cursor = db.cursor()
26                  cursor.execute("""SELECT
27                                  Items.ItemName
28                                  FROM Items
29                                  INNER JOIN
30                                  Booking_Items
31                                  ON
32                                  Booking_Items.ItemID
33                                  = Items.ItemID
34                                  WHERE
35                                  Booking_Items.BookingID
36                                  = ?
37                                  AND Items.ItemID =
38                                  ?""",(self.bookingDetails[0],self.ItemID)
39
40              item = cursor.fetchone()[0]
41          except TypeError:
42              pass
43
44          if item in itemsOrdered:
45              addedAlready = True
46
47          return item, addedAlready
```

35

```
return addedAlready
```

---

The algorithm above is used to check if an item that is about to be added has already been ordered, if it has then the quantity will increase (quantity increase is not part of this algorithm). I created a list (line 3) which was used to populate it (line 14-16) with all ordered items from the relevant booking. I then got the item name that was going to be added to the order (lines 21-28) and checked if the item was in the list that was populated with the ordered items (lines 32-33).

---

```
1     def checkQuantity(self):
2         OneQuantity = False
3         try:
4             with sqlite3.connect("restaurant.db") as
                    db:
5                 cursor = db.cursor()
6                 cursor.execute("""SELECT
7                                 Quantity
8                                 FROM Booking_Items
9                                 WHERE BookingID = ?
10                                AND ItemID =
                                    ?""", (self.bookingID, self.ItemID))
11                 itemQuantity = cursor.fetchone()[0]
12                 print(itemQuantity)
13
14                 if (int(itemQuantity)) == 1:
15                     OneQuantity = True
16
17                 return OneQuantity
18         except TypeError:
19             pass
```

---

The algorithm above will be used when the user wants to delete an item off an order. It is used to find out if the quantity of the item that wants to be deleted has a quantity of one or not. Lines 4-11 gets the quantity of the ordered item that the user wants to delete and if the quantity is equal to 1 then OneQuantity will be set to True if not then the returned OneQuantity will be False.

## 4.8 Settings

My client's computer system settings will not need any changes to run the application as I have made the application an executable however, if anyone wanted to alter the application then Python and PyQt4 would be needed.

## 4.9 Acknowledgements

I used <http://doc.qt.digia.com/4.6/stylesheet-reference.html> to help me with the style sheets for my application. I did not copy any coding but I did use the website's examples to help me understand what was needed to create the cascade style sheet. For example there was an example

" The background color used for the widget.



Examples:

```
QLabel background-color: yellow
QLineEdit background-color: rgb(255, 0, 0) "
```

In addition, I used <http://pyqt.sourceforge.net/Docs/PyQt4/qtsql.html> to help me display sql tables. The example that helped me the most was "QSqlTableModel offers a read-write model that works on a single SQL table at a time.

Example:

```
QSqlTableModel model;
model.setTable("employee");
model.setFilter("salary < 50000");
model.setSort(2, Qt.DescendingOrder);
model.select();
"
```

## 4.10 Code Listing

#### 4.10.1 add\_booking.py

---

```
1 import sys
2 import sqlite3
3 from PyQt4.QtCore import *
4 from PyQt4.QtGui import *
5 from table_display import *
6 import time
7
8 class AddBookingWindow(QWidget):
9
10     """this class creates a widget to add bookings"""
11
12     def __init__(self):
13         super().__init__()
14
15         self.main_layout = QVBoxLayout()
16         self.add_booking_layout = QGridLayout()
17         self.add_complete_layout = QHBoxLayout()
18
19         self.display_table = DisplayTable()
20         self.display_table.show_table("Bookings")
21         self.add_complete = QPushButton("Add Booking")
22
23         self.first_name_label = QLabel("First Name:")
24         self.last_name_label = QLabel("Last Name:")
25         self.date_label = QLabel("Date:")
26         self.time_label = QLabel("Time:")
```

```
27     self.number_of_people = QLabel("Number Of People:")
28     self.telephone_number = QLabel("Telephone Number:")
29     self.table_number_label = QLabel("Table Number:")
30
31     regExp = QRegExp("[a-zA-Z]+$")
32     Validator = QRegExpValidator(regExp)
33     self.input_first_name = QLineEdit()
34     self.input_first_name.setValidator(Validator)
35     self.input_first_name.setMaximumSize(300,30)
36     self.input_last_name = QLineEdit()
37     self.input_last_name.setValidator(Validator)
38
39     self.input_last_name.setMaximumSize(300,30)
40
41     self.input_number_of_people = QLineEdit()
42     regexp = QRegExp("\\d\\d\\d")
43     validator = QRegExpValidator(regexp)
44     self.input_number_of_people.setValidator(validator)
45     self.input_number_of_people.setMaximumSize(300,30)
46     self.input_number_of_people.setMaxLength(2)
47
48     regexp2 = QRegExp("[0-9]*$")
49     validator2 = QRegExpValidator(regexp2)
50
51     self.input_telephone_number = QLineEdit()
52     self.input_telephone_number.setValidator(validator2)
53     self.input_telephone_number.setMaximumSize(300,30)
54     self.input_telephone_number.setMaxLength(11)
```

```
55
56 self.select_table_number = QComboBox(self)
57 for each in range(1,17):
58     self.select_table_number.addItem(str(each))
59
60
61 #dates and times
62 self.date_edit = QDateEdit()
63 self.maximumdate = QDate(2050,1,30)
64 self.minimumdate = QDate.currentDate()
65 self.date_edit.setMaximumDate(self.maximumdate)
66 self.date_edit.setMinimumDate(self.minimumdate)
67 self.time_edit = QTimeEdit()
68 self.time_edit.setDisplayFormat("hh:mm")
69
70
71
72 #place holder text
73 self.input_first_name.setPlaceholderText("First name given")
74 self.input_last_name.setPlaceholderText("Last name given")
75 self.input_number_of_people.setPlaceholderText("Expected number")
76 self.input_telephone_number.setPlaceholderText("Telephone number given")
77
78 self.add_booking_layout.addWidget(self.first_name_label,0,0)
79 self.add_booking_layout.addWidget(self.last_name_label,1,0)
80 self.add_booking_layout.addWidget(self.date_label,2,0)
81 self.add_booking_layout.addWidget(self.time_label,3,0)
82 self.add_booking_layout.addWidget(self.number_of_people,4,0)
```

```
83     self.add_booking_layout.addWidget(self.telephone_number,5,0)
84     self.add_booking_layout.addWidget(self.table_number_label,6,0)
85
86     self.add_booking_layout.addWidget(self.input_first_name,0,1)
87     self.add_booking_layout.addWidget(self.input_last_name,1,1)
88     self.add_booking_layout.addWidget(self.date_edit,2,1)
89     self.add_booking_layout.addWidget(self.time_edit,3,1)
90     self.add_booking_layout.addWidget(self.input_number_of_people,4,1)
91     self.add_booking_layout.addWidget(self.input_telephone_number,5,1)
92     self.add_booking_layout.addWidget(self.select_table_number,6,1)
93     self.add_complete_layout.addWidget(self.add_complete)
94
95     #add layouts to main layout
96     self.main_layout.addWidget(self.display_table)
97     self.main_layout.addLayout(self.add_booking_layout)
98     self.main_layout.addLayout(self.add_complete_layout)
99     self.setLayout(self.main_layout)
100
101
102     #connections
103     self.add_complete.clicked.connect(self.add_booking)
104
105     def add_booking(self):
106         FirstName = self.input_first_name.text().capitalize()
107         LastName = self.input_last_name.text().capitalize()
108         TeleNumber = self.input_telephone_number.text()
109         try:
110             NumberOfPeople = int(self.input_number_of_people.text())
```

```

111     except ValueError:
112         pass
113     TableNumber = self.select_table_number.currentIndex() + 1
114     BookingDate = self.date_edit.text()
115     BookingTime = self.time_edit.text()
116
117     try:
118
119         if (len(FirstName) > 1) and (len(LastName) > 1) and
120             ((len(str(NumberOfPeople))) > 0 and (NumberOfPeople)>1) and
121             (len(str(TeleNumber)) == 11):
122
123             customer = (FirstName,LastName,TeleNumber)
124
125             with sqlite3.connect("restaurant.db") as db:
126                 cursor = db.cursor()
127                 sql = "insert into Customers(FirstName,LastName,TelephoneNo)
128                     values (?,?,?)"
129                 cursor.execute(sql,customer)
130                 db.commit()
131
132             with sqlite3.connect("restaurant.db") as db:
133                 cursor = db.cursor()
134                 cursor.execute("select CustomerID from Customers where
135                     TelephoneNo=? and FirstName=? and
136                     LastName=?", (TeleNumber,FirstName,LastName))
137                 customerid = cursor.fetchone()[0]
138                 print(customerid)

```

```

134         booking =
135             (customerid, TableNumber, NumberOfPeople, BookingDate, BookingTime)
136         print(booking)
137         with sqlite3.connect("restaurant.db") as db:
138             cursor = db.cursor()
139             sql = "insert into
140                 Bookings(CustomerID, TableNumber, NumberOfPeople, Date, Time)
141                 values (?, ?, ?, ?, ?)"
142             cursor.execute("PRAGMA foreign_keys = ON")
143             cursor.execute(sql, booking)
144             db.commit()
145
146         self.display_table.refresh()
147     else:
148         QMessageBox.about(self, "Error", "Please make sure you haven't left any
149             empty spaces")
150
151     except ValueError:
152         print("Booking unsuccessful")
153
154 if __name__ == "__main__":
155     application = QApplication(sys.argv)
156     window = AddBookingWindow()
157     window.show()
158     window.raise_()
159     application.exec()

```

#### 4.10.2 add\_item\_to\_menu.py

---

```
1 import sys
2 import sqlite3
3 from PyQt4.QtCore import *
4 from PyQt4.QtGui import *
5 from table_display import *
6
7 class AddItemToMenu(QWidget):
8     """this class creates a widget to add items to the menu"""
9
10    def __init__(self):
11        super().__init__()
12
13        self.display_table = DisplayTable()
14        self.display_table.show_table("Items")
15
16        #create layouts
17        self.main_layout = QVBoxLayout()
18        self.add_item_layout = QGridLayout()
19        self.add_complete_layout = QHBoxLayout()
20
21        #create buttons
22        self.add_complete = QPushButton("Add Item")
23
24        #create combobox for item type
25        self.select_item_type = QComboBox(self)
26        self.select_item_type.addItem("Dish")
```

191



```
27     self.select_item_type.addItem("Drink")
28
29     #labels
30     self.item_name_label = QLabel("Item Name : ")
31     self.item_price_label = QLabel("Item Price : ")
32     self.item_type_label = QLabel("Item Type : ")
33
34     #line edit
35     regexpp = QRegExp("[a-z | A-Z]{1,20}")
36     validatorr = QRegExpValidator(regexpp)
37     self.input_item_name = QLineEdit()
38     self.input_item_name.setValidator(validatorr)
39     self.input_item_name.setMaximumSize(300,30)
40
41     regexp = QRegExp("(^\\d|\\d\\d)(\\.\\d\\d)?$")
42     validator = QRegExpValidator(regexp)
43     self.input_item_price = QLineEdit()
44     self.input_item_price.setValidator(validator)
45     self.input_item_price.setMaximumSize(300,30)
46
47     #add labels to layout
48     self.add_item_layout.addWidget(self.item_name_label,0,0)
49     self.add_item_layout.addWidget(self.item_price_label,1,0)
50     self.add_item_layout.addWidget(self.item_type_label,2,0)
51
52     #add line edit to layout
53     self.add_item_layout.addWidget(self.input_item_name,0,1)
54     self.add_item_layout.addWidget(self.input_item_price,1,1)
```

```

55     self.add_item_layout.addWidget(self.select_item_type,2,1)
56
57     #add button to layout
58     self.add_complete_layout.addWidget(self.add_complete)
59
60     #add layouts/table to main layout
61     self.main_layout.addWidget(self.display_table)
62     self.main_layout.addLayout(self.add_item_layout)
63     self.main_layout.addLayout(self.add_complete_layout)
64     self.setLayout(self.main_layout)
65
66     #connection
67     self.add_complete.clicked.connect(self.add_item_to_menu)
68
69     self.display_table.refresh()
70
71     def add_item_to_menu(self):
72         ItemName = self.input_item_name.text().capitalize()
73         ItemPrice = self.input_item_price.text()
74         ItemType = self.select_item_type.currentIndex()
75         if ItemType == 0:
76             ItemType = 1
77         else:
78             ItemType = 2
79         MenuItem = (ItemName,ItemPrice,ItemType)
80         print(MenuItem)
81         if len(ItemName)>1 and (len(ItemPrice)>0):
82             with sqlite3.connect("restaurant.db") as db:

```

```
83         cursor = db.cursor()
84         sql = "insert into Items(ItemName,ItemPrice,ItemTypeID) values (?,?,?)"
85         cursor.execute("PRAGMA foreign_keys = ON")
86         cursor.execute(sql,MenuItem)
87         db.commit()
88
89         self.display_table.refresh()
90     else:
91         QMessageBox.about(self, "Error","Please make sure you have filled in the
           required fields")
92
93
94
95
194 if __name__ == "__main__":
96     application = QApplication(sys.argv)
97     window = AddItemToMenu()
98     window.show()
99     window.raise_()
100     application.exec()
```

---

### 4.10.3 add\_item\_to\_order.py

---

```
195 1 import sys
2   import sqlite3
3   from PyQt4.QtCore import *
4   from PyQt4.QtGui import *
5   from table_display import *
6   from cascade_style_sheet import *
7
8   class AddItemToOrder(QDialog):
9       orderitemAdded = pyqtSignal()
10      """this class creates a widget to add an item to order"""
11
12      def __init__(self,bookingDetails):
13          super().__init__()
14          self.setWindowTitle("Add Item To Order")
15          self.setMinimumSize(600,600)
16          self.bookingDetails = bookingDetails
17          self.setStyleSheet(css)
18
19          self.main_layout = QVBoxLayout()
20          self.add_item_layout = QGridLayout()
21          self.add_complete_layout = QHBoxLayout()
22
23          self.add_complete = QPushButton("Add Item")
24          self.itemID_label = QLabel("Item ID : ")
25          self.itemQuantity_label = QLabel("Item Quantity : ")
26
```

```
27     #line edit
28     regexp = QRegExp("^\\d\\d\\d?$")
29     validator = QRegExpValidator(regexp)
30     self.input_itemID = QLineEdit()
31     self.input_itemID.setValidator(validator)
32     self.input_itemID.setMaximumSize(133,30)
33     self.input_itemID.setAlignment(Qt.AlignLeft)
34     self.input_itemQuantity = QLineEdit()
35     self.input_itemQuantity.setValidator(validator)
36     self.input_itemQuantity.setMaximumSize(133,30)
37
38     self.item_table = DisplayTable()
39     self.item_table.show_table("Items")
40
41
42     self.add_item_layout.addWidget(self.itemID_label,0,0)
43     self.add_item_layout.addWidget(self.itemQuantity_label,1,0)
44     self.add_item_layout.addWidget(self.input_itemID,0,1)
45     self.add_item_layout.addWidget(self.input_itemQuantity,1,1)
46     self.add_complete_layout.addWidget(self.add_complete)
47
48     self.main_layout.addWidget(self.item_table)
49     self.main_layout.addLayout(self.add_item_layout)
50     self.main_layout.addLayout(self.add_complete_layout)
51
52     self.setLayout(self.main_layout)
53
54     self.add_complete.clicked.connect(self.add_item_to_order)
```

197

```
55
56 def add_item_to_order(self, bookingDetails):
57     bookingID = self.bookingDetails[0]
58     self.ItemID = self.input_itemID.text()
59     Quantity = self.input_itemQuantity.text()
60     MenuItem = (bookingID, self.ItemID, Quantity)
61     addedAlready = self.checkExistingItem()
62     print(addedAlready)
63
64     try:
65
66         if addedAlready == True:
67             with sqlite3.connect("restaurant.db") as db:
68                 cursor = db.cursor()
69                 cursor.execute("select Quantity from Booking_Items where ItemID=?
70                               and BookingID = ?", (self.ItemID, bookingID))
71                 dbquantity = cursor.fetchone()[0]
72
73                 newQuantity = dbquantity + int(Quantity)
74                 updateOrder = (newQuantity, self.ItemID)
75                 with sqlite3.connect("restaurant.db") as db:
76                     cursor = db.cursor()
77                     sql = "update Booking_Items set Quantity=? where ItemID=?"
78                     cursor.execute("PRAGMA foreign_keys = ON")
79                     cursor.execute(sql, updateOrder)
80                     db.commit()
81
82                 self.orderitemAdded.emit()
```

```
82
83         elif addedAlready == False:
84
85             with sqlite3.connect("restaurant.db") as db:
86                 cursor = db.cursor()
87                 sql = "insert into Booking_Items(BookingID,ItemID,Quantity) values
88                     (?,?)"
89                 cursor.execute("PRAGMA foreign_keys = ON")
90                 cursor.execute(sql,MenuItem)
91                 db.commit()
92
93                 self.orderitemAdded.emit()
94             except sqlite3.IntegrityError:
95                 QMessageBox.about(self, "Error", "Please make sure the item exists")
96
97     def checkExistingItem(self):
98         addedAlready = False
99         itemsOrdered = []
100         item = ""
101
102         with sqlite3.connect("restaurant.db") as db:
103             cursor = db.cursor()
104             cursor.execute("""SELECT
105                             Items.ItemName
106                             FROM Items
107                             INNER JOIN Booking_Items
108                             ON Booking_Items.ItemID = Items.ItemID
```

```

108             WHERE Booking_Items.BookingID = ?
109             """, (self.bookingDetails[0],))
110     items = cursor.fetchall()
111     for each in items:
112         itemsOrdered.append(each[0])
113
114     try:
115         with sqlite3.connect("restaurant.db") as db:
116             cursor = db.cursor()
117             cursor.execute("""SELECT
118                             Items.ItemName
119                             FROM Items
120                             INNER JOIN Booking_Items
121                             ON Booking_Items.ItemID = Items.ItemID
122                             WHERE Booking_Items.BookingID = ?
123                             AND Items.ItemID =
124                                 ?""", (self.bookingDetails[0], self.ItemID))
125             item = cursor.fetchone()[0]
126     except TypeError:
127         pass
128
129     if item in itemsOrdered:
130         addedAlready = True
131
132     return addedAlready

```



#### 4.10.4 assign\_table\_customer.py

```
1 import sqlite3
2 import sys
3 from PyQt4.QtCore import *
4 from PyQt4.QtGui import *
5 from table_display import *
6 import time
7 from cascade_style_sheet import *
8
9 class AssignCustomer(QDialog):
10     """this class will be used to either assign a customer that has
11        made a booking to a table or assign a customer that has not made a booking
12        to a table"""
13
14     def __init__(self, TableNumber):
15         super().__init__()
16         self.setWindowTitle("Assign customer to table {}".format(TableNumber))
17         self.setMinimumSize(600,600)
18         self.tableNumber = TableNumber
19         self.setStyleSheet(css)
20
21         self.titleFont = QFont()
22         self.titleFont.setPointSize(15)
23
24
25         self.todays_bookings_label = QLabel("Todays bookings for table
        {}".format(TableNumber))
```

200

```
26     self.todays_bookings_label.setFont(self.titleFont)
27     self.todays_bookings_label.setAlignment(Qt.AlignLeft)
28     self.todays_bookings_label.setFixedWidth(400)
29
30
31     self.main_assign_layout = QVBoxLayout()
32     self.choose_customer = QHBoxLayout()
33     self.create_combo_box(TableNumber)
34     self.add_customer_layout = QGridLayout()
35     self.create_complete_layout = QHBoxLayout()
36
37
38     self.choose_customer.addWidget(self.customer_combo_box)
39     self.select_customer = QPushButton("Select")
40     self.choose_customer.addWidget(self.select_customer)
41     self.select_customer.clicked.connect(self.select_connect)
42
43     #create buttons
44     self.create_complete = QPushButton("Create")
45     self.create_complete.clicked.connect(self.create_booking)
46
47     #labels
48     self.table_number_label = QLabel("Table Number : ")
49     self.number_of_people_label = QLabel("Number Of People : ")
50     self.time_arrived_label = QLabel("Time Of Arrival : ")
51     self.date_arrived_label = QLabel("Date Of Arrival : ")
52
53     self.systemtime = time.strftime("%H:%M")
```

```
54 self.system_time_label = QLineEdit(self.systemtime)
55 self.system_time_label.setReadOnly(True)
56 sizehint = self.system_time_label.sizeHint()
57 self.system_time_label.setMaximumSize(sizehint)
58
59 self.systemdate = time.strftime("%d/%m/%Y")
60 self.system_date_label = QLineEdit(self.systemdate)
61 self.system_date_label.setReadOnly(True)
62 self.system_date_label.setMaximumSize(sizehint)
63
64 self.display_table_number = QLineEdit("{0}".format(TableNumber))
65 self.display_table_number.setReadOnly(True)
66 self.display_table_number.setMaximumSize(sizehint)
67
68 regexp = QRegExp("^\\d\\d?$")
69 validator = QRegExpValidator(regexp)
70 self.input_number_of_people = QLineEdit()
71 self.input_number_of_people.setValidator(validator)
72 self.input_number_of_people.setMaximumSize(sizehint)
73
74
75 displayQuery = """SELECT
76                 Customers.FirstName,
77                 Customers.LastName,
78                 Bookings.NumberOfPeople,
79                 Bookings.Time
80                 FROM Customers
81                 INNER JOIN Bookings
```

```
82         ON Customers.CustomerID = Bookings.CustomerID
83         WHERE Bookings.Date = '{0}'
84         AND Bookings.TableNumber = {1}
85         """.format(self.systemdate, TableNumber)
86
87     self.display_customers = DisplayTable()
88     self.display_customers.show_results(displayQuery)
89
90
91     self.add_customer_layout.addWidget(self.table_number_label, 0, 0)
92     self.add_customer_layout.addWidget(self.display_table_number, 0, 1)
93     self.add_customer_layout.addWidget(self.time_arrived_label, 1, 0)
94     self.add_customer_layout.addWidget(self.date_arrived_label, 2, 0)
95     self.add_customer_layout.addWidget(self.system_time_label, 1, 1)
96     self.add_customer_layout.addWidget(self.system_date_label, 2, 1)
97     self.add_customer_layout.addWidget(self.number_of_people_label, 3, 0)
98     self.add_customer_layout.addWidget(self.input_number_of_people, 3, 1)
99     self.add_customer_layout.addWidget(self.create_complete, 4, 0, 2, 2)
100
101     self.assign_street_box = QGroupBox("Customer that has not booked in advance")
102     self.assign_street_box.setLayout(self.add_customer_layout)
103
104     self.assign_booked_box = QGroupBox("Customer that has booked in advance")
105     self.assign_booked_box.setLayout(self.choose_customer)
106
107     self.main_assign_layout.addWidget(self.todays_bookings_label)
108     self.main_assign_layout.addWidget(self.display_customers)
109     self.main_assign_layout.addWidget(self.assign_booked_box)
```

```

110         self.main_assign_layout.addWidget(self.assign_street_box)
111         self.setLayout(self.main_assign_layout)
112
113         self.exec_()
114
115     def create_booking(self):
116         #create bookingID for customer that has walked in
117         TableNumber = self.display_table_number.text()
118         CustomerID = 1
119         NumberOfPeople = self.input_number_of_people.text()
120         Date = self.systemdate
121         Time = self.systemtime
122
123         Booking = (CustomerID, TableNumber, NumberOfPeople, Date, Time)
124
125
126
127         if len(NumberOfPeople) > 0 and (int(NumberOfPeople)>0):
128
129             with sqlite3.connect("restaurant.db") as db:
130                 cursor = db.cursor()
131                 sql = "insert into
132                     Bookings(CustomerID,TableNumber,NumberOfPeople,Date,Time) values
133                     (?, ?, ?, ?, ?)"
134                 cursor.execute("PRAGMA foreign_keys = ON")
135                 cursor.execute(sql, Booking)
136                 db.commit()

```

```

136         with sqlite3.connect("restaurant.db") as db:
137             cursor = db.cursor()
138             cursor.execute("select * from Bookings where CustomerID = {?} and
                             TableNumber = {1} and NumberOfPeople = {2} and Date = '{3}' and
                             Time = '{4}'
                             ".format(CustomerID,TableNumber,NumberOfPeople,Date,Time))
139             self.bookingDetails = cursor.fetchone()
140
141         self.close()
142         return self.bookingDetails
143     else:
144         print("Please enter a valid number.")
145
146
147
148     def select_connect(self):
149         TodaysDate = time.strftime("%d/%m/%Y")
150         customerCurrentIndex = self.customer_combo_box.currentIndex()
151         print("Customer : {0}".format(customerCurrentIndex))
152         CustomerID = self.CustomerList[customerCurrentIndex]
153         print("Customer ID: {0}".format(CustomerID))
154
155         with sqlite3.connect("restaurant.db") as db:
156             cursor = db.cursor()
157             cursor.execute("select * from Bookings where CustomerID = {0} and
                             TableNumber = {1} and Date =
                             '{2}'".format(CustomerID,self.tableNumber,TodaysDate))
158             self.bookingDetails = cursor.fetchone()

```

```

159         print(self.bookingDetails)
160
161     self.close()
162
163     return self.bookingDetails
164
165     def create_combo_box(self, TableNumber):
166         self.CustomerList = []
167         CustomerLastName = []
168         TodaysDate = time.strftime("%d/%m/%Y")
169
170         ## get all customer IDs that are on table _
171         with sqlite3.connect("restaurant.db") as db:
172             cursor = db.cursor()
173             cursor.execute("select CustomerID from Bookings where TableNumber = {0}
174                             and Date = '{1}'".format(TableNumber, TodaysDate))
175             customers = cursor.fetchall()
176             for each in customers:
177                 self.CustomerList.append(each[0])
178
179         ## get all last names from previous fetchall
180         for customer in self.CustomerList:
181             with sqlite3.connect("restaurant.db") as db:
182                 cursor = db.cursor()
183                 cursor.execute("select LastName from Customers where CustomerID =
184                                 {0}".format(customer))
185                 customer = cursor.fetchone()
186                 CustomerLastName.append(customer[0])

```

```
185         #create combo, insert all last names from fetchall
186         self.customer_combo_box = QComboBox(self)
187         for each in CustomerLastName:
188             self.customer_combo_box.addItem(each)
189
190
191 if __name__ == "__main__":
192     TableNumber = 1
193     application = QApplication(sys.argv)
194     window = AssignCustomer(TableNumber)
195     window.show()
196     window.raise_()
197     application.exec()
```

---



#### 4.10.5 cascade\_style\_sheet.py

---

```
1
2  css = """QMainWindow{
3              background-color: #F9F9F9;
4              font-family: Verdana;
5              font-size: 12pt;}
6
7  QDialog{background-color: #F9F9F9;}
8
9  QToolBar{
10             background-color: #F9F9F9;
11             }
12
13  QPushButton{
14             font-family: Verdana;
15             font-size: 10pt;
16             font-weight: bold;
17             color: white;
18             background-color: #727272;
19             height: 20;
20             border: 0px}
21
22  QPushButton:pressed{
23             font-family: Verdana;
24             font-size: 11pt;
25             background-color: #CCCCCC;}
26
```

```
27 QLineEdit{
28     font-family: Verdana;
29     font-size: 9pt;}
30
31 QMenu{
32     background-color: #EAEAEA;
33     font-family: Verdana;
34     font-size: 9pt;
35     color: black;}
36
37 QMenu:item:selected:enabled{
38     background: rgb(220, 220, 220);}
39
40 QMenu:item:disabled{
41     color: rgb(220, 220, 220);}
42
43 QMenuBar{ background-color: #EAEAEA;
44     }
45
46 QMenuBar:item{
47     background-color: #EAEAEA;
48     font-family: Verdana;
49     font-size: 10pt;
50     color: rgb(106,106,106);}
51
52 QLabel{
53     font-family: Verdana;
54     font-size: 11pt;
```

```
55         color: rgb(70,70,70)}
56
57     QComboBox{
58         font-family: Verdana;
59         font-size: 10pt;
60         border-style: solid;
61         border-width: 1px;
62         color: rgb(70,70,70);}
63
64     QGroupBox{
65         font-family: Verdana;
66         font-size: 11pt;
67         color: rgb(70,70,70);}
68
69     QTableView{
70
71         font-family: Verdana;
72         font-size: 9pt;
73         color: rgb(70,70,70);
74         border-style: solid;
75         border-width: 1px;
76         border-color: rgb(200,200,200);
77         selection-background-color: rgb(255,255,255);
78         selection-color: rgb(70,70,70);}
79
80
81     QHeaderView:section{
82         background: white ;
```

```
83         font-family: Verdana;
84         font-size: 7pt;
85         border-style: solid;
86         border-width: 1px;
87         font-weight: bold;
88         border-color: rgb(200,200,200);
89         color: rgb(70,70,70);
90         height: 30px;
91     }
92
93     QHeaderView:section:pressed{
94         background: rgb(200,200,200);
95         font-family: Verdana;
96         font-size: 12pt;
97         border-style: solid;
98         border-width: 1px;
99         color: white;
100        height: 30px;}
101
102
103
104
105
106    " " "
```

---

#### 4.10.6 delete\_booking.py

---

```
1 import sys
2 import sqlite3
3 from PyQt4.QtCore import *
4 from PyQt4.QtGui import *
5 from table_display import *
6
7 class DeleteBookingWindow(QWidget):
8     """this class creates a widget to delete bookings"""
9
10    def __init__(self):
11        super().__init__()
12        self.setWindowTitle("Delete Booking")
13
14        self.main_layout = QVBoxLayout()
15        self.input_layout = QHBoxLayout()
16
17
18        self.display_table = DisplayTable()
19        self.display_table.show_table("Bookings")
20        self.bookingIDlabel = QLabel("Booking ID")
21        self.bookingIDlabel.setMaximumSize(133,20)
22
23        regexp = QRegExp("^\\d\\d\\d")
24        validator = QRegExpValidator(regexp)
25        self.input_bookingID = QLineEdit()
26        self.input_bookingID.setValidator(validator)
```

212

```

27         self.input_bookingID.setMaximumSize(self.input_bookingID.sizeHint())
28
29         self.delete_bookingID = QPushButton("Delete BookingID")
30         self.delete_bookingID.setMaximumSize(133,20)
31         self.delete_bookingID.clicked.connect(self.delete_booking)
32
33         self.input_layout.addWidget(self.bookingIDlabel)
34         self.input_layout.addWidget(self.input_bookingID)
35         self.input_layout.addWidget(self.delete_bookingID)
36
37         self.main_layout.addWidget(self.display_table)
38         self.main_layout.addLayout(self.input_layout)
39         self.setLayout(self.main_layout)
40
213 41
42     def delete_booking(self):
43         booking = self.input_bookingID.text()
44         try:
45             with sqlite3.connect("restaurant.db") as db:
46                 cursor = db.cursor()
47                 sql = ("delete from Bookings where BookingID = {0}".format(booking))
48                 cursor.execute("PRAGMA foreign_keys = ON")
49                 cursor.execute(sql)
50                 db.commit()
51
52                 self.display_table.refresh()
53
54         except sqlite3.OperationalError:

```

```
55         QMessageBox.about(self, "Error", "Please make sure you have filled in the
        required field")
56
57
58
59
60 if __name__ == "__main__":
61     application = QApplication(sys.argv)
62     window = DeleteBookingWindow()
63     window.show()
64     window.raise_()
65     application.exec()
```

---

#### 4.10.7 delete\_item\_off\_menu.py

---

```
1 import sys
2 import sqlite3
3 from PyQt4.QtCore import *
4 from PyQt4.QtGui import *
5 from table_display import *
6
7 class DeleteItemOffMenu(QWidget):
8     """this class creates a widget to delete items off the menu"""
9
10    def __init__(self):
11        super().__init__()
12        self.display_table = DisplayTable()
13        self.display_table.show_table("Items")
14
15        #create layouts
16        self.main_layout = QVBoxLayout()
17        self.delete_item_name_layout = QHBoxLayout()
18        self.delete_itemID_layout = QHBoxLayout()
19
20
21        #create buttons
22        self.delete_item_name = QPushButton("Delete Item")
23        self.delete_itemID = QPushButton("Delete Item")
24
25
26        #labels
```



```
27     self.item_name_label = QLabel("Item Name : ")
28     self.itemID_label = QLabel("Item ID : ")
29
30
31     #line edit
32     self.input_item_name = QLineEdit()
33     self.input_item_name.setMaximumSize(300,30)
34
35     regexp = QRegExp("^\\d\\d\\d\\d?$")
36     validator = QRegExpValidator(regexp)
37     self.input_itemID = QLineEdit()
38     self.input_itemID.setValidator(validator)
39     self.input_itemID.setMaximumSize(300,30)
40
41     #add widgets to delete item name layout
42     self.delete_item_name_layout.addWidget(self.item_name_label)
43     self.delete_item_name_layout.addWidget(self.input_item_name)
44     self.delete_item_name_layout.addWidget(self.delete_item_name)
45
46     #add widgets to delete itemID layout
47     self.delete_itemID_layout.addWidget(self.itemID_label)
48     self.delete_itemID_layout.addWidget(self.input_itemID)
49     self.delete_itemID_layout.addWidget(self.delete_itemID)
50
51     #add layouts to main layout
52     self.main_layout.addWidget(self.display_table)
53     self.main_layout.addLayout(self.delete_item_name_layout)
54     self.main_layout.addLayout(self.delete_itemID_layout)
```

```
55
56
57     #set layout
58     self.setLayout(self.main_layout)
59
60     #connections
61     self.delete_item_name.clicked.connect(self.delete_item_off_menu)
62     self.delete_itemID.clicked.connect(self.delete_itemID_off_menu)
63
64     def delete_item_off_menu(self):
65         item_name = self.input_item_name.text()
66         print(item_name)
67         if len(item_name)>1:
68             item_name = (item_name,)
69             with sqlite3.connect("restaurant.db") as db:
70                 cursor = db.cursor()
71                 sql = "delete from Items where ItemName = ?"
72                 cursor.execute(sql,item_name)
73                 db.commit()
74
75             self.display_table.refresh()
76         else:
77             QMessageBox.about(self,"Error","Please make sure you have filled in the
              required field ")
78
79     def delete_itemID_off_menu(self):
80         itemID = self.input_itemID.text()
81         print(itemID)
```

```
82         try:
83
84             with sqlite3.connect("restaurant.db") as db:
85                 cursor = db.cursor()
86                 sql = ("delete from Items where ItemID = {0}".format(itemID))
87                 cursor.execute(sql)
88                 db.commit()
89
90             self.display_table.refresh()
91
92         except (sqlite3.OperationalError) or AttributeError:
93             QMessageBox.about(self, "Error", "Please make sure you have filled in the
               required field correctly")
218 94 if __name__ == "__main__":
95     application = QApplication(sys.argv)
96     window = DeleteItemOffMenu()
97     window.show()
98     window.raise_()
99     application.exec()
```

---

#### 4.10.8 delete\_item\_off\_order.py

---

```
1 import sys
2 import sqlite3
3 from PyQt4.QtCore import *
4 from PyQt4.QtGui import *
5 from table_display import *
6 from cascade_style_sheet import *
7
8 class DeleteItemOffOrder(QDialog):
9     orderitemDeleted = pyqtSignal()
10     """this class will be used to delete an item off the order"""
11
12     def __init__(self, bookingDetails):
13         super().__init__()
14         self.setWindowTitle("Delete Item Off Order")
15         self.bookingDetails = bookingDetails
16         self.setMinimumSize(600,600)
17         self.setStyleSheet(css)
18
19         #create layouts
20         self.main_layout = QVBoxLayout()
21         self.delete_item_layout = QGridLayout()
22         self.delete_complete_layout = QHBoxLayout()
23
24         #create buttons
25         self.delete_complete = QPushButton("Delete Item")
26
```

```
27     #labels
28     self.itemID_label = QLabel("Item ID : ")
29     self.itemQuantity_label = QLabel("Item Quantity : ")
30
31     #line edit
32     regexp = QRegExp("^\\d\\d\\d?")
33     validator = QRegExpValidator(regexp)
34     self.input_itemID = QLineEdit()
35     self.input_itemID.setValidator(validator)
36     self.input_itemID.setMaximumSize(300,30)
37     self.input_itemQuantity = QLineEdit()
38     self.input_itemQuantity.setValidator(validator)
39     self.input_itemQuantity.setMaximumSize(300,30)
40
41     #table
42     self.item_table = DisplayTable()
43     query = """SELECT
44             Booking_Items.Quantity,
45             Items.ItemID,
46             Items.ItemName,
47             Items.ItemPrice
48         FROM Items
49         INNER JOIN Booking_Items
50         ON Booking_Items.ItemID = Items.ItemID
51         WHERE Booking_Items.BookingID = {0}
52         """.format(self.bookingDetails[0])
53     self.item_table.show_results(query)
54
```

```
55
56     #add labels to layout
57     self.delete_item_layout.addWidget(self.itemID_label,0,0)
58     self.delete_item_layout.addWidget(self.itemQuantity_label,1,0)
59
60     #add line edit to layout
61     self.delete_item_layout.addWidget(self.input_itemID,0,1)
62     self.delete_item_layout.addWidget(self.input_itemQuantity,1,1)
63     #add button to layout
64     self.delete_complete_layout.addWidget(self.delete_complete)
65
66     #add layouts to main layout
67     self.main_layout.addWidget(self.item_table)
68     self.main_layout.addLayout(self.delete_item_layout)
69     self.main_layout.addLayout(self.delete_complete_layout)
70     self.setLayout(self.main_layout)
71
72     #connections
73     self.delete_complete.clicked.connect(self.delete_item_off_order)
74
75     def delete_item_off_order(self,bookingDetails):
76         self.bookingID = self.bookingDetails[0]
77         self.ItemID = self.input_itemID.text()
78         Quantity = self.input_itemQuantity.text()
79
80         MenuItem = (self.bookingID,self.ItemID)
81
82         OneQuantity = self.checkQuantity()
```

```

83
84     if OneQuantity == False:
85         with sqlite3.connect("restaurant.db") as db:
86             cursor = db.cursor()
87             cursor.execute("select Quantity from Booking_Items where ItemID=? and
88                             BookingID = ?",(self.ItemID,self.bookingID))
89             dbquantity = cursor.fetchone()[0]
90
91             newQuantity = dbquantity - int(Quantity)
92
93             if newQuantity <= 0:
94
95                 with sqlite3.connect("restaurant.db") as db:
96                     cursor = db.cursor()
97                     sql = "delete from Booking_Items where BookingID = ? and ItemID =
98                           ? "
99                     cursor.execute("PRAGMA foreign_keys = ON")
100                     cursor.execute(sql,MenuItem)
101                     db.commit()
102
103             else:
104
105                 updateOrder = (newQuantity,self.ItemID)
106                 with sqlite3.connect("restaurant.db") as db:
107                     cursor = db.cursor()
108                     sql = "update Booking_Items set Quantity=? where ItemID=?"
109                     cursor.execute("PRAGMA foreign_keys = ON")
110                     cursor.execute(sql,updateOrder)
111                     db.commit()

```

```
109         self.orderitemDeleted.emit()
110
111
112     elif OneQuantity == True:
113
114         with sqlite3.connect("restaurant.db") as db:
115             cursor = db.cursor()
116             sql = "delete from Booking_Items where BookingID = ? and ItemID = ? "
117             cursor.execute("PRAGMA foreign_keys = ON")
118             cursor.execute(sql, MenuItem)
119             db.commit()
120
121         self.orderitemDeleted.emit()
122
223 def checkQuantity(self):
123     OneQuantity = False
124
125     try:
126         with sqlite3.connect("restaurant.db") as db:
127             cursor = db.cursor()
128             cursor.execute("""SELECT
129                             Quantity
130                             FROM Booking_Items
131                             WHERE BookingID = ?
132                             AND ItemID = ?""", (self.bookingID, self.ItemID))
133             itemQuantity = cursor.fetchone()[0]
134             print(itemQuantity)
135
136     if (int(itemQuantity)) == 1:
```



```
137         OneQuantity = True
138
139     return OneQuantity
140 except TypeError:
141     pass
```

---

#### 4.10.9 main\_window.py

---

```
1
2 import sys
3 import time
4
5 from PyQt4.QtCore import *
6 from PyQt4.QtGui import *
7 from manage_booking import *
8 from manage_order import *
9
10 from add_item_to_menu import *
11 from add_booking import *
225 12
13 from delete_item_off_menu import *
14 from delete_booking import *
15
16 from table_display import *
17 from search_order import *
18
19 from update_item_price import *
20 from update_booking import *
21
22 from radio_button_widget_class import *
23 from assign_table_customer import *
24 from cascade_style_sheet import *
25
26 class RestaurantWindow(QMainWindow):
```

```
27     """this class creates a main window to observe the restaurant"""
28
29     def __init__(self):
30         super().__init__()
31         self.setWindowTitle("Restaurant Simulation")
32         self.setStyleSheet(css)
33
34         self.TableOneOccupied = False
35         self.TableTwoOccupied = False
36         self.TableThreeOccupied = False
37         self.TableFourOccupied = False
38         self.TableFiveOccupied = False
39         self.TableSixOccupied = False
40         self.TableSevenOccupied = False
41         self.TableEightOccupied = False
42         self.TableNineOccupied = False
43         self.TableTenOccupied = False
44         self.TableElevenOccupied = False
45         self.TableTwelveOccupied = False
46         self.TableThirteenOccupied = False
47         self.TableFourteenOccupied = False
48         self.TableFifteenOccupied = False
49         self.TableSixteenOccupied = False
50
51
52         self.titleFont = QFont()
53         self.titleFont.setPointSize(20)
54         self.titleFont.setBold(True)
```

```
55
56     self.create_menu_bar()
57     self.create_tool_bar()
58
59     self.main_layout()
60     #stacked layouts
61
62     self.setMinimumSize(1080,800)
63
64     def create_tool_bar(self):
65         #create toolbar
66         self.main_screen_tool_bar = QToolBar()
67         self.orders_tool_bar = QToolBar()
68         self.bookings_tool_bar = QToolBar()
69         self.view_customers = QToolBar()
70         self.view_dishes = QToolBar()
71         self.view_drinks = QToolBar()
72
73
74         self.main_screen_label_bar = QAction("Main Screen",self)
75         self.main_screen_label_bar.setToolTip("This will direct you to main screen")
76         self.main_screen_tool_bar.addAction(self.main_screen_label_bar)
77         self.main_screen_label_bar.triggered.connect(self.main_layout)
78
79         self.orders_label_bar = QAction("Search Order",self)
80         self.orders_label_bar.setToolTip("Search an order by using a booking ID")
81         self.orders_tool_bar.addAction(self.orders_label_bar)
82         self.orders_label_bar.triggered.connect(self.search_order_connect)
```

```
83
84     self.bookings_label_bar = QAction("View Bookings",self)
85     self.bookings_label_bar.setToolTip("All bookings will be displayed")
86     self.bookings_tool_bar.addAction(self.bookings_label_bar)
87     self.bookings_label_bar.triggered.connect(self.view_bookings_connect)
88
89     self.view_customers_label = QAction("View Customers",self)
90     self.view_customers_label.setToolTip("All customers will be displayed")
91     self.view_customers.addAction(self.view_customers_label)
92     self.view_customers_label.triggered.connect(self.view_customers_connect)
93
94     self.view_dishes_label = QAction("View Dishes",self)
95     self.view_dishes_label.setToolTip("All dishes will be displayed")
96     self.view_dishes.addAction(self.view_dishes_label)
97     self.view_dishes_label.triggered.connect(self.view_dishes_connect)
98
99     self.view_drinks_label = QAction("View Drinks",self)
100    self.view_drinks_label.setToolTip("All drinks will be displayed")
101    self.view_drinks.addAction(self.view_drinks_label)
102    self.view_drinks_label.triggered.connect(self.view_drinks_connect)
103
104    self.addToolBar(self.main_screen_tool_bar)
105    self.addToolBar(self.orders_tool_bar)
106    self.addToolBar(self.bookings_tool_bar)
107    self.addToolBar(self.view_customers)
108    self.addToolBar(self.view_dishes)
109    self.addToolBar(self.view_drinks)
110
```

```

111     def create_menu_bar(self):
112         #actions
113         self.add_item_box = QAction("Add Item",self)
114         self.delete_item_box = QAction("Delete Item",self)
115         self.update_item_box = QAction("Update Item Price",self)
116
117         self.add_booking_box = QAction("Add Booking",self)
118         self.delete_booking_box = QAction("Delete Booking",self)
119         self.update_booking_box = QAction("Update Booking",self)
120
121         self.menu = QMenuBar()
122         self.menu_bar = self.menu.addMenu("Item Menu")
123         self.bookings_bar = self.menu.addMenu("Bookings")
124
125         self.setMenuBar(self.menu)
126
127         self.menu_bar.addAction(self.add_item_box)
128         self.menu_bar.addAction(self.delete_item_box)
129         self.menu_bar.addAction(self.update_item_box)
130
131         self.bookings_bar.addAction(self.add_booking_box)
132         self.bookings_bar.addAction(self.delete_booking_box)
133         self.bookings_bar.addAction(self.update_booking_box)
134
135         #connections
136         self.add_item_box.triggered.connect(self.add_item_connect)
137         self.delete_item_box.triggered.connect(self.delete_item_connect)
138

```

```

139         self.update_item_box.triggered.connect(self.update_item_connect)
140
141         self.add_booking_box.triggered.connect(self.add_booking_connect)
142         self.delete_booking_box.triggered.connect(self.delete_booking_connect)
143         self.update_booking_box.triggered.connect(self.update_booking_connect)
144
145     def main_layout(self):
146
147         #create layouts
148         self.main_layout = QVBoxLayout()
149         self.booking_layout = QVBoxLayout() #box 1,0
150         self.table_radio_layout = QVBoxLayout()
151
152
153         #radio button
154         tableList = []
155         for each in range(1,17):
156             tableList.append("Table {0}".format(each))
157
158         self.table_buttons = RadioButtonWidget("Table Numbers", "Please select a
            Table" , tableList)
159         self.select_table_button = QPushButton("Select Table")
160
161         self.select_table_button.clicked.connect(self.radio_button_connect)
162
163
164         self.table_radio_layout.addWidget(self.table_buttons)
165         self.table_radio_layout.addWidget(self.select_table_button)

```

166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193

```
#booking section
self.manage_bookings = QPushButton("Manage Bookings") # Manage bookings button
TodaysDate = time.strftime("%d/%m/%Y")
print(TodaysDate)

bookingQuery = """SELECT
    Customers.FirstName,
    Customers.LastName,
    Bookings.NumberOfPeople,
    Bookings.TableNumber,
    Bookings.Time
FROM Customers
INNER JOIN Bookings
ON Customers.CustomerID = Bookings.CustomerID
WHERE Bookings.Date = '{0}'
ORDER BY Bookings.Time
""".format(TodaysDate)

self.display_bookings = DisplayTable()
self.display_bookings.show_results(bookingQuery)

#connections
```



```

194         self.manage_bookings.clicked.connect(self.manage_booking_connect)
195
196
197         #add widgets to booking layout
198         self.todays_bookings_label = QLabel("Todays Bookings")
199         self.todays_bookings_label.setFont(self.titleFont)
200         self.todays_bookings_label.setFixedWidth(400)
201
202         self.booking_layout.addWidget(self.todays_bookings_label)
203         self.booking_layout.addWidget(self.display_bookings)
204         self.booking_layout.addWidget(self.manage_bookings)
205
206
207
208         #add layouts to main layout
209         self.main_layout.addLayout(self.table_radio_layout)
210         self.main_layout.addLayout(self.booking_layout)
211
212
213         #create a widget to display main layout
214         self.main_widget_layout = QWidget()
215         self.main_widget_layout.setLayout(self.main_layout)
216
217         self.setCentralWidget(self.main_widget_layout)
218
219
220
221     def radio_button_connect(self):

```

```

222     TableNumber = self.table_buttons.selected_button()
223     print("Table Number {0} Selected".format(TableNumber))
224
225     try:
226         if TableNumber == 1:
227             if self.TableOneOccupied == False:
228                 self.table1 = AssignCustomer(TableNumber)
229                 bookingDetails = self.table1.bookingDetails
230                 self.TableOneOrder = OrderWindow(bookingDetails)
231                 self.TableOneOccupied = True
232                 if self.TableOneOrder.Finished == True:
233                     self.TableOneOccupied = False
234             elif self.TableOneOccupied == True:
235                 bookingDetails = self.table1.bookingDetails
236                 self.TableOneOrder = OrderWindow(bookingDetails)
237                 if self.TableOneOrder.Finished == True:
238                     self.TableOneOccupied = False
239
240         elif TableNumber == 2:
241             if self.TableTwoOccupied == False:
242                 self.table2 = AssignCustomer(TableNumber)
243                 bookingDetails = self.table2.bookingDetails
244                 self.TableTwoOrder = OrderWindow(bookingDetails)
245                 self.TableTwoOccupied = True
246                 if self.TableTwoOrder.Finished == True:
247                     self.TableTwoOccupied = False
248             elif self.TableTwoOccupied == True:
249                 bookingDetails = self.table2.bookingDetails

```

```

250         self.TableTwoOrder = OrderWindow(bookingDetails)
251         if self.TableTwoOrder.Finished == True:
252             self.TableTwoOccupied = False
253
254     elif TableNumber == 3:
255         if self.TableThreeOccupied == False:
256             self.table3 = AssignCustomer(TableNumber)
257             bookingDetails = self.table3.bookingDetails
258             self.TableThreeOrder = OrderWindow(bookingDetails)
259             self.TableThreeOccupied = True
260             if self.TableThreeOrder.Finished == True:
261                 self.TableThreeOccupied = False
262         elif self.TableThreeOccupied == True:
263             bookingDetails = self.table3.bookingDetails
264             self.TableThreeOrder = OrderWindow(bookingDetails)
265             if self.TableThreeOrder.Finished == True:
266                 self.TableThreeOccupied = False
267
268     elif TableNumber == 4:
269         if self.TableFourOccupied == False:
270             self.table4 = AssignCustomer(TableNumber)
271             bookingDetails = self.table4.bookingDetails
272             self.TableFourOrder = OrderWindow(bookingDetails)
273             self.TableFourOccupied = True
274             if self.TableFourOrder.Finished == True:
275                 self.TableFourOccupied = False
276         elif self.TableFourOccupied == True:
277             bookingDetails = self.table4.bookingDetails

```

```

278         self.TableFourOrder = OrderWindow(bookingDetails)
279         if self.TableFourOrder.Finished == True:
280             self.TableFourOccupied = False
281
282     elif TableNumber == 5:
283         if self.TableFiveOccupied == False:
284             self.table5 = AssignCustomer(TableNumber)
285             bookingDetails = self.table5.bookingDetails
286             self.TableFiveOrder = OrderWindow(bookingDetails)
287             self.TableFiveOccupied = True
288             if self.TableFiveOrder.Finished == True:
289                 self.TableFiveOccupied = False
290         elif self.TableFiveOccupied == True:
291             bookingDetails = self.table5.bookingDetails
292             self.TableFiveOrder = OrderWindow(bookingDetails)
293             if self.TableFiveOrder.Finished == True:
294                 self.TableFiveOccupied = False
295
296     elif TableNumber == 6:
297         if self.TableSixOccupied == False:
298             self.table6 = AssignCustomer(TableNumber)
299             bookingDetails = self.table6.bookingDetails
300             self.TableSixOrder = OrderWindow(bookingDetails)
301             self.TableSixOccupied = True
302             if self.TableSixOrder.Finished == True:
303                 self.TableSixOccupied = False
304         elif self.TableSixOccupied == True:
305             bookingDetails = self.table6.bookingDetails

```

```
306         self.TableSixOrder = OrderWindow(bookingDetails)
307         if self.TableSixOrder.Finished == True:
308             self.TableSixOccupied = False
309
310
311     elif TableNumber == 7:
312         if self.TableSevenOccupied == False:
313             self.table7 = AssignCustomer(TableNumber)
314             bookingDetails = self.table7.bookingDetails
315             self.TableSevenOrder = OrderWindow(bookingDetails)
316             self.TableSevenOccupied = True
317             if self.TableSevenOrder.Finished == True:
318                 self.TableSevenOccupied = False
319         elif self.TableSevenOccupied == True:
320             bookingDetails = self.table7.bookingDetails
321             self.TableSevenOrder = OrderWindow(bookingDetails)
322             if self.TableSevenOrder.Finished == True:
323                 self.TableSevenOccupied = False
324
325     elif TableNumber == 8:
326         if self.TableEightOccupied == False:
327             self.table8 = AssignCustomer(TableNumber)
328             bookingDetails = self.table8.bookingDetails
329             self.TableEightOrder = OrderWindow(bookingDetails)
330             self.TableEightOccupied = True
331             if self.TableEightOrder.Finished == True:
332                 self.TableEightOccupied = False
333         elif self.TableEightOccupied == True:
```

```

334         bookingDetails = self.table8.bookingDetails
335         self.TableEightOrder = OrderWindow(bookingDetails)
336         if self.TableEightOrder.Finished == True:
337             self.TableEightOccupied = False
338
339     elif TableNumber == 9:
340         if self.TableNineOccupied == False:
341             self.table9 = AssignCustomer(TableNumber)
342             bookingDetails = self.table9.bookingDetails
343             self.TableNineOrder = OrderWindow(bookingDetails)
344             self.TableNineOccupied = True
345             if self.TableNineOrder.Finished == True:
346                 self.TableNineOccupied = False
347         elif self.TableNineOccupied == True:
348             bookingDetails = self.table9.bookingDetails
349             self.TableNineOrder = OrderWindow(bookingDetails)
350             if self.TableNineOrder.Finished == True:
351                 self.TableNineOccupied = False
352
353     elif TableNumber == 10:
354         if self.TableTenOccupied == False:
355             self.table10 = AssignCustomer(TableNumber)
356             bookingDetails = self.table10.bookingDetails
357             self.TableTenOrder = OrderWindow(bookingDetails)
358             self.TableTenOccupied = True
359             if self.TableTenOrder.Finished == True:
360                 self.TableTenOccupied = False
361         elif self.TableTenOccupied == True:

```

```
362         bookingDetails = self.table10.bookingDetails
363         self.TableTenOrder = OrderWindow(bookingDetails)
364         if self.TableTenOrder.Finished == True:
365             self.TableTenOccupied = False
366
367     elif TableNumber == 11:
368         if self.TableElevenOccupied == False:
369             self.table11 = AssignCustomer(TableNumber)
370             bookingDetails = self.table11.bookingDetails
371             self.TableElevenOrder = OrderWindow(bookingDetails)
372             self.TableElevenOccupied = True
373             if self.TableElevenOrder.Finished == True:
374                 self.TableElevenOccupied = False
375         elif self.TableElevenOccupied == True:
376             bookingDetails = self.table11.bookingDetails
377             self.TableElevenOrder = OrderWindow(bookingDetails)
378             if self.TableElevenOrder.Finished == True:
379                 self.TableElevenOccupied = False
380
381     elif TableNumber == 12:
382         if self.TableTwelveOccupied == False:
383             self.table12 = AssignCustomer(TableNumber)
384             bookingDetails = self.table12.bookingDetails
385             self.TableTwelveOrder = OrderWindow(bookingDetails)
386             self.TableTwelveOccupied = True
387             if self.TableTwelveOrder.Finished == True:
388                 self.TableTwelveOccupied = False
389         elif self.TableTwelveOccupied == True:
```

```

390         bookingDetails = self.table12.bookingDetails
391         self.TableTwelveOrder = OrderWindow(bookingDetails)
392         if self.TableTwelveOrder.Finished == True:
393             self.TableTwelveOccupied = False
394
395     elif TableNumber == 13:
396         if self.TableThirteenOccupied == False:
397             self.table13 = AssignCustomer(TableNumber)
398             bookingDetails = self.table13.bookingDetails
399             self.TableThirteenOrder = OrderWindow(bookingDetails)
400             self.TableThirteenOccupied = True
401             if self.TableThirteenOrder.Finished == True:
402                 self.TableThirteenOccupied = False
403         elif self.TableThirteenOccupied == True:
404             bookingDetails = self.table13.bookingDetails
405             self.TableThirteenOrder = OrderWindow(bookingDetails)
406             if self.TableThirteenOrder.Finished == True:
407                 self.TableThirteenOccupied = False
408
409     elif TableNumber == 14:
410         if self.TableFourteenOccupied == False:
411             self.table14 = AssignCustomer(TableNumber)
412             bookingDetails = self.table14.bookingDetails
413             self.TableFourteenOrder = OrderWindow(bookingDetails)
414             self.TableFourteenOccupied = True
415             if self.TableFourteenOrder.Finished == True:
416                 self.TableFourteenOccupied = False
417         elif self.TableFourteenOccupied == True:

```



```
418         bookingDetails = self.table14.bookingDetails
419         self.TableFourteenOrder = OrderWindow(bookingDetails)
420         if self.TableFourteenOrder.Finished == True:
421             self.TableFourteenOccupied = False
422
423     elif TableNumber == 15:
424         if self.TableFifteenOccupied == False:
425             self.table15 = AssignCustomer(TableNumber)
426             bookingDetails = self.table15.bookingDetails
427             self.TableFifteenOrder = OrderWindow(bookingDetails)
428             self.TableFifteenOccupied = True
429             if self.TableFifteenOrder.Finished == True:
430                 self.TableFifteenOccupied = False
431         elif self.TableFifteenOccupied == True:
432             bookingDetails = self.table15.bookingDetails
433             self.TableFifteenOrder = OrderWindow(bookingDetails)
434             if self.TableFifteenOrder.Finished == True:
435                 self.TableFifteenOccupied = False
436
437     elif TableNumber == 16:
438         if self.TableSixteenOccupied == False:
439             self.table16 = AssignCustomer(TableNumber)
440             bookingDetails = self.table16.bookingDetails
441             self.TableSixteenOrder = OrderWindow(bookingDetails)
442             self.TableSixteenOccupied = True
443             if self.TableSixteenOrder.Finished == True:
444                 self.TableSixteenOccupied = False
445         elif self.TableSixteenOccupied == True:
```

```

446         bookingDetails = self.table16.bookingDetails
447         self.TableSixteenOrder = OrderWindow(bookingDetails)
448         if self.TableSixteenOrder.Finished == True:
449             self.TableSixteenOccupied = False
450
451     except AttributeError:
452         pass
453
454     def add_item_connect(self):
455         self.add_menu_item = AddItemToMenu()
456         self.setCentralWidget(self.add_menu_item)
457
458     def delete_item_connect(self):
459         self.delete_menu_item = DeleteItemOffMenu()
460         self.setCentralWidget(self.delete_menu_item)
461
462     def view_customers_connect(self):
463         self.tool_bar_customers = DisplayTable()
464         self.tool_bar_customers.show_table("Customers")
465         self.setCentralWidget(self.tool_bar_customers)
466
467
468     def view_bookings_connect(self):
469         viewBooking = """SELECT
470                         Bookings.BookingID,
471                         Customers.CustomerID,
472                         Customers.FirstName,
473                         Customers.LastName,

```

```

474         Bookings.NumberOfPeople,
475         Bookings.TableNumber,
476         Bookings.Time,
477         Bookings.Date,
478         Customers.TelephoneNo
479     FROM Customers
480     INNER JOIN Bookings
481     ON Customers.CustomerID = Bookings.CustomerID
482     ORDER BY Bookings.Date,Bookings.Time"""
483
484     self.view_bookings = DisplayTable()
485     self.view_bookings.show_results(viewBooking)
486     self.setCentralWidget(self.view_bookings)
487
242 def manage_booking_connect(self):
488     self.manage_bookings = BookingWindow()
489     self.manage_bookings.add_button.clicked.connect(self.add_booking_connect)
490     #connection
491     self.manage_bookings.delete_button.clicked.connect(self.delete_booking_connect)
492     #connection
493     self.setCentralWidget(self.manage_bookings)
494
494 def update_item_connect(self):
495     self.update_item = UpdateItemPrice()
496     self.setCentralWidget(self.update_item)
497
498 def add_booking_connect(self):
499     self.add_booking = AddBookingWindow()

```

```

500         self.setCentralWidget(self.add_booking)
501
502     def delete_booking_connect(self):
503         self.delete_booking = DeleteBookingWindow()
504         self.setCentralWidget(self.delete_booking)
505
506
507     def view_dishes_connect(self):
508         filter_query = "ItemTypeID like '%1%'"
509
510         self.view_dishes_tool = DisplayTable()
511         self.view_dishes_tool.show_table("Items")
512         self.view_dishes_tool.model.setFilter(filter_query)
513         self.setCentralWidget(self.view_dishes_tool)
514
515     def view_drinks_connect(self):
516         filter_query = "ItemTypeID like '%2%'"
517
518         self.view_drinks_tool = DisplayTable()
519         self.view_drinks_tool.show_table("Items")
520         self.view_drinks_tool.model.setFilter(filter_query)
521         self.setCentralWidget(self.view_drinks_tool)
522
523     def search_order_connect(self):
524         self.search_order = SearchOrder()
525         self.setCentralWidget(self.search_order)
526
527     def update_booking_connect(self):

```

```
528         self.update_booking = UpdateBooking()
529         self.setCentralWidget(self.update_booking)
530
531
532     def main():
533         restaurant_simulation = QApplication(sys.argv) # create new application
534         restaurant_window = RestaurantWindow() #create new instance of main window
535         restaurant_window.show() #make instance visible
536         restaurant_window.raise_() #raise instance to top of window stack
537         restaurant_simulation.exec_() #monitor application for events
538
539     if __name__ == "__main__":
540         main()
```

---

#### 4.10.10 manage\_booking.py

```
1  #http://pyqt.sourceforge.net/Docs/PyQt4/qdate.html#currentDate
2
3  import sys
4  from PyQt4.QtCore import *
5  from PyQt4.QtGui import *
6
7  from table_display import *
8
9  class BookingWindow(QWidget):
10     """this class creates a widget to observe the bookings"""
11
12     def __init__(self):
13         super().__init__()
14
15         #create layouts
16         self.manage_layout = QVBoxLayout()
17         self.manage_booking = QHBoxLayout()
18
19         #create buttons
20         self.back_button = QPushButton("Back")
21         self.add_button = QPushButton("Add Booking")
22         self.delete_button = QPushButton("Delete Booking")
23
24         #add buttons to layouts
25         self.manage_booking.addWidget(self.add_button)
26         self.manage_booking.addWidget(self.delete_button)
```

```
27
28     self.display_booking_table = DisplayTable()
29     self.display_booking_table.show_table("Bookings")
30
31     self.manage_layout.addWidget(self.display_booking_table)
32     self.manage_layout.addLayout(self.manage_booking)
33     self.setLayout(self.manage_layout)
34
35
36 if __name__ == "__main__":
37     application = QApplication(sys.argv)
38     window = BookingWindow()
39     window.show()
40     window.raise_()
41     application.exec()
```

---

#### 4.10.11 manage\_order.py

```
1  #http://pyqt.sourceforge.net/Docs/PyQt4/qdate.html#currentDate
2
3  import sys
4  from PyQt4.QtCore import *
5  from PyQt4.QtGui import *
6  from table_display import *
7  from add_item_to_order import *
8  from print_invoice import *
9  from delete_item_off_order import *
10 from cascade_style_sheet import*
11
12 ##          bookingID = bookingDetails[0]
13 ##          customerID = bookingDetails[1]
14 ##          tableNumber = bookingDetails[2]
15 ##          numberPeople = bookingDetails[3]
16 ##          Date = bookingDetails[4]
17 ##          Time = bookingDetails[5]
18
19 class OrderWindow(QDialog):
20     """this class will be used to manage the orders"""
21
22     def __init__(self,bookingDetails):
23         super().__init__()
24         self.Finished = False
25         self.setFixedSize(1000,500)
26         self.setWindowTitle("Manage Order")
```



```
27     self.setStyleSheet(css)
28     self.bookingDetails = bookingDetails
29     self.CalcTotal()
30
31     self.add_button = QPushButton("Add")
32     self.delete_button = QPushButton("Delete")
33     self.finish_button = QPushButton("Finish")
34     self.preview_invoice = QPushButton("Invoice Preview")
35     self.invoice_button = QPushButton("Print Invoice")
36
37     self.add_button.clicked.connect(self.AddItem)
38     self.delete_button.clicked.connect(self.DeleteItem)
39     self.finish_button.clicked.connect(self.Finish)
40     self.invoice_button.clicked.connect(self.Invoice)
41     self.preview_invoice.clicked.connect(self.InvoicePreview)
42
43     self.drinks_label = QLabel("Drinks")
44     self.dishes_label = QLabel("Dishes")
45     self.table_number_label = QLabel("Table : {0} ".format(bookingDetails[2]))
46     self.date_label = QLabel("Date : {0} ".format(bookingDetails[4]))
47     self.time_label = QLabel("Time : {0} ".format(bookingDetails[5]))
48     self.number_people_label = QLabel("Number of people : {0}
49         ".format(bookingDetails[3]))
50     self.total_price_label = QLabel("Total Price : ")
51
52     self.total_price = QLineEdit("{0}".format(str(self.TotalPrice)))
53     self.total_price.setFixedWidth(150)
54     self.total_price.setReadOnly(True)
```

```
54
55
56 #tables
57 self.drinkQuery = """SELECT
58     Booking_Items.Quantity,
59     Items.ItemName,
60     Items.ItemPrice
61 FROM Items
62 INNER JOIN Booking_Items
63 ON Booking_Items.ItemID = Items.ItemID
64 WHERE Booking_Items.BookingID = {0}
65 AND Items.ItemTypeID = 2
66 """.format(bookingDetails[0])
67 self.drinks_ordered_table = DisplayTable()
68 self.drinks_ordered_table.show_results(self.drinkQuery)
69
70
71 self.dishQuery = """SELECT
72     Booking_Items.Quantity,
73     Items.ItemName,
74     Items.ItemPrice
75 FROM Items
76 INNER JOIN Booking_Items
77 ON Booking_Items.ItemID = Items.ItemID
78 WHERE Booking_Items.BookingID = {0}
79 AND Items.ItemTypeID = 1
80 """.format(bookingDetails[0])
81 self.dishes_ordered_table = DisplayTable()
```

```
82         self.dishes_ordered_table.show_results(self.dishQuery)
83
84
85         self.order_layout = QVBoxLayout()
86         self.order_information = QHBoxLayout()
87         self.items_ordered = QHBoxLayout()
88         self.price_layout = QHBoxLayout()
89         self.dishes_ordered = QVBoxLayout()
90         self.drinks_ordered = QVBoxLayout()
91         self.manage_order = QHBoxLayout()
92
93         #add widgets to layouts
94         self.manage_order.addWidget(self.add_button)
95         self.manage_order.addWidget(self.delete_button)
96         self.manage_order.addWidget(self.finish_button)
97         self.manage_order.addWidget(self.preview_invoice)
98         self.manage_order.addWidget(self.invoice_button)
99
100         self.dishes_ordered.addWidget(self.dishes_label)
101         self.drinks_ordered.addWidget(self.drinks_label)
102
103         self.order_information.addWidget(self.table_number_label)
104         self.order_information.addWidget(self.date_label)
105         self.order_information.addWidget(self.time_label)
106         self.order_information.addWidget(self.number_people_label)
107
108         self.drinks_ordered.addWidget(self.drinks_ordered_table)
109         self.dishes_ordered.addWidget(self.dishes_ordered_table)
```

```

110
111     self.price_layout.addWidget(self.total_price_label)
112     self.price_layout.addWidget(self.total_price)
113
114     self.items_ordered.addLayout(self.dishes_ordered)
115     self.items_ordered.addLayout(self.drinks_ordered)
116
117     ##add layouts to main order layout
118     self.booking_information_box = QGroupBox("Booking Information")
119     self.booking_information_box.setLayout(self.order_information)
120
121     self.items_ordered_box = QGroupBox("Items Ordered")
122     self.items_ordered_box.setLayout(self.items_ordered)
123
124     self.order_layout.addWidget(self.booking_information_box)
125     self.order_layout.addWidget(self.items_ordered_box)
126     self.order_layout.addLayout(self.price_layout)
127     self.order_layout.addLayout(self.manage_order)
128
129     self.setLayout(self.order_layout)
130
131     self.exec_()
132
133     def AddItem(self):
134         self.AddOrderItem = AddItemToOrder(self.bookingDetails)
135         self.AddOrderItem.orderitemAdded.connect(self.RefreshQuery)
136         self.AddOrderItem.exec_()
137

```

```

138     def DeleteItem(self):
139         self.DeleteOrderItem = DeleteItemOffOrder(self.bookingDetails)
140         self.DeleteOrderItem.orderitemDeleted.connect(self.RefreshQuery)
141         self.DeleteOrderItem.exec_()
142
143     def RefreshQuery(self):
144         self.dishes_ordered_table.show_results(self.dishQuery)
145         self.drinks_ordered_table.show_results(self.drinkQuery)
146         self.CalcTotal()
147
148     def CalcTotal(self):
149         self.TotalPrice = 0
150         self.price = []
151         self.quantity = []
152         with sqlite3.connect("restaurant.db") as db:
153             cursor = db.cursor()
154             cursor.execute("""SELECT
155                             Items.ItemPrice
156                             FROM Items
157                             INNER JOIN Booking_Items
158                             ON Booking_Items.ItemID = Items.ItemID
159                             WHERE Booking_Items.BookingID = ?
160                             """, (self.bookingDetails[0],))
161             price = cursor.fetchall()
162             for each in price:
163                 self.price.append(each[0])
164
165         with sqlite3.connect("restaurant.db") as db:

```

```

165         cursor = db.cursor()
166         cursor.execute("""SELECT
167                         Booking_Items.Quantity
168                         FROM Items
169                         INNER JOIN Booking_Items
170                         ON Booking_Items.ItemID = Items.ItemID
171                         WHERE Booking_Items.BookingID = ?
                        """, (self.bookingDetails[0],))
172
173         quantity = cursor.fetchall()
174         for each in quantity:
175             self.quantity.append(each[0])
176
177         for each in range (len(self.price)):
178             self.price[each] = self.price[each]*self.quantity[each]
179
180         for each in self.price:
181             self.TotalPrice += each
182
183
184
185
186
187     def Finish(self):
188         self.Finished = True
189         self.close()
190         return self.Finished
191

```

```
192     def Invoice(self):
193         self.Invoice = CustomerInvoice(self.bookingDetails)
194         self.Invoice.print_invoice()
195
196     def InvoicePreview(self):
197         self.Invoice = CustomerInvoice(self.bookingDetails)
198         self.Invoice.print_preview()
199
200
201
202
203
204
205
206
207 if __name__ == "__main__":
208     application = QApplication(sys.argv)
209     window = OrderWindow()
210     window.show()
211     window.raise_()
212     application.exec()
```

---

#### 4.10.12 new\_create\_tables\_cli.py

---

```
1 import sqlite3
2
3
4 def create_table(db_name, table_name, sql):
5     with sqlite3.connect(db_name) as db:
6         cursor = db.cursor()
7         cursor.execute("select name from sqlite_master where name=?", (table_name,))
8         result = cursor.fetchall()
9         keep_table = True
10        if len(result) == 1:
11            response = input("The table {0} already exists, do you wish to recreate it\n(y/n): ".format(table_name))
12            if response == 'y':
13                keep_table = False
14                print("The {0} table will be recreated - all existing data will be\nlost".format(table_name))
15                cursor.execute("drop table if exists {0}".format(table_name))
16                db.commit()
17            else:
18                print("The existing table was kept")
19        else:
20            keep_table = False
21        if not keep_table:
22            cursor.execute(sql)
23            db.commit()
24
```



```

25 def Type():
26     sql = """create table ItemType
27             (ItemTypeID integer,
28              Type text,
29              primary key(ItemTypeID))"""
30     create_table(db_name,"ItemType",sql)
31
32 def Items():
33     sql = """create table Items
34             (ItemID integer,
35              ItemName text,
36              ItemPrice real,
37              ItemTypeID integer,
38              primary key(ItemID),
39              foreign key(ItemTypeID) references ItemType(ItemTypeID))"""
40     create_table(db_name,"Items",sql)
41
42 def BookingItem():
43     sql = """create table Booking_Items
44             (BookingItemID integer,
45              BookingID integer,
46              ItemID integer,
47              Quantity integer,
48              primary key(BookingItemID),
49              foreign key(BookingID) references Bookings(BookingID) on delete cascade,
50              foreign key(ItemID) references Items(ItemID))"""
51     create_table(db_name,"Booking_Items",sql)
52

```

```
53
54 def Booking():
55     sql = """create table Bookings
56             (BookingID integer,
57              CustomerID integer,
58              TableNumber integer,
59              NumberOfPeople integer,
60              Date text,
61              Time text,
62              primary key(BookingID),
63              foreign key(CustomerID) references Customers(CustomerID),
64              foreign key(TableNumber) references Table_Numbers(TableNumber))"""
65     create_table(db_name,"Bookings",sql)
257 66
67 def Table():
68     sql = """create table Table_Numbers
69             (TableNumber integer,
70              primary key(TableNumber))"""
71     create_table(db_name,"Table_Numbers",sql)
72
73 #create a customer id for a walk in
74 def Customer():
75     sql = """create table Customers
76             (CustomerID integer,
77              FirstName text,
78              LastName text,
79              TelephoneNo integer,
80              primary key(CustomerID))"""
```

```

81     create_table(db_name,"Customers",sql)
82
83
84 if __name__ == "__main__":
85     db_name = "restaurant.db"
86     Table()
87     Type()
88     Items()
89     BookingItem()
90     Booking()
91     Customer()
92
93     data = ("Street","Customer","None")
94 with sqlite3.connect("restaurant.db") as db:
95     cursor = db.cursor()
96     sql = "insert into Customers (FirstName,LastName,TelephoneNo) values (?,?,?)"
97     cursor.execute("PRAGMA foreign_keys = ON")
98     cursor.execute(sql,data)
99     db.commit()
100
101     data = ("Dish",)
102     data2 = ("Drink",)
103
104 with sqlite3.connect("restaurant.db") as db:
105     cursor = db.cursor()
106     sql = "insert into ItemType (Type) values (?)"
107     cursor.execute(sql,data)
108     cursor.execute(sql,data2)

```

```
109         db.commit()
110
111     with sqlite3.connect("restaurant.db") as db:
112         cursor = db.cursor()
113         for each in range(1,17):
114             cursor.execute("insert into Table_Numbers (TableNumber) values (?)",
                             (each,))
```

---

#### 4.10.13 print invoice.py

```
1 import sqlite3
2 from PyQt4.QtCore import *
3 from PyQt4.QtGui import *
4 import sys
5
6 class CustomerInvoice(QDialog):
7     """This class provides a dialog box for invoices"""
8     def __init__(self, bookingDetails):
9         super().__init__()
10        self.bookingDetails = bookingDetails
11        self.bookingNo = self.bookingDetails[0] #bookingID
12        print(self.bookingNo)
13        self.TableNo = self.bookingDetails[2]
14        self.Date = self.bookingDetails[4]
15        self.Time = self.bookingDetails[5]
16
17        self.items = []
18        self.price = []
19        self.quantity = []
20
21
22        with sqlite3.connect("restaurant.db") as db:
23            cursor = db.cursor()
24            cursor.execute("""SELECT
25                            Items.ItemName
26                            FROM Items
```

```
27             INNER JOIN Booking_Items
28             ON Booking_Items.ItemID = Items.ItemID
29             WHERE Booking_Items.BookingID = ? """,(self.bookingNo,))
30 items = cursor.fetchall()
31 for each in items:
32     self.items.append(each[0])
33
34 with sqlite3.connect("restaurant.db") as db:
35     cursor = db.cursor()
36     cursor.execute("""SELECT
37                     Items.ItemPrice
38                     FROM Items
39                     INNER JOIN Booking_Items
40                     ON Booking_Items.ItemID = Items.ItemID
41                     WHERE Booking_Items.BookingID = ? """,(self.bookingNo,))
42 price = cursor.fetchall()
43 for each in price:
44     self.price.append(each[0])
45
46 with sqlite3.connect("restaurant.db") as db:
47     cursor = db.cursor()
48     cursor.execute("""SELECT
49                     Booking_Items.Quantity
50                     FROM Items
51                     INNER JOIN Booking_Items
52                     ON Booking_Items.ItemID = Items.ItemID
53                     WHERE Booking_Items.BookingID = ? """,(self.bookingNo,))
54 quantity = cursor.fetchall()
```

```
55         for each in quantity:
56             self.quantity.append(each[0])
57
58
59         for each in range (len(self.price)):
60             print(each)
61             self.price[each] = self.price[each]*self.quantity[each]
62
63
64
65     def create_html(self):
66         TotalPrice = 0
67         for each in self.price:
68             TotalPrice += each
69         html = ""
70         html += ""
71     <html>
72     <head>
73     <style>
74     table, th, td {
75
76         border: 3px solid black;
77         border-collapse: collapse;
78         width: 100%;
79     }
80         th, td
81             {
82                 padding: 10px;
```

```
83             text-align: center;
84         }
85
86
87     </style>
88 </head>
89 <body>""
90
91     html += ""<center> <h1>Linhs Chinese Restaurant</h1> </center>""
92     html += ""<p align = "right">
93         <br> 48A CARTER STREET <br>
94         FORDHAM - ELY <br>
95         CAMBRIDGESHIRE <br>
96         TEL: (01638) 721117 <br>
97         </p>
98         ""
99
100     html += ""<p>
101         <b>Date</b> &nbsp;   {0} <br>
102         <b>Time</b> &nbsp;   {1} <br> <br>
103         <b>Table Number</b> &nbsp;   {2} <br>
104         <b>Booking No.</b> &nbsp;   {3}
105         <p></p>
106         <table style = "width:100%", align="center">
107         <tr>
108             <th>Quantity</th>
109             <th>Item</th>
110             <th>Price (£)</th>
```



```

111         </tr>""".format(self.Date,self.Time,self.TableNo,self.bookingNo)
112
113     for count in range (len(self.items)):
114         html += """<tr>
115                 <td>{0}</td>
116                 <td>{1}</td>
117                 <td>{2}</td>
118             </tr>
119             """.format(self.quantity[count],self.items[count],self.price[count])
120
121         html+= """
122
123     </table>
124
125     <br>
126     <br> """
127     html += """ <center><b>    Total Price</b> : £ {0}</center>
128
129     <br>
130     <em>All meal rates are invlusive of VAT <br>
131     There is no Service Charge</em>
132 </body>
133 </html>""".format(TotalPrice)
134     return html
135
136     def print_preview(self):
137         html = self.create_html()
138         document = QTextDocument()

```

```
139         document.setHtml(html)
140         self.printer = QPrinter()
141         self.printer.setPaperSize(QSizeF(200, 220), QPrinter.Millimeter)
142         invoicePreview = QPrintPreviewDialog(self.printer, self)
143         invoicePreview.paintRequested.connect(document.print_)
144         invoicePreview.resize(1280,900)
145         invoicePreview.exec()
146
147
148     def print_invoice(self):
149         html = self.create_html()
150         self.printer = QPrinter()
151         dialog = QPrintDialog(self.printer, self)
152         if dialog.exec_():
265             document = QTextDocument()
153             document.setHtml(html)
154             document.print_(self.printer)
155
156
157 if __name__ == "__main__":
158     application = QApplication(sys.argv)
159     window = CustomerInvoice()
160     window.show()
161     window.raise_()
162     application.exec()
```

#### 4.10.14 radio\_button\_widget\_class.py

---

```
1
2 from PyQt4.QtGui import *
3
4 class RadioButtonWidget(QWidget):
5     """this class creates a group of radio buttons from a given list of labels"""
6
7     def __init__(self, label, instruction, button_list):
8         super().__init__()
9
10        self.title_label = QLabel(label)
11        self.radio_group_box = QGroupBox(instruction)
12        self.radio_button_group = QButtonGroup()
13
14        self.radio_button_list = []
15        for each in button_list:
16            self.radio_button_list.append(QRadioButton(each))
17
18        self.radio_button_list[0].setChecked(True)
19
20        self.radio_button_layout = QHBoxLayout()
21
22        counter = 1
23        for each in self.radio_button_list:
24            self.radio_button_layout.addWidget(each)
25            self.radio_button_group.addButton(each)
26            self.radio_button_group.setId(each, counter)
```

266

```
27         counter += 1
28
29     self.radio_group_box.setLayout(self.radio_button_layout)
30
31     self.main_layout = QVBoxLayout()
32     self.main_layout.addWidget(self.title_label)
33     self.main_layout.addWidget(self.radio_group_box)
34
35     self.setLayout(self.main_layout)
36
37     def selected_button(self):
38         return self.radio_button_group.checkedId()
```

---

#### 4.10.15 search\_order.py

---

```
1 import sys
2 import sqlite3
3 from PyQt4.QtCore import *
4 from PyQt4.QtGui import *
5 from table_display import *
6 from print_invoice import *
7
8 class SearchOrder(QWidget):
9     """this class will be used to search for orders using booking IDs"""
10
11     def __init__(self):
12         super().__init__()
13         booking = None
14         self.setWindowTitle("Delete Booking")
15         self.items_ordered_table = DisplayTable()
16         self.items_ordered_table.setFixedWidth(350)
17
18
19
20         self.main_layout = QVBoxLayout()
21         self.top_layout = QHBoxLayout()
22         self.input_layout = QGridLayout()
23
24
25         self.display_table = DisplayTable()
26         self.display_table.show_table("Bookings")
```

```
27     self.display_table.setFixedWidth(690)
28     self.bookingIDlabel = QLabel("Booking ID")
29     self.bookingIDlabel.setMaximumSize(100,20)
30
31     regexp = QRegExp("^\\d\\d\\d\\d?$")
32     validator = QRegExpValidator(regexp)
33     self.input_bookingID = QLineEdit()
34     self.input_bookingID.setValidator(validator)
35     self.input_bookingID.setMaximumSize(self.input_bookingID.sizeHint())
36     self.search_bookingID = QPushButton("Search Order")
37     self.search_bookingID.setMaximumSize(133,20)
38     self.search_bookingID.clicked.connect(self.search_order)
39
40     self.preview_button = QPushButton("Preview Invoice")
41     self.preview_button.setMaximumSize(133,20)
42     self.preview_button.clicked.connect(self.preview_invoice)
43
44     self.print_button = QPushButton("Print Invoice")
45     self.print_button.setMaximumSize(133,20)
46     self.print_button.clicked.connect(self.invoice_print)
47
48     self.input_layout.addWidget(self.bookingIDlabel,0,0)
49     self.input_layout.addWidget(self.input_bookingID,0,1)
50     self.input_layout.addWidget(self.search_bookingID,0,2)
51     self.input_layout.addWidget(self.preview_button,1,1)
52     self.input_layout.addWidget(self.print_button,2,1)
53
54     self.top_layout.addWidget(self.display_table)
```

```
55     self.top_layout.addLayout(self.input_layout)
56     self.main_layout.addLayout(self.top_layout)
57     self.main_layout.addWidget(self.items_ordered_table)
58     self.setLayout(self.main_layout)
59
60
61     def search_order(self):
62         self.booking = self.input_bookingID.text()
63         self.searchQuery = """SELECT
64                               Booking_Items.Quantity,
65                               Items.ItemName,
66                               Items.ItemPrice
67                             FROM Items
68                             INNER JOIN Booking_Items
69                             ON Booking_Items.ItemID = Items.ItemID
70                             WHERE Booking_Items.BookingID = {0}
71                               """.format(self.booking)
72         self.items_ordered_table.show_results(self.searchQuery)
73
74
75
76
77     def preview_invoice(self):
78
79         with sqlite3.connect("restaurant.db") as db:
80             cursor = db.cursor()
81             cursor.execute("select * from Bookings where BookingID = {0}
82                            ".format(self.booking))
```

```

82         bookingDetails = cursor.fetchone()
83
84         self.invoice = CustomerInvoice(bookingDetails)
85         self.invoice.print_preview()
86
87     def invoice_print(self):
88
89         with sqlite3.connect("restaurant.db") as db:
90             cursor = db.cursor()
91             cursor.execute("select * from Bookings where BookingID = {0}"
92                             ".format(self.booking)")
93             bookingDetails = cursor.fetchone()
94
95         self.invoice = CustomerInvoice(bookingDetails)
96         self.invoice.print_invoice()
97
98
99
100
101
102 if __name__ == "__main__":
103     application = QApplication(sys.argv)
104     window = DeleteBookingWindow()
105     window.show()
106     window.raise_()
107     application.exec()

```



#### 4.10.16 table\_display.py

---

```
1 import sys
2 import sqlite3
3
4 from PyQt4.QtSql import *
5 from PyQt4.QtCore import *
6 from PyQt4.QtGui import *
7
8 class DisplayTable(QWidget):
9     """this class will be used to display tables from the database"""
10
11     def __init__(self):
12         super().__init__()
13         self.displaySQLtable = QVBoxLayout()
14         self.setLayout(self.displaySQLtable)
15         self.db = None
16         self.model = None
17         self.open_database()
18
19
20     def display_results_layout(self):
21         self.results_table = QTableView()
22         self.results_layout = QVBoxLayout()
23         self.results_layout.addWidget(self.results_table)
24         self.results_widget = QWidget()
25         self.results_widget.setLayout(self.results_layout)
26         self.displaySQLtable.addWidget(self.results_widget)
```

```
27
28     def open_database(self):
29         if self.db:
30             self.close_database()
31         self.db = QSqlDatabase.addDatabase("QSQLITE")
32         self.db.setDatabaseName("restaurant.db")
33         opened_ok = self.db.open()
34         return opened_ok
35
36     def show_results(self, query):
37         self.display_results_layout()
38         if not self.model or not isinstance(self.model, QSqlQueryModel):
39             self.model = QSqlQueryModel()
40         self.model.setQuery(query)
41         self.results_table.setModel(self.model)
42         self.results_table.show()
43
44     def show_table(self, tableName):
45         self.display_results_layout()
46         if not self.model or not isinstance(self.model, QSqlTableModel):
47             self.model = QSqlTableModel()
48         self.model.setTable(tableName)
49         self.model.select()
50         self.results_table.setModel(self.model)
51         self.results_table.show()
52
53     def refresh(self):
54
```

```
55         self.results_table.setModel(self.model)
56         self.model.select()
57
58
59
60     if __name__ == "__main__":
61         application = QApplication(sys.argv)
62         window = DisplayTable()
63         window.show()
64         window.raise_()
65         application.exec()
```

---

#### 4.10.17 update\_booking.py

---

```
1 import sys
2 import sqlite3
3 from PyQt4.QtCore import *
4 from PyQt4.QtGui import *
5 from table_display import *
6 import time
7
8 class UpdateBooking(QWidget):
9     bookingAdded = pyqtSignal()
10     """this class will be used to update bookings"""
11
12     def __init__(self):
13         super().__init__()
14
15         self.main_layout = QVBoxLayout()
16         self.update_booking_layout = QGridLayout()
17
18         self.display_table = DisplayTable()
19         self.display_table.show_table("Bookings")
20
21         self.date_button = QPushButton("Update Date")
22         self.time_button = QPushButton("Update Time")
23         self.people_button = QPushButton("Update Number Of People")
24         self.table_button = QPushButton("Update Table")
25         self.telenumber_button = QPushButton("Update TeleNumber")
26
```

```
27     self.bookingID_label = QLabel("Booking ID you want to update: ")
28     self.date_label = QLabel("Date:")
29     self.time_label = QLabel("Time:")
30     self.number_of_people = QLabel("Number Of People:")
31     self.telephone_number = QLabel("Telephone Number:")
32     self.table_number_label = QLabel("Table Number:")
33
34     regexp = QRegExp("^\\d\\d\\d\\d")
35     validator = QRegExpValidator(regexp)
36     self.input_bookingID = QLineEdit()
37     self.input_bookingID.setValidator(validator)
38     self.input_bookingID.setMaximumSize(300,30)
39
40     self.input_number_of_people = QLineEdit()
41     regexp2 = QRegExp("^\\d\\d")
42     validator2 = QRegExpValidator(regexp2)
43     self.input_number_of_people.setValidator(validator2)
44     self.input_number_of_people.setMaximumSize(300,30)
45     self.input_number_of_people.setMaxLength(2)
46     self.input_number_of_people.setPlaceholderText("Expected number")
47
48
49     self.select_table_number = QComboBox(self)
50     self.select_table_number.setMaximumSize(300,30)
51     for each in range(1,17):
52         self.select_table_number.addItem(str(each))
53
54     #dates and times
```

```
55     self.date_edit = QDateEdit()
56     self.maximumdate = QDate(2050,1,30)
57     self.minimumdate = QDate(2015,1,1)
58     self.date_edit.setMaximumDate(self.maximumdate)
59     self.date_edit.setMinimumDate(self.minimumdate)
60     self.date_edit.setMaximumSize(300,30)
61     self.time_edit = QTimeEdit()
62     self.time_edit.setDisplayFormat("hh:mm")
63     self.time_edit.setMaximumSize(300,30)
64
65
66     self.update_booking_layout.addWidget(self.bookingID_label,0,0)
67     self.update_booking_layout.addWidget(self.date_label,1,0)
68     self.update_booking_layout.addWidget(self.time_label,2,0)
69     self.update_booking_layout.addWidget(self.number_of_people,3,0)
70     self.update_booking_layout.addWidget(self.table_number_label,4,0)
71
72     self.update_booking_layout.addWidget(self.input_bookingID,0,1)
73     self.update_booking_layout.addWidget(self.date_edit,1,1)
74     self.update_booking_layout.addWidget(self.time_edit,2,1)
75     self.update_booking_layout.addWidget(self.input_number_of_people,3,1)
76     self.update_booking_layout.addWidget(self.select_table_number,4,1)
77
78     self.update_booking_layout.addWidget(self.date_button,1,2)
79     self.update_booking_layout.addWidget(self.time_button,2,2)
80     self.update_booking_layout.addWidget(self.people_button,3,2)
81     self.update_booking_layout.addWidget(self.table_button,4,2)
82
```

```

83         #add layouts to main layout
84         self.main_layout.addWidget(self.display_table)
85         self.main_layout.addLayout(self.update_booking_layout)
86
87
88         #create a widget to display main layout
89         self.setLayout(self.main_layout)
90
91
92
93         #connections
94         self.date_button.clicked.connect(self.update_date)
95         self.time_button.clicked.connect(self.update_time)
96         self.people_button.clicked.connect(self.update_peopleNo)
97         self.table_button.clicked.connect(self.update_tableNo)
98
99
100     def update_date(self):
101         bookingID = self.input_bookingID.text()
102         BookingDate = self.date_edit.text()
103         UpdateDate = (BookingDate, bookingID)
104
105         with sqlite3.connect("restaurant.db") as db:
106             cursor = db.cursor()
107             sql = "update Bookings set Date=? where BookingID=?"
108             cursor.execute("PRAGMA foreign_keys = ON")
109             cursor.execute(sql, UpdateDate)
110             db.commit()

```

```

111
112         self.display_table.refresh()
113
114     def update_time(self):
115         bookingID = self.input_bookingID.text()
116         BookingTime = self.time_edit.text()
117         UpdateTime = (BookingTime, bookingID)
118
119         with sqlite3.connect("restaurant.db") as db:
120             cursor = db.cursor()
121             sql = "update Bookings set Time=? where BookingID=?"
122             cursor.execute("PRAGMA foreign_keys = ON")
123             cursor.execute(sql, UpdateTime)
124             db.commit()
279
125
126         self.display_table.refresh()
127
128
129     def update_peopleNo(self):
130         bookingID = self.input_bookingID.text()
131         try:
132             NumberOfPeople = int(self.input_number_of_people.text())
133             if len(str(NumberOfPeople)) > 0 and (NumberOfPeople > 0):
134                 UpdatePeople = (NumberOfPeople, bookingID)
135                 with sqlite3.connect("restaurant.db") as db:
136                     cursor = db.cursor()
137                     sql = "update Bookings set NumberOfPeople=? where BookingID=?"
138                     cursor.execute("PRAGMA foreign_keys = ON")

```



```

139         cursor.execute(sql, UpdatePeople)
140         db.commit()
141
142         self.display_table.refresh()
143
144     except ValueError or UnboundLocalError:
145         QMessageBox.about(self, "Error", "Please enter a suitable value")
146
147
148     def update_tableNo(self):
149         bookingID = self.input_bookingID.text()
150         TableNumber = self.select_table_number.currentIndex() + 1
151         UpdateTableNo = (TableNumber, bookingID)
152
153         with sqlite3.connect("restaurant.db") as db:
154             cursor = db.cursor()
155             sql = "update Bookings set TableNumber=? where BookingID=?"
156             cursor.execute("PRAGMA foreign_keys = ON")
157             cursor.execute(sql, UpdateTableNo)
158             db.commit()
159
160         self.display_table.refresh()
161
162
163
164
165
166 if __name__ == "__main__":

```

```
167     application = QApplication(sys.argv)
168     window = UpdateBooking()
169     window.show()
170     window.raise_()
171     application.exec()
```

---

#### 4.10.18 update\_item\_price.py

---

```
1 import sys
2 import sqlite3
3 from PyQt4.QtCore import *
4 from PyQt4.QtGui import *
5 from table_display import *
6
7 class UpdateItemPrice(QDialog):
8     """this class creates a widget to update prices of items on the menu"""
9
10    def __init__(self):
11        super().__init__()
12        self.main_layout = QVBoxLayout()
13        self.update_item_layout = QGridLayout()
14        self.update_complete_layout = QHBoxLayout()
15
16        self.display_table = DisplayTable()
17        self.display_table.show_table("Items")
18
19        self.update_complete = QPushButton("Update Item")
20
21        self.itemID_label = QLabel("Item ID : ")
22        self.item_price_label = QLabel("New Item Price : ")
23
24
25        regexp = QRegExp("^\\d\\d\\d\\d?$")
26        validator = QRegExpValidator(regexp)
```

282

```

27     self.input_itemID = QLineEdit()
28     self.input_itemID.setValidator(validator)
29     self.input_itemID.setMaximumSize(300,30)
30
31     regexp2 = QRegExp("(^\\d|\\d\\d)(\\.\\d\\d)?$")
32     validator2 = QRegExpValidator(regexp2)
33     self.input_item_price = QLineEdit()
34     self.input_item_price.setValidator(validator2)
35     self.input_item_price.setMaximumSize(300,30)
36
37
38     self.update_item_layout.addWidget(self.itemID_label,0,0)
39     self.update_item_layout.addWidget(self.item_price_label,1,0)
40     self.update_item_layout.addWidget(self.input_itemID,0,1)
41     self.update_item_layout.addWidget(self.input_item_price,1,1)
42     self.update_complete_layout.addWidget(self.update_complete)
43
44     self.main_layout.addWidget(self.display_table)
45     self.main_layout.addLayout(self.update_item_layout)
46     self.main_layout.addLayout(self.update_complete_layout)
47
48     self.setLayout(self.main_layout)
49
50     #connection
51     self.update_complete.clicked.connect(self.update_item)
52
53     def update_item(self):
54         ItemID = self.input_itemID.text()

```

```
55     ItemPrice = self.input_item_price.text()
56     UpdateItem = (ItemPrice,ItemID)
57     print(UpdateItem)
58     if len(ItemID)>0 and (len(ItemPrice)>0):
59
60         with sqlite3.connect("restaurant.db") as db:
61             cursor = db.cursor()
62             sql = "update Items set ItemPrice=? where ItemID=?"
63             cursor.execute("PRAGMA foreign_keys = ON")
64             cursor.execute(sql,UpdateItem)
65             db.commit()
66
67         self.display_table.refresh()
68     else:
69         QMessageBox.about(self,"Error","Please fill in the required fields")
70
71
72
73 if __name__ == "__main__":
74     application = QApplication(sys.argv)
75     window = UpdateItemPrice()
76     window.show()
77     window.raise_()
78     application.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

#### 5.3.4 Saving

#### 5.3.5 Limitations

### 5.4 Error Recovery

#### 5.4.1 Error 1





## 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