

Computing Coursework

Toby Kerslake

January 26, 2016

Contents

1 Analysis	5
1.1 Introduction	5
1.1.1 Client Identification	5
1.1.2 Define the current system	6
1.1.3 Describe the problems	6
1.1.4 Section appendix	7
1.2 Investigation	10
1.3 Investigation	11
1.3.1 The current system	11
1.3.2 The proposed system	24
1.4 Objectives	32
1.4.1 General Objectives	32
1.4.2 Specific Objectives	32
1.4.3 Core Objectives	33
1.4.4 Other Objectives	34
1.5 ER Diagrams and Descriptions	35
1.5.1 ER Diagram	35
1.5.2 Entity Descriptions	35
1.6 Object Analysis	36
1.6.1 Object Listing	36
1.6.2 Relationship diagrams	36
1.6.3 Class definitions	37
1.7 Other Abstractions and Graphs	38
1.8 Constraints	38
1.8.1 Hardware	38
1.8.2 Software	39
1.8.3 Time	39
1.8.4 User Knowledge	39
1.8.5 Access restrictions	39
1.9 Limitations	40
1.9.1 Areas which will not be included in computerisation	40
1.9.2 Areas considered for future computerisation	40
1.10 Solutions	41

1.10.1 Alternative solutions	41
1.10.2 Justification of chosen solution	44
2 Design	46
2.1 Overall System Design	46
2.1.1 Short description of the main parts of the system	46
2.1.2 System flowcharts showing an overview of the complete system	50
2.2 User Interface Designs	59
2.3 Hardware Specification	67
2.4 Program Structure	68
2.4.1 Top-down design structure charts	68
2.4.2 Algorithms in pseudo-code for each data transformation process	73
2.4.3 Object Diagrams	74
2.4.4 Class Definitions	75
2.5 Prototyping	75
2.6 Definition of Data Requirements	77
2.6.1 Identification of all data input items	77
2.6.2 Identification of all data output items	78
2.6.3 Explanation of how data output items are generated	79
2.6.4 Data Dictionary	81
2.6.5 Identification of appropriate storage media	83
2.7 Database Design	84
2.7.1 Normalisation	84
2.7.2 SQL Queries	89
2.8 Security and Integrity of the System and Data	92
2.8.1 Security and Integrity of Data	92
2.8.2 System Security	92
2.9 Validation	93
2.10 Testing	95
3 Testing	118
3.1 Test Plan	118
3.1.1 Original Outline Plan	119
3.1.2 Changes to Outline Plan	119
3.1.3 Original Detailed Plan	119
3.1.4 Changes to Detailed Plan	119
3.2 Test Data	120
3.2.1 Original Test Data	120
3.2.2 Changes to Test Data	120
3.3 Annotated Samples	120
3.3.1 Actual Results	120
3.3.2 Evidence	120
3.4 Evaluation	121
3.4.1 Approach to Testing	121

3.4.2	Problems Encountered	121
3.4.3	Strengths of Testing	121
3.4.4	Weaknesses of Testing	121
3.4.5	Reliability of Application	121
3.4.6	Robustness of Application	121
4	System Maintenance	122
4.1	Environment	123
4.1.1	Software	123
4.1.2	Usage Explanation	123
4.1.3	Features Used	123
4.2	System Overview	123
4.2.1	System Component	123
4.3	Code Structure	123
4.3.1	Particular Code Section	123
4.4	Variable Listing	123
4.5	System Evidence	123
4.5.1	User Interface	123
4.5.2	ER Diagram	123
4.5.3	Database Table Views	123
4.5.4	Database SQL	123
4.5.5	SQL Queries	123
4.6	Testing	123
4.6.1	Summary of Results	123
4.6.2	Known Issues	123
4.7	Code Explanations	123
4.7.1	Difficult Sections	123
4.7.2	Self-created Algorithms	123
4.8	Settings	123
4.9	Acknowledgements	123
4.10	Code Listing	123
4.10.1	Module 1	124
5	User Manual	125
5.1	Introduction	126
5.2	Installation	126
5.2.1	Prerequisite Installation	126
5.2.2	System Installation	126
5.2.3	Running the System	126
5.3	Tutorial	126
5.3.1	Introduction	126
5.3.2	Assumptions	126
5.3.3	Tutorial Questions	126
5.3.4	Saving	126
5.3.5	Limitations	126
5.4	Error Recovery	126

5.4.1	Error 1	126
5.4.2	Error 2	126
5.5	System Recovery	126
5.5.1	Backing-up Data	126
5.5.2	Restoring Data	126
6	Evaluation	127
6.1	Customer Requirements	128
6.1.1	Objective Evaluation	128
6.2	Effectiveness	128
6.2.1	Objective Evaluation	128
6.3	Learnability	128
6.4	Usability	128
6.5	Maintainability	128
6.6	Suggestions for Improvement	128
6.7	End User Evidence	128
6.7.1	Questionnaires	128
6.7.2	Graphs	128
6.7.3	Written Statements	128

Chapter 1

Analysis

1.1 Introduction

Client: Neil Green Job Description: Owns and manages an Independently run gym Contact Information: Sweat Gymnasium LLP 107a Clay Street Soham Ely Cambridgeshire CB7 5HL

Telephone: 01353 721864 Email: info@sweatgym.co.uk

1.1.1 Client Identification

My client is an independent gym owner who is struggling to keep track of his members payment details and use of the facilities. He is also my brother in law. He approached me about creating a system for him after I mentioned I needed a client for coursework and he mentioned that he is having issues with clients not paying him since he doesn't have an accurate, reliable system for keeping track of payments. He also has to manage staff members throughout the day and has to manage his business' accounts, which relies a lot on his current system for keeping track of his members payments. His gym has over a 1000 members which he keeps track of using excel spreadsheets. He made these spreadsheets himself so he has a reputable amount of I.T skills (I go more in depth regarding his and his employees I.T abilities in the Constraints section). I have accumulated the information for the following sections via an interview (presented in full in the appendix) and text messages/email and general conversation as I see him and several of his employees quite frequently but don't always have the chance to document the interaction.

1.1.2 Define the current system

His current system involves using a series of excel spreadsheets to document his clients payment, membership details, personal details and exercise plans. He does this manually by creating new spreadsheets and entering the clients data by hand. His clients give him this data by filling out a membership form and membership agreement and via discussion and another entry form if they need an exercise program made for them. My client uses one spreadsheet (presented in section: Investigation) to keep a record of all the members of the gym including details like their member ID, their contact information and membership details, and another for their payment information (presented in section: Investigation) as well as whether their up to date with these payments. Each member then has their own table which is used to keep track of any exercise routines they may have asked to have planned for them. When the applying member signs up they get given information regarding the gym including payments details so that the can pay via direct debit. Every month the owner checks his mandates to see who has and hasn't made their payment. When a member of the gym leaves all of their information is kept in case they return to their membership in the future. All of the exercise regimes are dated so that it can be referred back to if the member decides they want to change their regime. He then scans these sheets and saves them digitally and uses another physical sheet to document the new regime. A lot of the time he will immediately scan the sheet as members normally request a copy, plus he finds it useful to have a backup especially when dealing with physical forms. All of these spreadsheets and regime forms are kept on the owners workstation computer at the reception desk. Usually he will send the not print the spreadsheets unless he outsources his accounts to another employee or an accountant and he isn't comfortable supplying them with a full digital, or they just find it easier to deal with a physical version. He doesn't keep any of his clients debit/credit card information as all transactions are performed by the members through either direct debit, paying by card at the gym or cash.

1.1.3 Describe the problems

This system imposes some limitations. First of all, all the processes have to be dealt with manually meaning that all the information has to be entered into excel spreadsheets and not in an easier manner like through a digital form which would organise the data efficiently and automatically instead of the staff having to do it themselves. These spreadsheets are used to log any payments made by a client, whether its cash/card or by direct debit, as well as all the client information that needs to be stored. A password might need to be included in the new system as the current system also isn't secure as any piece of information can be changed at any time by anyone who has managed to access the staff workstation and to be sure of everything debit mandates would have to be looked over, but even then some clients may have paid by cash and thus wouldn't be recoverable. Their

is also the possibility of a computer error that could delete all the documents as they are not backed up anywhere. His gym also now accommodates over 1000 members meaning that he has a large number of spreadsheets for all his clients as well as having his main spreadsheet being exceptionally large meaning the program can lag, especially since the program isn't running on the greatest available hardware. Another issue is that it can be difficult to sort through and search the spreadsheets to find specific information regarding clients, for example if he needs to find they're contact information, excel doesn't have a tool to easily search for that. He also has no easy way to create invoices or just clean organised print outs of any information he needs.

1.1.4 Section appendix

Q. What is your current system?

Our current system involves entering data regarding our clients into excel spreadsheets to keep track of the data we need

Q. And what data is it that you record?

We use the first spreadsheet to record our clients names, contact information, and what membership they have, as well as another for payment details. Each member then has their own table for us to detail any exercise plans that they request for us to create.

Q. What are the limitations of this? This system means that everything has to be done manually and I won't know if I've forgotten to enter whether someone's paid or not as there is no way for me to know if I forgot to change part of the spreadsheet or if my client is trying not to pay. It also isn't very secure as anyone could edit it easily. The process can also be very time consuming.

Q. Is there any additional data you would like to be recorded in the new system? I would like to be able to store all of the clients information in one place instead of having to use multiple spreadsheets so I can organise all my data more easily. Though I'm pretty sure I don't need any additional information.

Q. What processes do you normally go through with your current system and how do you implement it? To use my current system I ask my clients to fill out two forms (presented in the investigation section) and then take the information they filled out and enter them digitally into the appropriate excel spreadsheets.

Q. How would you like to operate the new system? I would like to be able to operate the system similar to how I operate my current one with me being able to enter information from forms my clients filled out, or perhaps even with digital forms I can email them to fill out.

Q. What inputs and outputs would your new system need? I would like to be able to print out a clients information easily in an organised fashion and also be able to create print outs of any other information I would like, possibly to create

an invoice or general report for the client. I would also need some physical forms if possible for my less technically inclined members.

Q. What hardware would this need to run on? This would be run off a reasonably mid spec laptop purchased four years ago. All I really know is it has an i5 processor.

Questions For Client

Q. What is your current system?

OUR CURRENT SYSTEM INVOLVES ENTERING DATA REGARDING OUR CLIENTS INTO EXCEL SPREADSHEETS TO KEEP TRACK OF THE DATA WE NEED.

Q. And what data is it that you record?

WE USE THE SAME SPREADSHEET TO RECORD OUR CLIENTS NAMES, CONTACT INFORMATION, PAYMENT DETAILS, AND THE TYPE OF MEMBERSHIP THEY HAVE. EACH MEMBER HAS THEIR OWN SPREADSHEET FOR US TO DETAIL ANY EXERCISE PLANS THAT THEY REQUEST FOR US TO CREATE.

Q. What are the limitations of this?

THIS SYSTEM MEANS THAT EVERYTHING HAS TO BE DONE MANUALLY AND I WON'T KNOW IF I'VE FORGOTTEN TO ENTER WHETHER SOMEONE PAID OR NOT AS THERE IS NO WAY FOR ME TO KNOW IF I FORGOT TO CHANGE PART OF THE SPREADSHEET OR IF MY CLIENT IS TRYING NOT TO PAY. IT ALSO ISN'T VERY SECURE AS ANYONE COULD EDIT IT EASILY. THIS PROCESS CAN ALSO BE VERY TIME CONSUMING.

Q. Is there any additional data you would like to be recorded in the new system?

I WOULD LIKE TO BE ABLE TO STORE ALL OF THE CLIENTS INFORMATION IN ONE PLACE INSTEAD OF HAVING TO USE MULTIPLE SPREADSHEETS SO I CAN ORGANISE ALL MY DATA MORE EASILY. THOUGH I'M PRETTY SURE I DON'T NEED ANY ADDITIONAL INFORMATION.

1.jpg

Figure 1.1: 1st page of the questions asked to my client

1.2 Investigation

Q. What processes do you normally go through with your current system and how do you implement it?

TO USE MY CURRENT SYSTEM I ASK MY CLIENT TO
FILL OUT 2 FORMS AND THEN TAKE THE INFORMATION THEY
FILLED OUT AND ENTER THEM DIGITALLY INTO THE APPROPRIATE
EXCEL SPREADSHEETS

Q. How would you like to operate the new system?

I WOULD LIKE TO BE ABLE TO OPERATE THE SYSTEM SIMILAR
TO HOW I OPERATE THE CURRENT ONE WITH ME BEING ABLE TO
ENTER INFORMATION FROM FORMS MY CLIENTS FILLED OUT OR PERHAPS
EVEN DIGITAL FORMS I CAN EMAIL THEM TO FILL OUT.

Q. What inputs and outputs would your new system need?

I WOULD LIKE TO BE ABLE TO PRINT OUT A CLIENT'S INFORMATION
EASILY IN AN ORGANISED FASHION AND ALSO BE ABLE TO CREATE
PRINT OUTS OF ANY OTHER INFORMATION I WOULD LIKE, POSSIBLY
TO CREATE AN INVOICE OR GENERAL REPORT FOR THE CLIENT. I
WOULD ALSO NEED TO PRODUCE SOME PHYSICAL FORMS IF POSSIBLE
FOR MY LESS TECHNICALLY INCLINED MEMBERS.

Q. What hardware would this need to run on?
THIS WOULD NEED TO RUN ON A REASONABLY MID SPEC
LAPTOP PURCHASED 4 YEARS AGO. ALL I REALLY KNOW IS IT
HAS AN i5 PROCESSOR.

Signed



Toby Kerslake

Signed



Neil Green

2.jpg

Figure 1.2: 2nd page of the questions asked to my client

1.3 Investigation

1.3.1 The current system

The current system is made of completely manual functions only using computers for data entry and storing the spreadsheets in which the data is entered. I do my best to explain this system below through the use of hierarchy charts, flowcharts, data flow diagrams and the all the physical and digital forms used in the process.

Data sources and destinations

The table below presents all the data currently used in the system, where its come from, where its being stored and gives an example of said data.

Source	Data	Example Data	Destination
Applying Member	Name, Address, Telephone Number, Membership Type, Registration Date, Payment Type, Join Date, Initial Payment Method, How Much	Toby Kerslake, 123 Fakestreet, 01353 666677, Couples, 5/11/2014, Direct Debit, 10/11/2014,	Member Spreadsheet
Applying Member	Initial Payment Type, Registration Date, How Much	Cash, 5/11/2014, £30	Payment Spreadsheet
Member of Staff	Member Number,	100301	Payment Spreadsheet
Member of Staff	A new month is added to the spreadsheet every month and the member of staff enters whether it's been paid for by each member	June '12 Paid	Payment Spreadsheet
Member Interviewed by Staff (and filling out a form)	Exercise Regime	10 Pushups 15kg weights - 20 minutes	Member Programme Card

Figure 1.3: Data Sources and Destinations for the current system

Algorithms

Structure Tables

Gym Sign Up Process

- 1 1. Information **is** attained **from** the client
 - 2 1.1 My client gives a form to their client to be promptly filled out
 - 3 1.2 The client then hands the sheet back into reception
 - 4 1.3 The information provided on this sheets then enter manually into an excel spreadsheet
 - 5 1.4 Do they have an exercise plan? Then the plan **is** created based on an interview **with** the client **while** they fill out a form
 - 6 2. Create Membership Card
 - 7 2.1 Make sure **all** necessary details are correct
 - 8 2.2 **print** Membership card
-

Payment Process

- 1 1. Client enters
 - 2 2. Are they paying upfront?
 - 3 2.1. Process their payment
 - 4 2.2. Enter the appropriate information into the correct spreadsheet
 - 5 2.3 Allow them into the facility
 - 6 3. Have they payed by direct debit?
 - 7 3.1. Check to see **if** the payment went through
 - 8 3.2. If it did allow them into the facility
 - 9 3.3 If the payment didnt go through then ask them to pay upfront **and return** to section 2
 - 10 3.4 If they refuse to pay ask them to leave
-

Pseudocode

Gym Sign Up Process

```
1
2 START
3
4 Function GetInfo:
5     Client fills out form
6     Hands form into gym owner
7     Owner enters the information into an excel
        spreadsheet
8     IF they have an exercise plan:
```

```
9           Owner creates a plan based on an
          interview with client and new form
          and enters this into its
10      own spreadsheet
11          Owner creates Membership Card
12          Owner checks to make sure all the info is
              correct
13          Owner prints membership card and gives it to
              the new member
14
15  GetInfo
16
17  END
```

Payment Process

```
1  Start
2
3  Function ClientPayment:
4      Client enters
5      IF paying upfront:
6          PayingUpFront
7      IF paying by direct debit:
8          PayingDirectDebit
9      IF they refuse to pay:
10         AskThemToLeave
11
12  Function PayingUpFront:
13      Owner processes payment
14      Process their payment
15      Enter payment information into the
          appropriate spreadsheet
16      Allow them into the facility
17
18  Function PayingDirectDebit:
19      Check to see if the payment went through
20      IF payment didn't go through:
21          IF they want to pay up front:
22              PayingUpFront
23          ELSE IF they refuse to pay:
24              AskThemToLeave
25      ELSE IF the payment did go through:
26          Allow them into the facility
27
28  Function AskThemToLeave:
29      Ask them to leave
```

30
31 ClientPayment
32
33 END

Flow Charts

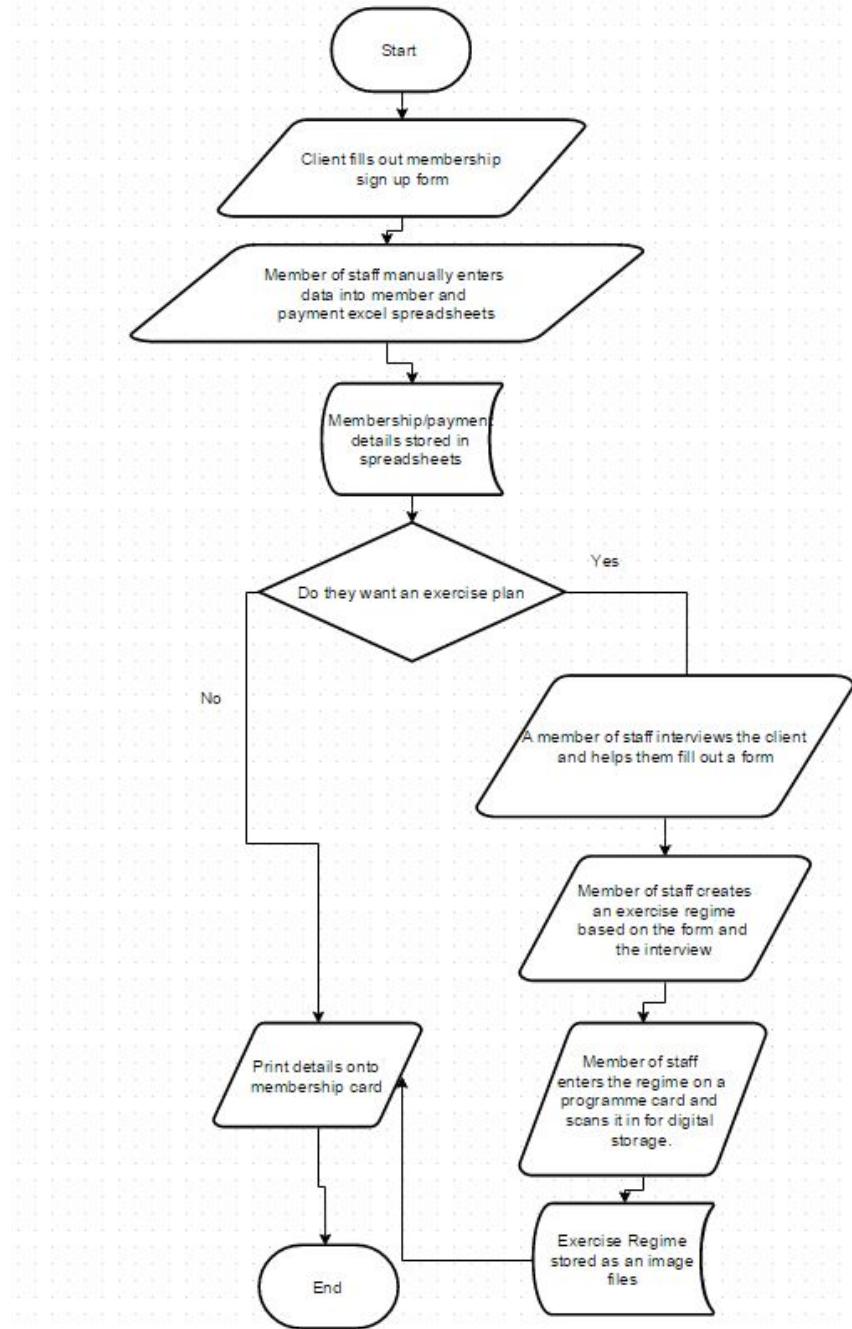


Figure 1.4: Algorithm for the gym sign up process. The form details the process of an applying member giving a staff member their details, the staff member entering them into the spreadsheets and then going through all the extra processes that are required if the applying member wants an exercise regime.

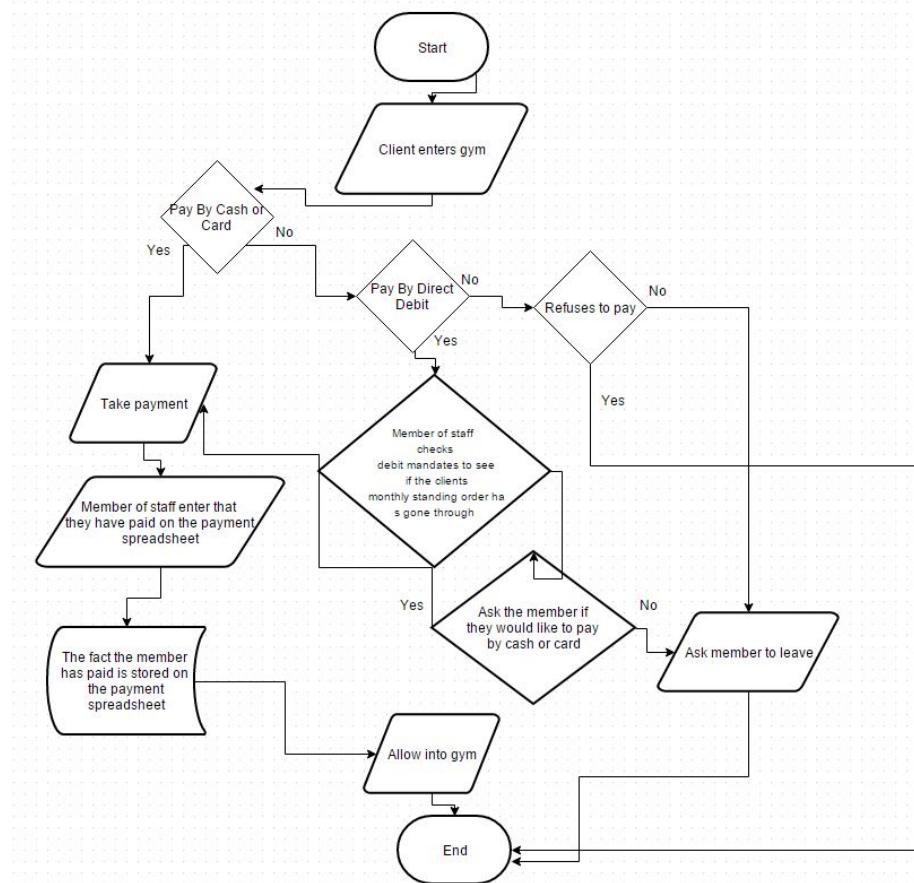


Figure 1.5: Algorithm for the payment process. This details the process the member and staff members go through each month when the member pays either through cash/card or a monthly standing order, and how they deal with people who refuse to pay.

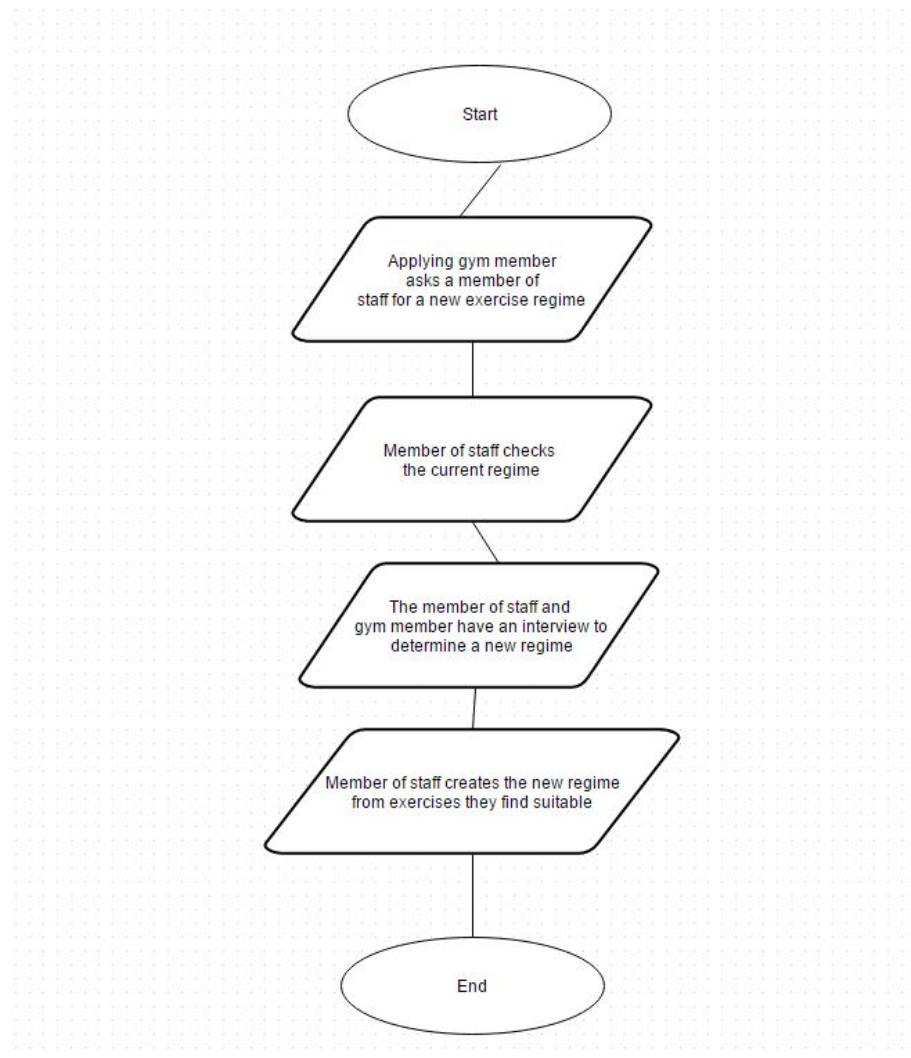


Figure 1.6: Algorithm for the process of creating a new exercise regime. This details how the gym member interviews with a staff member to create a new regime based on exercises they deem appropriate.

Data flow diagram

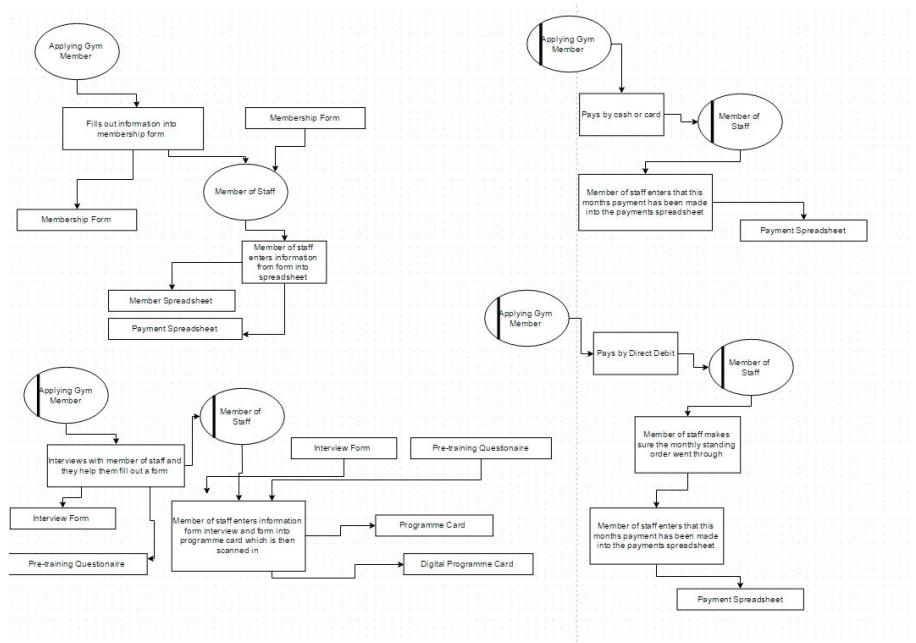


Figure 1.7: Data Flow Diagram For The Current System. These diagrams show where all the data in the current system comes from, where it goes and the process it goes through to get there.

Toby Kerslake

Candidate No. 44108

Centre No. 22151

Input Forms, Output Forms, Report Formats

Membership Form

Name:

Address:

Telephone Number:

Membership Number:

Type of Membership:

(Please note: Couples Membership only applies if both members of the couple are registered at the same address)

Join date:

Membership fee paid: £20 £35 Other (please circle)

How paying: Monthly Standing order (25th) Pay as you go (please circle)

Frequency of payments: Monthly on the 25th of every month

Six Monthly

Annually

Pay-As-You-Go

Figure 1.8: This is the Membership Sign up form. This is printed and handed to applying members upon sign up and can either be filled out at the premises or filled out elsewhere and handed back in to reception. It is used to gather basic information for the membership and payment details spreadsheets.

**Standing Order Form**

(To be completed and handed in at Customers Bank)

Sort Code: Account Number:

Name:.....

Address:.....
.....

Membership Number:.....

First Payment Date:

Frequency: Monthly on 25th of every month

Amount: £.....

Reference:

(Name and Reference Number – Please ensure that this reference is given to your bank when setting up the standing order, otherwise we may not be able to track your payment.)

Customer Signature:

To be paid to:**Sweat Gymnasium LLP****HSBC****Sort Code: 40-20-38****5 Butter Market****Ely****Account Number: 91342703****Cambridgeshire CB7 4PA**

107A Clay Street, Soham, Ely, Cambridgeshire, CB7 5HL

T: 01353 721 864 E: info@sweatgym.co.uk

www.sweatgym.co.uk

Company Registration No: OC370760

VAT Registration No: GB 12R 7802 95

Figure 1.9: This is the Standing Order Form. This is printed and handed to the user along with the membership form and is for the member to fill in and hand to their bank so they make the correct payment. Some of this form can be filled in with assistance from a member of staff. For example their membership number.

Pre-Training Questionnaire

The Pre-Training Questionnaire is designed to help us assess your exercise and medical history, so that our Instructors are able to recommend the best and safest way for you train. The Pre-Training Questionnaire must be completed during Induction, so that we can assess your abilities fully and reduce the risk of injury and ensure that you get the best out of your work out.

1. Have you previously attended another Gymnasium? (If so where?).....
.....
2. Whilst attending a previous Gymnasium, did you follow an individual exercise programmes designed specifically for yourself by their Instructors?.....
.....
3. How long since you have trained regularly? How often did you train?
4. Have you ever used Free-Weights before? If so, up to what weight?
5. Which of Sweat Gymnasium's facilities do you hope to be using?
 - Cardiovascular Exercise Room
 - Free-Weights Room
 - Fitness Classes
6. Do you feel that you need any further guidance or instruction before using our exercise equipment or free-weights?
7. Do you have any previous medical history which could affect your ability to use any of the facilities at Sweat Gymnasium LLP, which we should be aware of? (If so, please give details.)

Figure 1.10: Regime Interview Part 1

.....

I agree that the information above which I have provided is accurate to the best of my knowledge. I am fully aware that I may be required to supply further information regarding any of the above or my medical history, before being able to use the facilities of Sweat Gymnasium LLP. I understand that refusal to follow the Instructor's guidance when using the facilities at Sweat Gymnasium LLP can result in the inability to train at Sweat Gymnasium LLP.

Printed Name:

Signature:

Date:

Instructors' recommendation: (please circle)

Full Induction	Refresher Induction	Personalised Fitness Programme
GP Referral	No Further Instruction Necessary	

Instructors Additional Notes:

.....

.....

Instructors Signature: Date:

Figure 1.11: Regime Interview Part 2. The regime interview/Pre-Training Questionnaire is filled out by the applying gym member and a member of staff to determine what the members regime should appropriately consist of. This information is compiled into different exercises and routines stored on the members programme card shown in the next figure. Afterwards these sheets are disposed of.

Name Membership No..... Programme Card

Figure 1.12: The Programme Card is where a Gym members personal regime is written down and organised, with 2 columns to determine the number of reps/weights involved with the exercise. The other columns that contain slashes have no purpose and are just there to fill space. This shows the forms poor design and thus it will probably have to be redesigned for use with the new system.

Membership No.	Surname	First Name		Address	Tel No.
100001	Fakeman	Tobias	P	4 Fake Street, Soham, Ely	01353 624031/07846200759
100002	Notreal	George	C	1 Fake Road, Soham, Ely, Cambs CB7 5EP	7990584355
100003	Notreal	Amy	C	1 Fake Road, Soham, Ely, Cambs CB7 5EP	7990584355

Figure 1.13: Member Details Spreadsheet Part 1

Type of Membership	How Paid	Amount	S.O or Cash	When Joined
Peak - 6 Months Upfront	£160 cash upfront (01.02.13-01.08.13)	£160	Cash upfront	23.06.12
Couples Peak (100003 - A. Notreal)	£47 paid for August	£47	S.O.	23.06.12
Couples Peak (100002 - G. Notreal)	£0 - see above	£0 - see a	S.O.	23.06.12

Figure 1.14: Member Details Spreadsheet Part 2

Registration Fee Paid	Date of Induction	Comments
£20 cash paid 23.06.12	25.06.12	6 months peak upfront (01.02.13 - 01.08.13)
£35 cash paid 23.06.12	25.06.12	Changed to Couples Peak (see 100003 - J. Fletcher) from August 2013
£0.00 23.06.12	Hattie: 28.06.12	Changed to Couples Peak (see 100002 - A. Fletcher) from August 2013

Figure 1.15: Member Details Spreadsheet Part 3. All the information stored in this spreadsheet was gathered from previous forms or noted down by a member of staff during the members induction and then entered into the spreadsheet. This is stored locally on a workstation in the gym where only authorised users can gain access to it. The spreadsheets are rarely printed out except in special circumstances like accounting purposes.

Membership No.	Full Name	Type of Membership	How Paid	Amount	Registration Fee (Date)
100001	Tobias Fakeman	Peak - 6 Months Upfront	£160 cash upfront (July-Dec)	£160	£20 paid 23.06.12
100002	George Notreal	Couples Peak (see 100003 - A. Notreal)	£47 paid for August	£47	£35 paid 23.06.12
100003	Amy Notreal	Couples Peak (see 100002 - G. Notreal)	£0 paid for August - see above	£0	£0 - see above

Figure 1.16: Payment Details Spreadsheet part 1

How Pay?	June '12	July '12	Aug '12	Sept '12	Oct '12	Nov '12	Dec '12	Jan '13	Feb '13	Mar '13	Apr '13	May '13	June '13
cash upfront	x	Paid	Paid	Paid	Paid	Paid	Paid		Paid	Paid	Paid	Paid	Paid
S.O.	x	Paid	Paid	Paid	Paid	Paid	Paid	Paid	Paid	Paid	Paid	Paid	Paid
S.O.	x	Paid	Paid	Paid	Paid	x	x	x	x	x	x	x	x

Figure 1.17: Payment Details Spreadsheet part 2. This spreadsheet contains payment details collected from the previous forms and during the induction period. It also keeps track of every month and whether the member has paid for said month.

1.3.2 The proposed system

My proposed system uses a database to keep track of all the data that my client previously kept in an excel spreadsheet using a program created in the python programming language using the PyQt 4 library. This is because its the language I'm most comfortable with and allows for the creation of intuitive UI and the creation of an executable program using the software cx freeze ensuring that the system will be easily usable by my client.

Data sources and destinations

Sources and	Definitions -		Proposed	2.JPG
What is it	Source	Process	Example	Destination
Name	Member	Member enters his name into digital form/ writes it on physical form and is then entered into the database either manually or automatically via the form	Toby Kerslake	Members Table in Database
Address	Member	Member enters his address	123 fake lane	Members Table in Database
Tel. Num	Member	Member enters his number	01353 778934	Members Table in Database
Type of Membership	Member	Member enters his desired membership type	Couples	Members Table in Database
Join Date	Member	Member enters date in which he desires to join	12/05/2015	Members Table in Database
Date of Induction	Member	Member enters the date of their induction	15/05/2015	Members Table in Database
How Paid	Member	Member enters how they have payed for their membership	Card	Members Table in Database

Figure 1.18: Data Sources and Destinations

Amount	Member	How much the member paid for registration	£35	Members Database
Registration Fee	Owner	How much the members registration cost	£35	Members Database
Registration Date	Member	The date of registration	12/05/2015	Members Database
Payment Type	Member	Member enters how they are paying	Pay as you go	Members Database
Date Of Payment	Member of Staff	The date which payment is due	12/06/2015	Payment Table in
How Much?	Member Of Staff	How Much they have to pay	£20	Payment Table in
Paid	Member of Staff	Whether the client has paid for a given month	Yes	Payment Table in
Exercise Name	Member/Owner	The name of an exercise	10 Star Jumps	Exercise Database
Exercise Description	Member of staff/owner	Description of the exercise	Jumping up and Down	Exercise Database

Sources and Definitions - Proposed.JPG

Figure 1.19: Data Sources and Destinations 2

Specific Description	Member of staff/owner	More specific description of the exercise more tailored for the members needs	10 star jumps to increase <u>indurance</u>	Regime Table in Database
Start Date	Member of staff/owner	The date the member started this regime	5/11/14	Regime Table In Database
End Date	Member of staff/owner	The date the member finished doing this regime	17/01/15	Regime Table In Database

Proposed Sources 3.JPG

Figure 1.20: Data Sources and Destinations 3

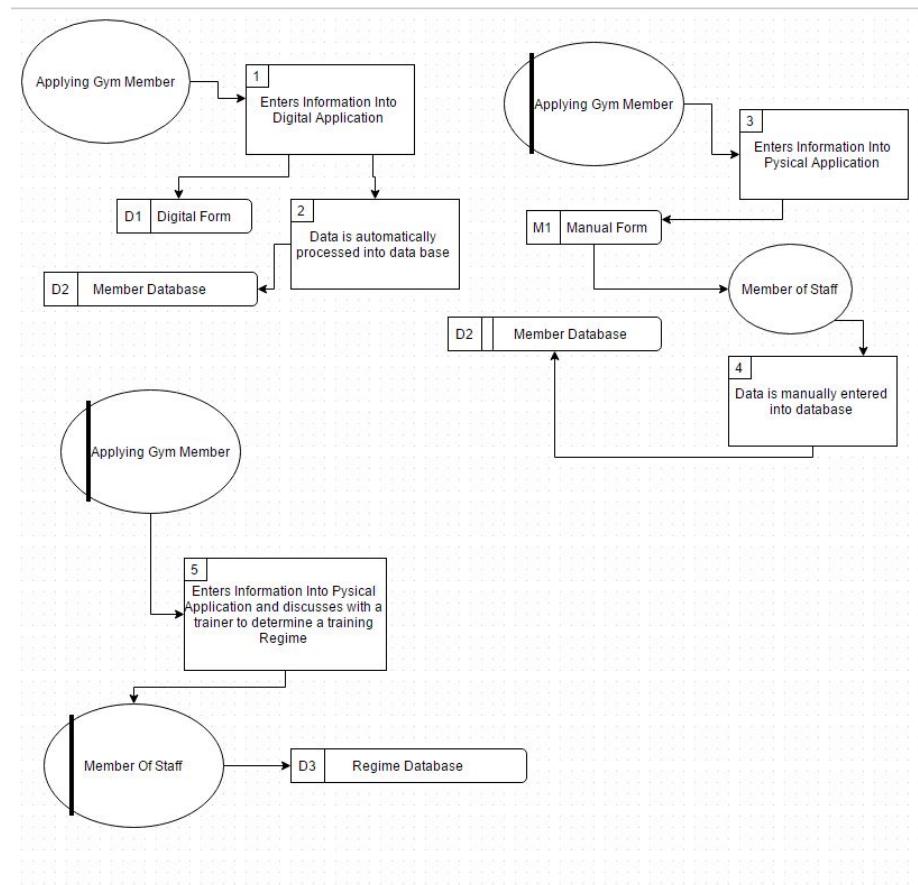
Data flow diagram

Figure 1.21: Data Flow Diagram

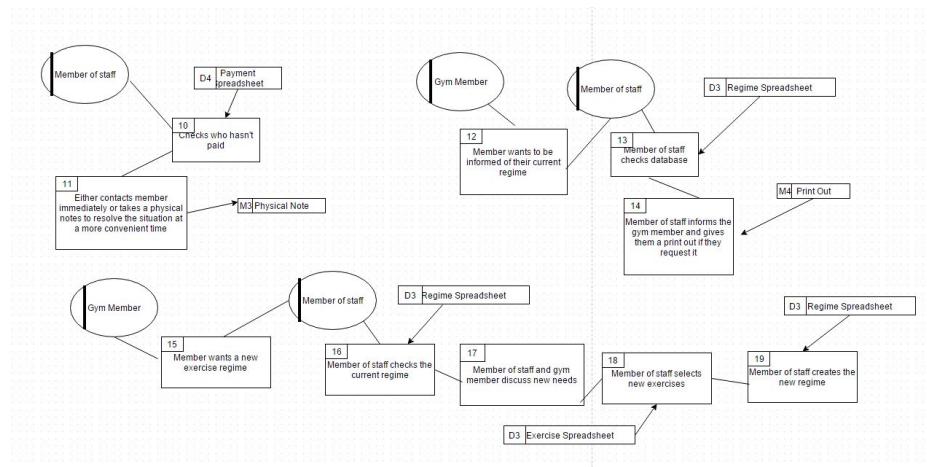


Figure 1.22: Data Flow Diagram

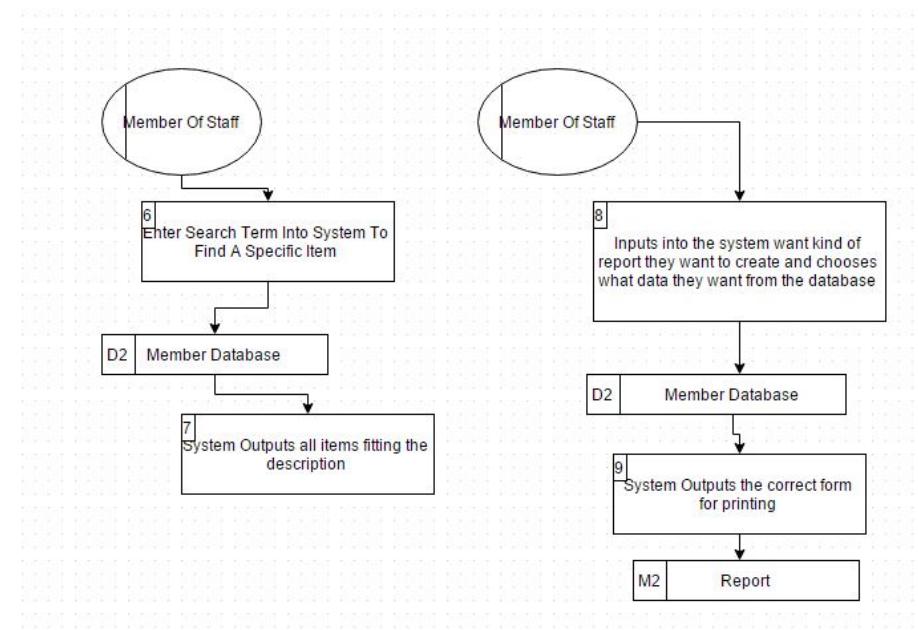


Figure 1.23: Data Flow Diagram. These diagrams show where all the data in the proposed system comes from, where it goes and the process it goes through to get there.

Data dictionary

Entity	Attribute	Data Type	Length	Constrain	Description
Member	Member No	Int	(10)	Primary Key	The unique membership ID
Member	Name	String	(30)	Not Null	The name of the member
Member	Address	String	(30)	Not Null	The members address
Member	Telephone Number	String	(10)	Not Null	The members preferred contact number
Member	Type of Membership	String	(10)	Not Null	The type of membership
Member	Join Date	Datetime	(8)	Not Null	The date the member joined
Member	Date Of Induction	DateTime	(8)	Not Null	The date of the members induction
Member	How Paid	String	(10)	Not Null	How the member payed for their membership
Member	Amount	int	(2)	Not Null	How much the member payed for

Figure 1.24: Data Dictionary

					their membership
Member	Registration Fee	int	(2)	Not Null	How much does the member have to pay for registration
Member	Registration Date	Date/Time	(10)	Not Null	The date of registration
Payment Details	Member No	Int	(10)	Primary Key	The unique membership ID
Payment Details	Payment Type	String	(22)	Not Null	How the client pays
Payment Details	Date Of Payment	Datetime	(8)	Not Null	The Date the payment is due
Payment Details	How Much	Int	(2)	Not Null	How much the client needs to pay
Payment Details	Paid	Boolean	(1)	Not Null	Whether the client has paid
Regime	Member No	Int	(10)	Primary Key	The unique membership ID
Regime	ExerciseID	int	(10)	Not Null	The unique exercise identifier
Exercise	ExerciseID	int	(10)	Primary Key	The unique exercise identifier

Figure 1.25: Data Dictionary 2

Exercise	Name	String	(20)	Not Null	The name of the exercise
Exercise	Description	String	(50)	Not Null	Description of the exercise
Regime	Specific Description	String	(50)	Not Null	More specific description of the exercise tailored for the member
Regime	Start Date	Datetime	(8)	Not Null	The date the member started the regime
Regime	End Date	Datetime	(8)	Not Null	The date the member finished doing the regime

Figure 1.26: Data Dictionary 3

Volumetrics

I have decided on an initial size of 2000 members since the gym is nearing that number of members and will inevitably achieve this amount of clients. My client has said the amount of clients he gains isn't consistent as people often register in different sized groups. Although over the past year he has gained 560 new members and roughly 400 members the year before, and the year before that. This means that we can estimate that in about a year and a half the gym will exceed 2000 members. The size of the database can be increased at a later date when needed.

Each member has 11 - 16 fields of data in total, with each data field taking up 1 KB of data.

$$16 * 1\text{KB} * 2000 = 32000 \text{ KB} / 1024 = 31.25 \text{ MB}$$

There is also the exercise tables and regime tables which will take up roughly 10,000 KB of data assuming each member will have an average of 10 exercises each in their regime, each exercise taking up 1KB of data. The exercise table would then likely come up to 300 KB of data as currently the owner estimates that he currently offers 300 different variations of exercises.

$$10,300 / 1024 = 10.05 \text{ MB}$$

The actual program should then take up a small amount of space that will probably take up about 3MB.

$$31.25 \text{ MB} + 10.05 \text{ MB} + 3\text{MB} = 44.30 \text{ MB}$$

1.4 Objectives

1.4.1 General Objectives

- Clear/easy to understand layout structure for member/client records viewing/observation
- Clear/easy to understand layout structure for all member/client data input
- Clear/easy to understand layout for creating and printing invoices/ member records
- Clear/easy to understand digital/physical input forms for users to enter data
- Secure system that is password protected so that only appropriate staff members can use the system
- Certain options like deleting items from the database restricted to only an administrator/ separate password that only higher level staff members can access
- Presentable, minimalistic GUI that can be easily understood and interacted with by the users
- Appropriate Screen outputs for example dialog boxes and the presentation of the database presented in the GUI mockup.

1.4.2 Specific Objectives

Client/Member Record Viewing/Data Input

- Easy to use Simple Gui with clearly labelled buttons to choose an option (as shown in the mock-ups I presented my client).
- There should be buttons for opening a table, add to it, editing it, deleting an item, creating an invoice or physical record, and searching for something specific.

- Each of these buttons should open up either drop down menus or entirely new windows, depending on which is more user friendly, and display further options within those commands.
- The interface for entering new information should be easy to use and clearly labelled for adding each attribute to the specified table.
- There should be an easy/efficient way to switch and search between all of the tables in the database.
- Avoid more novice users accidentally deleting items by having a secondary password for higher level access.
- Store backups of the database online using a system like dropbox or GitHub so that in the event of data corruption there would be recent version to backup to
- Storing the latest version of the spreadsheet locally also adds an extra layer of security as the most recent version won't be easily accessible except by people with direct access to the workstation.

System Outputs

- Easy to create reports containing all of a specific client's information in an organised manner on as little paper as legibly possible.
- Easy to create invoices for members based off of the information stored in the tables.
- Reports should include Invoices, database table printouts, member reports giving details selected from the databases.
- These reports should be printable so that they can be given to members and stored physically

Input Forms

- Easy to fill out digital input forms for entering new data into the database tables instead of a more complicated SQL query based system.
- Printable versions of these forms that can be created to be filled out by hand by a client that can then have their information manually entered into the system by a member of staff.

1.4.3 Core Objectives

- Easy to use Simple GUI with clearly labelled buttons to choose an option (as shown in the mock-ups I presented my client).
- There should be buttons for opening a table, adding to it, editing it, deleting an item, creating an invoice or physical record, and searching for something specific.

- Each of these buttons should open up either drop down menus or entirely new windows, depending on which is more user friendly, and display further options within those commands.
- The interface for entering new information should be easy to use and clearly labelled for adding each attribute to the specified table.
- There should be an easy/efficient way to switch and search between all of the tables in the database.
- Avoid more novice users accidentally deleteing items by having a secondary password for higher level access.
- Store backups of the database online using a system like dropbox or GitHub so that in the event of data corruption there would be recent version to backup to
- Storing the latest version of the spreadsheet locally also adds an extra layer of security as the most recent version wont be easily accessable except by people with direct access to the workstation.
- Easy to create reports containing all of a specific clients information in an organised manner on as little paper as legibly possible.
- Easy to create invoices for members based off of the information stored in the tables.
- Reports should include Invoices, database table printouts, member reports giving details selected from the databases.
- These reports should be printable so that they can be given to members and stored physically

1.4.4 Other Objectives

- Easy to fill out digital input forms for entering new data into the database tables instead of a more complicated sql query based system.
- Printable versions off these forms that can be created to be filled out by hand by less technically inclined clients that can then have their information manually entered into the system by a member of staff.

1.5 ER Diagrams and Descriptions

1.5.1 ER Diagram

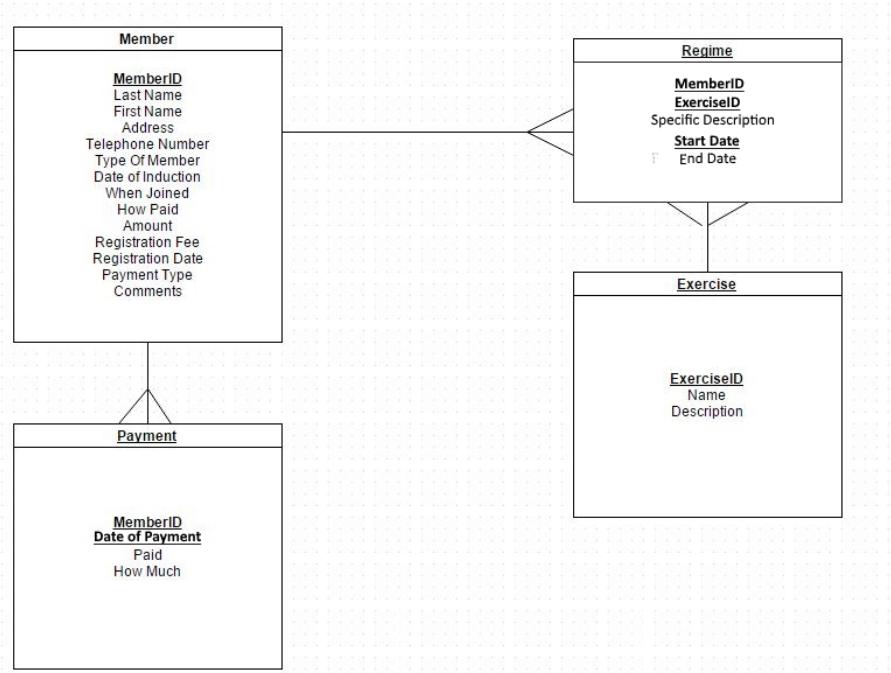


Figure 1.27: ER Diagram

1.5.2 Entity Descriptions

Membership Details(**Membership No.**, Last Name, First Name, Address, Telephone No., Type Of Membership, Date of Induction, When Joined, How Paid, amount, registration fee, Registration Date, Payment Type, Comments)

Payment Details(**Membership No.**,**Date of Payment**, How Much, Paid)

Regime(**Membership No.**,**ExerciseID**, Specific Description, **Start Date**, End Date)

Exercise(**ExerciseID**, Name, Description)

1.6 Object Analysis

1.6.1 Object Listing

Membership Details

Payment Details

Regime

Exercise

1.6.2 Relationship diagrams

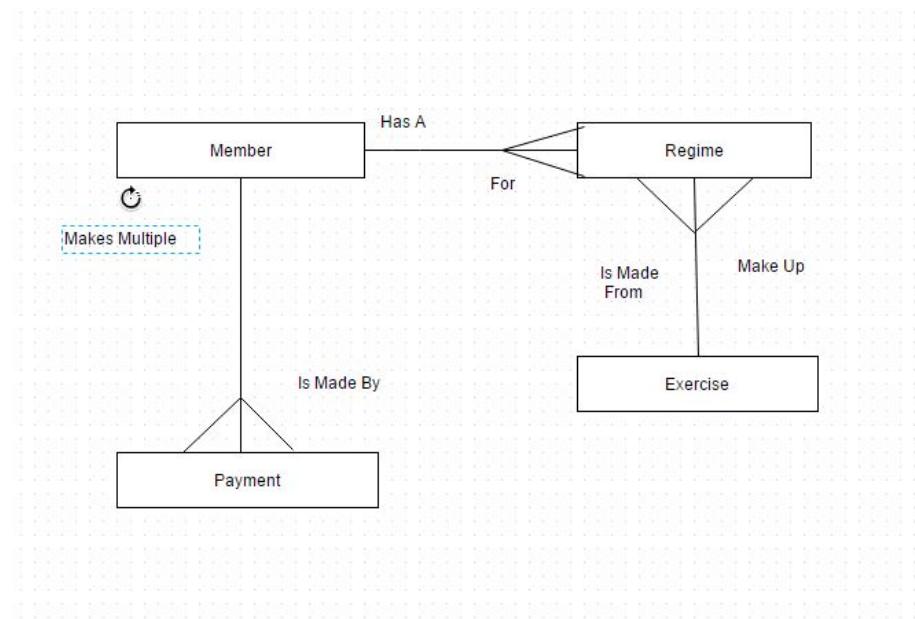


Figure 1.28: Relationship Diagram

1.6.3 Class definitions

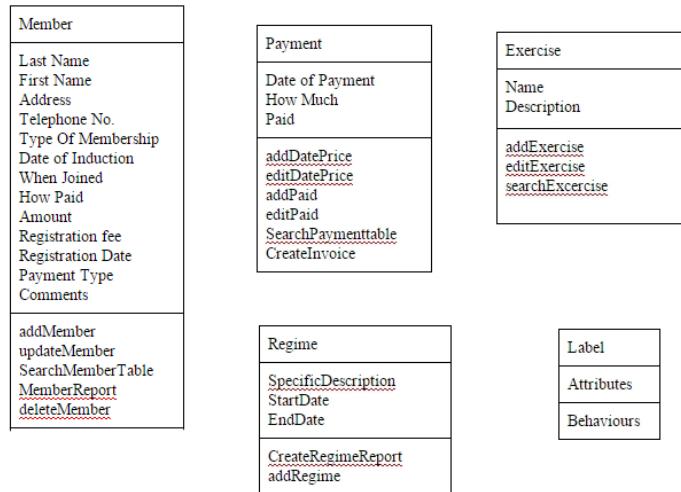


Figure 1.29: Definitions

1.7 Other Abstractions and Graphs

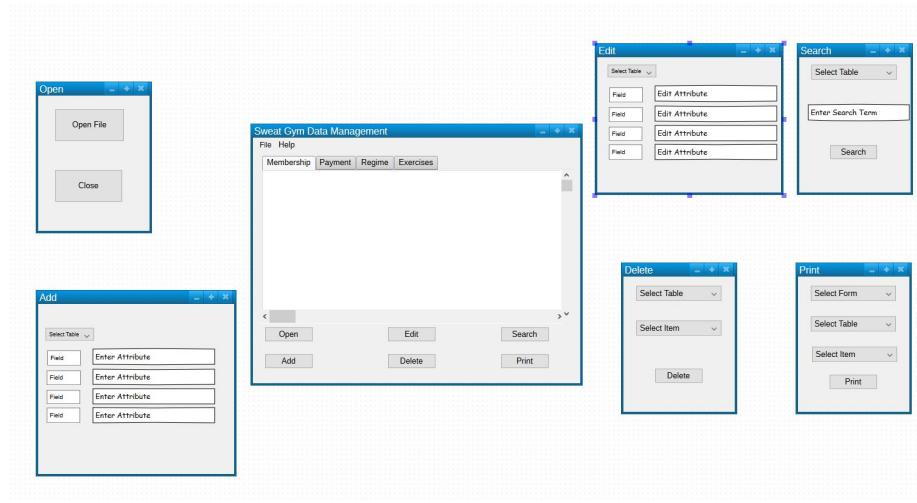


Figure 1.30: This is the early preliminary designs for the GUI for the proposed system. It shows a main window (with a white box representing where the database will be presented) containing 6 push buttons that open into the 6 respective dialog boxes.

1.8 Constraints

1.8.1 Hardware

The owner currently uses a Notebook laptop powered by an intel 1st generation i5 processor and 4GB of RAM. He already uses this laptop to run his current system which uses more resources and power than this system will hopefully require, but if not he still has more than enough horse power to run the new system. His use of a laptop is convenient as it means his system is portable so he doesn't need to do any data entry (if required) confined to his office and away from his client if he doesn't want too and it also means he has a battery so in the case of a power outage he wont lose any data. Though it is worth noting that he may soon be upgrading to a more powerful desktop soon so while this lowers portability, he has more power for the proposed system and he can easily have a battery backup in the form of a UPS(Uninteruptable Power Supply).

All members will be running this program off the same workstation so the databases don't need to be primarily stored online and stored locally as the files don't need to be portable. This also adds a level of security.

The only hardware constraint this laptop proposes is the size and resolution of the screen. As it is only 1440p x 900p and 15 inches diagonally this doesn't give him much room to observe over 1300 clients without excessive amounts of scrolling. Fortunately this could be easily solved by either upgrading to a desktop or simply using an external monitor of a higher resolution.

1.8.2 Software

My client has no preference on what software can or cannot be used as long as its easily accessible to him as he isn't the most computer literate person. Its worth noting that although he has no particular requirements for software he isn't willing to learn a new operating system so it will have to run on windows 7. This is not a problem since the proposed system will be developed on a windows os and should be fully compatable. The client wants the system to be efficient so the databases are going to be programmed in SQL and the system itself is going to have two levels of password protection so that the data stays secure.

1.8.3 Time

My client has given me no personal deadline for this project so the only one I have is the one set by AQA which at the time of writing is April 2015. Although my client doesn't need the new system immediately he is willing to use it as soon as possible and willing to test it before its finished providing its in a fully functioning state.

1.8.4 User Knowledge

Although the owner and staff members aren't specifically trained in I.T they are all completely able to use a computer when given the necessary instruction and support. Most of the staff only use computers for email and web browsing, and occasioinally writing things with a word processing package, as well as the current system which is computer based.

1.8.5 Access restrictions

Access to the proposed system should only be available to members of sweat gymnasium staff and thus the system may need a password although my client insists that only authorised people will have access to the computer, which is password protected itself so the password may be set to optional incase he wants to deactiate it. Due to the system containing private and confidential data on individuals it will have to comply with the Data Protection Act and appropriate action will be taken to do so though the care of the programs security is mostly

in the hands of the staff and owner. Although the password for the program may be made optional I insisted that the database be password protected so that people without clearance can't open the database in another browser for malicious or fraudulent purposes. If a password is used for the program which it likely will be then a secondary password will be used for the delete option so that no novice users can accidentally delete an item as only higher users who are authorised (like the manager) will have access to that password.

1.9 Limitations

1.9.1 Areas which will not be included in computerisation

For the less technically inclined who don't have email or don't feel that digital forms are of convenience there will be physical forms for them to fill out that would then require their data to be entered into the proposed system manually by the owner or a member of staff. The forms that need to be filled out to determine an exercise regime/programme will also stay physical as they have some very specific questions that can't easily be answered with enough detail in a digital form since the applying member may need assistance from a member of staff and consultation from a trainer.

1.9.2 Areas considered for future computerisation

Although the invoices and printout records of each member could be stored digitally this would be a bad idea for a number of reasons. First of all the printouts could need updating at anytime and if they need a new physical record the gym will need an up to date one instead of an older one they saved from a couple of months before. Another reason for this is that I've been told by the owner that he would prefer to keep hard copies of invoices over digital so that he not only has a physical record of the invoice for accountancy reasons and so he can send one to any of his clients, even the ones with little computer knowledge. The forms also change repeatedly so if they were saved an older version could be printed out instead by accident. This will be considered for future computerisation as it may come in handy if the owner decides he may want digital copies of forms kept if he believes he's going to need to repeatedly print off an unchanged form in the future, or needs a digital record of a table in a previous state. So a function to save a form to a text file or binary file that can be reread by the program to be printed off in the future may be added.

1.10 Solutions

1.10.1 Alternative solutions

Improving Spreadsheets

- Advantages
 - More familiar structure to the current system
 - Easier to edit to adapt to new features
 - Easy to make back ups
 - Microsoft support available if anything goes wrong with the software
 - Gets rid of all the data duplication in the current spreadsheets
- Disadvantages
 - Still prone to the same security issues as the old system
 - All data still has to be entered manually
 - No easy and specific way to sort through data
 - No easy to create invoices or printable records

Online HTML Based Application

- Advantages
 - Easy to access using any computer running any operating system and even mobile devices
 - Low system requirements as it runs in browser based on HTML
 - Cloud based so backups can be kept securely on error checking hardware/servers
 - SQL has great web integration for databases
 - Wouldn't require that anything would need to be installed
- Disadvantages
 - I don't know HTML/CSS and javascript that well and would have to spend large amounts of development time learning the language
 - Online applications have massive security risks and this could open the system up to attacks like SQL injection
 - If the gym's internet connection becomes inadequate for any reason then the system couldn't be accessed
 - Web hosting can be very expensive

Paper Based Manual System

- Advantages

- Easy for everyone to understand as it requires no technology
- Can be used immediately as there will be basically no development time

- Disadvantages

- Requires a large amount of space to store so many physical documents
- No easy way of making backups except for maybe carbon copies
- Can easily be damaged and destroyed/lost
- Has no easy search method and could easily become un-organised
- No way to edit documents

Command Line Application Programmed in Python implementing a database saved to text or binary files

- Advantages

- Relatively easy to program with a short development time
- Easy to use if users are taught appropriately
- Can be programmed to do anything required and easily modified in the future
- Saving the databases to one of these formats is relatively easy to program and manage.

- Disadvantages

- Commandline can be difficult for some people to understand
- Executing the program may be difficult for some users
- Using text files and binary files is very insecure as they can be easily edited, manipulated and read by anyone with moderate I.T knowledge.
- Using these formats means that all sorting methods and search methods have to be programmed by me instead of using a function from a library like sqlite3 resulting in a considerable use of time and resources that isn't necessary.

Command Line Application Programmed in Python implementing a database programmed and managed in SQL

- Advantages

- Relatively easy to program with a short development time

- Easy to use if users are taught appropriately
- Can be programmed to do anything required and easily modified in the future
- Uses relational databases that will be easily understandable as it will be designed to my clients specifications when compared to the previous excel spreadsheets as well as being more secure than a text or binary file as well as being easier to manipulate with the sqlite3 library.
- With the database being saved as a database file, if the program encounters some form of unknown bug down the line the database can still be opened using another SQL browser program

- Disadvantages

- Commandline can be difficult for some people to understand
- Executing the program may be difficult for some
- SQL is slightly more difficult to program compared to just using text files or binary files
- SQL still isn't totally secure as using digital input forms means that it may be susceptible to SQL injection and other security faults.

GUI Based Application Programmed in Python using the PyQt Library implementing a database programmed and managed in SQL.

- Advantages

- Easy to program as I have experience with both python and PyQt so the development time will be quite short
- Can be programmed to do anything the client requests
- Easy to use as entirely GUI based so users only need basic computer knowledge
- That program can be made into an executable using cx freeze so its easy to open and doesn't require the install of any additional programs
- Can be programmed to do anything required and easily modified in the future
- Uses relational databases that will be easily understandable as it will be designed to my clients specifications when compared to the previous excel spreadsheets as well as being more secure than a text or binary file as well as being easier to manipulate with the sqlite3 library.

- With the database being saved as a database file, if the program encounters some form of unkown bug down the line the database can still be opened using another SQL browser program
- Disadvantages
 - Not as easy an adjustment as updating the current spreadsheets would be
 - Not as easy to create as a command line based program
 - SQL is slightly more difficult to programm compared to just using text files or binary files
 - SQL still isn't totally secure as using digital input forms means that it may be susceptable to SQL injection and other security faults.

1.10.2 Justification of chosen solution

My Chosen solution is too use an application with a GUI programmed in python with the library PyQt implementing a database programmed and managed in SQL. This is because of a number of reasons:

- I am familiar with python and PyQt and know how to use them as well as knowing how to use and disect their documentation if I ever get lost
- Python is easy to install and I can easily use it to debug the program on the gyms workstation if needed incase of emergency though the program will be distributed to my client as a executable
- If its more accessable to the client I can even make an executable of the program using a program like cx freeze
- Programming in python gives me the opportunity to present and demonstrate my knowledge of complex programming features like Object Oriented Programming and recursive programming
- Python gives me he flexibility to program the system to my clients exact specification as its a fully featured programming language
- The use of relational databases can be implemented using python and the SQLite3 library meaning it will be easy to store and manipulate the data required using SQL as oppossed to just a text file or a binary file or an Excel spreadsheet
- The use of SQL databases is that it is much more secure than simply saving the data to a text file, or pickling the data to create a binary file, and even more secure than a password protected or encrypted Excel spreadsheet. It also means that in the event that the program encounters an issue that the database can be opened in another SQL Browser, even if the interface

wont be as suited to the clients needs as the one I will have programmed in the chosen solution.

- This also means that the database can be presented in any format/design that is desired as long as its possible to program

Chapter 2

Design

2.1 Overall System Design

2.1.1 Short description of the main parts of the system

- Main Parts
 - Database View Interface
 - * The main window for accessing the entire system. It is designed so that all the dialog boxes and other windows can be opened from within it and gives a main, tabbed layout for the tables.
 - * Presented as a database window with 6 push buttons that open dialog boxes for different functions
 - * the 6 push buttons are labelled:
 - Open
 - Add
 - Edit
 - delete
 - Search
 - Print
 - * Each table in the database is displayed in a different tab
 - * Main window displays database in a box above function buttons

- * Has a menu bar at the top that provides shortcuts to the functions and an about section with a help section that contains a user manual
- * Add Interface
 - Dialog box that contains forms that allow the user to enter data specific to the table they want to add an item to
 - Drop down menu to select table
 - Fields with editable text boxes/line edits to enter attributes
 - confirm button
- * Edit Interface
 - Dialog box that contains forms that allow the user to edit data specific to the table they want to add an item
 - Drop down menu to select table
 - Fields with editable text boxes/line edits to enter attributes
 - confirm button
- * Delete Interface
 - Dialog box allowing the user to delete a specific item from any of the tables
 - Select table drop down menu
 - Select item drop down menu
 - Delete button
 - Delete all items button
 - require a password to be entered first in a dialog box
- * Search Interface
 - Dialog box allowing the user to search through any or all of the tables to find a specific search term/query
 - Select table drop down menu
 - Search term QLineEdit
 - Search push button
- * Print Interface
 - Dialog box allowing the user to select a type of form to print out based on selected information like a list of members, an invoice, or a regime.

- 3 Combo boxes allowing the user to select the type of form, the table they want the information from, and the information.
 - A push button allowing the user to confirm they want to print the selected details.
- * Password Interface
- Dialog box that presents the user an outlet for entering the password for the system
 - Enter password lineEdit allowing the user to enter their password
 - Enter password push Button allowing the user to confirm the password they've entered

Toby Kerslake

Candidate No. 44108

Centre No. 22151

2.1.2 System flowcharts showing an overview of the complete system

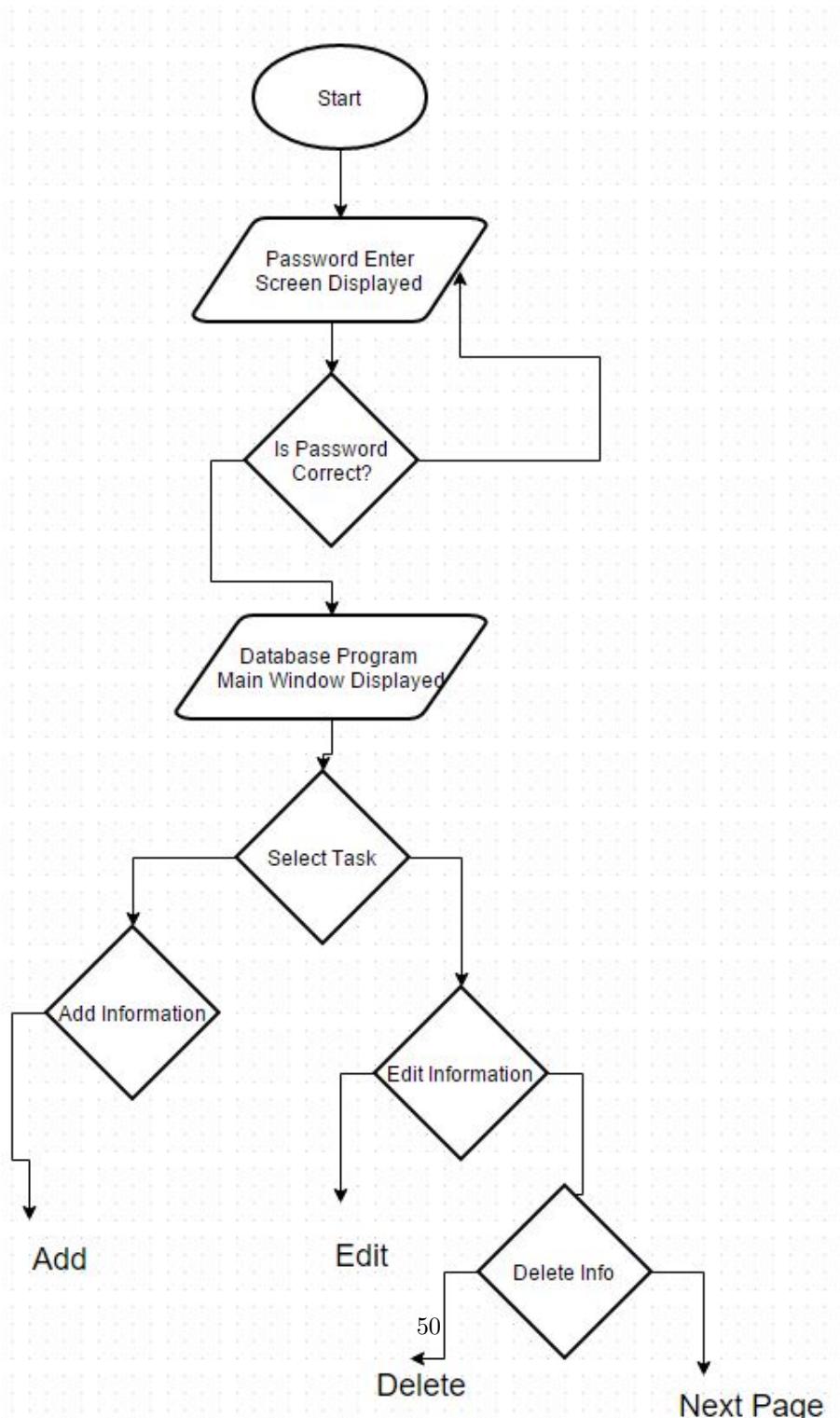


Figure 2.1: Flowchart

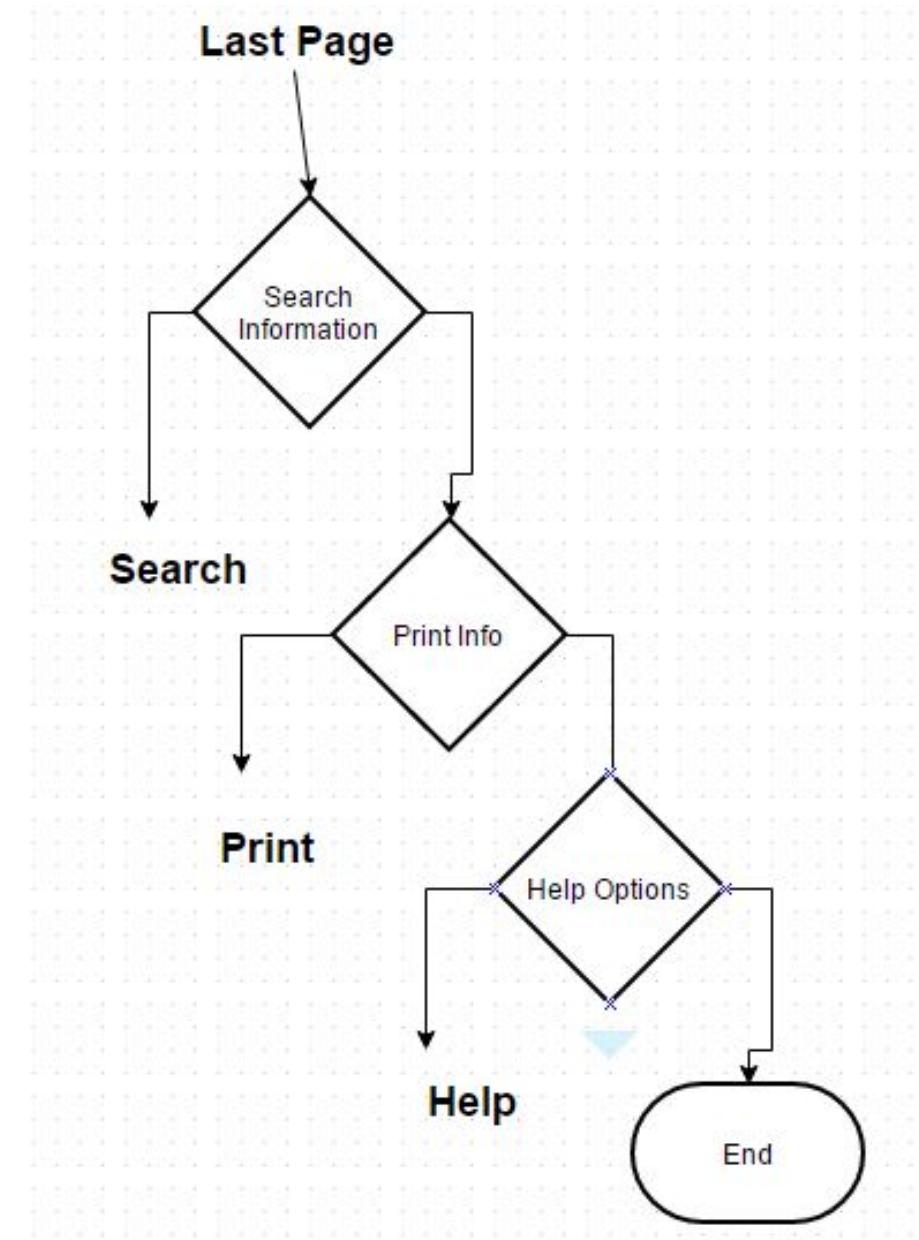


Figure 2.2: Flowchart

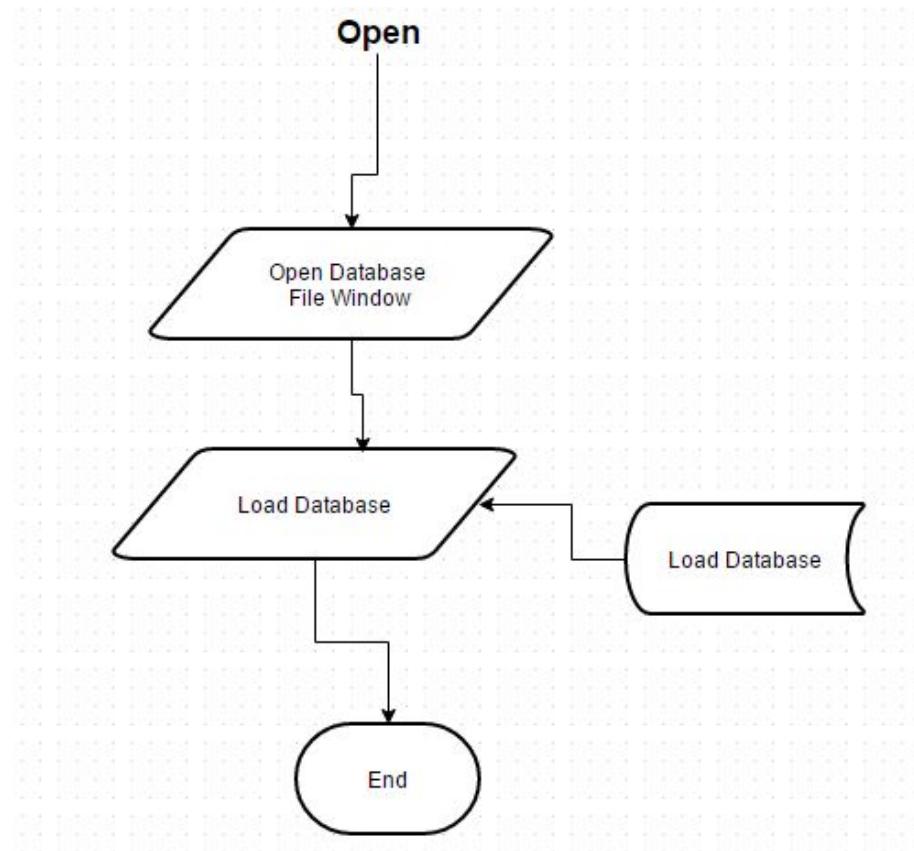


Figure 2.3: Flowchart

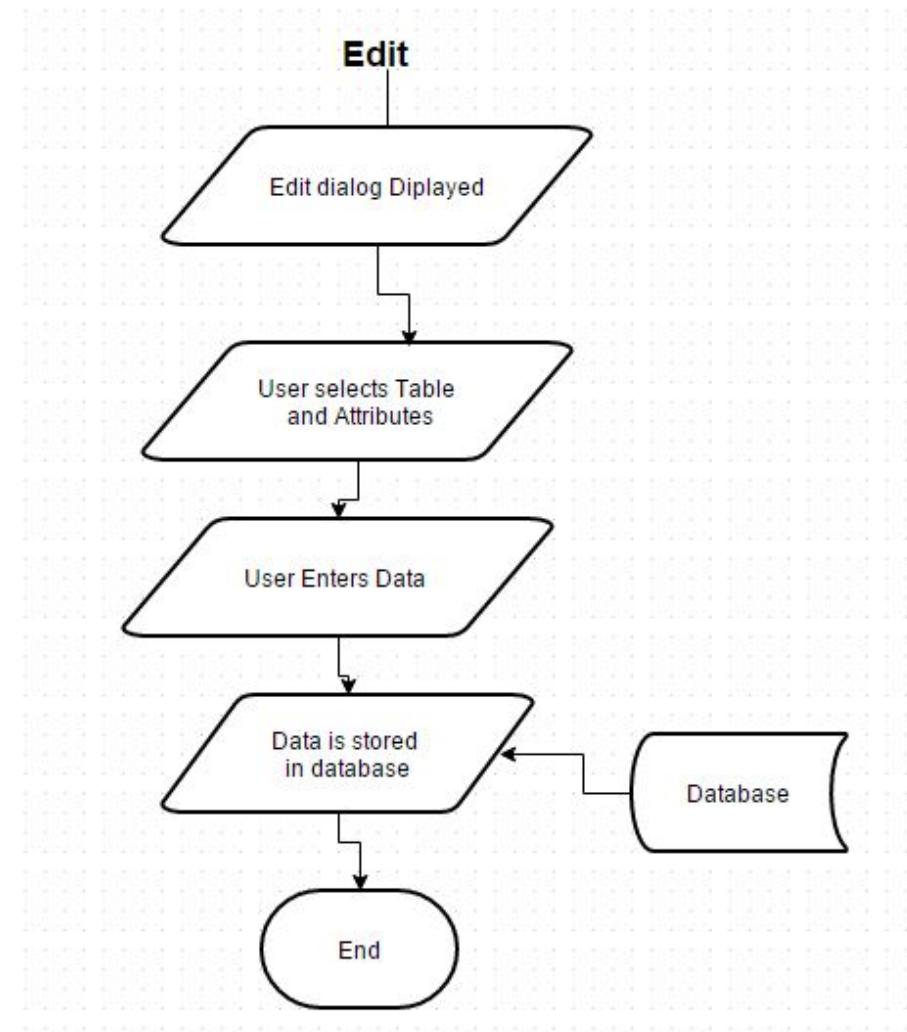


Figure 2.4: Flowchart

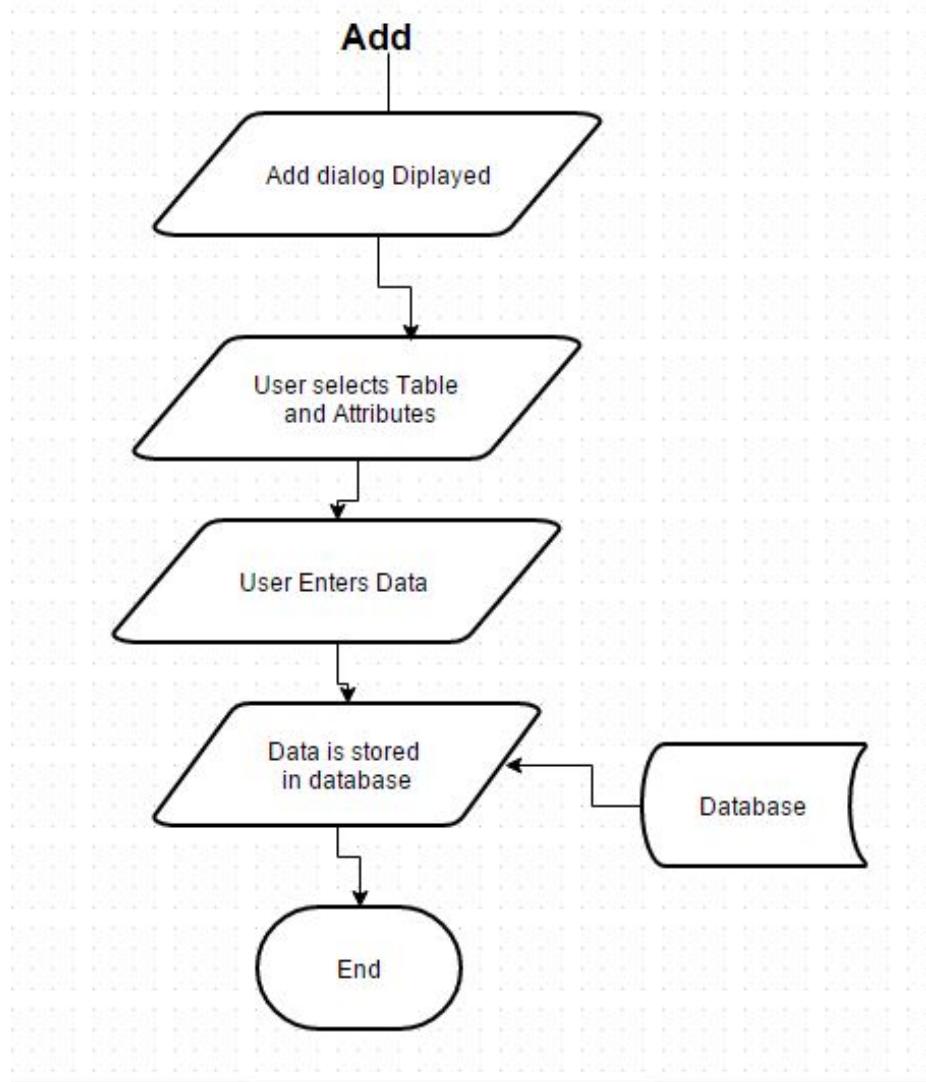


Figure 2.5: Flowchart

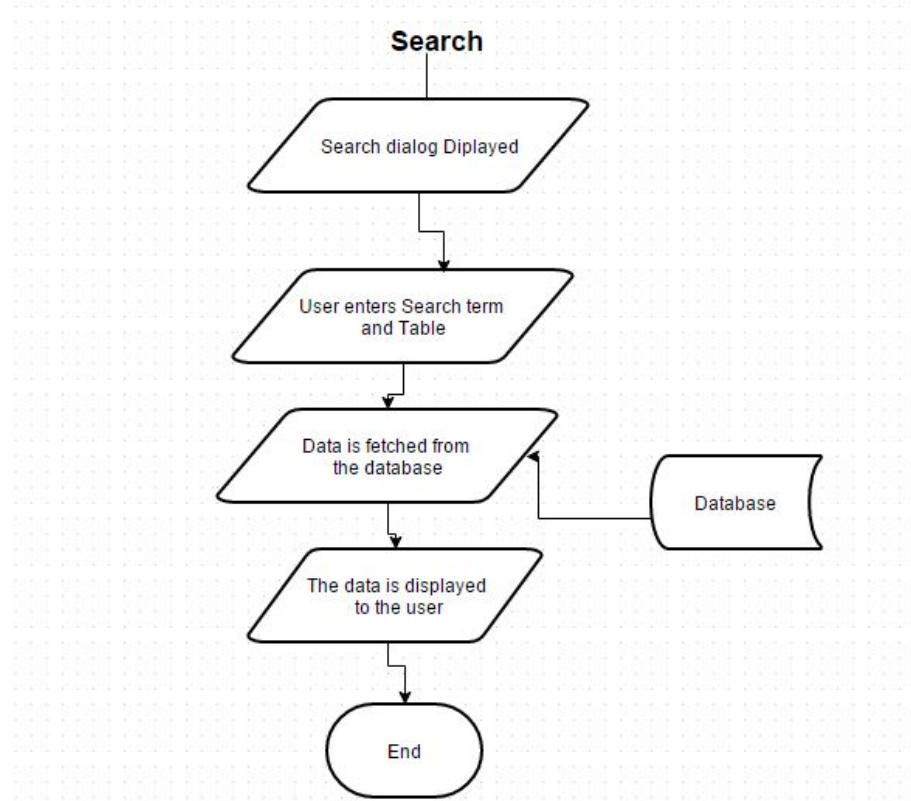


Figure 2.6: Flowchart

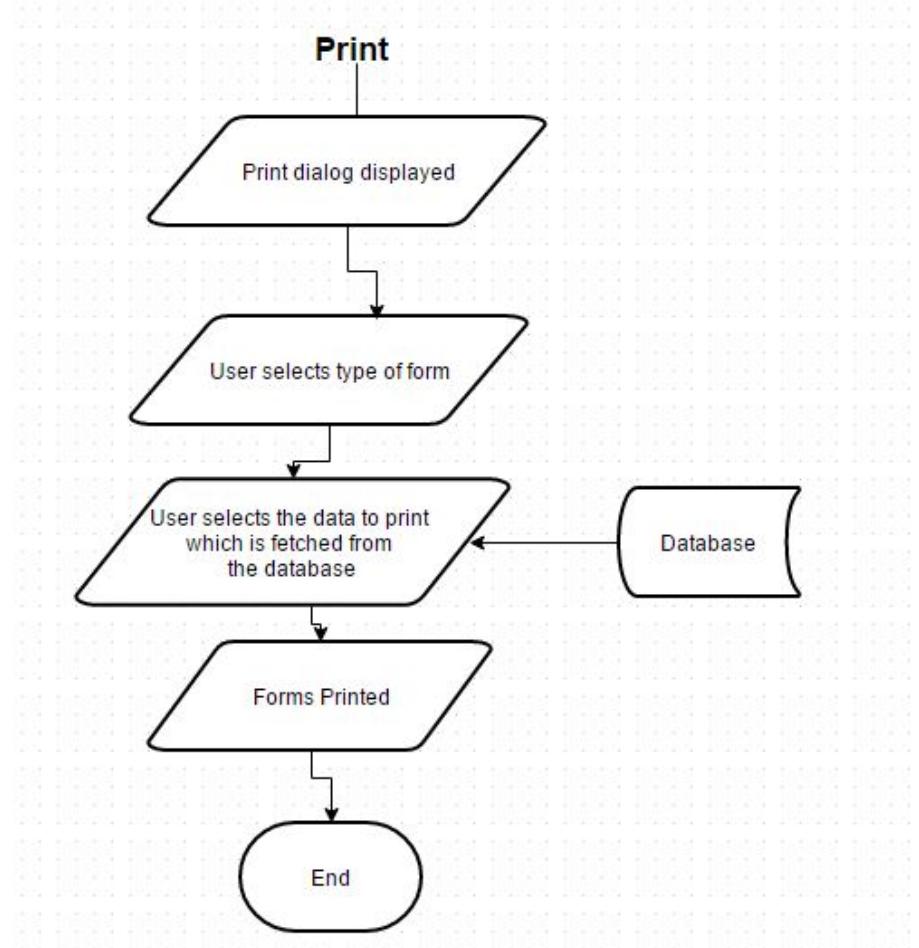


Figure 2.7: Flowchart

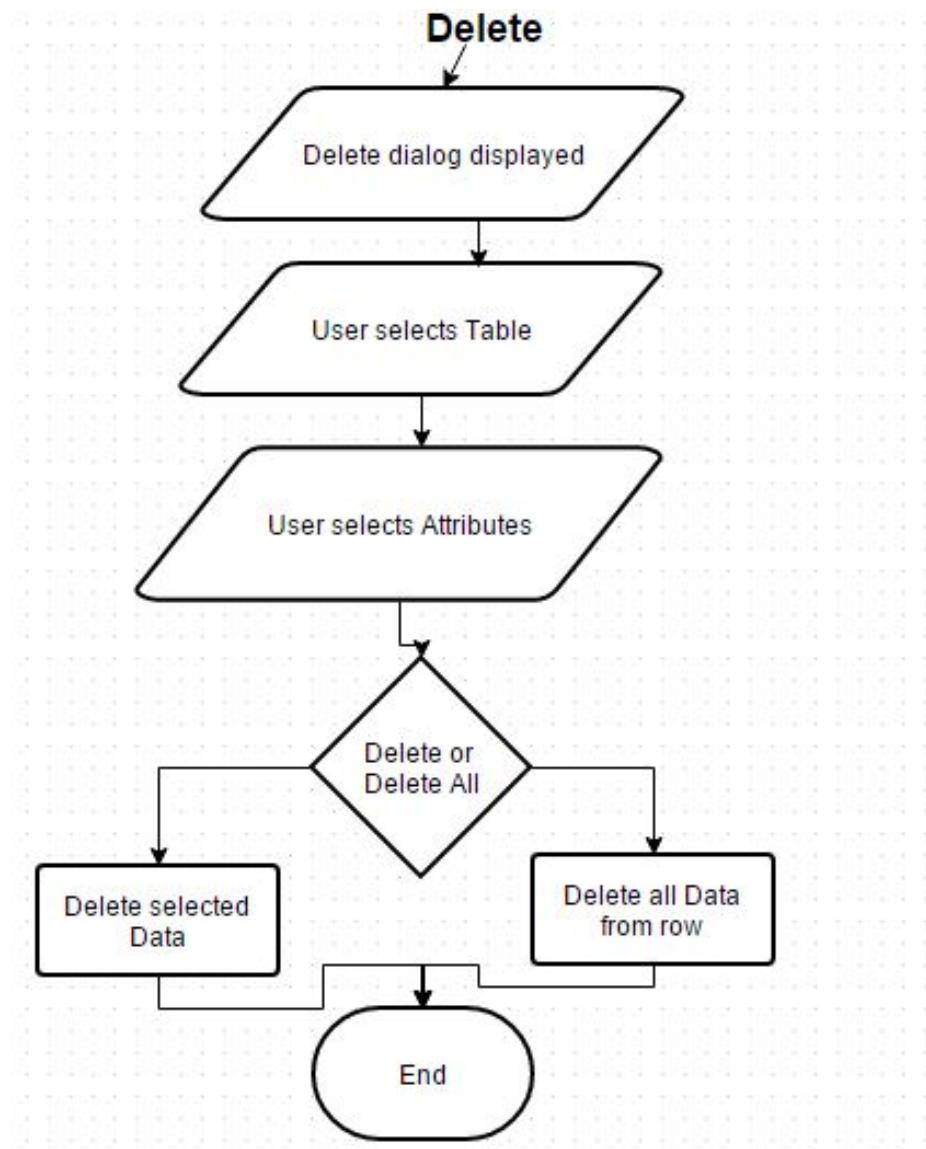


Figure 2.8: Flowchart

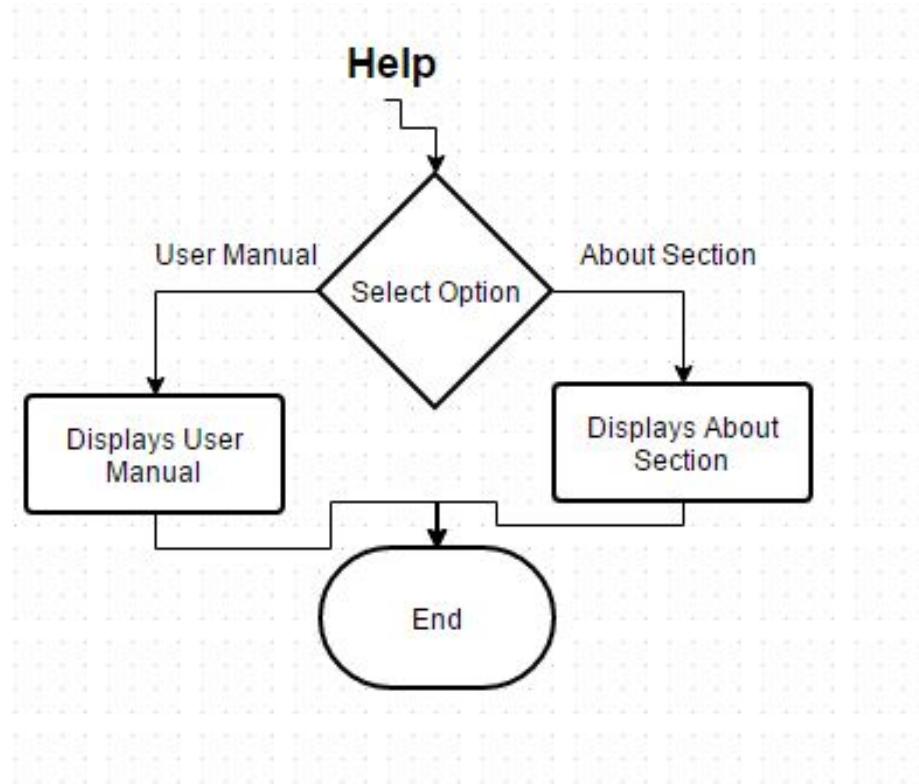
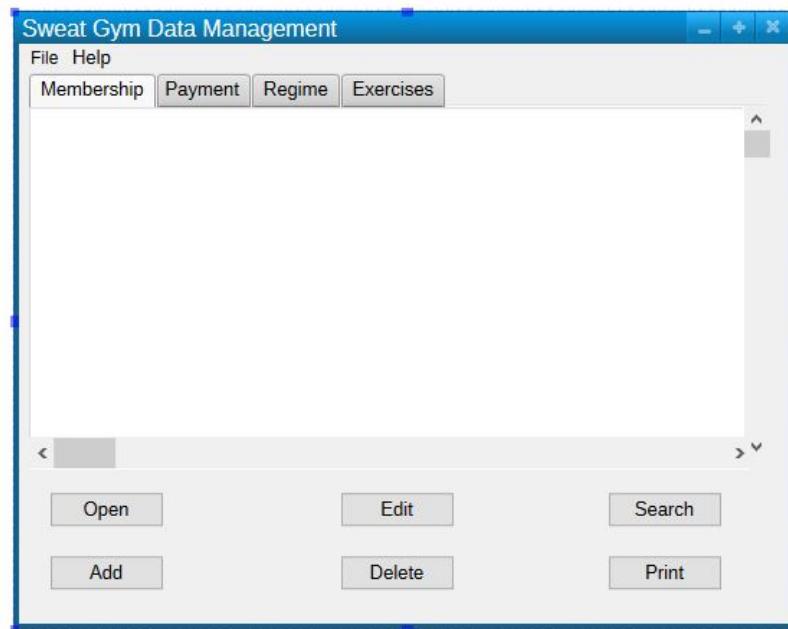


Figure 2.9: Flowchart

2.2 User Interface Designs



This is the design for the main window. The window would allow the user to get to any of the other dialogs. This includes all the programs main functions including opening the database file, adding or editing items in the database, deleting items from the database, searching through the database and printing different documents based on information from the database. It also has the main view for the tables displayed in a tabbed layout for each table. It also has shortcuts to all the functions in the file dropdown menu and links to a digital user manual and an about section in the help drop down menu.

Figure 2.10: GUI Design

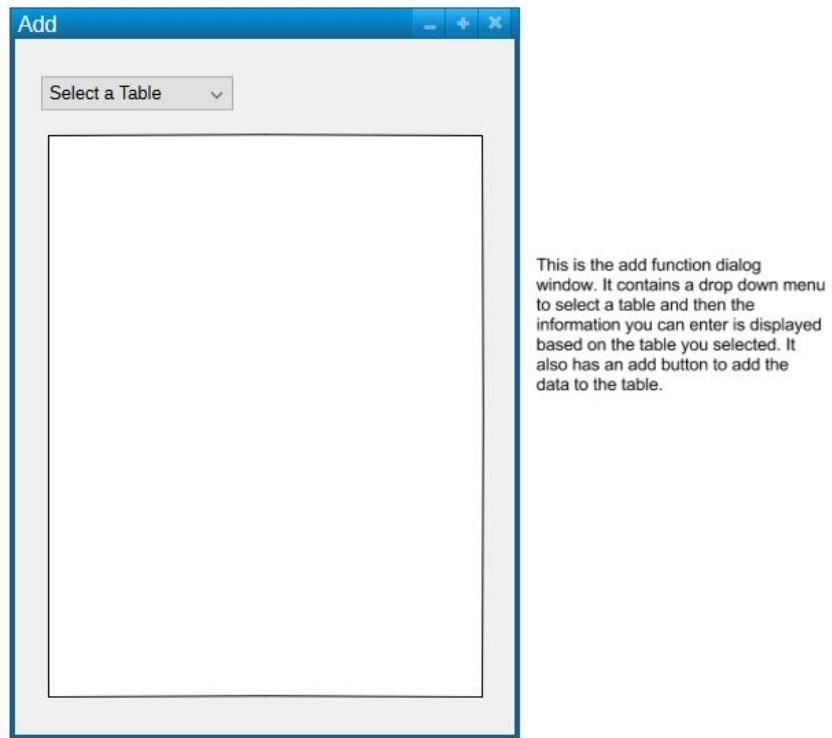


Figure 2.11: GUI Design

Add

Membership

Member Name	<input type="text" value="Enter Text"/>
Member ID	<input type="text" value="Enter Text"/>
Membership Type	<input type="text" value="Enter Text"/>
How Paid?	<input type="text" value="Enter Text"/>
Date of Induction	<input type="text" value="Enter Text"/>
Member Address	<input type="text" value="Enter Text"/>

The following designs show how the table appears with different data to be entered based off of which table is selected.

Figure 2.12: GUI Design

Add

Payment

Payment Type	Enter Text
Member ID	Enter Text
Date of Payment	Enter Text
How Paid?	Enter Text
How Much	Enter Text
Paid	Yes <input type="button" value="▼"/>

Figure 2.13: GUI Design

Add

Regime

Member ID	<input type="text" value="Enter Text"/>
Exercise Name	<input type="text" value="Enter Text"/>
Specific Description	<input type="text" value="Enter Text"/>
Stat Date	<input type="text" value="Enter Text"/>
End Date	<input type="text" value="Enter Text"/>

Figure 2.14: GUI Design



Figure 2.15: GUI Design

Edit

Membership

Member ID: 67 - Marcus ...

Member ID

Membership Type

How Paid?

Date of Induction

Member Address

Enter Text

Enter Text

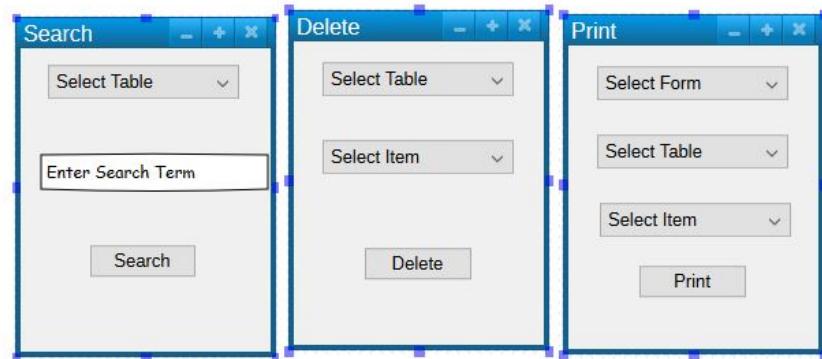
Enter Text

Enter Text

Enter Text

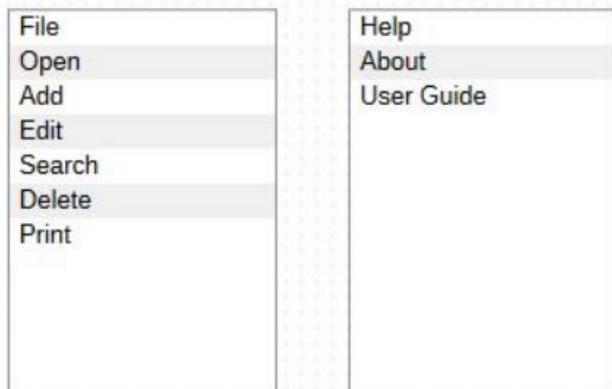
This is the edit window. It looks similar to the add window except it contains a combobox to select the primary key for the entity the user wants to edit.

Figure 2.16: GUI Design



These are the designs for the Search, Delete and Print dialog boxes. The search Dialog box contains a combobox to select the table to search through, although it does contain an option to search through all of the tables. It contains a line edit for the search term and a push button to execute the search. The delete dialog box contains 2 comboboxes to select the table and select the item. There is an option for all items in the combobox. There's a delete push button to execute the function but it opens up a new dialog for a password to be entered so an inexperienced user won't accidentally delete any data. The print dialog contains 3 comboboxes to select the form you want to print and the specific data you want in that type of form. It can be executed by the print push button.

Figure 2.17: GUI Design



The designs here show the drop down menus for the file and help options in the toolbar. Below this is the dialog box for entering the password to use the program and delete data. The dialog contains a qlineedit for the password to be entered and a push button to submit the password.

Figure 2.18: GUI Design

2.3 Hardware Specification

The owner currently uses a Notebook laptop powered by an intel 1st generation i5 processor and 4GB of RAM. He already uses this laptop to run his current system which uses more resources and power than this system will hopefully require, but if not he still has more than enough horse power to run the new

system. His use of a laptop is convenient as it means his system is portable so he doesn't need to do any data entry (if required) confined to his office and away from his client if he doesn't want too and it also means he has a battery so in the case of a power outage he wont lose any data. Though it is worth noting that he may soon be upgrading to a more powerful desktop soon so while this lowers portability, he has more power for the proposed system and he can easily have a battery backup in the form of a UPS(Uninteruptable Power Supply). All members will be running this program off the same workstation so the databases will be stored locally. This wont be a problem as the laptop has a 500 gb hard drive that wont be filled up too quickly and has lots of room for databases and the program. The program will be output through a display with a resolution of at least 600 * 800 (like the one used in the laptop) to accomodate the amount of screen real estate the program will require and the program will be operated and have data inputed using mouse/trackpad and keyboard (also contained within my clients laptop). The system will not use any additional peripherals. The system will also need to run on a windows os (windows 7, 8, 8.1, or 10) as thats what its been developed on and intended to run on, though it may also be compatable with MacOS and many Linux distributions as python is fairly portable and compatable with all of those languages. Though its worth noting that the program will only be compiled to work on a windows pc as thats the operating system used by my client.

2.4 Program Structure

2.4.1 Top-down design structure charts

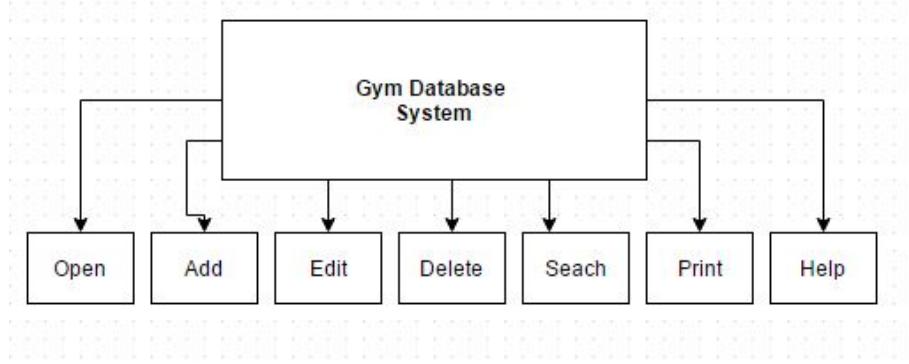


Figure 2.19: Structure Chart

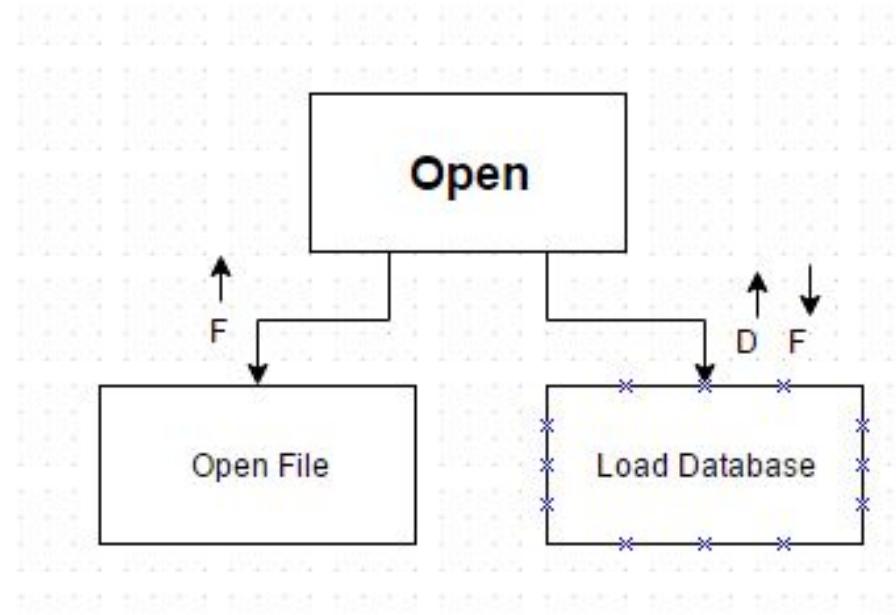


Figure 2.20: Structure Chart

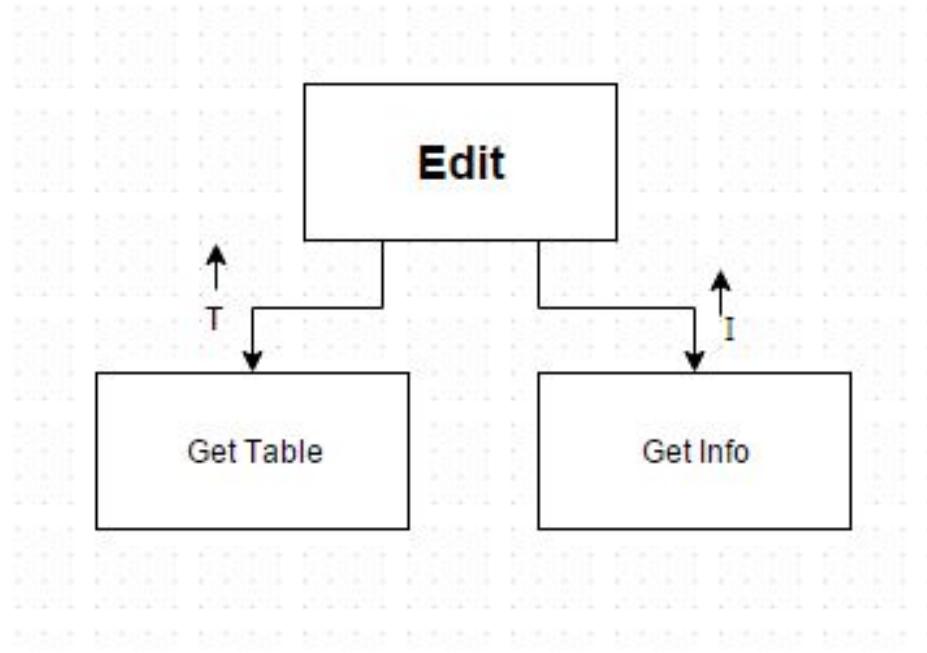


Figure 2.21: Structure Chart

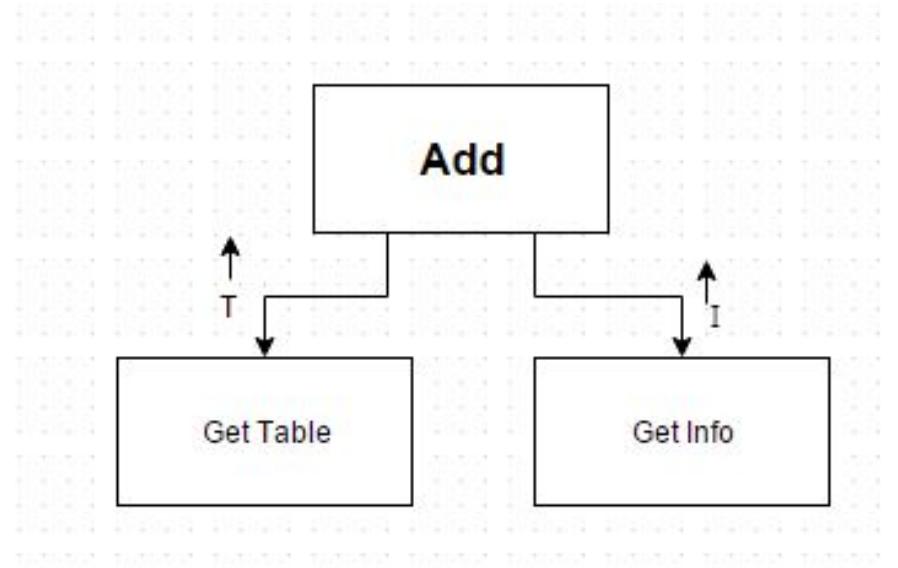


Figure 2.22: Structure Chart

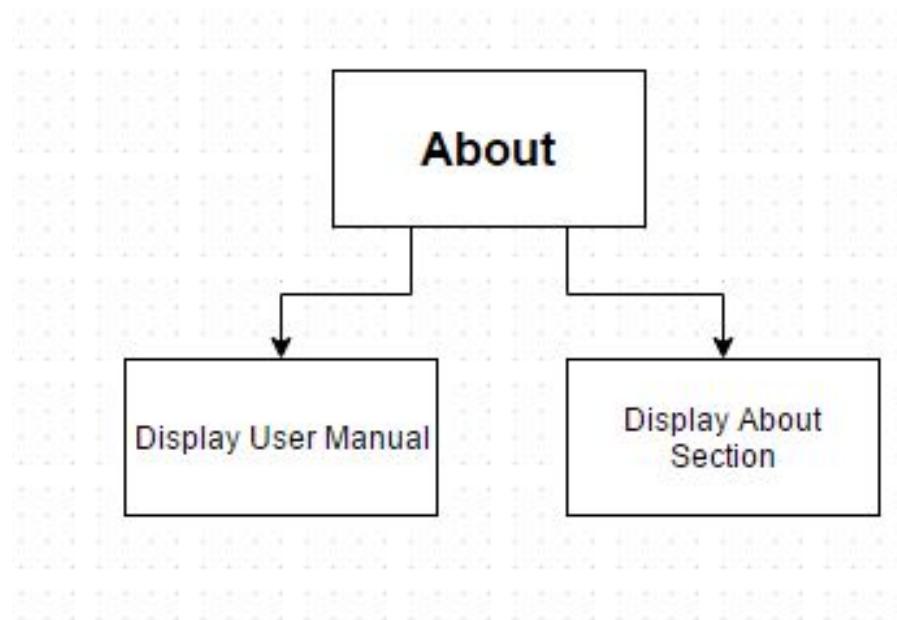


Figure 2.23: Structure Chart

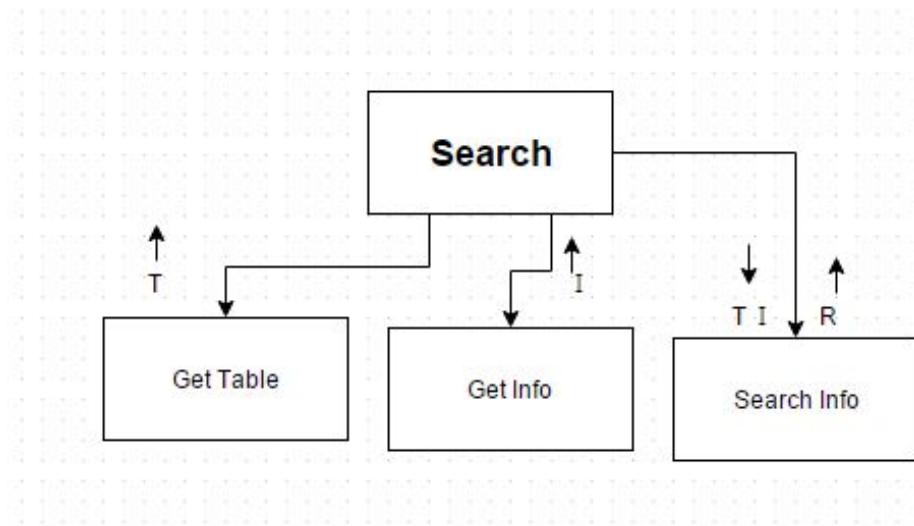


Figure 2.24: Structure Chart

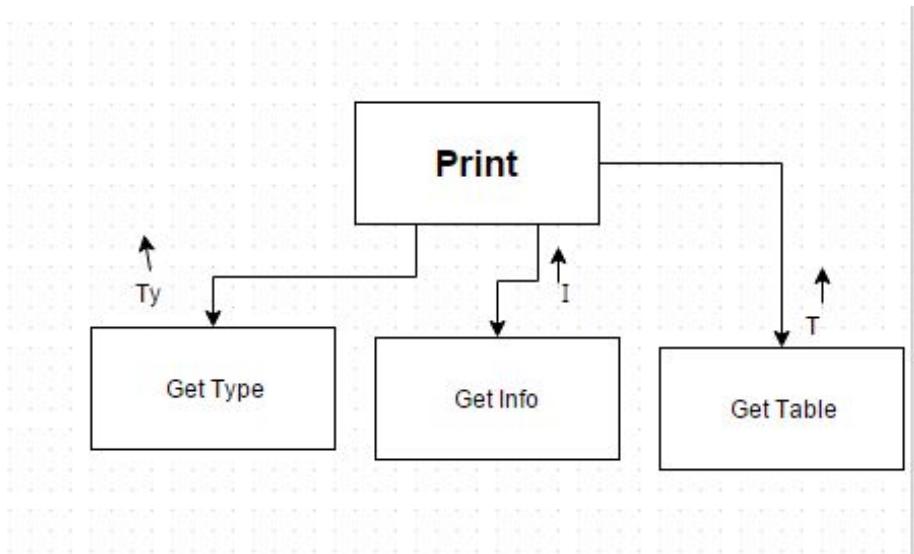


Figure 2.25: Structure Chart

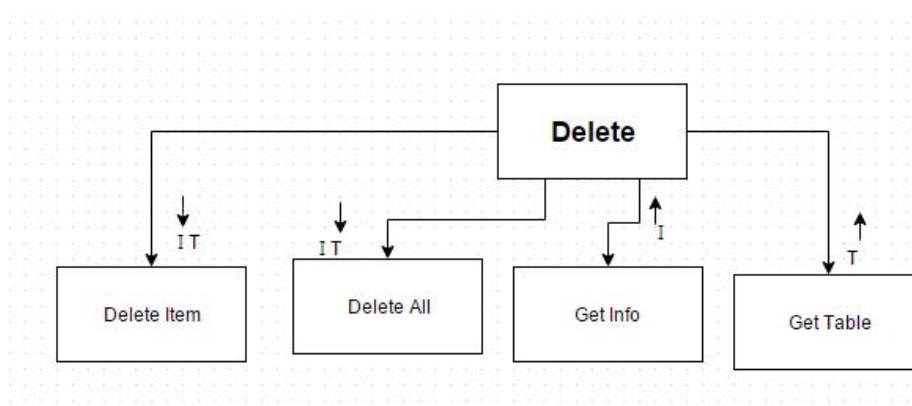


Figure 2.26: Structure Chart

2.4.2 Algorithms in pseudo-code for each data transformation process

```

1 START
2
3
4 Function FetchTotalForInvoice (database, person)
5     sql connect database as db
6     db.cursor <-- """Select How Much Where MemberID
7         = "105" Payments"""
8     data <-- cursor.execute
9     total <-- 0
10    for amount in data:
11        total <-- total + data
12    return total
13 END
14
15
16 Function GenerateMemberID(database)
17     sql connect database as db
18     db.cursor <-- """Select Count (Distinct
19         MemberID) From Members"""
20     HighestPlace <-- cursor.execute
21     MemberID <--- HighestPlace +1
22     return MemberID
23
24 END

```

```

25 START
26
27 Function GenerateExerciseID(database)
28     sql connect database as db
29     db.cursor <--- """Select Count (Distinct
30         ExerciseID) From Exercises"""
31     HighestPlace <--- cursor.execute
32     ExerciseID <--- HighestPlace +1
33     return ExerciseID
34
35 END

```

All the other instances of data transformation in my program are accurately represented in the SQL Queries section of this coursework as my other data transformation processes are all done primarily in SQL and don't require any pseudocode.

2.4.3 Object Diagrams

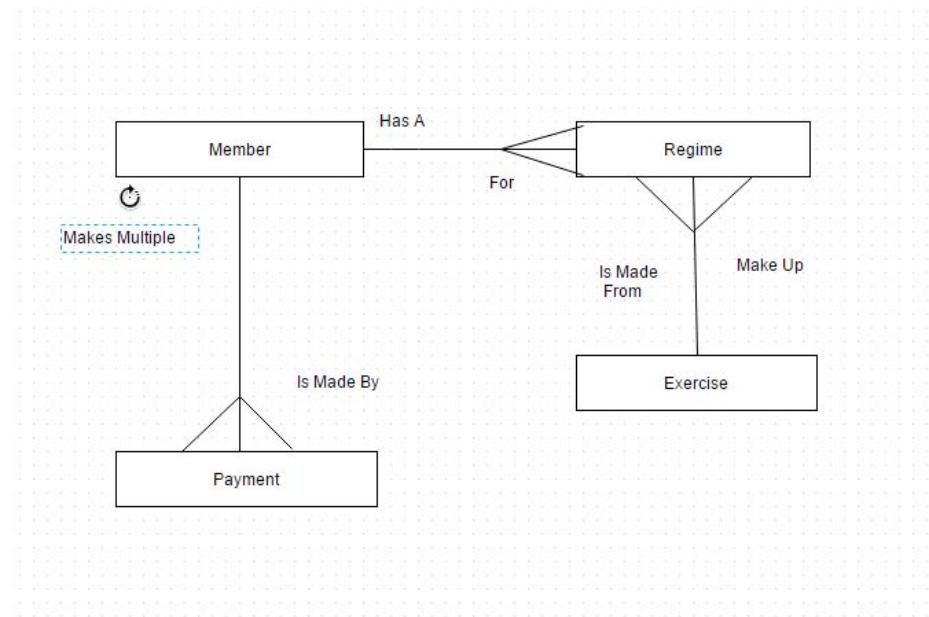


Figure 2.27: Relationship Diagram

2.4.4 Class Definitions

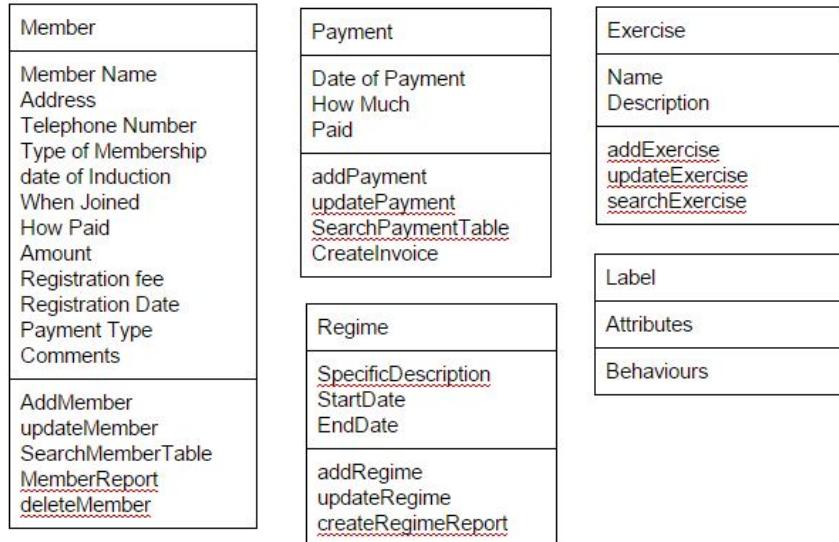


Figure 2.28: Class Definitions

2.5 Prototyping

For the prototype I created various functional GUI's to test how my program will look, feel and operate for my client and serveral commandline python functions to test some of the functionality. This allowed me to make sure the program was coming along to my clients expectations.

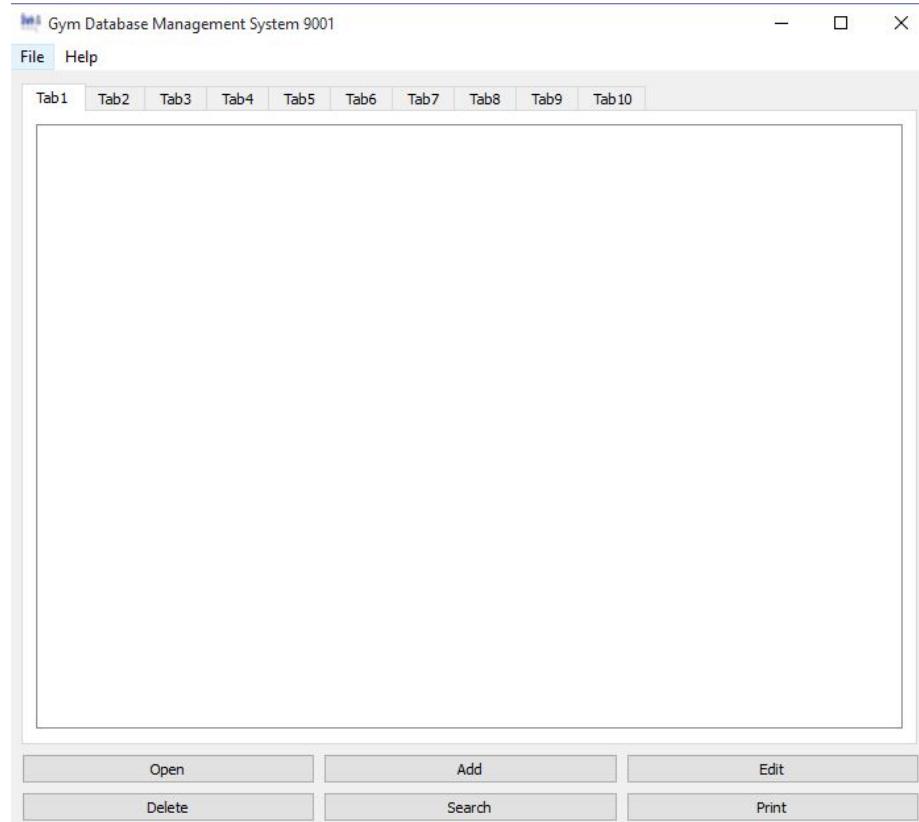
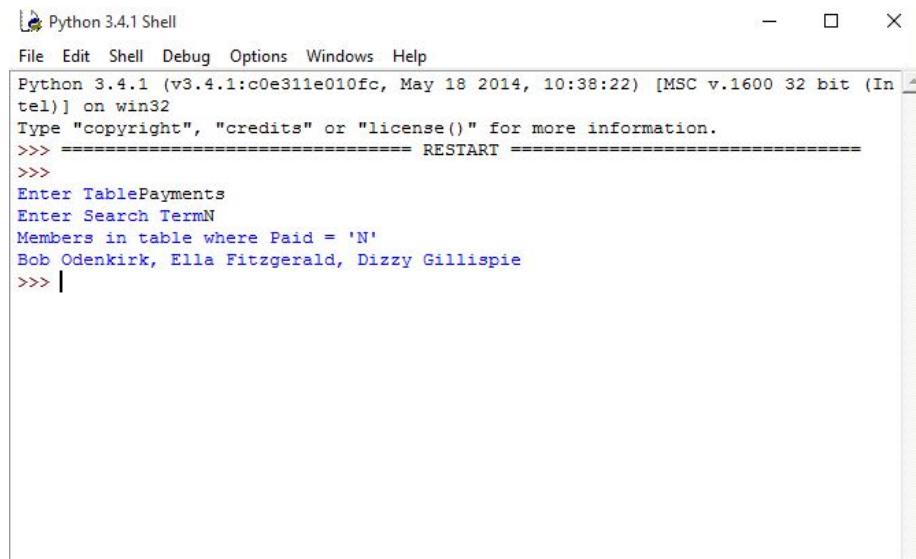


Figure 2.29: Prototype Main Gui Window

This shows the window for my main Gui. The blank white space is where the database will go once the program is complete, with each tab representing a different table. My client used this and seemed pleased with the potential of the programm but felt like he needed some functionality to get an idea of how the program will work.



The screenshot shows a Windows-style application window titled "Python 3.4.1 Shell". The menu bar includes File, Edit, Shell, Debug, Options, Windows, and Help. The main window displays a Python shell session:

```
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 10:38:22) [MSC v.1600 32 bit (In tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Enter TablePayments
Enter Search TermN
Members in table where Paid = 'N'
Bob Odenkirk, Ella Fitzgerald, Dizzy Gillispie
>>> |
```

Figure 2.30: Prototype Search Commandline Window

This shows a commandline interface for a search function I made to show the potential of the functionality of the program for my client. He seemed impressed by the potential problems this search function could solve for when he needs to find any specific information quickly like how in this example he could find out everyone who had outstanding payments for membership.

2.6 Definition of Data Requirements

2.6.1 Identification of all data input items

- Member Name
- Member Address
- Member Telephone Number
- Type of Membership
- Join Date
- Date of Induction
- How Paid
- Amount Paid

- Registration Fee
- Registration Date
- Payment Type
- Date of Payment
- How Much
- Paid?
- Exercise Name
- Exercise Description
- Specific Description
- Start Date
- End Date

2.6.2 Identification of all data output items

- Member Details Printout
- Invoice Printout
- Memberlist Printout
- Member Regime

Output to Database

- Member Name
- Member Address
- Member Telephone Number
- Type of Membership
- Join Date
- Date of Induction
- How Paid
- Amount Paid
- Registration Fee
- Registration Date
- Payment Type
- Date of Payment

- How Much
- Paid?
- Exercise Name
- Exercise Description
- Specific Description
- Start Date
- End Date

2.6.3 Explanation of how data output items are generated

Output	How the Output is Generated
Member Name	Input by the user
Member Address	Input by the user
Member Telephone Number	Input by the user
Type of Membership	Input by the user
Join Date	Input by the user
Date of Induction	Input by the user
How Paid	Input by the user
Amount Paid	Input by the user
Registration Fee	Input by the user
Registration Date	Input by the user

Figure 2.31: Output Explanations

Payment Type	Input by the user
Date of Payment	Input by the user
How Much	Input by the user
Paid?	Input by the user
Exercise Name	Input by the user
Exercise Description	Input by the user
Specific Description	Input by the user
Start Date	Input by the user
End Date	Input by the user
Member Details Printout	Fetched from the Member and Payment Details Database
Invoice Printout	Fetched from the Member and Payment Details Database
Memberlist Printout	Fetched from the Member Details Database
Member Regime	Fetched from the Regime Database

Figure 2.32: Output Explanations

2.6.4 Data Dictionary

Entity	Attribute	Data Type	Length	Constrain	Description
Member	Member No	Int	(10)	Primary Key	The unique membership ID
Member	Name	String	(30)	Not Null	The name of the member
Member	Address	String	(30)	Not Null	The members address
Member	Telephone Number	String	(10)	Not Null	The members preferred contact number
Member	Type of Membership	String	(10)	Not Null	The type of membership
Member	Join Date	Datetime	(8)	Not Null	The date the member joined
Member	Date Of Induction	DateTime	(8)	Not Null	The date of the members induction
Member	How Paid	String	(10)	Not Null	How the member payed for their membership
Member	Amount	int	(2)	Not Null	How much the member payed for

Figure 2.33: Data Dictionary

					their membership
Member	Registration Fee	int	(2)	Not Null	How much does the member have to pay for registration
Member	Registration Date	Date/Time	(10)	Not Null	The date of registration
Payment Details	Member No	Int	(10)	Primary Key	The unique membership ID
Payment Details	Payment Type	String	(22)	Not Null	How the client pays
Payment Details	Date Of Payment	Datetime	(8)	Not Null	The Date the payment is due
Payment Details	How Much	Int	(2)	Not Null	How much the client needs to pay
Payment Details	Paid	Boolean	(1)	Not Null	Whether the client has paid
Regime	Member No	Int	(10)	Primary Key	The unique membership ID
Regime	ExerciseID	int	(10)	Not Null	The unique exercise identifier
Regime	Specific Description	String	(50)	Not Null	More specific description of the exercise

Figure 2.34: Data Dictionary 2

					tailored for the member
Regime	Start Date	Datetime	(8)	Not Null	The date the member started the regime
Regime	End Date	Datetime	(8)	Not Null	The date the member finished doing the regime
Exercise	ExerciseID	int	(10)	Primary Key	The unique exercise identifier
Exercise	Name	String	(20)	Not Null	The name of the exercise
Exercise	Description	String	(50)	Not Null	Description of the exercise

Figure 2.35: Data Dictionary 3

2.6.5 Identification of appropriate storage media

The system only needs to be accessed by the one workstation in my clients gym meaning that the database files can be stored locally as he won't need to access the files on any other system. Although in saying that the files will be backed up in the gym owners dropbox account so that if the system the databases are stored on breaks then the most recent version of the files will be retrievable and then the files can be accessed on a different system in case of an emergency, adding a certain level of security.

2.7 Database Design

2.7.1 Normalisation

ER Diagrams

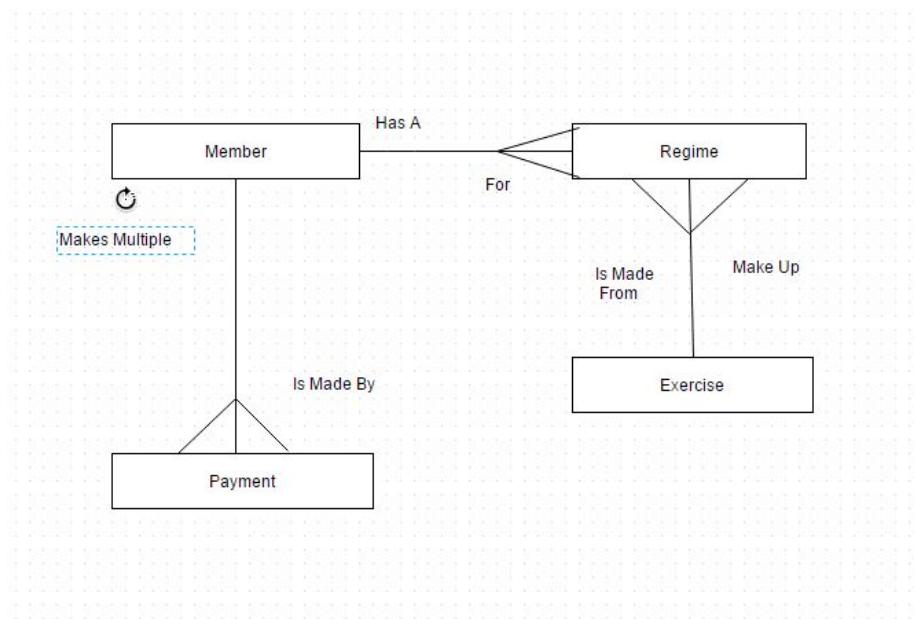


Figure 2.36: ER Diagram

Entity Descriptions

Membership Details(Membership No., Last Name, First Name, Address, Telephone No., Type Of Membership, Date of Induction, When Joined, How Paid, amount, registration fee, Registration Date, Payment Type, Comments)

Payment Details(Membership No.,Date of Payment, How Much, Paid)

Regime(Membership No.,ExerciseID, Specific Description, Start Date, End Date)

Exercise(ExerciseID, Name, Description)

1NF to 3NF

Un-Normalised Form	1NF	3NF																
MemberID Name Address Telephone No Type of Membership Date of Induction When Joined How Paid Amount Registration Fee Registration Date Payment Type Comments Date of Payment How Much Paid Exercise ID Name Description Specific Description StartDate EndDate	<table border="1"> <tr> <td>Repeating</td><td>Not Repeating</td></tr> <tr> <td>ExerciseID Name Description MemberID StartDate EndDate Specific Description</td><td>MemberID Name Address Telephone No Type of Membership Date of Induction When Joined How Paid Amount Registration Fee Registration Date Payment Type Comments Date of Payment How Much Paid</td></tr> </table>	Repeating	Not Repeating	ExerciseID Name Description MemberID StartDate EndDate Specific Description	MemberID Name Address Telephone No Type of Membership Date of Induction When Joined How Paid Amount Registration Fee Registration Date Payment Type Comments Date of Payment How Much Paid	<table border="1"> <tr> <td>Member</td><td>Payment</td></tr> <tr> <td>MemberID Name Address Telephone No Type of Membership Date of Induction When Joined How Paid Amount Registration Fee Registration Date Payment Type Comments</td><td>Member ID Date of Payment How Much Paid</td></tr> <tr> <td>Regime</td><td></td></tr> <tr> <td>MemberID ExerciseID Specific Description StartDate EndDate</td><td>MemberID ExerciseID Specific Description StartDate EndDate</td></tr> <tr> <td>Exercise</td><td></td></tr> <tr> <td>ExerciseID Name Description</td><td>ExerciseID Name Description</td></tr> </table>	Member	Payment	MemberID Name Address Telephone No Type of Membership Date of Induction When Joined How Paid Amount Registration Fee Registration Date Payment Type Comments	Member ID Date of Payment How Much Paid	Regime		MemberID ExerciseID Specific Description StartDate EndDate	MemberID ExerciseID Specific Description StartDate EndDate	Exercise		ExerciseID Name Description	ExerciseID Name Description
Repeating	Not Repeating																	
ExerciseID Name Description MemberID StartDate EndDate Specific Description	MemberID Name Address Telephone No Type of Membership Date of Induction When Joined How Paid Amount Registration Fee Registration Date Payment Type Comments Date of Payment How Much Paid																	
Member	Payment																	
MemberID Name Address Telephone No Type of Membership Date of Induction When Joined How Paid Amount Registration Fee Registration Date Payment Type Comments	Member ID Date of Payment How Much Paid																	
Regime																		
MemberID ExerciseID Specific Description StartDate EndDate	MemberID ExerciseID Specific Description StartDate EndDate																	
Exercise																		
ExerciseID Name Description	ExerciseID Name Description																	
2NF																		
Repeating	Not Repeating																	
Exercise ID Name Description MemberID ExerciseID StartDate EndDate Specific Description	<table border="1"> <tr> <td>MemberID Name Address Telephone No Type of Membership Date of Induction When Joined How Paid Amount Registration Fee Registration Date Payment Type Comments Date of Payment How Much Paid</td></tr> </table>	MemberID Name Address Telephone No Type of Membership Date of Induction When Joined How Paid Amount Registration Fee Registration Date Payment Type Comments Date of Payment How Much Paid																
MemberID Name Address Telephone No Type of Membership Date of Induction When Joined How Paid Amount Registration Fee Registration Date Payment Type Comments Date of Payment How Much Paid																		

2.7.2 SQL Queries

SQL QUERY	DESCRIPTION
<pre>"""insert into Members(MemberID,MemberName,Address Telephone No,Type of Membership,Date of Induction,When Joined,How Paid,Amount,Registration Fee,Registration Date,Payment Type,Comments) values ('{0}', '{1}', '{2}', '{3}', '{4}', '{5}', '{6}', '{7}', '{8}', '{9}', '{10}', '{11}', '{12}') """.format(MemberID,MemberName,Address Telephone No,Type of Membership,Date of Induction,When Joined,How Paid,Amount,Registration Fee,Registration Date,Payment Type,Comments)</pre>	<p>Add a new member to the Members Table. The information is acquired from an input form made in the gui. This input form contains fields for the user to enter the input that is then passed into this query under the variable names MemberName,Address Telephone No,Type of Membership,Date of Induction,When Joined,How Paid,Amount,Registration Fee,Registration Date,Payment Type,Comments and one variable named MemberID that is autogenerated by the program.</p>
<pre>"""insert into Payments(MemberID, Date of Payment, How Much, Paid) values ('{0}', '{1}', '{2}', '{3}') """.format(MemberID, Date of Payment, How Much, Paid)</pre>	<p>Add a new payment to the Payments Table. The information is acquired from an input form made in the gui. This input form contains fields for the user to enter the input that is then passed into this query under the variable names MemberID, Date of Payment, How Much, Paid.</p>
<pre>"""insert into Exercises(ExerciseID, Name, Description) values ('{0}', '{1}', '{2}') """.format(ExerciseID, Name, Description)</pre>	<p>Add a new exercise to the Exercise Table. The information is acquired from an input form made in the gui. This input form contains fields for the user to enter the input that is then passed into this query under the variable names Name, Description and one variable named ExerciseID that is autogenerated by the program.</p>
<pre>"""insert into Regimes(MemberID,ExerciseID,Specific Description,StartDate,EndDate) values ('{0}', '{1}', '{2}', '{3}', '{4}') """.format(MemberID,ExerciseID,Specific Description,StartDate,EndDate)</pre>	<p>Add a new regime to the Regime Table. The information is acquired from an input form made in the gui. This input form contains fields for the user to enter the input that is then passed into this query under the variable names MemberID,ExerciseID,Specific Description,StartDate,EndDate.</p>
<pre>"""updateMembers set Name = '{0}'</pre>	<p>Edit the details of a member in the Members Table. The information is</p>

Figure 2.38: SQL Queries

<pre> Address ='{1}' Telephone No ='{2}' Type of Membership ='{3}' Date of Induction ='{4}' When Joined ='{5}' How Paid ='{6}' Amount ='{7}' Registration Fee ='{8}' Registration Date ='{9}' Payment Type ='{10}' Comments ='{11}' WHERE MemberID ='{12}' """.format((MemberName,Address Telephone No,Type of Membership,Date of Induction,When Joined,How Paid,Amount,Registration Fee,Registration Date,Payment Type,Comments,MemberID) </pre>	<p>acquired from an input form made in the gui. This input form contains fields for the user to enter the input that is then passed into this query under the variable names MemberName,Address Telephone No,Type of Membership,Date of Induction,When Joined,How Paid,Amount,Registration Fee,Registration Date,Payment Type,Comments. The forms also contains a combo box which displays all the members names, accompanied by their memberID's to allow the user to select which member they want to edit the information of.</p>
<pre> """update Payments set Paid = '{0}' How Much ='{1}' WHERE MemberID ='{2}' and Date of Payment ='{3}' """.format((MemberID, Date of Payment, How Much, Paid) </pre>	<p>Edit the details of a payment in the Payments Table. The information is acquired from an input form made in the gui. This input form contains fields for the user to enter the input that is then passed into this query under the variable names How Much and Paid. The forms also contains 2 combo boxes which displays all the members names, accompanied by their memberID's and their payments organised by Date of Payment to allow the user to select which member and month they want to edit the information of.</p>
<pre> Name ='{0}' Description ='{1}' WHERE ExerciseID ='{2}' """.format(Name,Description,ExerciseID) </pre>	<p>Edit the details of an exercise in the Exercise Table. The information is acquired from an input form made in the gui. This input form contains fields for the user to enter the input that is then passed into this query under the variable names Name and Description. The forms also contains a combo box which displays all the Exercises names, accompanied by their exerciseID's to allow the user to select which exercise they want to edit the information of.</p>

Figure 2.39: SQL Queries

<pre>"""update Regime set Specific Description = '{0}' Start Date = '{1}' End Date = '{2}' WHERE MemberID = '{3}' and ExerciseID = '{3}''''.format((Specific Description, Start Date, End Date, MemberID, ExerciseID)</pre>	<p>Edit the details of a regime in the Regime Table. The information is acquired from an input form made in the gui. This input form contains fields for the user to enter the input that is then passed into this query under the variable names Specific Description, Start Date and End Date. The forms also contains 2 combo boxes which displays all the members names, accompanied by their memberID's and their exercises organised by their ExerciseID to allow the user to select which member and regime they want to edit the information of.</p>
<pre>"""delete from '{0}' where '{1}' = '{2}' ''''.format(Members,ColumnMemberID,MemberID)</pre>	<p>Delete an item in a table. The information is acquired from an input form made in the gui. The input form contains 3 comboboxes allowing the user to select the table, the column, and the row.</p>
<pre>"""select from Members where MemberID = '{0}' ''''.format(MemberID)</pre>	<p>Search for a specific item in a table. The information is acquired from an input form made in the gui. The input form contains a combobox allowing the user to select the table and a lineEdit allowing the user to enter the search term. This data is then passed into this query.</p>
<pre>"""select Name from Members""""</pre>	<p>Fetches the information for a form, in this case a list of all the members in the Member Table. The kind of form and data passed depends on the information input by the user using the print interface, which has several comboboxes allowing the user to select the kind of form and the data used.</p>

Figure 2.40: SQL Queries

2.8 Security and Integrity of the System and Data

2.8.1 Security and Integrity of Data

The system will have data entered through combo boxes/drop down menus where possible to avoid the user entering bad data but for a large amount of the fields (like a members name) has to be a raw input typed by the user with a keyboard. All the data input will be checked for errors. e.g making sure the data type is correct - strings can't be entered where an integer is required.

2.8.2 System Security

The system will be protected behind a password that is only known by certain users/members of staff at the gym and a secondary password will be required to use certain functions like the function to delete any of the data or edit anything important.

2.9 Validation

Item	Example	Validation/Verification	Comments
Password	password2	Lookup Check	Makes sure that they password entered is the correct one that's been preset by the user
Name	Cab Calloway	Presence Check	Makes sure somethings entered
Address	2, Fake Street	Presence Check	Makes sure somethings entered
Telephone No	01353 669878	Presence Check	Makes sure somethings entered
Date of Induction	11/05/2015	Presence Check Type Check	Makes sure somethings entered and it's in a date time format
When Joined	14/05/2015	Presence Check Type Check	Makes sure somethings entered and it's in a date time format
Amount	50	Presence Check Type Check	Makes sure somethings entered and it's an integer

Figure 2.41: Validation

Registration Fee	50	Presence Check Type Check	Makes sure somethings entered and it's an integer
Registration Date	12/05/2015	Presence Check Type Check	Makes sure somethings entered and it's in a date time format
Comments	Has Asthma	Presence Check	Makes sure somethings entered
Date of Payment	12/05/2015	Presence Check Type Check	Makes sure somethings entered and it's in a date time

Figure 2.42: Validation

			format
How Much	50	Presence Check Type Check	Makes sure somethings entered and it's an integer
Specific Description	10 Push Ups	Presence Check	Makes sure somethings entered
StartDate	12/06/2015	Presence Check Type Check	Makes sure somethings entered and it's in a date time format
EndDate	12/09/2015	Presence Check Type Check	Makes sure somethings entered and it's in a date time format
Name	Push Ups	Presence Check	Makes sure somethings entered
Description	Vertical Lifts .etc....	Presence Check	Makes sure somethings entered

Figure 2.43: Validation

2.10 Testing

Outline Plan

Test Series	Purpose of Test Series	Testing Strategy	Strategy Rationale
1	Test the flow and navigation between the different menus and interfaces included in the GUI.	Top-Down Testing	
2	Test the validation methods for the data input by the user	Bottom-Up Testing	Each component will be tested after development.
3	Test that all the input data is stored in the correct place	Black Box Testing	
4	Test algorithms to make sure the output is correct	White Box Testing	
5	Test that the system meets the specification.	Acceptance Testing	

Figure 2.44: Outline Plan

Detailed Plan

Test Series and Number	Purpose	Description	Test Data	Test Data Type	Expected Result	Actual Result	Evidence in Appendix
1.1	Test that the password enter dialog responds correctly to the wrong password	The password dialog should reopen prompting the user to try again	" " - Leave the password lineEdit blank and then click the enter button	Erroneous	The password dialog will reopen		
1.2	Test that the password responds correctly to the correct password	The password dialog should close opening the main interface	"password" - Enter the correct password and then click the enter button	Normal	The main interface will open after the password dialog closes		

Figure 2.45: Detailed Plan

1.3	Test that the open button works correctly	The windows file explorer should open to allow the user to select the right database file	Click the open button	Normal	The windows file explorer should open while the main interface remains open		
1.4	Test that the open shortcut in the toolbar	The windows file explorer should	Click the open button	Normal	The windows file explorer should		

Figure 2.46: Detailed Plan

	works correctly	open to allow the user to select the right database file			open while the main interface remains open		
1.5	Test that the table view in the main interface changes to accommodate the opened database	The table view should change to include data from the database selected with each table appearing in a different table	Open a database	Normal	The table view will now contain information stored in the database file		

Figure 2.47: Detailed Plan

1.6	Test that the Add button works correctly	The Add dialog box should open allowing the user to select and input the data they want to add to the database	Click the add button	Normal	The Add dialog box should open while the main interface remains open		
1.7	Test that the Add shortcut in the toolbar works correctly	The Add dialog box should open allowing the user to select	Click the add button	Normal	The Add dialog box should open while the main interface		

Figure 2.48: Detailed Plan

		and input the data they want to add to the database			remains open		
1.8	Test that the select table combo box works correctly in the Add dialog.	The section in the layout should fill with a form with input options based on the table selected	Change the select table combo box option to all <u>avaibale</u> tables	Normal	The input form should change to contain information and input options based on the selected table		
1.9	Test that the Add dialog responds correctly after the Add button has been pushed	The dialog box should close returning the <u>users</u> focus to the main interface	Enter all appropriate data and hit the Add button	Normal	The dialog box should close with the main interface still open		

Figure 2.49: Detailed Plan

1.10	Test that the Edit button works correctly	The Edit dialog box should open allowing the user to select and input the data they want to edit in the database	Click the edit button	Normal	The Edit dialog box should open while the main interface remains open		
------	---	--	-----------------------	--------	---	--	--

Figure 2.50: Detailed Plan

1.11	Test that the Edit shortcut in the toolbar works correctly	The Edit dialog box should open allowing the user to select and input the data they want to edit in the database	Click the edit button	Normal	The Edit dialog box should open while the main interface remains open		
1.12	Test that the select table combo box works correctly in the Edit dialog.	The section in the layout should fill with a form with input options based on the table selected	Change the select table combo box option to all <u>avaibale</u> tables	Normal	The input form should change to contain information and input options based on the selected table		

Figure 2.51: Detailed Plan

1.13	Test that the Edit dialog responds correctly after the Edit button has been pushed	The dialog box should close returning the <u>users</u> focus to the main interface	Enter all appropriate data and hit the Edit button	Normal	The dialog box should close with the main interface still open		
1.14	Test that the Delete button works correctly	The Delete dialog box should open allowing	Click the Delete button	Normal	The Delete dialog box should open while the		

Figure 2.52: Detailed Plan

		the user to select the information they want to delete while the main interface remains open			main interface is still open		
1.15	Test that the delete shortcut in the toolbar works correctly	The Delete dialog box should open allowing the user to select the information they want to delete while the main interface remains open	Click the Delete shortcut	Normal	The Delete dialog box should open while the main interface is still open		

Figure 2.53: Detailed Plan

1.16	Test that the Select Table combobox in the delete dialog works correctly	The Select Item combobox should then include different options for the user to select	Select all possible options from the Select Table combo box	Normal	The Select Item combo box should fill with data based off of the selected table		
1.17	Test that	The	Click the	Normal	The		

Figure 2.54: Detailed Plan

	the Delete dialog box responds correctly to the Delete or Delete All button being clicked	dialog box should close bringing the Main Interface into the users focus	Delete and Delete All Buttons		Delete Dialog should close leaving just the Main Interface Open		
1.18	Test that the Search Button works correctly	The search Dialog should open allowing the user to enter the information they wish to search for	Click on the Search button	Normal	The Search dialog should open while the Main Interface stays open in the background		

Figure 2.55: Detailed Plan

1.19	Test that the Search shortcut in the toolbar works correctly	The search Dialog should open allowing the user to enter the information they wish to search for	Click on the Search Shortcut	Normal	The Search dialog should open while the Main Interface stays open in the background		
1.20	Test that the Search button in	This should display the	Enter all appropriate information	Normal	A new Results Dialog should		

Figure 2.56: Detailed Plan

		the Search dialog works correctly	results of the search for the user	on and click the Search button		open closing the Search dialog but keep the Main Interface open in the background		
1.21	Test that the Close button in the result dialog functions correctly	The Results dialog should close drawing the user's focus to the Main Interface again	Click on the Close Button	Normal	The Results dialog should close while the Main Interface window is brought back to the user's attention			

Figure 2.57: Detailed Plan

1.22	Test the Print Button works correctly	The Print Dialog should open so that the user can select the form and information they want to print	Click on the Print Button	Normal	The Print Dialog should open leaving the Main Interface open in the Background		
1.23	Test that the comboboxes in the print function work	The comboboxes should change the information	Select all the different options in all 3 comboboxes	Normal	The Comboboxes should have different options		

Figure 2.58: Detailed Plan

	correctly	on in the subsequent combobox when an item is selected in one of them			based on what's entered in the other comboboxes		
1.24	Test that the Print button in the Print Dialog works correctly	The Print dialog should close bringing the Main Interface back into the user's focus	Click on the Print Button	Normal	The Print Dialog should close leaving only the Main Interface Open		

Figure 2.59: Detailed Plan

1.25	Test that the User Manual Option in the toolbar works	A digital version of the User Manual should open in another window so that the user can easily identify how to perform an operation within the program	Click on the User Manual Shortcut	Normal	The User Manual should open up in a dialog box leaving the Main Interface Open		
1.26	Test that the About option in the toolbar works	The about section should open so that the	Click on the About option	Normal	A Dialog box containing information about		

Figure 2.60: Detailed Plan

		user can identified the programs current version, creator and any other necessary information			the program should be opened while the Main Interface remains open in the program		
--	--	---	--	--	---	--	--

Figure 2.61: Detailed Plan

2.1	Verify that the password entered is correct	The password will match the one stored inside the program	The correct password	Normal	The password will be correct and the program will carry on as so		
			The incorrect password	Erroneous	The program will not continue until the correct password is entered and the user will be prompted as such		
2.2	Verify that the file opened is a database (.db) file	Opening the wrong type of file should result in an error	A Database File A Word	Normal Erroneous	The program will continue and load the database The		

Figure 2.62: Detailed Plan

			File	s	program will prompt the user with an error		
2.3	Verify that the information entered in the Add input form is of the correct type	If the data is not the correct type the user should receive an error message	Appropriate Data Incorrect data type(s)	Normal Erroneous	The program will continue with the data The user will receive an error message and be prompted to change the necessary data		

Figure 2.63: Detailed Plan

2.4	Verify that the information entered in the Edit input form is of the correct type	If the data is not the correct type the user should receive an error message	Appropriate Data Incorrect data type(s)	Normal Erroneous	The program will continue with the data The user will receive an error message and be prompted to change the necessary data		
-----	---	--	--	---------------------	--	--	--

Figure 2.64: Detailed Plan

2.5	Verify that the search term entered in the Search Dialog actually yields results	If the search comes up without results the user should receive a message saying so	A correct search term An incorrect search term	Normal Erroneous	The program continues on with the search results The user receives a message and is prompted to use a different search term		
3.1	Verify that all Member details that are input are added to the database	All the details should be added to the correct place in the database	Member information	Normal	The data should be added into the appropriate place in the database		

Figure 2.65: Detailed Plan

3.2	Verify that all Payment details that are input are added to the database	All the details should be added to the correct place in the database	Payment information	Normal	The data should be added into the appropriate place in the database		
3.3	Verify that all Exercise details that are input are added to the	All the details should be added to the correct place in the	Exercise information	Normal	The data should be added into the appropriate place in the database		

Figure 2.66: Detailed Plan

	database	database					
3.4	Verify that all Regime details that are input are added to the database	All the details should be added to the correct place in the database	Regime information	Normal	The data should be added into the appropriate place in the database		
3.5	Verify that all Member details that are input are updated in the database	All the correct details should be updated in the database	Member information	Normal	The data should be edited into the appropriate place in the database		
3.6	Verify that all Payment details that are input are updated in the database	All the correct details should be updated in the database	Payment information	Normal	The data should be edited into the appropriate place in the database		

Figure 2.67: Detailed Plan

	database	database					
3.7	Verify that all Exercise details that are input are updated in the database	All the correct details should be updated in the database	Exercise information	Normal	The data should be edited into the appropriate place in the database		
3.8	Verify that all Regime details that are	All the correct details should be updated	Regime information	Normal	The data should be edited into the appropriate		

Figure 2.68: Detailed Plan

	input are updated in the database	in the database			te place in the database		
3.9	Verify that all Member details that are input are deleted in the database	All the correct details should be deleted from the database	Member information	Normal	The data should be deleted from the appropriate place in the database		
3.10	Verify that all Payment details that are input are deleted in the database	All the correct details should be deleted from the database	Payment information	Normal	The data should be deleted from the appropriate place in the database		

Figure 2.69: Detailed Plan

3.11	Verify that all Exercise details that are input are deleted in the database	All the correct details should be deleted from the database	Exercise information	Normal	The data should be deleted from the appropriate place in the database		
3.12	Verify that all Regime details that are input are deleted in the database	All the correct details should be deleted from the database	Regime information	Normal	The data should be deleted from the appropriate place in the database		
4.1	Verify that the search	The search	Search some	Normal	The search		

Figure 2.70: Detailed Plan

	Search function works correctly	function should allow the user to input a search term and then query that database and return the results	information that is known to be in the database		function should return the correct information		
4.2	Test that the Invoice function correctly calculates the total money a client needs to pay	The invoice function should add together the sums of all the unpaid months of a member	A member that the user already knows the total money owed of	Normal	The function should print the correct total along with other information		

Figure 2.71: Detailed Plan

4.3	Test that the Print function fetches the correct data	The Print function fetches data from the database to print in the forms	Any table and form that the user knows the output of	Normal	The function should retrieve the correct data		
5	Make sure that the program fulfills the specification laid out in the analysis and	Run through the program making sure every function and aspect	Add some information to the database and run through all the different methods	Normal	Program fulfills the specification		

Figure 2.72: Detailed Plan

	design	meets this specification	of data manipulation and view the results				
--	--------	--------------------------	---	--	--	--	--

Figure 2.73: Detailed Plan

Implementation

Main

```
1. #Gymnasium Database Management System
2.
3. import sys
4.
5. from gym_delete_dialog_class import *
6. from gym_print_dialog_class import *
7. from gym_search_dialog_class import *
8. from gym_edit_dialog_class import *
9. from gym_add_edit_dialog_class import *
10. from gym_add_dialog_class import *
11. from gym_about_dialog_class import *
12. from gym_password_dialog_class import *
13.
14. from data_browser import *
15.
16. from PyQt4.QtCore import *
17. from PyQt4.QtGui import *
18.
19. class AppWindow(QMainWindow):
20.     """creates the main window"""
21.
22.     #constructor
23.     def __init__(self):
24.         super().__init__()
25.         self.setWindowTitle("Gym Database Management System 9001")#sets the title for the
window
26.         self.setWindowIcon(QIcon("Logo.png"))#sets the window icon
27.
28.         #variable for database
```

```

29.     self.file = None
30.
31.     #toolbars
32.     self.open_push_button = QPushButton("Open")#sets the main button for opening the Open
GUI and function
33.     self.add_push_button = QPushButton("Add")#sets the main button for opening the Add GUI
and function
34.     self.edit_push_button = QPushButton("Edit")#sets the main button for opening the Edit GUI
and function
35.     self.delete_push_button = QPushButton("Delete")#sets the main button for opening the
Delete GUI and function
36.     self.search_push_button = QPushButton("Search")#sets the main button for opening the
Search GUI and function
37.     self.print_push_button = QPushButton("Print")#sets the main button for opening the Print
GUI and function
38.
39.     self.tab_bar = QTabWidget() #creates the widget to display the tables in a tabular layout
40.
41.     self.tabs = {}#creates a dictionary for the table names/tab names
42.     self.tabNames = ['Members','Payments','Regime','Exercise']#The 4 tab/table names
43.     for count in range(4):
44.         self.tabs["{0}".format(self.tabNames[count])] = BrowseDataWidget()
45.
        self.tab_bar.addTab(self.tabs["{0}".format(self.tabNames[count])],"{0}".format(self.tabNames
[count]))
46.     #a for loop that creates a new tab and adds a "BrowseDataWidget" into each of them
47.
48.     self.labels = {"Members": ["MemberID", "Name", "Address", "Telephone
Number", "Membership Type", "Induction Date", "Join Date", "How Paid", "Amount",
"RegistrationFee", "Registration Date", "PaymentType", "Comments"],

49.         "Payments": ["MemberID", "Payment Date", "How Much", "Paid"],

50.         "Regime": ["MemberID", "ExerciseID", "Specific Description", "Start Date", "End
Date"],
```

```
51.         "Exercise": ["ExerciseID", "Name", "Description"]}
```

52. #labels for the table headers that are referenced later in the program

53.

```
54. self.toolBar = QMenuBar()#creates a menu bar
```

```
55. self.file_menu = self.toolBar.addMenu("File")#adds File option to the tool bar
```

```
56. self.help_menu = self.toolBar.addMenu("Help")#adds Help option to the tool bar
```

```
57. self.about = self.help_menu.addAction("About Gym Database Management System
```

9001")#adds options into the Help menu

```
58. self.open_shortcut = self.file_menu.addAction("Open")#adds Open shortcut to the File
```

menu

```
59. self.add_shortcut = self.file_menu.addAction("Add")#adds Add shortcut to the File menu
```

```
60. self.edit_shortcut = self.file_menu.addAction("Edit")#adds Edit shortcut to the File menu
```

```
61. self.delete_shortcut = self.file_menu.addAction("Delete")#adds Delete shortcut to the File
```

menu

```
62. self.search_shortcut = self.file_menu.addAction("Search")#adds Search shortcut to the File
```

menu

```
63. self.print_shortcut = self.file_menu.addAction("Print")#adds Print shortcut to the File menu
```

64.

65.

```
66. #layout
```

```
67. self.layout1 = QHBoxLayout()
```

```
68. self.layout2 = QHBoxLayout()
```

```
69. self.layout3 = QVBoxLayout()
```

```
70. self.layout1.addWidget(self.open_push_button)
```

```
71. self.layout1.addWidget(self.add_push_button)
```

```
72. self.layout1.addWidget(self.edit_push_button)
```

73.

```
74. self.layout2.addWidget(self.delete_push_button)
```

```
75. self.layout2.addWidget(self.search_push_button)
```

```
76. self.layout2.addWidget(self.print_push_button)
```

77.

```
78. self.layout3.addWidget(self.tab_bar)
```

```
79. self.layout3.setLayout(self.layout1)
```

```

80.    self.layout3.addLayout(self.layout2)

81.

82.    self.mainWidget = QWidget()
83.    self.setMenuWidget(self.toolBar)
84.    self.mainWidget.setLayout(self.layout3)
85.    self.setCentralWidget(self.mainWidget)

86.

87.    #connections
88.    #connections linking the main pushbuttons to their respective functions
89.    self.open_push_button.clicked.connect(self.open_file_menu)
90.    self.delete_push_button.clicked.connect(self.delete)
91.    self.print_push_button.clicked.connect(self.print_stuff)
92.    self.search_push_button.clicked.connect(self.search)
93.    self.edit_push_button.clicked.connect(self.edit)
94.    self.add_push_button.clicked.connect(self.add)
95.    self.about.triggered.connect(self.about_the_program)
96.    self.open_shortcut.triggered.connect(self.open_file_menu)
97.    self.add_shortcut.triggered.connect(self.add)
98.    self.edit_shortcut.triggered.connect(self.edit)
99.    self.delete_shortcut.triggered.connect(self.delete)
100.   self.search_shortcut.triggered.connect(self.search)
101.   self.print_shortcut.triggered.connect(self.print_stuff)
102.   #Keyboard Shortcuts for accessing the 6 main Functions
103.   self.connect(QShortcut(QKeySequence("Ctrl+o"), self), SIGNAL('activated()'),
104.                 self.open_file_menu)
105.   self.connect(QShortcut(QKeySequence("Ctrl+a"), self), SIGNAL('activated()'), self.add)
106.   self.connect(QShortcut(QKeySequence("Ctrl+e"), self), SIGNAL('activated()'), self.edit)
107.   self.connect(QShortcut(QKeySequence("Ctrl+d"), self), SIGNAL('activated()'),
108.                 self.delete)
109.   self.connect(QShortcut(QKeySequence("Ctrl+s"), self), SIGNAL('activated()'),
110.                 self.search)
111.   self.connect(QShortcut(QKeySequence("Ctrl+p"), self), SIGNAL('activated()'),
112.                 self.print_stuff)

```

```

109.         self.connect(QShortcut(QKeySequence("Ctrl+h"), self), SIGNAL('activated()'),
110.             self.about_the_program)
111.
112.
113.     def open_file_menu(self):
114.         self.file = QFileDialog.getOpenFileName(caption="Open Database", filter = "Database
file (*.db *.dat)")#opens the windows folder tree allowing the user to select the database they
want to open. File type restricted to .db or .dat files
115.         try: #Restricts the user to only opening correct database or a version of it
116.             for item in self.tabNames:
117.                 self.tabs["{0}".format(item)].UpdateTable(self.file)#loads selected database into
tabs
118.                 self.tabs["{0}".format(item)].PopulateTable(item, self.labels[item])#populates
the tabs with relevant information
119.             except NameError:
120.                 return
121.             except sqlite3.OperationalError:
122.                 return
123.
124.
125.     def delete(self):
126.         self.password("niel")#sets delete password to "niel" and opens the password function
127.         delete_dialog = DeleteDialog(self.file)#opens delete dialog
128.         delete_dialog.exec_()
129.
130.     def print_stuff(self):
131.         print_dialog = PrintDialog(self.file)#opens print dialog
132.         print_dialog.exec_()
133.
134.     def search(self):
135.         search_dialog = SearchDialog(self.file)#opens search dialog
136.         search_dialog.exec_()

```

```
137.  
138.     def edit(self):  
139.         edit_dialog = EditDialog(self.file)#opens edit dialog  
140.         edit_dialog.exec_()  
141.  
142.     def add(self):  
143.         add_dialog = AddDialog(self.file)#opens add dialog  
144.         add_dialog.exec_()  
145.  
146.     def about_the_program(self):  
147.         about_dialog = AboutDialog()#opens about dialog  
148.         about_dialog.exec_()  
149.  
150.     def password(self,currentPass):  
151.         passWord = ""#sets entered password to the wrong password  
152.         while passWord != currentPass:#while the correct password hasn't been entered  
153.             password_dialog = PasswordDialog()#opens password dialog  
154.             password_dialog.exec_()  
155.             passWord = password_dialog.close_method()  
156.  
157.  
158.     def main():  
159.         gym_program = QApplication(sys.argv)#creates application  
160.         gym_window = AppWindow()#creates Main Window  
161.         gym_window.resize(700,600)#Locks initial window size  
162.         password_dialog = gym_window.password("sweatGym9001_niel")  
163.         gym_window.show()  
164.         gym_window.raise_()  
165.         gym_program.exec_()  
166.  
167.  
168.     if __name__ == "__main__":  
169.         main()
```

AddEditMemberTableWidget

```
1. from PyQt4.QtGui import *
2. from gym_add_function import *
3. from gym_edit_function import *
4.
5. class AddEditMemberTableWidget(QWidget):
6.     """This class creates the widget for the members table in the add and edit dialog boxes"""
7.
8.     def __init__(self):
9.         super().__init__()
10.
11.     #create widgets
12.
13.     self.lineEdits = {}
14.
15.     for count in range(13):
16.         self.lineEdits["self.enter_text {}".format(count)] = QLineEdit("")#creates 13
QLineEdits, 1 for each column in the table
17.         self.memberID_Label = QLabel("MemberID")
18.         self.name_Label = QLabel("Name")
19.         self.address_Label = QLabel("Address")
20.         self.telephone_number_Label = QLabel("Telephone Number")
21.         self.membership_type_Label = QLabel("Membership Type")
22.         self.induction_date_Label = QLabel("Induction Date")
23.         self.join_date_Label = QLabel("Join Date")
24.         self.how_paid_Label = QLabel("How Paid")
25.         self.amount_Label = QLabel("Amount")
26.         self.registration_fee_Label = QLabel("Registration Fee")
27.         self.registration_date_Label = QLabel("Registration Date")
28.         self.payment_type_Label = QLabel("Payment Type")
29.         self.comments_Label = QLabel("Comments")
```

```

30.
31.     #create member layout
32.
33.     self.mainMemberLayout = QVBoxLayout()
34.     self.memberContentLayout = QHBoxLayout()
35.     self.memberLabelLayout = QVBoxLayout()
36.     self.memberInputLayout = QVBoxLayout()
37.
38.     self.memberLabelLayout.addWidget(self.memberID_Label)
39.     self.memberLabelLayout.addWidget(self.name_Label)
40.     self.memberLabelLayout.addWidget(self.address_Label)
41.     self.memberLabelLayout.addWidget(self.telephone_number_Label)
42.     self.memberLabelLayout.addWidget(self.membership_type_Label)
43.     self.memberLabelLayout.addWidget(self.induction_date_Label)
44.     self.memberLabelLayout.addWidget(self.join_date_Label)
45.     self.memberLabelLayout.addWidget(self.how_paid_Label)
46.     self.memberLabelLayout.addWidget(self.amount_Label)
47.     self.memberLabelLayout.addWidget(self.registration_fee_Label)
48.     self.memberLabelLayout.addWidget(self.registration_date_Label)
49.     self.memberLabelLayout.addWidget(self.payment_type_Label)
50.     self.memberLabelLayout.addWidget(self.comments_Label)
51.     for count in range(13):
52.         self.memberInputLayout.addWidget(self.lineEdits["self.enter_text
{0}"].format(count)])#adds each QLineEdit to the layout
53.
54.     self.memberContentLayout.addLayout(self.memberLabelLayout)
55.     self.memberContentLayout.addLayout(self.memberInputLayout)
56.     self.mainMemberLayout.addLayout(self.memberContentLayout)
57.
58.     self.setLayout(self.mainMemberLayout)
59.
60.     def addMemberItems(self,database):
61.         #method for adding items to the member table

```

```

62.    addMember(database,self.lineEdits["self.enter_text 0"].text(),self.lineEdits["self.enter_text
1"].text(),self.lineEdits["self.enter_text 2"].text(),self.lineEdits["self.enter_text
3"].text(),self.lineEdits["self.enter_text 4"].text(),self.lineEdits["self.enter_text
5"].text(),self.lineEdits["self.enter_text 6"].text(),self.lineEdits["self.enter_text
7"].text(),self.lineEdits["self.enter_text 8"].text(),self.lineEdits["self.enter_text
9"].text(),self.lineEdits["self.enter_text 10"].text(),self.lineEdits["self.enter_text
11"].text(),self.lineEdits["self.enter_text 12"].text()))

63.

64. def editMemberItems(self,database):
65.     #method for editting items in the member table
66.     columns
67.     =["Name","Address","TelephoneNumber","MembershipType","InductionDate","JoinDate","HowP
aid","Amount","RegistrationFee","RegistrationDate","PaymentType","Comments"]
68.     items = ""
69.     for count in range(1,12):
70.         if self.lineEdits["self.enter_text {0}".format(count)].text() != "":
71.             if count == 1 or 2 or 3 or 4 or 8 or 11 or 12:
72.                 items +=(columns[count-1] + "=" + self.lineEdits["self.enter_text
{0}".format(count)].text() + " ,")
73.             else:
74.                 items +=(columns[count-1] + "=" + self.lineEdits["self.enter_text
{0}".format(count)].text() + " ,")
75.
76.     items = items[:-1]
77.     editMember(database,items,self.lineEdits["self.enter_text 0"].text())

```

AddEditPaymentTableWidget

```
1.  from PyQt4.QtGui import *
2.  from gym_add_function import *
3.  from gym_edit_function import *
4.
5.  class AddEditPaymentTableWidget(QWidget):
6.      """This class creates the widget for the payment table in the add and edit dialog boxes"""
7.
8.      def __init__(self):
9.          super().__init__()
10.
11.         #create widgets
12.
13.         self.lineEdits = {}
14.
15.         for count in range(4):
16.             self.lineEdits["self.enter_text {0}".format(count)] = QLineEdit()#creates 4 QLineEdits, 1
for each column in the payments table
17.
18.         self.memberID_Label = QLabel("MemberID")
19.         self.payment_date_Label = QLabel("Payment Date")
20.         self.how_much_Label = QLabel("How Much")
21.         self.paid_label = QLabel("Paid")
22.
23.         # create payment layout
24.
25.         self.paymentLayout = QVBoxLayout()
26.         self.paymentContentLayout = QHBoxLayout()
27.         self.paymentLabelLayout = QVBoxLayout()
28.         self.paymentInputLayout = QVBoxLayout()
29.
30.         self.paymentLabelLayout.addWidget(self.memberID_Label)
```

```

31.     self.paymentLabelLayout.addWidget(self.payment_date_Label)
32.     self.paymentLabelLayout.addWidget(self.how_much_Label)
33.     self.paymentLabelLayout.addWidget(self.paid_label)
34.     for count in range(4):
35.         self.paymentInputLayout.addWidget(self.lineEdits["self.enter_text
{0}".format(count)])#adds all the QLineEdits to the layout
36.
37.     self.paymentContentLayout.addLayout(self.paymentLabelLayout)
38.     self.paymentContentLayout.addLayout(self.paymentInputLayout)
39.     self.paymentLayout.addLayout(self.paymentContentLayout)
40.
41.     self.setLayout(self.paymentLayout)
42.
43. def addPaymentItems(self, database):
44.     #method for adding items to the payment table
45.     addPayment(database,self.lineEdits["self.enter_text
0"].text(),self.lineEdits["self.enter_text 1"].text(),self.lineEdits["self.enter_text
2"].text(),self.lineEdits["self.enter_text 3"].text())
46.
47. def editPaymentItems(self,database):
48.     #method for editing items in the payment table
49.     columns = ["HowMuch","Paid"]
50.     items = ""
51.     for count in range(2,4):
52.         print(count)
53.         if self.lineEdits["self.enter_text {0}".format(count)].text() != "":
54.             items +==(columns[count-2] + "=" + self.lineEdits["self.enter_text
{0}".format(count)].text() + ",") 
55.     items = items[:-1]
56.
57.     editPayment(database,items,self.lineEdits["self.enter_text
0"].text(),self.lineEdits["self.enter_text 1"].text())

```

AddEditRegimeTableWidget

```
1.  from PyQt4.QtGui import *
2.  from gym_add_function import *
3.  from gym_edit_function import *
4.
5.  class AddEditRegimeTableWidget(QWidget):
6.      """This class creates the widget for the regime table in the add and edit dialog boxes"""
7.
8.      def __init__(self):
9.          super().__init__()
10.
11.         #create widgets
12.
13.         self.lineEdits = {}
14.
15.         for count in range(5):
16.             self.lineEdits["self.enter_text {}".format(count)] = QLineEdit()#creates 5 QLineEdits, 1
for each column in the regime database
17.
18.         self.memberID_Label = QLabel("MemberID")
19.         self.exerciseID_Label = QLabel("Exercise ID")
20.         self.start_date_Label = QLabel("Start Date")
21.         self.end_date_Label = QLabel("End Date")
22.         self.description_Label = QLabel("Description")
23.
24.         # create regime layout
25.
26.         self.regimeLayout = QVBoxLayout()
27.         self.regimeContentLayout = QHBoxLayout()
28.         self.regimeLabelLayout = QVBoxLayout()
29.         self.regimeInputLayout = QVBoxLayout()
```

```

30.
31.     self.regimeLabelLayout.addWidget(self.memberID_Label)
32.     self.regimeLabelLayout.addWidget(self.exerciseID_Label)
33.     self.regimeLabelLayout.addWidget(self.start_date_Label)
34.     self.regimeLabelLayout.addWidget(self.end_date_Label)
35.     self.regimeLabelLayout.addWidget(self.description_Label)
36.     for count in range(5):
37.         self.regimeInputLayout.addWidget(self.lineEdits["self.enter_text
{0}".format(count)])#adds each QLineEdit to the layout
38.
39.     self.regimeContentLayout.setLayout(self.regimeLabelLayout)
40.     self.regimeContentLayout.setLayout(self.regimeInputLayout)
41.     self.regimeLayout.setLayout(self.regimeContentLayout)
42.
43.     self.setLayout(self.regimeLayout)
44.
45.
46. def addRegimeItems(self,database):
47.     #method for adding items to the regime table
48.     addRegime(database,self.lineEdits["self.enter_text 0"].text(),self.lineEdits["self.enter_text
1"].text(),self.lineEdits["self.enter_text 2"].text(),self.lineEdits["self.enter_text
3"].text(),self.lineEdits["self.enter_text 4"].text())
49.
50. def editRegimeItems(self,database):
51.     #method for editing items in the regime table
52.     columns = ["StartDate","EndDate", "SpecificDescription"]
53.     items = ""
54.     for count in range(2,5):
55.         if self.lineEdits["self.enter_text {0}".format(count)].text() != "":
56.             if count == 4:
57.                 items +=(columns[count-2] + "=" + self.lineEdits["self.enter_text
{0}".format(count)].text() + " ,")
58.             else:

```

```
59.         items +=(columns[count-2] + "=" + self.lineEdits["self.enter_text
{0}"].format(count)).text() + ","
60.     items = items[:-1]
61.
62.     editRegime(database,items,self.lineEdits["self.enter_text
0"].text(),self.lineEdits["self.enter_text 1"].text())
```

AddEditExerciseTableWidget

```
1.  from PyQt4.QtGui import *
2.  from gym_add_function import *
3.  from gym_edit_function import *
4.
5.  class AddEditExerciseTableWidget(QWidget):
6.      """This class creates the widget for the exercise table in the add and edit dialog boxes"""
7.
8.      def __init__(self):
9.          super().__init__()
10.
11.         #create widgets
12.
13.         self.lineEdits = {}
14.
15.         for count in range(3):
16.             self.lineEdits["self.enter_text {}".format(count)] = QLineEdit()#creates 3 QLineEdits, 1
for each column in the table
17.
18.         self.exerciseID_Label = QLabel("Exercise ID")
19.         self.name_Label = QLabel("Name")
20.         self.description_Label = QLabel("Description")
21.
22.
23.         # create exercise layout
24.         self.exerciseLayout = QVBoxLayout()
25.         self.exerciseContentLayout = QHBoxLayout()
26.         self.exerciseLabelLayout = QVBoxLayout()
27.         self.exerciseInputLayout = QVBoxLayout()
28.
29.         self.exerciseLabelLayout.addWidget(self.exerciseID_Label)
30.         self.exerciseLabelLayout.addWidget(self.name_Label)
```

```
31.     self.exerciseLabelLayout.addWidget(self.description_Label)
32.
33.     for count in range(3):
34.         self.exerciseInputLayout.addWidget(self.lineEdits["self.enter_text
{0}".format(count)])#adds each QLineEdit to the layout
35.
36.     self.exerciseContentLayout.addLayout(self.exerciseLabelLayout)
37.     self.exerciseContentLayout.addLayout(self.exerciseInputLayout)
38.     self.exerciseLayout.addLayout(self.exerciseContentLayout)
39.
40.     self.setLayout(self.exerciseLayout)
41.
42. def addExerciseItems(self, database):
43.     #method for adding items to the exercise table
44.     addExercise(database,self.lineEdits["self.enter_text 0"].text(),self.lineEdits["self.enter_text
1"].text(),self.lineEdits["self.enter_text 2"].text())
45.
46.
47. def editExerciseItems(self,database):
48.     #method for editing items in the exercise table
49.     columns = ["Name","Description"]
50.     items = ""
51.     for count in range(1,3):
52.         if self.lineEdits["self.enter_text {0}".format(count)].text() != "":
53.             items +=(columns[count-1] + "=" + self.lineEdits["self.enter_text
{0}".format(count)].text() + " ,")
54.     items = items[:-1]
55.
56.     editExercise(database,items,self.lineEdits["self.enter_text 0"].text())
```

BrowseDatabaseWidget

```
1.  from PyQt4.QtGui import *
2.  from PyQt4.QtSql import *
3.  from sqlite3 import *
4.
5.  from gym_database_class import *
6.
7.  class BrowseDataWidget(QWidget):
8.      """A widget for displaying Database data"""
9.
10.     def __init__(self):
11.
12.         self.loadDataBase = None
13.         super().__init__()
14.         self.layout = QVBoxLayout()
15.
16.         self.table_view = QTableView()
17.
18.         self.layout.addWidget(self.table_view)
19.
20.         self.setLayout(self.layout)
21.
22.         self.database = None
23.
24.
25.     def PopulateTable(self,item,labels):
26.         #method for adding all the information from the database to the table view based on
27.         #the currently opened database
28.         self.database = Database(self.loadDataBase)
29.         self.database.loadDatabase()
30.         data = self.database.getAllData(item)
31.         model = QStandardItemModel()
```

```
31.     model.setHorizontalHeaderLabels(labels)
32.     row = 0
33.     for item in data:
34.         for column in range(len(item)):
35.             if item[column] == "None":
36.                 print(item[column])
37.                 item = (" ")
38.             else:
39.                 StandardItem = QStandardItem("{}".format(item[column]))
40.                 model.setItem(row, column, StandardItem)
41.             row += 1
42.     self.table_view.setModel(model)
43.
44.
45.
46.
47.
48.
49.     def UpdateTable(self,newDataBase):
50.         #method for loading a new database when the user opens a new database or reopens a
database
51.         self.loadDataBase = DataBase
```

Print Function

```
1. import sqlite3
2.
3. def getMemberInfo(database,memberID):
4.     #gets the items from the database for a member info printout
5.
6.     con = sqlite3.connect(database)
7.
8.     with con:
9.
10.        cur = con.cursor()
11.        cur.execute("SELECT * FROM MEMBERS WHERE MEMBERID = "+memberID)
12.        result = cur.fetchall()
13.        return result
14.
15. def getInvoice(database,memberID):
16.     #gets the items from the database for a print out of an invoice
17.
18.     con = sqlite3.connect(database)
19.
20.     with con:
21.
22.        cur = con.cursor()
23.        cur.execute("SELECT * FROM PAYMENTS WHERE MEMBERID = "+memberID)
24.        result = cur.fetchall()
25.        cur.execute("SELECT Name FROM MEMBERS WHERE MEMBERID = "+memberID)
26.        name = cur.fetchall()
27.
28.        return result,name
29.
30. def getRegime(database,memberID):
31.     #gets the items from the database for an invoice printout
32.
```

```
33. con = sqlite3.connect(database)
34.
35. with con:
36.
37.     cur = con.cursor()
38.     cur.execute("SELECT * FROM REGIME WHERE MEMBERID = "+memberID)
39.     result = cur.fetchall()
40.     cur.execute("SELECT Name FROM MEMBERS WHERE MEMBERID = "+memberID)
41.     name = cur.fetchall()
42.
43. return result,name
```

Search Function

```
1. import sqlite3
2.
3. def searchQuery(database,table,constraint):
4.     #function that searches the correct table in the database with the correct constraints
5.
6.     con = sqlite3.connect(database)
7.
8.     with con:
9.
10.        cur = con.cursor()
11.
12.        if table == "MEMBERS":
13.            cur.execute("SELECT * FROM "+table+" WHERE MEMBERID Like "+constraint+" OR Name
Like "+constraint+" OR Address Like "+constraint+" OR TelephoneNumber Like "+constraint+
" OR MembershipType Like "+constraint+" OR InductionDate Like "+constraint+" OR JoinDate Like
"+constraint+" OR HowPaid Like "+constraint+" OR Amount Like "+constraint+" OR
RegistrationFee Like "+constraint+" OR RegistrationDate Like "+constraint+" OR PaymentType
Like "+constraint+" OR Comments Like "+constraint)
14.
15.        result = cur.fetchall()
16.
17.        return result
18.
19.    if table == "PAYMENTS":
20.
21.        cur.execute("SELECT * FROM "+table+" WHERE MEMBERID Like "+constraint+" OR
PaymentDate Like "+constraint+" OR HowMuch Like "+constraint+" OR Paid Like "+constraint)
22.
23.        result = cur.fetchall()
24.
25.        return result
26.
27.    if table == "REGIME":
28.
29.        cur.execute("SELECT * FROM "+table+" WHERE MEMBERID Like "+constraint+" OR
EXERCISEID Like "+constraint+" OR SpecificDescription Like "+constraint+" OR StartDate Like
"+constraint+" OR EndDate Like "+constraint)
30.
31.        result = cur.fetchall()
32.
33.        return result
34.
35.    if table == "EXERCISE":
```

```
24.     cur.execute("SELECT * FROM "+table+" WHERE EXERCISEID Like "+constraint+" OR  
Name Like "+constraint+" OR Description Like "+constraint)  
25.     result = cur.fetchall()  
26.     return result
```

Add Function

```
1. import sqlite3
2.
3. def addMember(database, memberID, name, address, telNumber, membershipType,
   inductionDate, joinDate, howPaid, amount, registrationDate, registrationFee, paymentType,
   comments):
4.     #function for adding items to the member table
5.     con = sqlite3.connect(database)
6.
7.     with con:
8.
9.         sql = "INSERT INTO Members (MemberID, Name, Address, TelephoneNumber,
   MembershipType, InductionDate, JoinDate, HowPaid, Amount, RegistrationDate, RegistrationFee,
   PaymentType, Comments) VALUES
   ('"+memberID+"','"+name+"','"+address+"','"+telNumber+"','"+membershipType+"','"+inducti
   onDate+"','"+joinDate+"','"+howPaid+"','"+amount+"','"+registrationDate+"','"+registrationFe
   e+"','"+paymentType+"','"+comments+"')"
10.    print(sql)
11.    cur = con.cursor()
12.    cur.execute(sql)
13.
14.
15. def addPayment(database,memberID,paymentDate,howMuch,paid):
16.     #function for adding items to the payment table
17.     con = sqlite3.connect(database)
18.
19.     with con:
20.
21.         cur = con.cursor()
22.         cur.execute("INSERT INTO Payments (MemberID, PaymentDate, HowMuch, Paid) VALUES
   ("+memberID+","+paymentDate+","+howMuch+","+paid+")")
23.
24. def addRegime(database,memberID,exerciseID,specificDescription,startDate,endDate):
```

```
25. #function for adding items to the regime table
26. con = sqlite3.connect(database)
27.
28. with con:
29.
30.     cur = con.cursor()
31.     cur.execute("INSERT INTO Regime(MemberID, ExerciseID, SpecificDescription, StartDate,
32.                                         EndDate) VALUES
33.                                         ("+memberID+","+exerciseID+","+specificDescription+","+startDate+","+endDate+"))
34.                                         #function for adding items to the exercise table
35.     con = sqlite3.connect(database)
36.
37.     with con:
38.
39.         cur = con.cursor()
40.         cur.execute("INSERT INTO Exercise(ExerciseID, Name, Description) VALUES
41.                                         ("+exerciseID+","+name+","+description+"))")
```

Edit Function

```
1. import sqlite3
2.
3. def editMember(database,items,memberID):
4.     #function for editing items in the member table
5.     con = sqlite3.connect(database)
6.
7.     with con:
8.
9.         sql = "UPDATE Members SET "+items+" WHERE MemberID = "+memberID
10.        cur = con.cursor()
11.        cur.execute(sql)
12.
13. def editPayment(database,items,memberID,paymentDate):
14.     #function for editing items in the payment table
15.     con = sqlite3.connect(database)
16.
17.     with con:
18.         sql = "UPDATE Payments SET "+items+" WHERE MemberID="+memberID+" AND
19.               PaymentDate = "+paymentDate
20.         cur = con.cursor()
21.
22. def editRegime(database,items,memberID,exerciseID):
23.     #function for editing items in the regimetable
24.     con = sqlite3.connect(database)
25.
26.     with con:
27.         sql = "UPDATE Regime SET "+items+" WHERE MemberID = "+memberID+" AND ExerciseID
28.               = "+exerciseID
29.         cur = con.cursor()
30.         cur.execute(sql)
```

```
31. def editExercise(database,items,exerciseID):
32.     #function for editing items in the exercise table
33.     con = sqlite3.connect(database)
34.
35.     with con:
36.         sql = "UPDATE Exercise SET "+items+" WHERE EXERCISEID = "+exerciseID
37.         cur = con.cursor()
38.         cur.execute(sql)
```

Delete Function

```
1. import sqlite3
2.
3. def deleteQueryPrimaryKey(database,table,item,constraint):
4.     #method for deleting items from the Members or Exercise Tables as they have a primary key
5.
6.     con = sqlite3.connect(database)
7.
8.     with con:
9.
10.        cur = con.cursor()
11.        cur.execute("DELETE FROM "+table+" WHERE "+item+" = "+constraint)
12.
13. def deleteQueryCompositeKey(database,table,item1,item2,constraint1,constraint2):
14.     #method for deleting items from the payments or regimes tables as they have composite keys
15.     con = sqlite3.connect(database)
16.
17.     with con:
18.
19.        cur = con.cursor()
20.        sql = "DELETE FROM "+table+" WHERE "+item1+" = "+constraint1+" AND "+item2+" =
21.             "+constraint2
22.        print(sql)
23.
24. def deleteAll(database,table):
25.     #delete all items from any table
26.     con = sqlite3.connect(database)
27.
28.     with con:
29.
30.        cur = con.cursor()
31.        cur.execute("DELETE FROM "+table)
```

```
32.  
33.  
34. def getItems(database,table):  
35.     #function to retrieve all the data from the tables  
36.  
37.     con = sqlite3.connect(database)  
38.  
39.     with con:  
40.  
41.         cur = con.cursor()  
42.         cur.execute("SELECT * FROM "+table)  
43.  
44.         results = cur.fetchall()  
45.         items = []  
46.  
47.         for count in range(len(results)):  
48.             column = results[count]  
49.             if table == "MEMBERS":  
50.                 items.append(str(column[0])+". "+str(column[1]))  
51.             if table == "PAYMENTS":  
52.                 cur.execute("SELECT Name FROM MEMBERS WHERE MEMBERID="+str(column[0]))  
53.                 memberName = cur.fetchall()  
54.                 for part in memberName:  
55.                     items.append(str(column[0])+". "+str(part[0])+" - "+str(column[1]))  
56.             if table == "REGIME":  
57.                 cur.execute("SELECT Name FROM MEMBERS WHERE MEMBERID="+str(column[0]))  
58.                 memberName = cur.fetchall()  
59.                 cur.execute("SELECT Name FROM EXERCISE WHERE EXERCISEID="+str(column[1]))  
60.                 exerciseName = cur.fetchall()  
61.                 name = ""  
62.                 exercise = ""  
63.                 for part in memberName:  
64.                     name = str(part[0])
```

```
65.     for part in exerciseName:  
66.         exercise = str(part[0])  
67.         items.append(str(column[0])+". "+name+" - "+str(column[1])+". "+exercise)  
68.     if table == "EXERCISE":  
69.         items.append(str(column[0])+". "+str(column[1]))  
70.  
71. return items
```

Database Class

```
1. import sqlite3
2.
3. class Database:
4.     def __init__(self, db_name):
5.         self.db_name = db_name
6.
7.     def loadDatabase(self):
8.         #method that connects to the right database based on the file that was opened by the user
9.         with sqlite3.connect(self.db_name) as db:
10.             cursor = db.cursor()
11.
12.     def getAllData(self,table):
13.         #method for retrieving all the data from a specific table in the database with a Select Query
14.         with sqlite3.connect(self.db_name) as db:
15.             allData = []
16.             cursor = db.cursor()
17.             query = "Select * From " + str(table)
18.             cursor.execute(query)
19.             data = cursor.fetchall()
20.             for item in data:
21.                 allData.append(item)
22.         return allData
```

AddEdit Dialog

```
1.  from PyQt4.QtGui import *
2.  from add_edit_member_table_widget_class import *
3.  from add_edit_payment_table_widget_class import *
4.  from add_edit_regime_table_widget_class import *
5.  from add_edit_exercise_table_widget_class import *
6.  from gym_add_function import *
7.
8.  class AddEditDialog(QDialog):
9.      """This class creates the dialog window for Adding items to the database and answering
them"""
10.
11.     def __init__(self, database):
12.         super().__init__()
13.
14.         self.table_combo_box = QComboBox()
15.         self.table_combo_box.addItem("Members")
16.         self.table_combo_box.addItem("Payments")
17.         self.table_combo_box.addItem("Regimes")
18.         self.table_combo_box.addItem("Exercises")
19.         self.add_edit_button = QPushButton()
20.         self.database = database
21.
22.         self.stackLayout = QStackedLayout()
23.         self.mainLayout = QVBoxLayout()
24.         self.topLayout = QHBoxLayout()
25.
26.         self.setWindowTitle(" ")
27.         self.setWindowIcon(QIcon("WindowIcon.png"))
28.
29.         self.memberLayoutWidget = AddEditMemberTableWidget()
30.         self.paymentLayoutWidget = AddEditPaymentTableWidget()
31.         self.regimeLayoutWidget = AddEditRegimeTableWidget()
```

```
32.     self.exerciseLayoutWidget = AddEditExerciseTableWidget()
33.     self.stackedLayout.addWidget(self.memberLayoutWidget)
34.     self.stackedLayout.addWidget(self.paymentLayoutWidget)
35.     self.stackedLayout.addWidget(self.regimeLayoutWidget)
36.     self.stackedLayout.addWidget(self.exerciseLayoutWidget)
37.
38.     self.topLayout.addWidget(self.table_combo_box)
39.     self.topLayout.addWidget(self.add_edit_button)
40.     self.mainLayout.setLayout(self.topLayout)
41.     self.mainLayout.addWidget(self.stackedLayout)
42.
43.     self.setLayout(self.mainLayout)
44.
45.     #connections
46.     self.table_combo_box.currentIndexChanged.connect(self.updateTable)#when the
comboboxes currently selected table is changed the updateTable method is run
47.
48.
49.     def updateTable(self,index):
50.         #method for chaning the widget based on the table selected in the combobox
51.         self.stackedLayout.setCurrentIndex(index)
```

Add Dialog

```
1. from PyQt4.QtGui import *
2. from gym_add_edit_dialog_class import *
3.
4. class AddDialog(AddEditDialog):
5.     """This class creates the dialog window for Adding items to the database"""
6.
7.     def __init__(self, database):
8.         super().__init__(database)
9.
10.        self.add_edit_button.setText("Add")
11.        self.setWindowTitle("Add")
12.
13.        #connections
14.        self.add_edit_button.clicked.connect(self.addItems)
15.
16.    def addItems(self):
17.        #method for using the correct methods to add items to a table, with the
18.        #table changing based on which widget is selected from the stacked layout
19.        if self.stackedLayout.currentIndex() == 0:
20.            self.memberLayoutWidget.addMemberItems(self.database)
21.        if self.stackedLayout.currentIndex() == 1:
22.            self.paymentLayoutWidget.addPaymentItems(self.database)
23.        if self.stackedLayout.currentIndex() == 2:
24.            self.regimeLayoutWidget.addRegimeItems(self.database)
25.        if self.stackedLayout.currentIndex() == 3:
26.            self.exerciseLayoutWidget.addExerciseItems(self.database)
27.
28.        self.close()
```

Edit Dialog

```
1. from PyQt4.QtGui import *
2. from gym_add_edit_dialog_class import *
3.
4. class EditDialog(AddEditDialog):
5.     """This class creates the dialog window for editing items in the database"""
6.
7.     def __init__(self, database):
8.         super().__init__(database)
9.
10.        self.add_edit_button.setText("Edit")
11.        self.setWindowTitle("Edit")
12.
13.        #connections
14.        self.add_edit_button.clicked.connect(self.editItems)
15.
16.    def editItems(self):
17.        #method for using the correct methods to edit items in a table, with the table changing
18.        #based on which widget is selected from the stacked layout
19.        if self.stackedLayout.currentIndex() == 0:
20.            self.memberLayoutWidget.editMemberItems(self.database)
21.        if self.stackedLayout.currentIndex() == 1:
22.            self.paymentLayoutWidget.editPaymentItems(self.database)
23.        if self.stackedLayout.currentIndex() == 2:
24.            self.regimeLayoutWidget.editRegimeItems(self.database)
25.        if self.stackedLayout.currentIndex() == 3:
26.            self.exerciseLayoutWidget.editExerciseItems(self.database)
27.        self.close()
```

Delete Dialog

```
1. from PyQt4.QtGui import *
2. from gym_delete_function import *
3.
4. class DeleteDialog(QDialog):
5.     """This class creates the dialog window for deleting items from the database"""
6.
7.     def __init__(self, database):
8.         super().__init__()
9.
10.        self.database = database
11.
12.        #create widgets
13.        self.table_select_combo_box = QComboBox()
14.        self.item_select_combo_box = QComboBox()
15.        self.delete_push_button = QPushButton("Delete")
16.        self.delete_all_push_button = QPushButton("Delete All Items")
17.
18.        self.table_select_combo_box.addItem("Select Table")
19.        self.table_select_combo_box.addItem("Members")
20.        self.table_select_combo_box.addItem("Payments")
21.        self.table_select_combo_box.addItem("Regimes")
22.        self.table_select_combo_box.addItem("Exercises")
23.
24.        self.item_select_combo_box.addItem("Select Item")
25.
26.        #create layout
27.        self.layout = QVBoxLayout()
28.
29.        #add widgets to layout
30.        self.layout.addWidget(self.table_select_combo_box)
31.        self.layout.addWidget(self.item_select_combo_box)
32.        self.layout.addWidget(self.delete_push_button)
```

```

33.     self.layout.addWidget(self.delete_all_push_button)

34.

35.     #set the window layout

36.     self.setLayout(self.layout)

37.     self.setWindowTitle("Delete")

38.     self.setWindowIcon(QIcon("WindowIcon.png"))

39.

40.     #connections

41.     self.table_select_combo_box.currentIndexChanged.connect(self.itemComboBoxPopulate)

42.     self.delete_push_button.clicked.connect(self.deleteItems)

43.     self.delete_all_push_button.clicked.connect(self.deleteAllItems)

44.

45. def itemComboBoxPopulate(self):
46.     #method for populating the item select combobox with the correct items and formating
47.     #from the correct tables
48.     if self.table_select_combo_box.currentIndex() == 0:
49.         self.item_select_combo_box.clear()
50.         self.item_select_combo_box.addItem("Select Item")
51.     if self.table_select_combo_box.currentIndex() == 1:
52.         items = getItems(self.database, "MEMBERS")
53.         self.item_select_combo_box.clear()
54.         for count in range(len(items)):
55.             self.item_select_combo_box.addItem(items[count])
56.     if self.table_select_combo_box.currentIndex() == 2:
57.         items = getItems(self.database, "PAYMENTS")
58.         self.item_select_combo_box.clear()
59.         for count in range(len(items)):
60.             self.item_select_combo_box.addItem(items[count])
61.     if self.table_select_combo_box.currentIndex() == 3:
62.         items = getItems(self.database, "REGIME")
63.         self.item_select_combo_box.clear()
64.         for count in range(len(items)):
65.             self.item_select_combo_box.addItem(items[count])

```

```

65.     if self.table_select_combo_box.currentIndex() == 4:
66.         items = getItems(self.database, "EXERCISE")
67.         self.item_select_combo_box.clear()
68.         for count in range(len(items)):
69.             self.item_select_combo_box.addItem(items[count])
70.
71.     def deleteItems(self):
72.         #method for deleting the correct items from the correct table
73.         if self.table_select_combo_box.currentIndex() == 0:
74.             return
75.         if self.table_select_combo_box.currentIndex() == 1:
76.
77.             deleteQueryPrimaryKey(self.database, "MEMBERS", "MEMBERID", str(self.item_select_combo_box.
78. currentText())[0])
79.
80.             if self.table_select_combo_box.currentIndex() == 2:
81.                 sep = " - "
82.
83.                 deleteQueryCompositeKey(self.database, "PAYMENTS", "MEMBERID", "PAYMENTDATE", str(self.item_
84. select_combo_box.currentText())[0], str(self.item_select_combo_box.currentText()).split(sep,
85. , 1)[1])
86.             if self.table_select_combo_box.currentIndex() == 3:
87.                 sep = " - "

```

#method for deleting every item in a certain table

```
88.     if self.table_select_combo_box.currentIndex() == 0:
89.         return
90.     if self.table_select_combo_box.currentIndex() == 1:
91.         deleteAll(self.database, "MEMBERS")
92.     if self.table_select_combo_box.currentIndex() == 2:
93.         deleteAll(self.database, "PAYMENTS")
94.     if self.table_select_combo_box.currentIndex() == 3:
95.         deleteAll(self.database, "REGIME")
96.     if self.table_select_combo_box.currentIndex() == 4:
97.         deleteAll(self.database, "EXERCISE")
```

Search Dialog

```
1. from PyQt4.QtGui import *
2. from gym_search_results_dialog_class import *
3. from gym_search_function import *
4.
5. class SearchDialog(QDialog):
6.     """This class creates the dialog window for searching for something"""
7.
8.     def __init__(self, database):
9.         super().__init__()
10.
11.        self.database = database
12.
13.        #create widgets
14.        self.table_combo_box = QComboBox()
15.        self.search_box = QLineEdit()
16.        self.search_push_button = QPushButton("Search")
17.
18.        self.table_combo_box.addItem("Select Table")
19.        self.table_combo_box.addItem("MEMBERS")
20.        self.table_combo_box.addItem("PAYMENTS")
21.        self.table_combo_box.addItem("REGIME")
22.        self.table_combo_box.addItem("EXERCISE")
23.        self.search_box.setPlaceholderText("Enter Search Term")
24.
25.        #create layout
26.        self.layout = QVBoxLayout()
27.
28.        #add widgets to layout
29.        self.layout.addWidget(self.table_combo_box)
30.        self.layout.addWidget(self.search_box)
31.        self.layout.addWidget(self.search_push_button)
32.
```

```
33.     #set the window layout
34.     self.setLayout(self.layout)
35.     self.setWindowTitle("Search")
36.     self.setWindowIcon(QIcon("WindowIcon.png"))
37.
38.     #connections
39.     self.search_push_button.clicked.connect(self.search)
40.
41.
42.     def search(self):
43.         #method for searching the database
44.         results =
45.             searchQuery(self.database,self.table_combo_box.currentText()+''+self.search_box.text()+'')
46.             results_dialog = ResultsDialog(results)
```

Search Results Dialog

```
1.  from PyQt4.QtGui import *
2.
3.  class ResultsDialog(QDialog):
4.      """This class creates the dialog window for the search results"""
5.
6.      def __init__(self,searchResults):
7.          super().__init__()
8.
9.          self.searchResults = searchResults
10.
11.         #create widgets
12.         self.results = QTextEdit()
13.
14.         #create layout
15.         self.layout = QVBoxLayout()
16.
17.         #add widgets to layout
18.         self.layout.addWidget(self.results)
19.
20.         #set the window layout
21.         self.setLayout(self.layout)
22.         self.setWindowTitle("Results")
23.         self.setWindowIcon(QIcon("WindowIcon.png"))
24.
25.         self.searchResults = searchResults
26.         self.populateResultsBox()
27.
28.     def populateResultsBox(self):
29.         #method for populating the editText in the results dialog with the results from the
30.         #search sql query
31.         self.string = ""
32.         for item in self.searchResults:
```

```
32.     self.string = ((self.string)+"\n\n"+str(item))
```

```
33.     self.results.setText(self.string)
```

Print Dialog

```
1.  from PyQt4.QtGui import *
2.  from PyQt4.QtCore import *
3.  from gym_print_sql_function import *
4.  from gym_delete_function import *
5.
6.  class PrintDialog(QDialog):
7.      """This class creates the dialog window for creating forms and printing them"""
8.
9.      def __init__(self,database):
10.         super().__init__()
11.
12.         self.database = database
13.
14.         #create widgets
15.         self.form_combo_box = QComboBox()
16.         self.item_select_combo_box = QComboBox()
17.         self.print_push_button = QPushButton("Print")
18.
19.         self.form_combo_box.addItem("Select Form")
20.         self.form_combo_box.addItem("Invoice")
21.         self.form_combo_box.addItem("Member Details")
22.         self.form_combo_box.addItem("Regime")
23.
24.
25.         #create layout
26.         self.layout = QVBoxLayout()
27.
28.         #add widgets to layout
29.         self.layout.addWidget(self.form_combo_box)
30.         self.layout.addWidget(self.item_select_combo_box)
31.         self.layout.addWidget(self.print_push_button)
32.
```

```

33.     #set the window layout
34.     self.setLayout(self.layout)
35.     self.setWindowTitle("Print")
36.     self.setWindowIcon(QIcon("Hugobells.png"))
37.
38.     #connections
39.     self.form_combo_box.currentIndexChanged.connect(self.itemComboBoxPopulate)
40.     self.print_push_button.clicked.connect(self.printInfo)
41.
42.     def printFunction(self,itemToPrint):
43.         #function that sends the textEdit to be printed and allows the user to select a printer
44.         #through the print dialog
45.         dialog = QPrintDialog()
46.         if dialog.exec_() == QDialog.Accepted:
47.             itemToPrint.print_(dialog.printer())
48.
49.     def generateMemberInfo(self):
50.         #method that creates the textEdit containg the correct information for a member info
51.         #print out
52.         columns
53.         =[ "MemberID", "Name", "Address", "TelephoneNumber", "MembershipType", "InductionDate", "Join
54.         Date", "HowPaid", "Amount", "RegistrationFee", "RegistrationDate", "PaymentType", "Comments"]
55.         sep = "."
56.         info = getMemberInfo(self.database,
57.         str(self.item_select_combo_box.currentText()).split(sep,1)[0])
58.         infoToPrint = QTextEdit()
59.         infoString = ""
60.         for item in info:
61.             count = 0
62.             for piece in item:
63.                 infoToPrint.setText(infoToPrint.toPlainText()+columns[count]+": "+str(piece)+"\n")
64.                 count += 1
65.         self.printFunction(infoToPrint)

```

```

61.
62.     def generateInvoice(self):
63.         #method that creates the textEdit containg the correct information for a print out of an
64.         invoice
65.         columns = ["MemberID", "Payment Date", "How Much", "Paid"]
66.         sep = "."
67.         info, name = getInvoice(self.database,
68.             str(self.item_select_combo_box.currentText()).split(sep, 1)[0])
69.         infoToPrint = QTextEdit()
70.         for list in name:
71.             for word in name:
72.                 for item in word:
73.                     infoToPrint.setText("Member Name : "+str(item)+"\n\n")
74.                     for item in info:
75.                         infoToPrint.setText(infoToPrint.toPlainText() + columns[count] + " : "+str(piece)+"\n")
76.                         count += 1
77.                         infoToPrint.setText(infoToPrint.toPlainText() + "\n")
78.                     self.printFunction(infoToPrint)
79.
80.     def generateRegime(self):
81.         #method that creates the textEdit containg the correct information for a members
82.         regime print out
83.         columns = ["MemberID", "ExerciseID", "Description", "Start Date", "End Date"]
84.         sep = "."
85.         info, name = getRegime(self.database,
86.             str(self.item_select_combo_box.currentText()).split(sep, 1)[0])
87.         infoToPrint = QTextEdit()
88.         for list in name:
89.             for word in name:
90.                 for item in word:
91.                     infoToPrint.setText("Member Name : "+str(item)+"\n\n")

```

```
90.     for item in info:
91.         count = 0
92.         for piece in item:
93.             infoToPrint.setText(infoToPrint.toPlainText() + columns[count] + " : " + str(piece) + "\n")
94.             count += 1
95.         infoToPrint.setText(infoToPrint.toPlainText() + "\n")
96.     self.printFunction(infoToPrint)
97.
98. def itemComboBoxPopulate(self):
99.     if self.form_combo_box.currentIndex() == 0:
100.         self.item_select_combo_box.clear()
101.         self.item_select_combo_box.addItem("Select Item")
102.     else:
103.         items = getItems(self.database, "MEMBERS")
104.         self.item_select_combo_box.clear()
105.         for count in range(len(items)):
106.             self.item_select_combo_box.addItem(items[count])
107.
108. def printInfo(self):
109.     if self.form_combo_box.currentIndex() == 1:
110.         self.generateInvoice()
111.     if self.form_combo_box.currentIndex() == 2:
112.         self.generateMemberInfo()
113.     if self.form_combo_box.currentIndex() == 3:
114.         self.generateRegime()
```

About Dialog

```
1.  from PyQt4.QtGui import *
2.  from PyQt4.QtCore import *
3.  import webbrowser
4.
5.  class AboutDialog(QDialog):
6.      """This is the about page"""
7.
8.      def __init__(self):
9.          super().__init__()
10.
11.     #create widgets
12.     self.text = QTextEdit()
13.     self.text.setPlainText("Created By Toby Kerslake \n\n\n\n User Manual available for
download below")
14.     self.text.setReadOnly(True)#set the textEdit so the user can't rewrite the text
15.     self.userManualButton = QPushButton("User Manual")
16.
17.     #create layout
18.     self.layout = QVBoxLayout()
19.
20.
21.     #add widgets to layout
22.
23.     self.layout.addWidget(self.text)
24.     self.layout.addWidget(self.userManualButton)
25.
26.
27.     #set the window layout
28.     self.setLayout(self.layout)
29.     self.setWindowTitle("About")
30.     self.setWindowIcon(QIcon("WindowIcon.png"))
31.
```

```
32.     #connections
33.     self.userManualButton.clicked.connect(self.openUserManual)
34.
35.     def openUserManual(self):
36.         #method that opens up the user manual inside the users default browser
37.
    webbrowser.open('https://www.dropbox.com/s/61n4soosnvo1cnc/User%20Manual.docx?dl=0'
)
```

Password Dialog

```
1.  from PyQt4.QtGui import *
2.
3.  class PasswordDialog(QDialog):
4.      """This class creates a dialog box for a password"""
5.
6.      def __init__(self):
7.          super().__init__()
8.
9.          #create widgets
10.         self.password_lineEdit = QLineEdit()
11.         self.password_lineEdit.setPlaceholderText("Password")
12.         self.password_lineEdit.setEchoMode(QLineEdit.Password)
13.         self.enterButton = QPushButton("Enter Password")
14.         self.closeButton = QPushButton("Close")
15.
16.         #create and set layout
17.         self.layoutMain = QVBoxLayout()
18.         self.layoutHorizontal = QHBoxLayout()
19.         self.layoutMain.addWidget(self.password_lineEdit)
20.         self.layoutHorizontal.addWidget(self.enterButton)
21.         self.layoutHorizontal.addWidget(self.closeButton)
22.         self.layoutMain.addLayout(self.layoutHorizontal)
23.         self.setLayout(self.layoutMain)
24.         self.setWindowTitle("Password")
25.         self.setWindowIcon(QIcon("WindowIcon.png"))
26.
27.         #connections
28.         self.enterButton.clicked.connect(self.close)
29.
30.     def close_method(self):
31.         #method for returning the entered password to check if it is correct by the main
```

program file

32. **return self.password_lineEdit.text()**

Testing

Outline Plan

Test Series	Purpose of Test Series	Testing Strategy	Strategy Rationale
1	Test the flow and navigation between the different menus and interfaces included in the GUI.	Top-Down Testing	
2	Test the validation methods for the data input by the user	Bottom-Up Testing	Each component will be tested after development.
3	Test that all the input data is stored in the correct place	Black Box Testing	
4	Test algorithms to make sure the output is correct	White Box Testing	
5	Test that the system meets the specification.	Acceptance Testing	

Changes to the Test Plan

I have made some changes to the detailed plan since the design section to accommodate for the lack of boundary data in the original plan. The changes were made in test series 2.

Detailed Plan

Test Series and Number	Purpose	Description	Test Data	Test Data Type	Expected Result	Actual Result	Evidence in Appendix
1.1	Test that the password enter	The password dialog should	“ “ - Leave the password lineEdit	Erroneous	The password dialog will	As Expected	

	dialog responds correctly to the wrong password	reopen prompting the user to try again	blank and then click the enter button		reopen		
1.2	Test that the password responds correctly to the correct password	The password dialog should close opening the main interface	"password" - Enter the correct password and then click the enter button	Normal	The main interface will open after the password dialog closes	The password dialog closes and then the main interface opens	Fig. 1
1.3	Test that the open button works correctly	The windows file explorer should open to allow the user to select the right database file	Click the open button	Normal	The windows file explorer should open while the main interface remains open	The main interface remains open but shifts out of focus as the window file explorer dialog becomes the main focus	
1.4	Test that the open shortcut in the toolbar works correctly	The windows file explorer should open to allow the user to select the right database file	Click the open button	Normal	The windows file explorer should open while the main interface remains open	The main interface remains open but shifts out of focus as the window file explorer dialog becomes the main	

						focus	
1.5	Test that the table view in the main interface changes to accomodate the opened database	The table view should change to include data from the database selected with each table appearing in a different table	Open a database	Normal	The table view will now contain information stored in the database file	The table view now contains the information stored on the database	Fig. 2
1.6	Test that the Add button works correctly	The Add dialog box should open allowing the user to select and input the data they want to add to the database	Click the add button	Normal	The Add dialog box should open while the main interface remains open	The Add dialog opens and shift into focus over the main interface which remains open in the background	Fig. 3
1.7	Test that the Add shortcut in the toolbar works correctly	The Add dialog box should open allowing the user to select and input the data they	Click the add button	Normal	The Add dialog box should open while the main interface remains open	The Add dialog opens and shift into focus over the main interface which remains	

		want to add to the database				open in the background	
1.8	Test that the select table combo box works correctly in the Add dialog.	The section in the layout should fill with a form with input options based on the table selected	Change the select table combo box option to all available tables - "Member", "Payment", "Regime", "Exercise"	Normal	The input form should change to contain information and input options based on the selected table	The input form changes widgets to display the appropriate input form selected - The member widget was displayed when the member option was selected and the same applies for payment, regime and exercise	
1.9	Test that the Add dialog responds correctly after the Add button has been pushed	The dialog box should close returning the users focus to the main interface	Enter all appropriate data and hit the Add button	Normal	The dialog box should close with the main interface still open	The dialog box closes and the main interface remains open	
1.10	Test that the Edit	The Edit dialog	Click the edit button	Normal	The Edit dialog	The edit dialog	

	button works correctly	box should open allowing the user to select and input the data they want to edit in the database			box should open while the main interface remains open	box opens and switches into focus while the main interface remains in the background	
1.11	Test that the Edit shortcut in the toolbar works correctly	The Edit dialog box should open allowing the user to select and input the data they want to edit in the database	Click the edit button	Normal	The Edit dialog box should open while the main interface remains open	The edit dialog box opens and switches into focus while the main interface remains in the background	
1.12	Test that the select table combo box works correctly in the Edit dialog.	The section in the layout should fill with a form with input options based on the table selected	Change the select table combo box option to all available tables - "Member", "Payment", "Regime", "Exercise"	Normal	The input form should change to contain information and input options based on the selected table	The input form changes widgets to display the appropriate input form selected - The member widget was displayed when the member	Fig. 4

						option was selected and the same applies for payment, regime and exercise	
1.13	Test that the Edit dialog responds correctly after the Edit button has been pushed	The dialog box should close returning the users focus to the main interface	Enter all appropriate data and hit the Edit button	Normal	The dialog box should close with the main interface still open	The dialog box closes and the main interface remains open	
1.14	Test that the Delete button works correctly	The Delete dialog box should open allowing the user to select the information they want to delete while the main interface remains open	Click the Delete button	Normal	The Delete dialog box should open while the main interface is still open	The delete dialog box opens and switches into focus while the main interface remains in the background	
1.15	Test that the delete shortcut	The Delete dialog box	Click the Delete shortcut	Normal	The Delete dialog box	The delete dialog box	

	in the toolbar works correctly	should open allowing the user to select the information they want to delete while the main interface remains open			should open while the main interface is still open	opens and switches into focus while the main interface remains in the background	
1.16	Test that the Select Table combobox in the delete dialog works correctly	The Select Item combobox should then include different options for the user to select	Select all possible options from the Select Table combo box	Normal	The Select Item combo box should fill with data based off of the selected table	The select item combo box fills with data retrieved from the selecting table in the database	Fig. 5
1.17	Test that the Delete dialog box responds correctly to the Delete or Delete All button being clicked	The dialog box should close bringing the Main Interface into the users focus	Click the Delete and Delete All Buttons	Normal	The Delete Dialog should close leaving just the Main Interface Open	The delete dialog box closes while the main interface stays open and switches back into focus	
1.18	Test that the Search Button	The search Dialog should	Click on the Search button	Normal	The Search dialog should	The search dialog box	

	works correctly	open allowing the user to enter the information they wish to search for			open while the Main Interface stays open in the background	opens and switches into focus while the main interface remains in the background	
1.19	Test that the Search shortcut in the toolbar works correctly	The search Dialog should open allowing the user to enter the information they wish to search for	Click on the Search Shortcut	Normal	The Search dialog should open while the Main Interface stays open in the background	The search dialog box opens and switches into focus while the main interface remains in the background	
1.20	Test that the Search button in the Search dialog works correctly	This should display the results of the search for the user	Enter all appropriate information and click the Search button	Normal	A new Results Dialog should open closing the Search dialog but keep the Main Interface open in the background	The search dialog closes and the result dialog opens in its place while the Main Interface window remains open in the background	

						nd	
1.21	Test that the Close button in the result dialog functions correctly	The Results dialog should close drawing the user's focus to the Main Interface again	Click on the Close Button	Normal	The Results dialog should close while the Main Interface window is brought back to the user's attention	The result dialog closes and the Main Interface window opens and is brought back to the user's focus	
1.22	Test the Print Button works correctly	The Print Dialog should open so that the user can select the form and information they want to print	Click on the Print Button	Normal	The Print Dialog should open leaving the Main Interface open in the Background	The Print dialog box opens and switches into focus while the main interface remains in the background	
1.23	Test the Print shortcut in the toolbar works correctly	The Print Dialog should open so that the user can select the form and information they want to print	Click on the Print Button	Normal	The Print Dialog should open leaving the Main Interface open in the Background	The Print dialog box opens and switches into focus while the main interface remains in the	Fig. 6

						background	
1.24	Test that the comboboxes in the print function work correctly	The comboboxes should change the information in the subsequent combobox when an item is selected in one of them	Select all the different options in all 3 comboboxes	Normal	The Comboboxes should have different options based on what's entered in the other comboboxes	The comboboxes fill with different items based on the currently selected index in the other comboboxes	
1.25	Test that the Print button in the Print Dialog works correctly	The Print dialog should close bringing the Main Interface back into the user's focus	Click on the Print Button	Normal	The Print Dialog should close leaving only the Main Interface Open	The print dialog closes shifting the main interface window back into the user's function	
1.26	Test that the User Manual Option in the toolbar works	A digital version of the User Manual should open in another window so that the user can easily identify	Click on the User Manual Shortcut	Normal	The User Manual should open up in a dialog box leaving the Main Interface Open	The user manual dialog box opens while the main interface remains open in the background	

		how to perform an operation within the program					
1.27	Test that the About option in the toolbar works	The about section should open so that the user can identified the programs current version, creator and any other necessary information	Click on the About option	Normal	A Dialog box containing information about the program should be opened while the Main Interface remains open in the program	A Dialog box with information about the program opens while the main interface remains open in the background	
2.1	Verify that the password entered is correct	The password will match the one stored inside the program	The correct password The incorrect password	Normal Erroneous	The password will be correct and the program will carry on as so The program will not continue until the correct password is entered and the user will	The Password was correct and the program carried on The password dialog closed and reopened as the password entered was not correct	

					be prompted as such		
2.2	Verify that the file opened is a database (.db) file	Opening the wrong type of file should result in an error	A Database File A Word File	Normal Erroneous	The program will continue and load the database The program will prompt the user with an error	The program continued and loaded the selected database The program doesn't load the database but continues on	
2.3	Verify that the information entered in the Add input form is of the correct type	If the data is not the correct type the user should receive an error message	Appropriate Data Incorrect data type(s)	Normal Erroneous	The program will continue with the data The user will receive an error message and be prompted to change the necessary data	The program continues and the info is added to the database The user receives an error message and is able to change the data through the widget	Fig. 7
2.4	Verify that the	If the data is	Appropriate Data	Normal	The program	The program	

	information entered in the Edit input form is of the correct type	not the correct type the user should receive an error message	Incorrect data type(s)	Erroneous	will continue with the data	continues and the info is edited in the database	
2.5	Verify that the search term entered in the Search Dialog actually yields results	If the search comes up without results the user should receive a message saying so	A correct search term An incorrect search term	Normal Erroneous	The program continues on with the search results The user receives a message and is prompted to use a different search term	A dialog box containing the correct results is displayed and the program continues as normal The results window displays that there are no results and the user is prompted to try different search terms	Fig. 8

2.6	Verify that any Add inputs only accept a certain limit of characters	Certain items have a limit of characters like the Telephone Number should only allow up to 11 characters	013536678 29 011111922 313123213 123123	Normal Boundary	The program accepts the input The program rejects the input and display an error message	The program accepts the input The program accepted the boundary data	Fig. 11
2.7	Verify that any Edit inputs only accept a certain limit of characters	Certain items have a limit of characters like the Telephone Number should only allow up to 11 characters	013536678 29 011111922 313123213 123123	Normal Boundary	The program accepts the input The program rejects the input and display an error message	The program accepts the input The program accepted the boundary data	Fig. 12
2.8	Verify that the primary or composite keys exists during an edit function	An item can't be edited if its primary key(s) can not be identified	MemberID: 5 MemberID: 99780453	Normal Boundary	The program accepts the input The program rejects the input and displays an error	The program accepts the input The program rejects the input but doesn't display	Fig. 13

					message	an error message	
3.1	Verify that all Member details that are input are added to the database	All the details should be added to the correct place in the database	Member information MemberID: 15 Name: Toby Kerslake All Other Items: Leave Blank	Normal	The data should be added into the appropriate place in the database	The correct memberID and Name are added to the database with the rest of the fields remaining blank	
3.2	Verify that all Payment details that are input are added to the database	All the details should be added to the correct place in the database	Payment information MemberID: 6 PaymentDate: 17-03-2015	Normal	The data should be added into the appropriate place in the database	The correct memberID and Payment Date are added to the database with the rest of the fields remaining blank	
3.3	Verify that all Exercise details that are input are added to the database	All the details should be added to the correct place in the database	Exercise information ExerciseID: 4 Name: Push Ups	Normal	The data should be added into the appropriate place in the database	The correct Exerciser ID and Name are added to the database with the rest of the fields remaining blank	Fig. 9

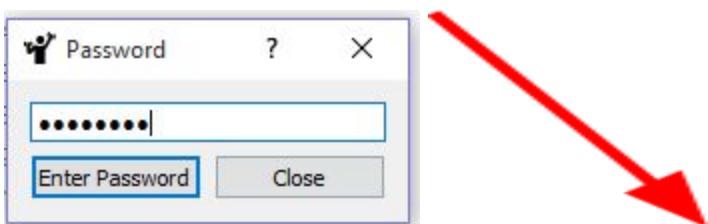
3.4	Verify that all Regime details that are input are added to the database	All the details should be added to the correct place in the database	Regime information MemberID: 3 ExerciseID: 22	Normal	The data should be added into the appropriate place in the database	The correct memberID and exerciseID are added to the database with the rest of the fields remaining blank	
3.5	Verify that all Member details that are input are updated in the database	All the correct details should be updated in the database	Member information MemberID: 6 Name: Toby Smith	Normal	The data should be edited into the appropriate place in the database	The member row with the memberID 6 has its name changed to Toby Smith	
3.6	Verify that all Payment details that are input are updated in the database	All the correct details should be updated in the database	Payment information MemberID: 5 paymentDate: 12-02-2015 Paid: 0	Normal	The data should be edited into the appropriate place in the database	The payment row with the memberID 5 and payment Date 12-02-2015 has its paid column changed to 0	
3.7	Verify that all Exercise details that are	All the correct details should be	Exercise information ExerciseID: 3	Normal	The data should be edited into the	The exercise row with the exerciseID	

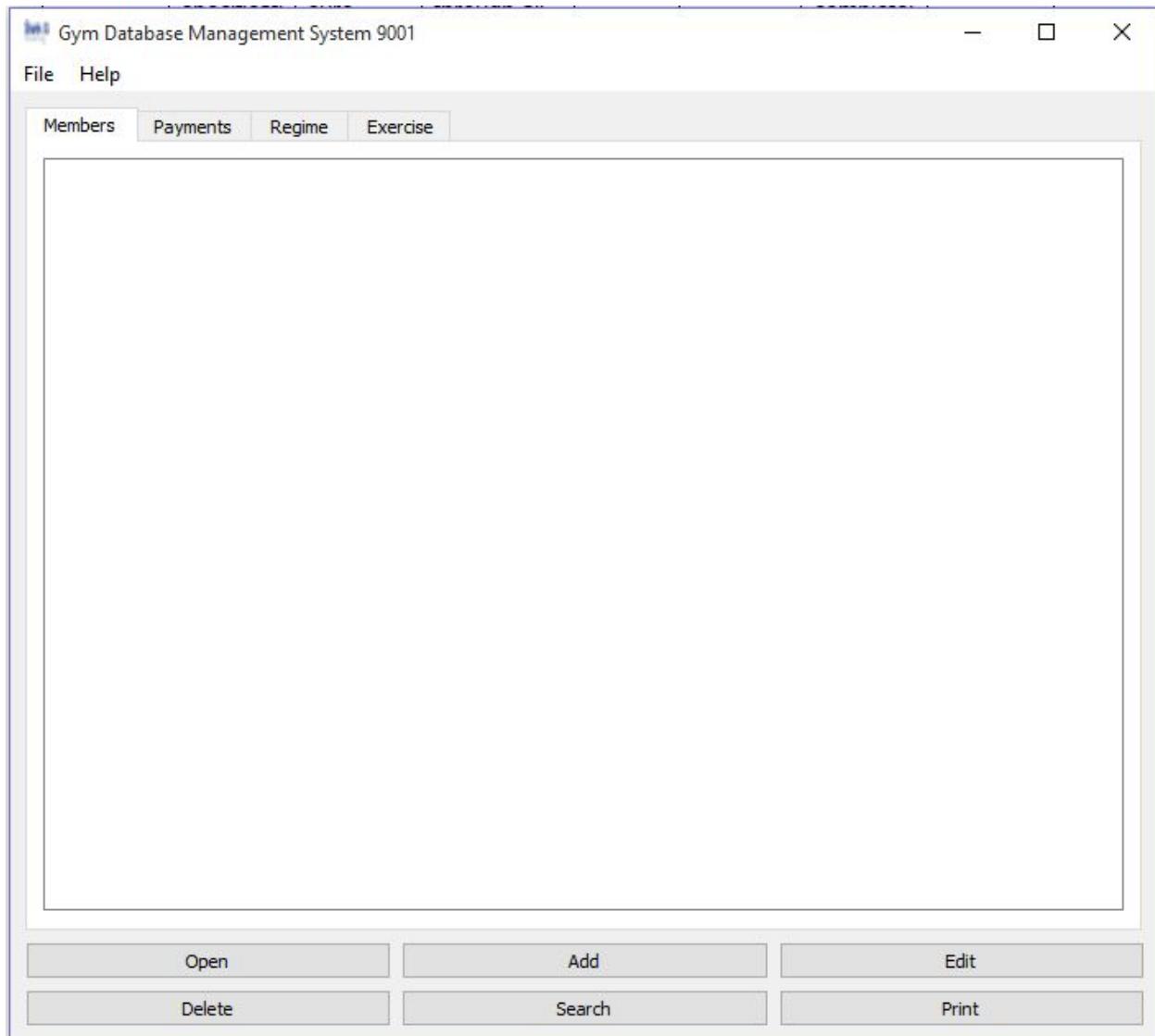
	input are updated in the database	updated in the database	Name: Pull Ups		appropriate place in the database	d 3 has its name changed to Pull Ups	
3.8	Verify that all Regime details that are input are updated in the database	All the correct details should be updated in the database	Regime information MemberID: 4 ExerciseID: 2 StartDate: 12-03-2016	Normal	The data should be edited into the appropriate place in the database	The regime row with the memberID 4 and exerciseID 2 has its startDate changed to 12-03-2016	
3.9	Verify that all Member details that are input are deleted in the database	All the correct details should be deleted from the database	Member information memberID: 5	Normal	The data should be deleted from the appropriate place in the database	The column with the memberID 5 is removed from the database	
3.10	Verify that all Payment details that are input are deleted in the database	All the correct details should be deleted from the database	Payment information memberID: 6 PaymentDate: 13-02-2016	Normal	The data should be deleted from the appropriate place in the database	The column with the memberID 6 and the Payment Date 13-02-2016 is removed from the database	

3.11	Verify that all Exercise details that are input are deleted in the database	All the correct details should be deleted from the database	Exercise information ExerciseID: 2	Normal	The data should be deleted from the appropriate place in the database	The column with the ExerciseID 2 is removed from the database	
3.12	Verify that all Regime details that are input are deleted in the database	All the correct details should be deleted from the database	Regime information MemberID: 7 ExerciseID: 4	Normal	The data should be deleted from the appropriate place in the database	The column with the memberID 7 and the ExerciseID 4 is removed from the database	
4.1	Verify that the Search function works correctly	The search function should allow the user to input a search term and then query that database and return the results	Search some information that is known to be in the database Table: Member Search Term: 1	Normal	The search function should return the correct information	The function returned all of the information from the members table with the memberID “1”	Fig. 10
4.2	Test that the Invoice function correctly	The invoice function should add	A member that the user already knows the	Normal	The function should print the correct	An Invoice containing the correct	

	calculates the total money a client needs to pay	together the sums of all the unpaid months of a member	total money owed of memberID: 1		total (£120) along with other information	total (£120) along with all the other relevant information	
4.3	Test that the Print function fetches the correct data	The Print function fetches data from the database to print in the forms	Any table and form that the user knows the output of	Normal	The function should retrieve the correct data	The function retrieves the correct data	
5	Make sure that the program fulfills the specification laid out in the analysis and design	Run through the program making sure every function and aspect meets this specification	Add some information to the database and run through all the different methods of data manipulation and view the results	Normal	Program fulfills the specification	The program fulfills the specification completely	

Fig 1.





This diagram shows how upon entering the correct password into the dialog box it will open up into the main interface allowing the user to access the rest of the program.

Fig 2.

Gym Database Management System 9001

File Help

Members Payments Regime Exercise

MemberID	Name	Address	Telephone Number	Membership Type	Induction Date	Join Date	How Paid	Amount	Registration Date
1 5	Toby	5 Henley Way	01353667827	PAYG	2015-05-05	2015-05-05	Card	50	12-04-20

< >

Open Add Edit
Delete Search Print

This diagram demonstrates how after selecting a database from the open dialog window its loaded and presented in the main interfaces tableview

Fig. 3

Gym Database Management System 9001

File Help

Members Payments Regime Exercise

MemberID	Payment Date	How Much	Paid
1 3	0	123	123

Open Add Edit
Delete Search Print

Gym Database Management System 9001

File Help

Members Payments Regime

Add

Members Add

MemberID	Payment Date
1 3	0

MemberID
Name
Address
Telephone Number
Membership Type
Induction Date
Join Date
How Paid
Amount
Registration Fee
Registration Date
Payment Type
Comments

Open Add Edit
Delete Search Print

This diagram shows the Add dialog open while the main interface remains in the background after the “Add” push button has been clicked by the user.

Fig. 4

The image displays two overlapping windows, both titled "Edit". The window on the left is titled "Members" and contains fields for MemberID, Name, Address, Telephone Number, Membership Type, Induction Date, Join Date, How Paid, Amount, Registration Fee, Registration Date, Payment Type, and Comments. The window on the right is titled "Payments" and contains fields for MemberID, Payment Date, How Much, and Paid. The "Edit" button is highlighted in blue in both windows.

This diagram above demonstrates how when changing the selected index on the combobox the input form changes in the edit dialog box.

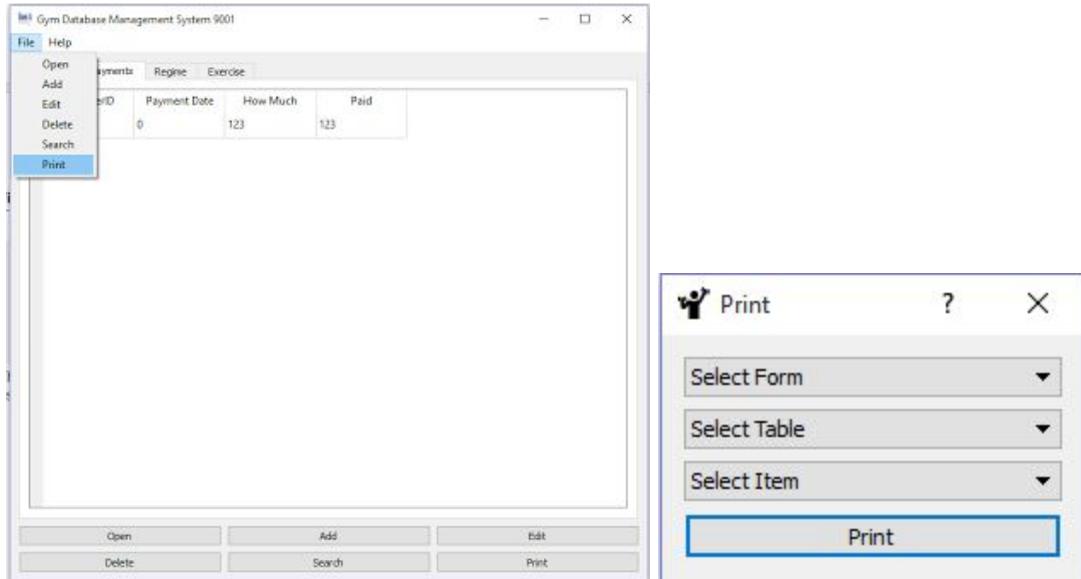
Fig. 5

The image shows two side-by-side windows. The left window has a "Select Table" dropdown set to "Select Table" and a "Select Item" dropdown set to "Select Item". It features a "Delete" button and a "Delete All Items" button, with the "Delete" button highlighted in blue. The right window has a "Select Table" dropdown set to "Exercises" and a "Select Item" dropdown set to "6. None". It also features a "Delete" button and a "Delete All Items" button, with the "Delete" button highlighted in blue.

This diagram show how after a table has been selected from the select table combo box the

select item combo box populates its index's with the different items from that table inn the currently loaded database.

Fig. 6



This image demonstrates how the print dialog open after the print shortcut is selected by the user in the file menu in the tool bar.

Fig. 7

Gym Database Management System 9001

File Help

	MemberID	Name	Address	Telephone Number	Membership Type	Induction Date	Join Date
1	5	Toby	5 Henley Way	01353667827	PAYG	2015-05-09	2015-05-09

Add

Members Add

MemberID	12
Name	Victor Bahman
Address	5 Hugo Street
Telephone Number	07598282881
Membership Type	PAYG
Induction Date	12-06-2015
Join Date	12-06-2015
How Paid	Card
Amount	50
Registration Fee	50
Registration Date	12-06-2015
Payment Type	Monthly
Comments	N/A

Edit Delete Search Print

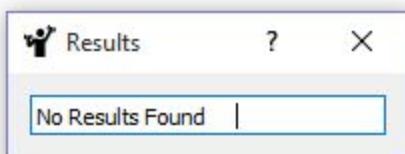
The screenshot shows a Windows application window titled "Gym Database Management System 9001". The menu bar includes "File" and "Help". Below the menu is a tab bar with "Members", "Payments", "Regime", and "Exercise", where "Members" is selected. The main area displays a table with the following data:

	MemberID	Name	Address	Telephone Number	Membership Type	Induction Date	Join Date
1	5	Toby	5 Henley Way	01353667827	PAYG	2015-05-09	2015-05-09
2	12	Victor Bahman	5 Hugo Street	07598282881	PAYG	12-06-2015	12-06-2015

Below the table are several buttons: Open, Add, Edit, Delete, Search, and Print.

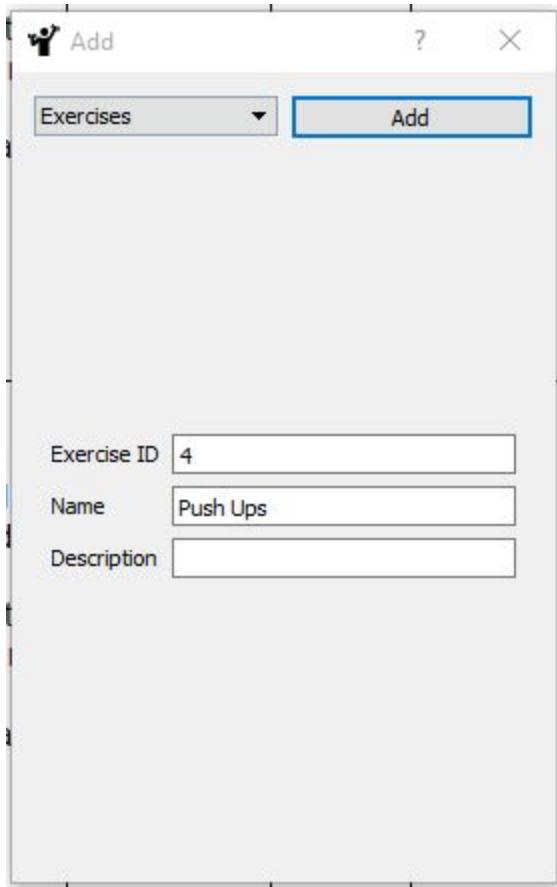
The figures above demonstrate how the information entered into the input form in the Add dialog is inserted into the currently loaded database and verifies how it's the correct data types as the program has proceeded to enter the data and not prompted the user to change their information.

Fig. 8



This figure shows the results window when the entered search term doesn't yield any results.

Fig. 9



Gym Database Management System 9001

File Help

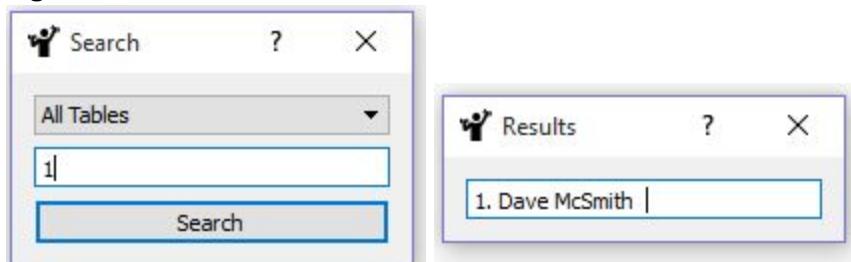
Members Payments Regime Exercise

	ExerciseID	Name	Description
1	6	None	None
2	6	None	None
3	2	22	
4	4	Push Ups	

Open Add Edit
Delete Search Print

The above figures show how after entering the test data and clicking the Add push button the data was inserted into the Exercise table in the open database.

Fig. 10



The above figure shows how when the test data is entered and search through with the programs search function the correct member information is returned.

Fig. 11

The figure consists of two screenshots of a software application. The top screenshot shows an 'Add' form for a 'Members' record. The bottom screenshot shows a table view of the 'Members' data.

Add Form (Top Screenshot):

- MemberID: 15
- Name: (empty)
- Address: (empty)
- Telephone Number: 011111922313123213123123 (highlighted with a red box)
- Membership Type: (empty)
- Induction Date: (empty)
- Join Date: (empty)
- How Paid: (empty)
- Amount: (empty)
- Registration Fee: (empty)
- Registration Date: (empty)
- Payment Type: (empty)
- Comments: (empty)

A red arrow points from the text 'Boundary Input' to the highlighted telephone number field.

Table View (Bottom Screenshot):

	MemberID	Name	Address	Telephone Numbe	Membership Type	Induction Date
4	5					
5	6					
6	7					
7	8					
8	9					
9	23					
10	24					
11	25					
12	26					
13	27					
14	28					
15	54					
16	55					
17	15			0111119223131...		

A red arrow points from the text 'Accepted Boundary Data' to the database table, indicating the successful insertion of the boundary data.

Buttons at the bottom of the table view:

- Open
- Add
- Edit
- Delete
- Search
- Print

The above figure demonstrates how after entering some boundary data into the add input form

the program incorrectly accepts it as a valid input.

Fig. 12

The figure consists of two screenshots of a software application. The top screenshot shows an 'Edit' form for a member with MemberID 1. The 'Telephone Number' field contains the invalid value '011111922313123213123123'. A red arrow points from the text 'Boundary Input' to this field. The bottom screenshot shows a list of members in a table. The 'Telephone Number' column for the second member (MemberID 2) contains the value '01353782765', which is also highlighted with a red box. A red arrow points from the text 'Accepted Boundary Data' to this value.

MemberID	Name	Address	Telephone Number	Membership Type	Induction Date	Comments
1	Toby Kerslake	4 main street	0111119223131...	PAYG	2015-02-01	
2	Dave Wrong	17 Real Street	01353782765	PAYG	2016-03-03	
3	Geoffrey Smith	17 Main Street	01364557892	PAYG	2015-02-03	
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						

The above figure demonstrates how after entering some boundary data into the edit input form the program incorrectly accepts it as a valid input.

Fig. 13

The figure consists of two windows from a Gym Database Management System.

The top window is an "Edit" dialog for a "Members" record. It contains fields for MemberID (99780453), Name (Brain Fallo), and other optional fields like Address, Telephone Number, Membership Type, Induction Date, Join Date, How Paid, Amount, Registration Fee, Registration Date, Payment Type, and Comments. The "Edit" button at the top right is highlighted in blue.

The bottom window shows a list of members in a table format. The columns are MemberID, Name, Address, Telephone Number, Membership Type, Induction Date, and a small upward arrow icon. The data in the table is as follows:

	MemberID	Name	Address	Telephone Number	Membership Type	Induction Date
1	1	Toby Kerslake	4 main street	0111119223131...	PAYG	2015-02-01
2	3	Dave Wrong	17 Real Street	01353782765	PAYG	2016-03-03
3	4	Geoffrey Smith	17 Main Street	01364557892	PAYG	2015-02-03
4	5					
5	6					
6	7					
7	8					
8	9					
9	23					
10	24					
11	25					
12	26					
13	27					
14	28					

At the bottom of the member list window are buttons for Open, Add, Edit, Delete, Search, and Print.

The above figure demonstrates how after entering data into an edit form with an invalid primary key the input gets rejected and not added to the table despite an error message not being displayed.

System Maintenance

Environment

Software used when creating my program

- Python 3 (the programming Language)
- Idle (for writing the main program python file)
- Notepad ++ (for writing the rest of the python files)
- PyQt4 (the library used for creating the graphical user interface (GUI))
- SQLite3 (The library used to create the sql database in python)
- CX_Freeze (For compiling my program)

Explanation of Usage

Python 3:

- The language I'm most familiar with
- Open source meaning it's free for commercial use
- Portable meaning it's compatible with most operating systems, even Dos
- Object Oriented and Event Driven meaning it was easy to create this sort of program
- A vast amount of pre existing libraries meaning a large amount of modules are usable

IDLE:

- Automatically downloaded with Python
- Designed especially for python
- Easy to execute code as you don't have to assign a path like in notepad++ or sublime text
- User friendly design and layout
- Has a debug feature making testing easier

Notepad++:

- Tabular layout which makes it more manageable to work with multiple files at once
- More advanced features than IDLE like blank operations

PyQt4:

- Designed to easily allow the implementation of GUI in Python

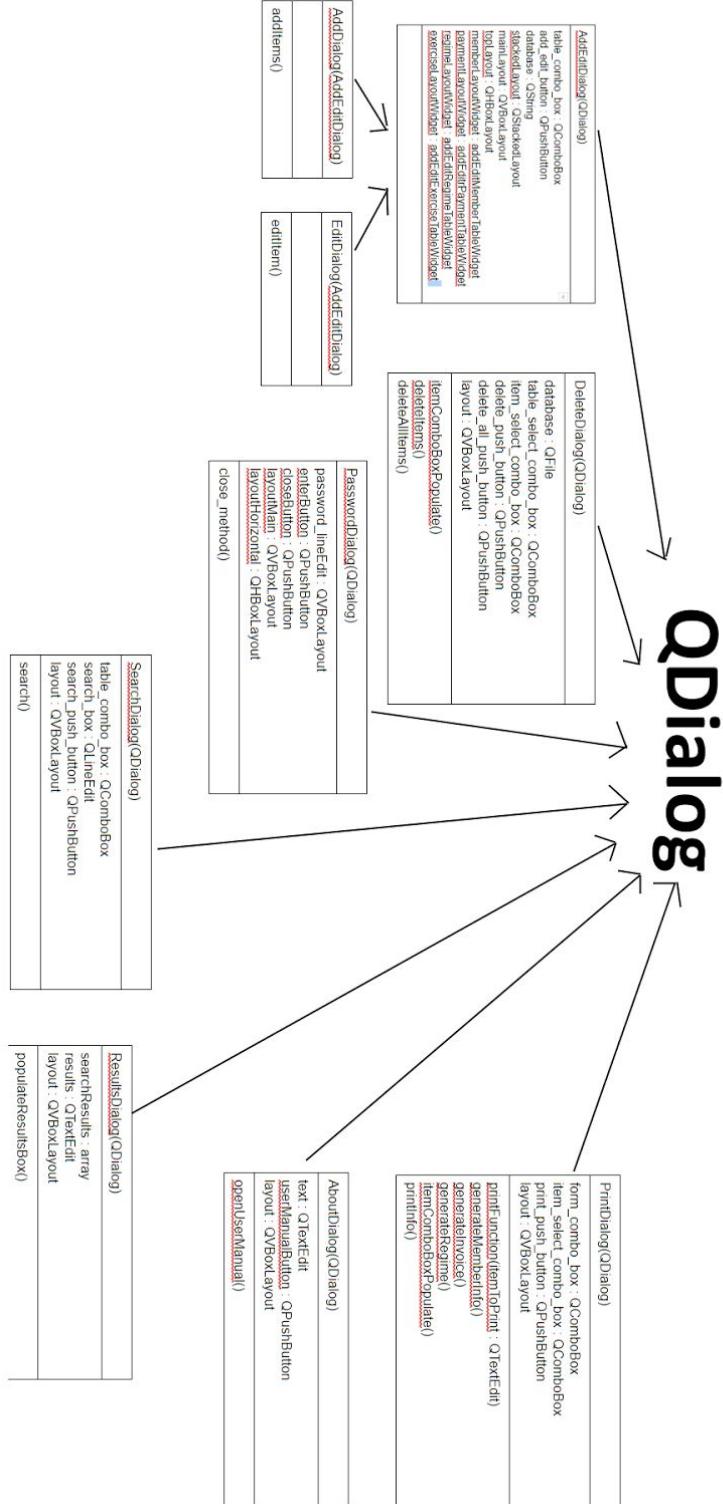
SQLite3:

- Simple but usable version of MySQL allowing me to make databases easily but lacks some more advanced features
- Lacks security but this has been made up for by security in the main program those this made it easier to create the program and the database
- Already installed with Python3

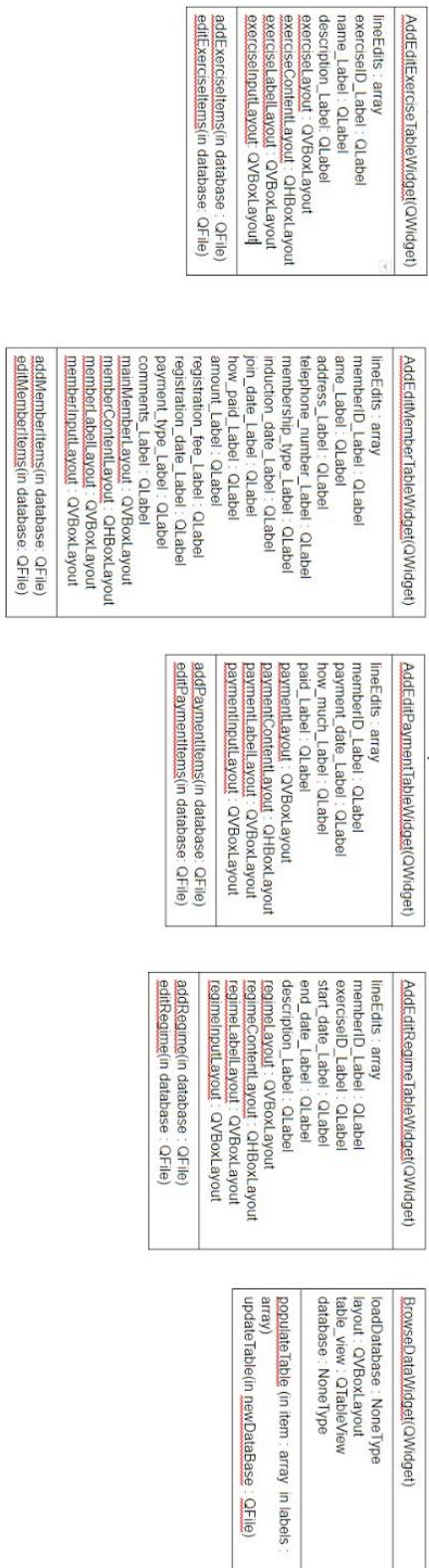
CX_Freeze

- Used to compile python programs
- Compatible with Python 3 unlike most python compilers that only work with Python 2
- Make it easier to distribute to my client and to install on my client's computer
- Allows the inclusion of images, databases and other file types

Class Diagrams



QWidget



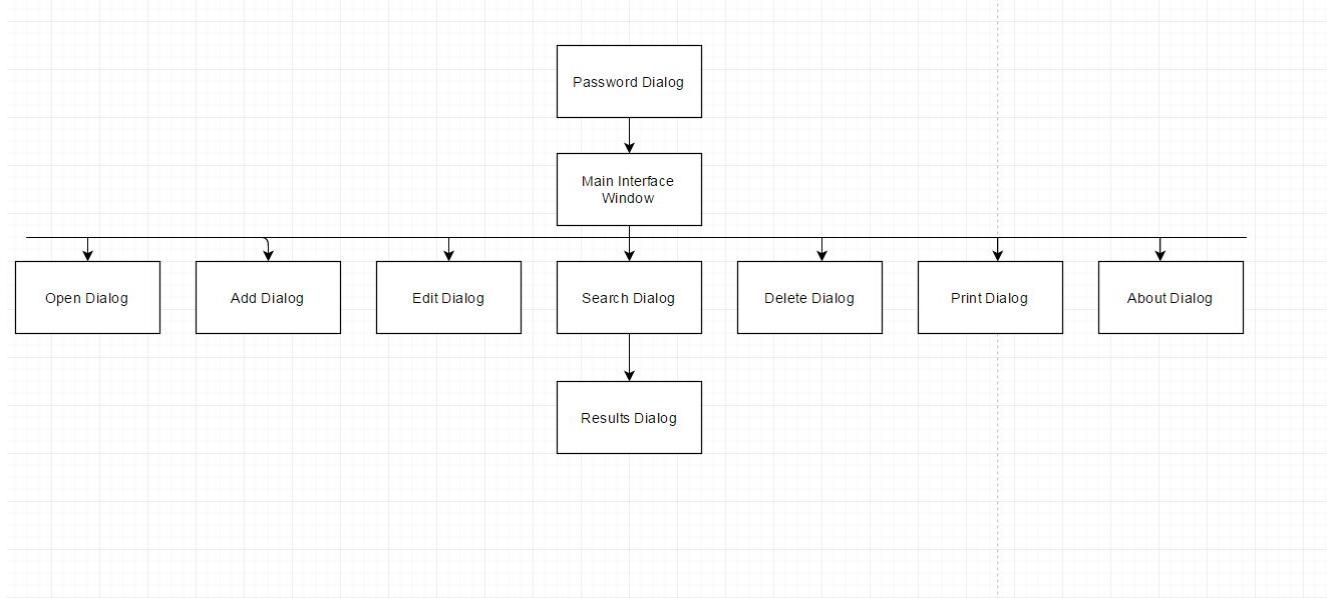
QMainWindow

```
AppWindow(QMainWindow)
file : NoneType
open_push_button : QPushButton
add_push_button : QPushButton
edit_push_button : QPushButton
search_push_button : QPushButton
delete_push_button : QPushButton
print_push_button : QPushButton
tab_bar : QTabWidget
tabs : array
tabNames : array
labels : nested array
toolBar : QToolBar
layout1 : QHBoxLayout
layout2 : QHBoxLayout
layout3 : QVBoxLayout
mainWidget : QWidget
mainWidget()

open_file_menu()
delete()
print_stuff()
search()
edit()
add()
about_the_program()
password(currentPass : string)
main()
```

```
Database()
db_name : QFile
loadDatabase()
get>All>Data(in table : string)
```

Navigation Diagram



CODE STRUCTURE

All of the code reviewed in this part of the system maintenance section is available in the Implementation section.

Add Edit Member Table Widget

This code is structured as a class so that it can parent both add and edit table widgets in the system. I did this because not only does it prevent duplication of code in the system as the Member Table Widget needs to be used for the Add and Edit input forms. This also reduced development time and makes debugging errors easier and quicker and still provides the full functionality required.

Add Edit Payment Table Widget

This code is structured as a class so that it can parent both add and edit table widgets in the system. I did this because not only does it prevent duplication of code in the system as the Payment Table Widget needs to be used for the Add and Edit input forms. This also reduced development time and makes debugging errors easier and quicker and still provides the full functionality required.

Add Edit Regime Table Widget

This code is structured as a class so that it can parent both add and edit table widgets in the system. I did this because not only does it prevent duplication of code in the system as the Regime Table Widget needs to be used for the Add and Edit input forms. This also reduced

development time and makes debugging errors easier and quicker and still provides the full functionality required.

Add Edit Exercise Table Widget

This code is structured as a class so that it can parent both add and edit table widgets in the system. I did this because not only does it prevent duplication of code in the system as the Exercise Table Widget needs to be used for the Add and Edit input forms. This also reduced development time and makes debugging errors easier and quicker and still provides the full functionality required.

Browse Data Widget

This code is used every time a database is loaded so it's structured to be efficient. It does this by inheriting functions from the gym database class and its only use is to load a database and set its layout.

Gym Password Dialog Class

This code is used frequently throughout the program when it's first opened and when the delete function is started. Its structured to display the password dialog, close, and if the password was entered incorrectly the repetition of the window is performed in a for loop where the class is called and not within the class itself.

Gym Search Dialog Class

This code is used to search through the database. It calls certain function from the Search Function in order for the code to be easily reusable.

Gym Search Results Dialog Class

This code is used to display the results of a user's search query.

Gym About Dialog Class

This code is used to display the about section of the system. It imports the webbrowser module in order to open a link to the user manual.

Gym Add Dialog Class

This class contains a small amount of code just to make the Gym Add Edit Dialog Class (which it inherits) specific to an Add dialog.

Gym Add Edit Dialog Class

This code is structured as a class so that it can parent both add and edit dialog classes in the system. I did this because not only does it prevent duplication of code in the system as it needs to be used for the Add and Edit input forms. This also reduced development time and makes debugging errors easier and quicker and still provides the full functionality required.

Gym Edit Dialog Class

This class contains a small amount of code just to make the Gym Add Edit Dialog Class (which it inherits) specific to an Edit dialog.

Gym Database Class

This code is used to load a database and to use sql to get data from a specific table in the database. Its methods are inherited by the Browse Data Widget to separate the PyQt operations and the SQL functions to make the code more maintainable and more reusable.

Gym Delete Dialog Class

This code is used to delete items through the database. It calls certain function from the Delete Function in order for the code to be easily reusable.

VARIABLE LIST

AddEditMemberTableWidget

Variable	Purpose
self.lineEdits	An array to store a series of other variables assigned to the values self.enter_text{0}.format(count)
self.memberID_Label	A string of text letting the user know that the QLineEdit adjacent is for the user to enter a memberID
self.name_Label	A string of text letting the user know that the QLineEdit adjacent is for the user to enter a name
self.address_Label	A string of text letting the user know that the QLineEdit adjacent is for the user to enter an address
self.telephone_number_Label	A string of text letting the user know that the QLineEdit adjacent is for the user to enter a telephone number
self.membership_type_Label	A string of text letting the user know that the QLineEdit adjacent is for the user to enter a membership type
self.induction_date_Label	A string of text letting the user know that the QLineEdit adjacent is for the user to enter an

	induction date
self.join_date_Label	A string of text letting the user know that the QLineEdit adjacent is for the user to enter a join date
self.how_paid_Label	A string of text letting the user know that the QLineEdit adjacent is for the user to enter a method of payment
self.amount_Label	A string of text letting the user know that the QLineEdit adjacent is for the user to enter a amount of money
self.registration_fee_Label	A string of text letting the user know that the QLineEdit adjacent is for the user to enter a registration fee
self.registration_date_Label	A string of text letting the user know that the QLineEdit adjacent is for the user to enter a registration date
self.payment_type_Label	A string of text letting the user know that the QLineEdit adjacent is for the user to enter a payment type
self.comments_Label	A string of text letting the user know that the QLineEdit adjacent is for the user to enter any additional comments
self.mainMemberLayout	The main Vertical Box layout for the member table widget for all the other layouts to be added to
self.memberContentLayout	A Horizontal Box layout for the label and input layouts to be added to
self.memberLabelLayout	A Vertical Box layout for all the Label QLabels to be added to
self.memberInputLabelLayout	A Vertical Box Layout for all the Input QLineEdits to be added to
columns	A local variable to list the required names of all the columns in the member table to pass into an sql statement
items	A string for all the sql for the items being edited to be passed into the editMember

	function
count	A variable for counting through the number 1 through 12

AddEditPaymentTableWidget

Variable	Purpose
self.lineEdits	An array to store a series of other variables assigned to the values self.enter_text{0}.format(count)
self.memberID_Label	A string of text letting the user know that the QLineEdit adjacent is for the user to enter a memberID
self.payment_date_Label	A string of text letting the user know that the QLineEdit adjacent is for the user to enter a payment date
self.how_much_Label	A string of text letting the user know that the QLineEdit adjacent is for the user to enter an amount of money
self.paid_Label	A string of text letting the user know that the QLineEdit adjacent is for the user to enter a boolean value describing whether a client has paid
self.paymentLayout	The main Vertical Box layout for the payment table widget for all the other layouts to be added to
self.paymentContentLayout	A Horizontal Box layout for the label and input layouts to be added to
self.paymentLabelLayout	A Vertical Box layout for all the Label QLabels to be added to
self.paymentInputLabelLayout	A Vertical Box Layout for all the Input QLineEdits to be added to
columns	A local variable to list the required names of all the columns in the payment table to pass into an sql statement

items	A string for all the sql for the items being edited to be passed into the editPayment function
count	A variable for counting through the number 2 through 3

AddEditRegimeTableWidget

Variable	Purpose
self.lineEdits	An array to store a series of other variables assigned to the values self.enter_text{0}.format(count)
self.memberID_Label	A string of text letting the user know that the QLineEdit adjacent is for the user to enter a memberID
self.exerciseID_Label	A string of text letting the user know that the QLineEdit adjacent is for the user to enter an exerciseID
self.start_date_Label	A string of text letting the user know that the QLineEdit adjacent is for the user to enter a start date
self.end_date_Label	A string of text letting the user know that the QLineEdit adjacent is for the user to enter an end date
self.description_Label	A string of text letting the user know that the QLineEdit adjacent is for the user to enter a description
self.regimeLayout	The main Vertical Box layout for the regimetable widget for all the other layouts to be added to
self.regimeContentLayout	A Horizontal Box layout for the label and input layouts to be added to
self.regimeLabelLayout	A Vertical Box layout for all the Label QLabels to be added to
self.regimeInputLabelLayout	A Vertical Box Layout for all the Input QLineEdits to be added to

columns	A local variable to list the required names of all the columns in the regime table to pass into an sql statement
items	A string for all the sql for the items being edited to be passed into the editRegimefunction
count	A variable for counting through the number 2 through 4

AddEditExerciseTableWidget

self.lineEdits	An array to store a series of other variables assigned to the values self.enter_text{0}.format(count)
self.exerciseID_Label	A string of text letting the user know that the QLineEdit adjacent is for the user to enter a exerciseID
self.name_Label	A string of text letting the user know that the QLineEdit adjacent is for the user to enter a name
self.description_Label	A string of text letting the user know that the QLineEdit adjacent is for the user to enter a description of an exercise
self.exerciseLayout	The main Vertical Box layout for the exercise table widget for all the other layouts to be added to
self.exerciseContentLayout	A Horizontal Box layout for the label and input layouts to be added to
self.exerciseLabelLayout	A Vertical Box layout for all the Label QLabels to be added to
self.exerciseInputLabelLayout	A Vertical Box Layout for all the Input QLineEdits to be added to
columns	A local variable to list the required names of all the columns in the exercise table to pass into an sql statement
items	A string for all the sql for the items being

	edited to be passed into the editExerciseFunction
count	A variable for counting through the number 1 through 2

BrowseDataWidget

Variable	Purpose
self.loadDataBase	A variable for the currently loaded database
self.layout	A Vertical Box layout for the databrowser
self.table_view	A table view to display the sql table appropriately
self.database	A variable for the database in use
data	A local variable to store all the data from the current database in an array
model	A local variable to set the model of the format of the data from the sql database
row	A local variable initially set to 0 to increment to set items in each row
item	A local variable that stores the current item from all the data array
column	A local variable initially set to 0 to increment to set items in each column
standardItem	A local variable to set the item to the table view

PrintDialog

Variable	Purpose
self.form_combo_box	A combo box to hold the different forms available for the user to print

self.item_select_combo_box	A combo box to hold the different items available for the user to print
self.print_push_button	A Push Button for the user to confirm what they want to print and open a print dialog
self.layout	A Vertical Box layout for the PrintDialog
dialog	A local variable that's assigned the QPrintDialog
itemToPrint	A QTextEdit of the items to print passed into the function
columns	A local variable that stores all the column names for the member table
info	A local variable that holds the return value from a function that gets all the required data
infoToPrint	A local variable for a text edit to hold all the information being printed
infoString	A local variable that's a string for all the data going into the text edit
item	A local variable that's assigned an item from the info array
count	A local variable that's incremented to assign a value from the columns array to add to the infoString variable
piece	A local variable that's assigned to each piece of data in the item array
name	A local variable that assigns the list of the member's name
items	A local variable that assigns the items returned by the getItems() function
list	A local variable that's assigned to the list inside the name variable
word	A local variable that's assigned to the name inside the variable list

PasswordDialog

Variable	Purpose
self.password_lineEdit	A QLineEdit for the password to be entered by the user
self.enterButton	A QPushButton for the user to click to enter the password they've input
self.closeButton	A QPushButton that allows the user to close the dialog
self.layoutMain	The main layout for the dialog box
self.layoutHorizontal	A Horizontal layout for the enter and close push buttons to be added to

ResultsDialog

Variable	Purpose
self.searchResults	An array variable for assigning the search results from the search function
self.results	A QTextEdit that stores all the results in a text format
self.layout	the layout for the results dialog
string	A Local variable which stores each item in a string format
item	A local variable that's assigned to each result from the results array

SearchDialog

Variable	Purpose
self.table_combo_box	A combo box for allowing the user to select the table they want
self.search_box	A line edit for the user to enter their search

	term
self.search_push_button	A push button for the user to confirm their search
self.layout	A vertical box layout for all the widgets to be added to
results	A local variable that assigns all the data returned by the SearchQuery function
resultsDialog	A local variable assigned to an execution of the resultsDialog imported from the gym_search_results_dialog_class file

AboutDialog

Variable	Purpose
self.text	A text edit to store the appropriate text to describe the program to the user
self.userManualButton	A push button allowing the user to open a link to download a pdf of the user manual
self.layout	A Vertical layout for all the widgets to be added to

AddDialog

Variable	Purpose
No Variables	

EditDialog

Variable	Purpose
No Variables	

AddEditDialog

Variable	Purpose
self.table_combo_box	A combo box to allow the user to select a table
self.add_edit_button	A push button that confirms the user's input data
self.database	A variable that's assigned the current database
self.stackLayout	A stacked layout to assign all of the different widgets used
self.mainLayout	The main layout for all the other layouts to be added to
self.topLayout	The layout for the add_edit_button and table_combo_box to be added to
self.memberLayoutWidget	A widget that contains all the widgets for an input form for a member input
self.paymentLayoutWidget	A widget that contains all the widgets for an input form for a payment input
self.regimeLayoutWidget	A widget that contains all the widgets for an input form for a regime input
self.exerciseLayoutWidget	A widget that contains all the widgets for an input form for an exercise input

Database

Variable	Purpose
self.db_name	A variable for assigning the name of the opened database
db	A local variable for assigning the open database
cursor	A local variable for assigning the cursor for the sql statements
allData	A local variable for assigning all the data from the database

query	A local variable for assigning the query
data	A local variable to assign an array of data in a for loop
item	A local variable to assign an item from the data array in a for loop

DeleteDialog

Variable	Purpose
self.table_select_combo_box	A Combo box allowing the user to select a table
self.item_select_combo_box	A combo box allowing the user to select an item
self.delete_push_button	A push button to confirm the users wants to delete a piece of data
self.delete_all_push_button	A push button for the user the delete all data in a table
self.layout	A variable for assigning the dialogs Vertical layout
items	A local variable for assigning an array of all the items from the selected table
count	A local variable to be incremented in a for loop
sep	A local variable assigned to a string that indicates where an item should be split to be passed as a parameter into a function

AppWindow

Variable	Purpose
self.file	A variable that stores the opened database file
self.open_push_button	A push button that lets the user start the open function

self.add_push_button	A push button that lets the user start the add function
self.edit_push_button	A push button that lets the user start the edit function
self.delete_push_button	A push button that lets the user start the delete function
self.search_push_button	A push button that lets the user start the search function
self.print_push_button	A push button that lets the user start the print function
self.tab_bar	A tab bar that allows the user to select a table to view
self.tabs	A variable that assigns the tabs
self.tabNames	A variable that assigns an array for the tab names
count	A local variable that increments from 1 in a for loop
self.labels	A variable to assign an array for the labels
self.toolBar	A toolbar that contains file and help options
self.file_menu	The file menu in the toolbar
self.help_menu	The help menu in the toolbar
self.about	The about option from the help menu
self.open_shortcut	The open option from the file menu
self.add_shortcut	The add option from the file menu
self.edit_shortcut	The edit option from the file menu
self.delete_shortcut	The delete option from the file menu
self.search_shortcut	The search option from the file menu
self.print_shortcut	The print option from the file menu
self.layout1	The main vertical layout for main window

self.layout2	The second layout for the main window
self.layout3	The third layout for the main window
self.mainWidget	The main widget for the app window
delete_dialog	A local variable that assigns a delete dialog
print_dialog	A local variable that assigns a print dialog
search_dialog	A local variable that assigns a search dialog
edit_dialog	A local variable that assigns an edit dialog
add_dialog	A local variable that assigns an add dialog
about_dialog	A local variable that assigns an about dialog
passWord	A local variable that assigns what the current password for the current password dialog
currentPass	A local variable that assigns the return from the print_dialog that returns what the user enters into the password dialog
password_dialog	A local variable that assigns a password dialog
gym_program	A local variable that assigns the main execution of the program
gym_window	A local variable that assigns the main window of the program

EXPLANATION OF DETAILED ALGORITHMS

Main Window Class

```
1. import sys
2.
3. from gym_delete_dialog_class import *
4. from gym_print_dialog_class import *
5. from gym_search_dialog_class import *
6. from gym_edit_dialog_class import *
7. from gym_add_edit_dialog_class import *
8. from gym_add_dialog_class import *
9. from gym_about_dialog_class import *
10. from gym_password_dialog_class import *
11.
12. from PyQt4.QtCore import *
13. from PyQt4.QtGui import *
14.
15. class AppWindow(QMainWindow):
16.     """creates the main window"""
17.
18.     #constructor
19.     def __init__(self):
20.         super().__init__()
21.         self.setWindowTitle("Gym Database Management System")#sets the title for the window
22.         self.setWindowIcon(QIcon("Logo.png"))#sets the window icon
23.
24.         #variable for database
25.         self.file = None
26.
27.         #toolbars
28.         self.open_push_button = QPushButton("Open")#sets the main button for opening the Open GUI and
function
29.         self.add_push_button = QPushButton("Add")#sets the main button for opening the Add GUI and function
30.         self.edit_push_button = QPushButton("Edit")#sets the main button for opening the Edit GUI and function
31.         self.delete_push_button = QPushButton("Delete")#sets the main button for opening the Delete GUI and
function
```

```

32.     self.search_push_button = QPushButton("Search")#sets the main button for opening the Search GUI and
function
33.     self.print_push_button = QPushButton("Print")#sets the main button for opening the Print GUI and
function
34.
35.     self.tab_bar = QTabWidget() #creates the widget to display the tables in a tabular layout
36.
37.     self.tabs = {}#creates a dictionary for the table names/tab names
38.     self.tabNames = ['Members','Payments','Regime','Exercise']#The 4 tab/table names
39.     for count in range(4):
40.         self.tabs["{0}".format(self.tabNames[count])] = BrowseDataWidget()
41.
42.         self.tab_bar.addTab(self.tabs["{0}".format(self.tabNames[count])],"{0}".format(self.tabNames[count]))
43. #a for loop that creates a new tab and adds a "BrowseDataWidget" into each of them
44.
45.     self.labels = {"Members": ["MemberID", "Name", "Address", "Telephone Number", "Membership
46. Type", "Induction Date", "Join Date", "How Paid", "Amount", "RegistrationFee", "Registration
47. Date", "PaymentType", "Comments"], "Payments": ["MemberID", "Payment Date", "How Much", "Paid"], "Regime": ["MemberID", "ExerciseID", "Specific Description", "Start Date", "End Date"], "Exercise": ["ExerciseID", "Name", "Description"]}
48. #labels for the table headers that are referenced later in the program
49.
50.     self.toolBar = QMenuBar()#creates a menu bar
51.     self.file_menu = self.toolBar.addMenu("File")#adds File option to the tool bar
52.     self.help_menu = self.toolBar.addMenu("Help")#adds Help option to the tool bar
53.     self.about = self.help_menu.addAction("About Gym Database Management System 9001")#adds options
into the Help menu
54.     self.open_shortcut = self.file_menu.addAction("Open")#adds Open shortcut to the File menu
55.     self.add_shortcut = self.file_menu.addAction("Add")#adds Add shortcut to the File menu
56.     self.edit_shortcut = self.file_menu.addAction("Edit")#adds Edit shortcut to the File menu
57.     self.delete_shortcut = self.file_menu.addAction("Delete")#adds Delete shortcut to the File menu
58.     self.search_shortcut = self.file_menu.addAction("Search")#adds Search shortcut to the File menu
59.     self.print_shortcut = self.file_menu.addAction("Print")#adds Print shortcut to the File menu
60.
61.
62.     #layout
63.     self.layout1 = QHBoxLayout()
64.     self.layout2 = QHBoxLayout()

```

```

65.     self.layout3 = QVBoxLayout()
66.     self.layout1.addWidget(self.open_push_button)
67.     self.layout1.addWidget(self.add_push_button)
68.     self.layout1.addWidget(self.edit_push_button)
69.
70.     self.layout2.addWidget(self.delete_push_button)
71.     self.layout2.addWidget(self.search_push_button)
72.     self.layout2.addWidget(self.print_push_button)
73.
74.     self.layout3.addWidget(self.tab_bar)
75.     self.layout3.setLayout(self.layout1)
76.     self.layout3.setLayout(self.layout2)
77.
78.     self.mainWidget = QWidget()
79.     self.setMenuWidget(self.toolBar)
80.     self.mainWidget.setLayout(self.layout3)
81.     self.setCentralWidget(self.mainWidget)
82.
83.     #connections
84.     #connections linking the main pushbuttons to their respective functions
85.     self.open_push_button.clicked.connect(self.open_file_menu)
86.     self.delete_push_button.clicked.connect(self.delete)
87.     self.print_push_button.clicked.connect(self.print_stuff)
88.     self.search_push_button.clicked.connect(self.search)
89.     self.edit_push_button.clicked.connect(self.edit)
90.     self.add_push_button.clicked.connect(self.add)
91.     self.about.triggered.connect(self.about_the_program)
92.     self.open_shortcut.triggered.connect(self.open_file_menu)
93.     self.add_shortcut.triggered.connect(self.add)
94.     self.edit_shortcut.triggered.connect(self.edit)
95.     self.delete_shortcut.triggered.connect(self.delete)
96.     self.search_shortcut.triggered.connect(self.search)
97.     self.print_shortcut.triggered.connect(self.print_stuff)
98.     #Keyboard Shortcuts for accessing the 6 main Functions
99.     self.connect(QShortcut(QKeySequence("Ctrl+o"), self), SIGNAL('activated()'), self.open_file_menu)
100.    self.connect(QShortcut(QKeySequence("Ctrl+a"), self), SIGNAL('activated()'), self.add)
101.    self.connect(QShortcut(QKeySequence("Ctrl+e"), self), SIGNAL('activated()'), self.edit)
102.    self.connect(QShortcut(QKeySequence("Ctrl+d"), self), SIGNAL('activated()'), self.delete)
103.    self.connect(QShortcut(QKeySequence("Ctrl+s"), self), SIGNAL('activated()'), self.search)

```

```

104.     self.connect(QShortcut(QKeySequence("Ctrl+p"), self), SIGNAL('activated()'), self.print_stuff)
105.     self.connect(QShortcut(QKeySequence("Ctrl+h"), self), SIGNAL('activated()'), self.about_the_program)
106.
107.
108.
109.     def open_file_menu(self):
110.         self.file = QFileDialog.getOpenFileName(caption="Open Database", filter = "Database file (*.db
*.dat)")#opens the windows folder tree allowing the user to select the database they want to open. File type
restricted to .db or .dat files
111.         try:#Restricts the user to only opening correct database or a version of it
112.             for item in self.tabNames:
113.                 self.tabs["{0}".format(item)].UpdateTable(self.file)#loads selected database into tabs
114.                 self.tabs["{0}".format(item)].PopulateTable(item, self.labels[item])#populates the tabs with
relevant information
115.             except NameError:
116.                 return
117.             except sqlite3.OperationalError:
118.                 return
119.
120.
121.     def delete(self):
122.         self.password("niel")#sets delete password to "niel" and opens the password function
123.         delete_dialog = DeleteDialog(self.file)#opens delete dialog
124.         delete_dialog.exec_()
125.
126.     def print_stuff(self):
127.         print_dialog = PrintDialog(self.file)#opens print dialog
128.         print_dialog.exec_()
129.
130.     def search(self):
131.         search_dialog = SearchDialog(self.file)#opens search dialog
132.         search_dialog.exec_()
133.
134.     def edit(self):
135.         edit_dialog = EditDialog(self.file)#opens edit dialog
136.         edit_dialog.exec_()
137.
138.     def add(self):
139.         add_dialog = AddDialog(self.file)#opens add dialog

```

```

140.     add_dialog.exec_()
141.
142.     def about_the_program(self):
143.         about_dialog = AboutDialog()#opens about dialog
144.         about_dialog.exec_()
145.
146.     def password(self,currentPass):
147.         passWord = """#sets entered password to the wrong password
148.         while passWord != currentPass:#while the correct password hasn't been entered
149.             password_dialog = PasswordDialog()#opens password dialog
150.             password_dialog.exec_()
151.             passWord = password_dialog.close_method()
152.
153.
154. def main():
155.     gym_program = QApplication(sys.argv)#creates application
156.     gym_window = AppWindow()#creates Main Window
157.     gym_window.resize(700,600)#Locks initial window size
158.     password_dialog = gym_window.password("a")
159.     gym_window.show()
160.     gym_window.raise_()
161.     gym_program.exec_()
162.
163.
164. if __name__ == "__main__":
165.     main()

```

This is the main window class for the main interface and the centre of the entire program. Here I use several functions to allow the user to start each of the main functions and several layouts to appropriately assign the buttons, the toolbar and the Table View. In this code the class inherits QMainWindow instead of QDialog since it's the main gui.

Print Dialog Class

```

1.  from PyQt4.QtGui import *
2.  from PyQt4.QtCore import *
3.  from gym_print_sql_function import *
4.  from gym_delete_function import *
5.
6.  class PrintDialog(QDialog):

```

```

7.     """This class creates the dialog window for creating forms and printing them"""
8.
9.     def __init__(self,database):
10.         super().__init__()
11.
12.         self.database = database
13.
14.         #create widgets
15.         self.form_combo_box = QComboBox()
16.         self.item_select_combo_box = QComboBox()
17.         self.print_push_button = QPushButton("Print")
18.
19.         self.form_combo_box.addItem("Select Form")
20.         self.form_combo_box.addItem("Invoice")
21.         self.form_combo_box.addItem("Member Details")
22.         self.form_combo_box.addItem("Regime")
23.
24.
25.         #create layout
26.         self.layout = QVBoxLayout()
27.
28.         #add widgets to layout
29.         self.layout.addWidget(self.form_combo_box)
30.         self.layout.addWidget(self.item_select_combo_box)
31.         self.layout.addWidget(self.print_push_button)
32.
33.         #set the window layout
34.         self.setLayout(self.layout)
35.         self.setWindowTitle("Print")
36.         self.setWindowIcon(QIcon("Hugobells.png"))
37.
38.         #connections
39.         self.form_combo_box.currentIndexChanged.connect(self.itemComboBoxPopulate)
40.         self.print_push_button.clicked.connect(self.printInfo)
41.
42.     def printFunction(self,itemToPrint):
43.         #function that sends the textEdit to be printed and allows the user to select a printer through the print dialog
44.         dialog = QPrintDialog()
45.         if dialog.exec_() == QDialog.Accepted:

```

```

46.     itemToPrint.print_(dialog.printer())
47.
48.     def generateMemberInfo(self):
49.         #method that creates the textEdit containg the correct information for a member info print out
50.         columns
51.         =["MemberID","Name","Address","TelephoneNumber","MembershipType","InductionDate","JoinDate","HowPaid","Amount",
52.          "RegistrationFee","RegistrationDate","PaymentType","Comments"]
53.         sep = "."
54.         info = getMemberInfo(self.database, str(self.item_select_combo_box.currentText()).split(sep,1)[0])
55.         infoToPrint = QTextEdit()
56.         infoString = ""
57.         for item in info:
58.             count = 0
59.             for piece in item:
60.                 infoToPrint.setText(infoToPrint.toPlainText()+columns[count]+": "+str(piece)+"\n")
61.                 count += 1
62.         self.printFunction(infoToPrint)
63.
64.     def generateInvoice(self):
65.         #method that creates the textEdit containg the correct information for a print out of an invoice
66.         columns = ["MemberID","Payment Date","How Much","Paid"]
67.         info,name = getInvoice(self.database, str(self.item_select_combo_box.currentText()).split(sep,1)[0])
68.         infoToPrint = QTextEdit()
69.         for list in name:
70.             for word in list:
71.                 for item in word:
72.                     infoToPrint.setText("Member Name : "+str(item)+"\n\n")
73.                     for item in info:
74.                         count = 0
75.                         for piece in item:
76.                             infoToPrint.setText(infoToPrint.toPlainText()+columns[count]+": "+str(piece)+"\n")
77.                             count += 1
78.                             infoToPrint.setText(infoToPrint.toPlainText()+"\n")
79.         self.printFunction(infoToPrint)
80.
81.     def generateRegime(self):
82.         #method that creates the textEdit containg the correct information for a members regime print out
83.         columns = ["MemberID","ExerciseID","Description","Start Date","End Date"]
84.         info,name = getRegime(self.database, str(self.item_select_combo_box.currentText()).split(sep,1)[0])

```

```

83.     infoToPrint = QTextEdit()
84.     for list in name:
85.         for word in name:
86.             for item in word:
87.                 infoToPrint.setText("Member Name : "+str(item)+"\n\n")
88.     for item in info:
89.         count = 0
90.         for piece in item:
91.             infoToPrint.setText(infoToPrint.toPlainText()+columns[count]+": "+str(piece)+"\n")
92.             count += 1
93.         infoToPrint.setText(infoToPrint.toPlainText()+"\n")
94.     self.printFunction(infoToPrint)
95.
96.     def itemComboBoxPopulate(self):
97.         if self.form_combo_box.currentIndex() == 0:
98.             self.item_select_combo_box.clear()
99.             self.item_select_combo_box.addItem("Select Item")
100.        else:
101.            items = getItems(self.database, "MEMBERS")
102.            self.item_select_combo_box.clear()
103.            for count in range(len(items)):
104.                self.item_select_combo_box.addItem(items[count])
105.
106.    def printInfo(self):
107.        if self.form_combo_box.currentIndex() == 1:
108.            self.generateInvoice()
109.        if self.form_combo_box.currentIndex() == 2:
110.            self.generateMemberInfo()
111.        if self.form_combo_box.currentIndex() == 3:
112.            self.generateRegime()

```

This code is for the print dialog. This uses 3 methods for getting and organising the information for the 3 different printable forms as well as another method for actually printing the form. These methods inherit multiple functions from the gym_print_sql_function.

Gym Edit Function

```
1. import sqlite3
2.
3. def editMember(database,items,memberID):
4.     #function for editing items in the member table
5.     con = sqlite3.connect(database)
6.
7.     with con:
8.
9.         sql = "UPDATE Members SET "+items+" WHERE MemberID = "+memberID
10.        cur = con.cursor()
11.        cur.execute(sql)
12.
13.    def editPayment(database,items,memberID,paymentDate):
14.        #function for editing items in the payment table
15.        con = sqlite3.connect(database)
16.
17.        with con:
18.            sql = "UPDATE Payments SET "+items+" WHERE MemberID="+memberID+" AND PaymentDate = "+paymentDate
19.            cur = con.cursor()
20.            cur.execute(sql)
21.
22.    def editRegime(database,items,memberID,exerciseID):
23.        #function for editing items in the regimetable
24.        con = sqlite3.connect(database)
25.
26.        with con:
27.            sql = "UPDATE Regime SET "+items+" WHERE MemberID = "+memberID+" AND ExerciseID = "+exerciseID
28.            cur = con.cursor()
29.            cur.execute(sql)
30.
31.    def editExercise(database,items,exerciseID):
32.        #function for editing items in the exercise table
33.        con = sqlite3.connect(database)
34.
35.        with con:
36.            sql = "UPDATE Exercise SET "+items+" WHERE EXERCISEID = "+exerciseID
37.            cur = con.cursor()
38.            cur.execute(sql)
```

This code is the sql for the Gym Edit Function. There is 4 different functions for each table.

All of them update their respective tables by passing in a string of items made out of the

inputs from the Gym Edit Dialog where their respective primary key(s) are the same as another parameter(s).

User Manual

Introduction

Purpose

The purpose of this system is to allow easy management of multiple clients information regarding their contact information, payments information, exercise regimes, as well as details of the exercises for a gym

It achieves this with the following features:

- Creation of member, payment, regime and exercise data - doesn't allow the same member ID's and other Primary/composite keys to be set
- The storage and editing of the created data - The data can be searched through and deleted using the user interface
- The creation and printing of invoices, member details, and regime details to send to clients or to keep hard copies

Intended Audience

The intended audience for this system was Neil Green - an independant local gym owner. Despite the a minor window icon and main window title change the interface could be used by any other gym that has the same requirements and organisation for their client data.

Installation

Prerequisites

The system has been compiled to a windows executable(.exe) so no software prerequisites are required to make use of this system as long as the users running windows.

The following is required to make use of every feature of this system:

- Windows 7, 8, 8.1, or 10
- A keyboard and mouse for input
- A monitor with a minimum resolution of 1024x768
- A hard disk drive or solid state drive for file and system storage with a minimum of 44.30 MB of free data
- A minimum of 512MB of RAM to carry out processing
- A minimum of a single core processor clocked at 1.2ghz to carry out processing
- A printer for printing invoices, regimes, and member info

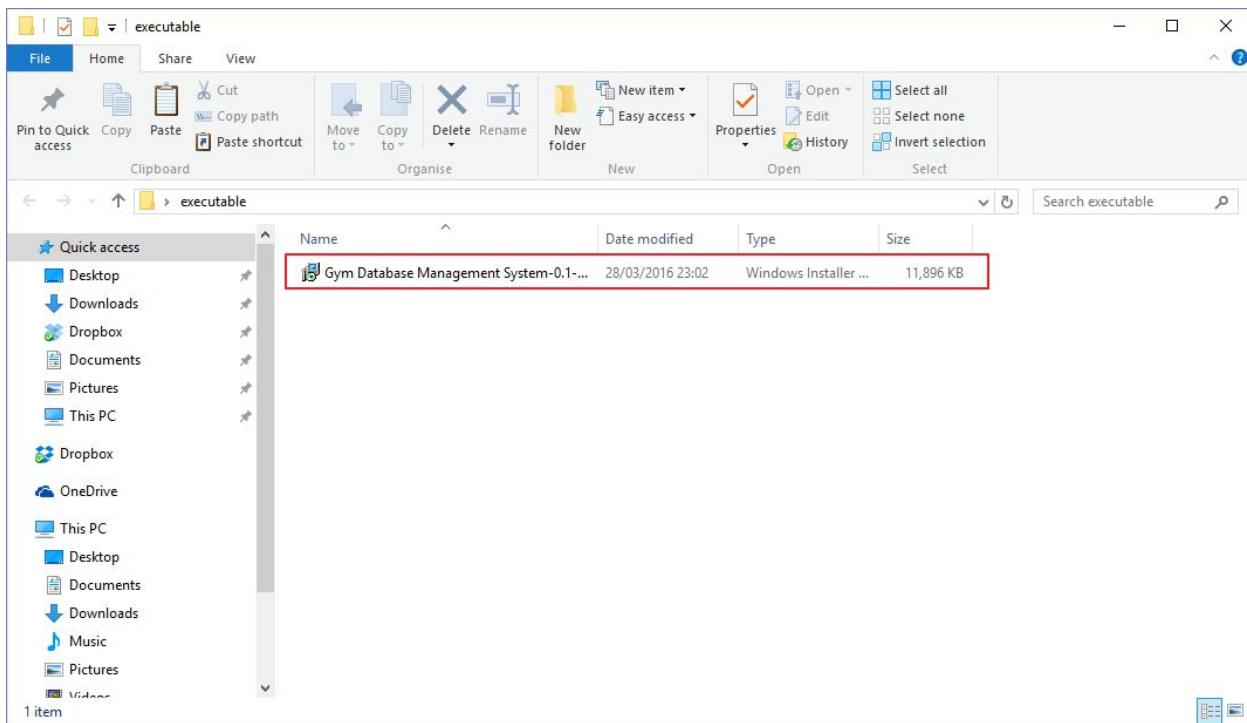
Operating Systems

The system was developed, tested and compiled exclusively on a workstation running windows 10 64-Bit, meaning the current executable of the system will only run on a 64-Bit version of windows, but a version can be compiled for windows, mac os and Linux based operating systems at 32-Bit or 64-Bit at a later date if my client changes os.

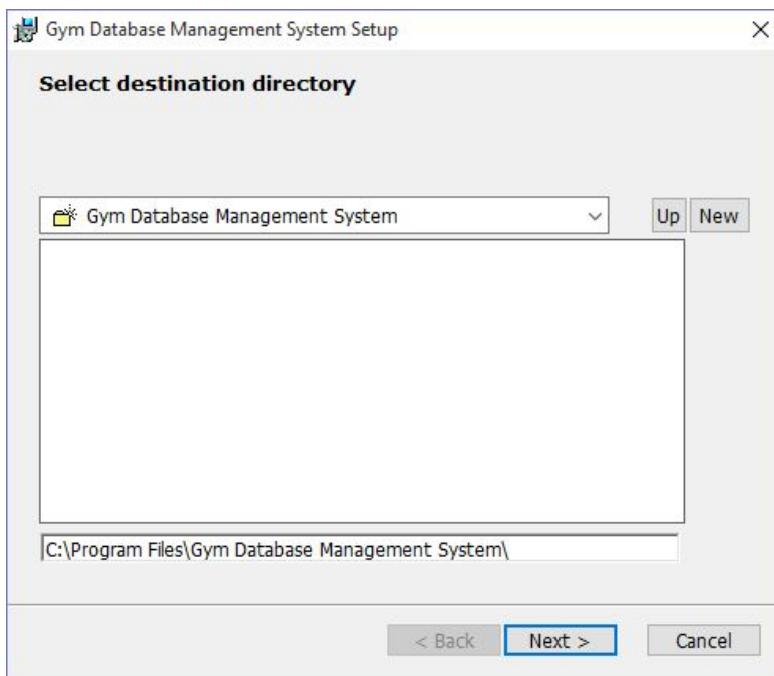
System Installation

The system was compiled to a windows installer prior to distribution, to install the system follow these steps:

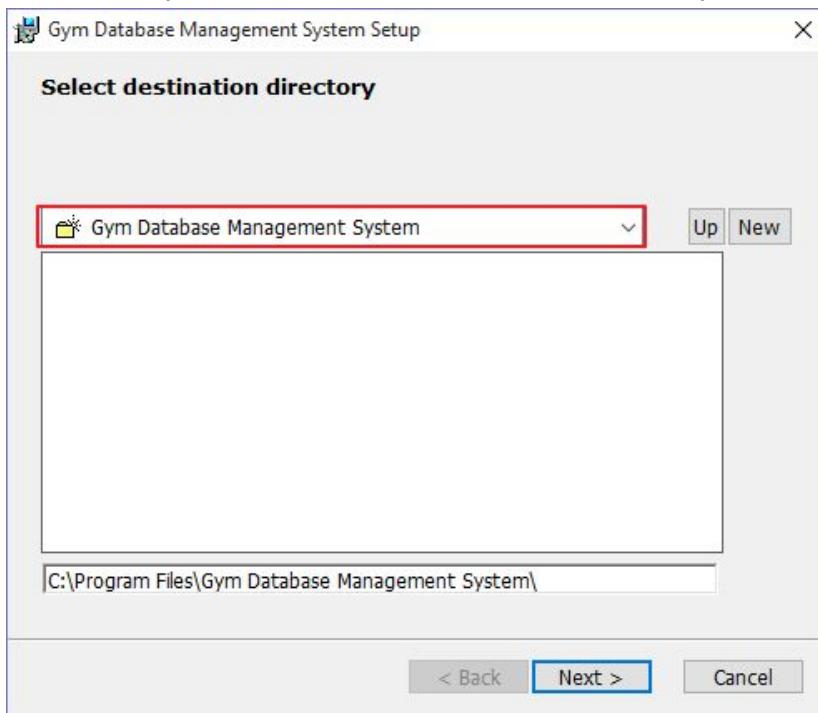
1. Locate the directory where the installer has been placed, in this example it's placed on the desktop in a folder labelled executable.



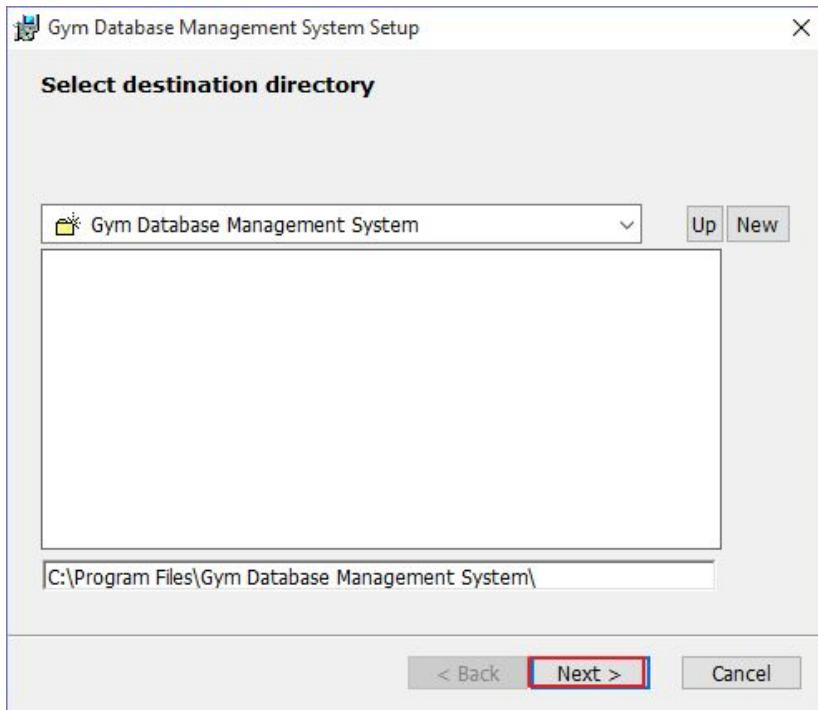
2. Double click on the installer to start the installation wizard



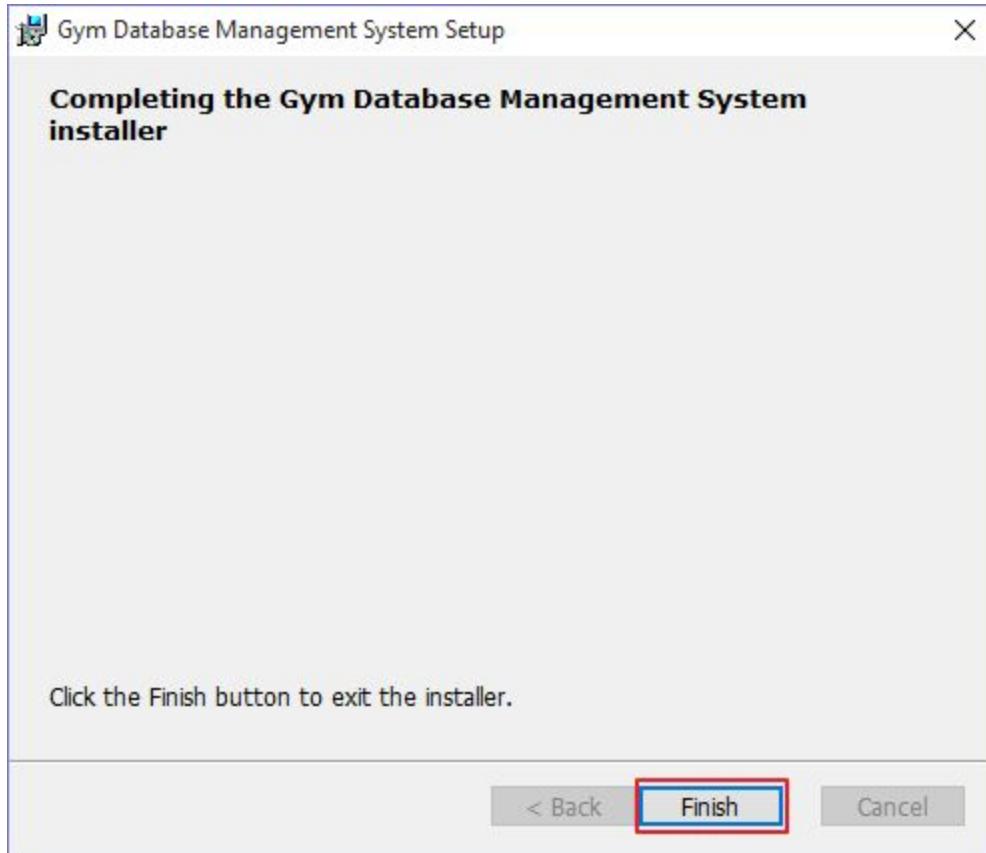
3. Now you can chose the location to install the system using the drop down box



4. Click the “Next” push buttons to progress through the installation



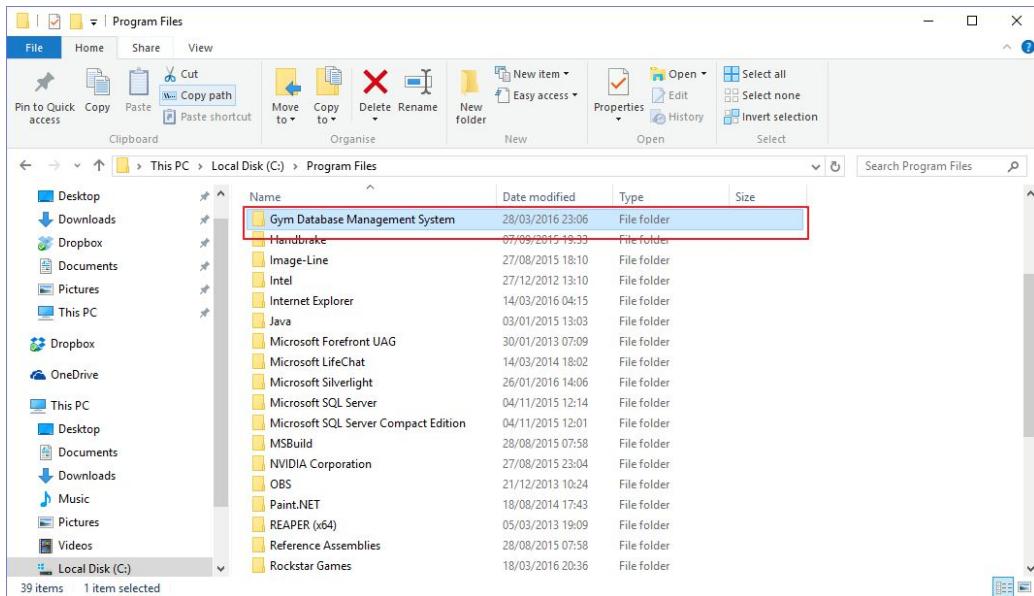
5. If the operating system appears asking for permission to install the system, click the “Yes” pushbutton.
6. Click the “Finish” push button to end the installation.



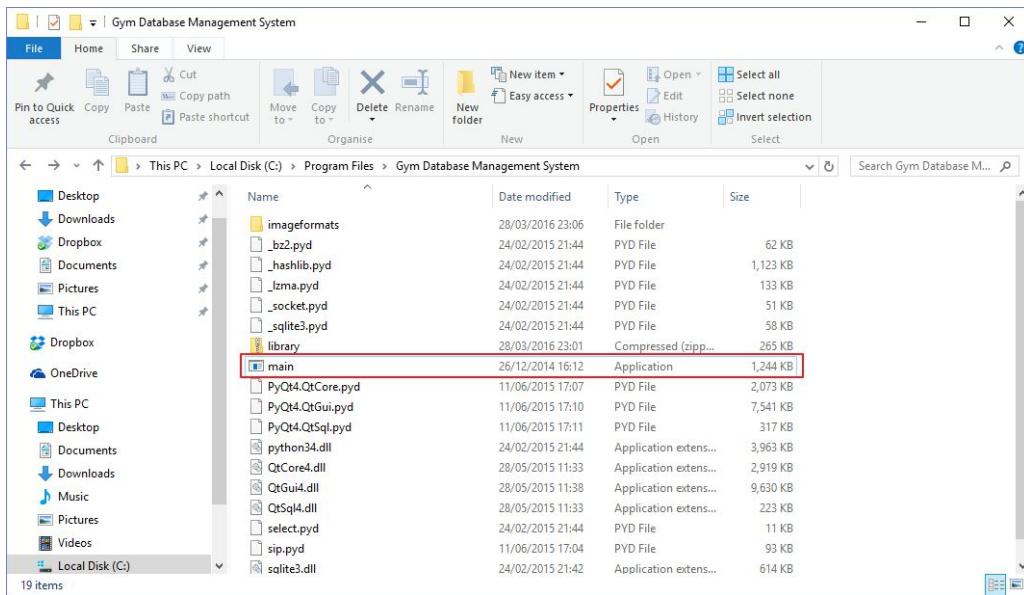
Running The System

Once the system has been installed following the instructions above the system can be ran by doing the following:

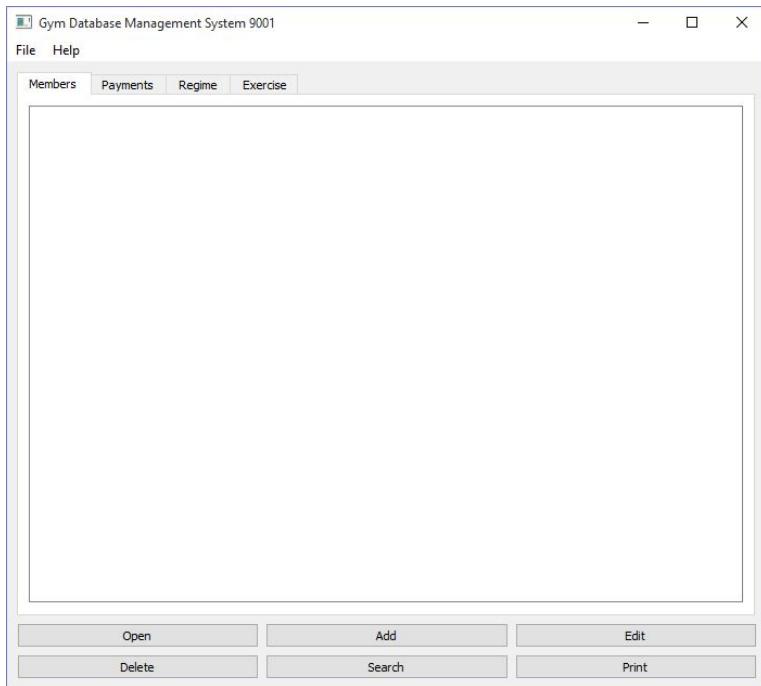
1. Navigate to the Folder where the system has been installed .



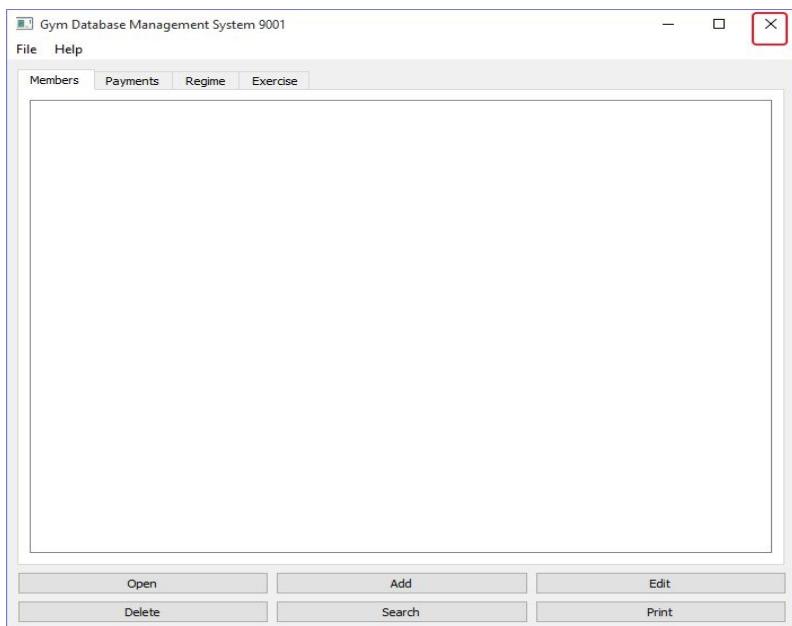
2. Double click on the executable with the left mouse button to start running the system.



3. Once the system is loaded it will be displayed on the screen.



4. Congratulations! The system is now running. The system can be closed at any time by clicking the “X” button in the top right corner



Tutorial

Introduction

In this section I will break down how to use each part of the system in the sections below, each sub-section will include a feature of the system explained with detailed text instructions and annotated diagrams to assist you in being able to completely use the system to its full ability.

Assumptions

There's no assumptions of computer knowledge made during the creation of the system and user manual so no assumed computer knowledge is included in the following tutorials. The only assumption that's made in the following tutorials is that the system is running, which is demonstrated in the above section "Running the System".

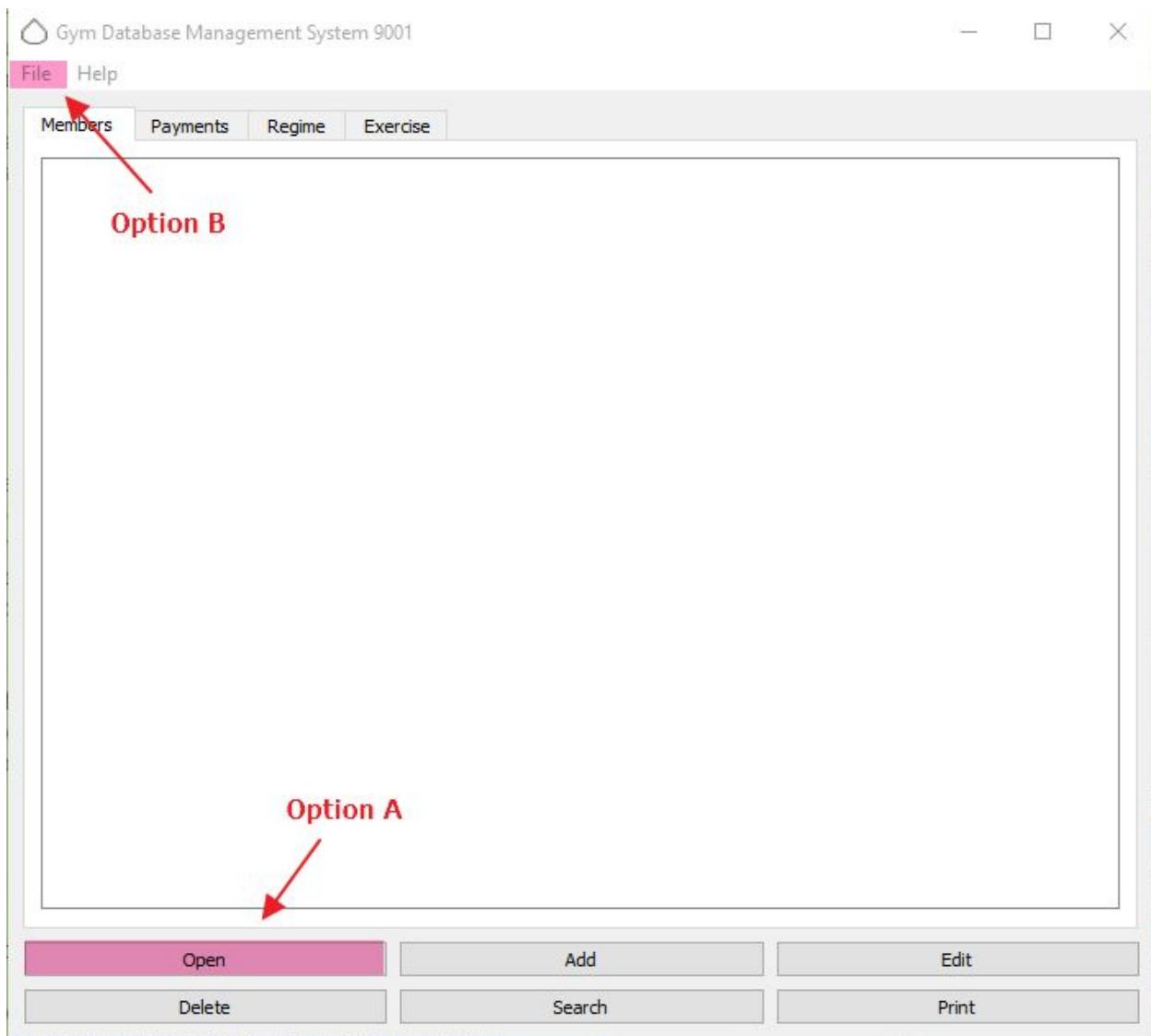
Contents

Tutorial	Page Number
Opening a database file	8
Adding a member	11
Adding a payment	16
Adding a regime	20
Adding an exercise	24
Editing a member	28
Editing a payment	32
Editing a regime	36
Editing an exercise	40
Deleting an item from a table	44
Deleting an entire table	48
Searching through a table	52
Printing member Info/Invoice/Regime Info	56
Accessing the about section	60
Navigating the table view	62
Entering Password	65

Opening a database file

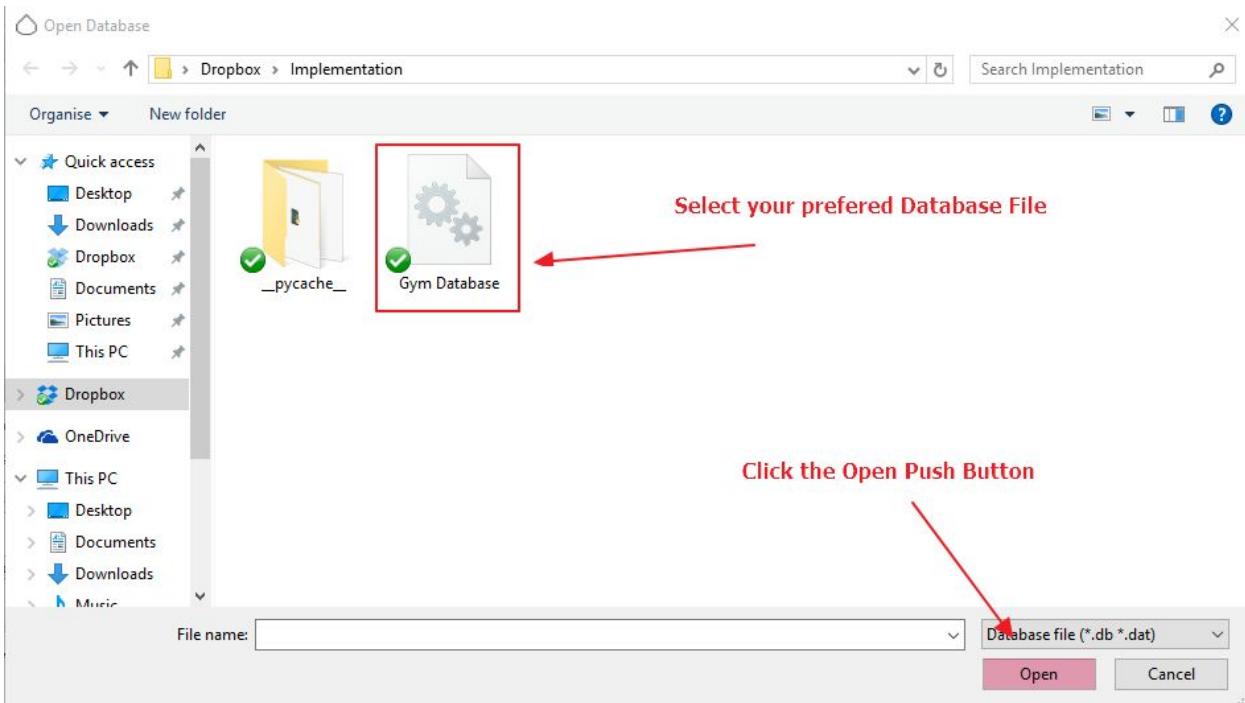
To use this program first an existing database file(provided with the program in the installation folder) needs to be opened and loaded into the program.

1. There are 3 ways to open a database file.
 - a. Find the “Open” Push Button and left click it.
 - b. Go to the Tool bar and highlight the “File” menu item and scroll down to the “Open” item and left click.
 - c. Press Ctrl + O.





2. After starting the open function a windows file dialog will open where you will need to locate your preferred database file and left click the “Open” Push Button.



3. The Selected database will now show up in the Table view in a tabbed layout.

The screenshot shows a software interface titled "Gym Database Management System 9001". At the top, there is a menu bar with "File" and "Help" options. Below the menu is a tab bar with four tabs: "Members", "Payments", "Regime", and "Exercise". The "Members" tab is currently selected, indicated by a red border around its tab and a red arrow pointing to it from the text "Tabbed Layout displaying all the tables". The main area displays a table with the following data:

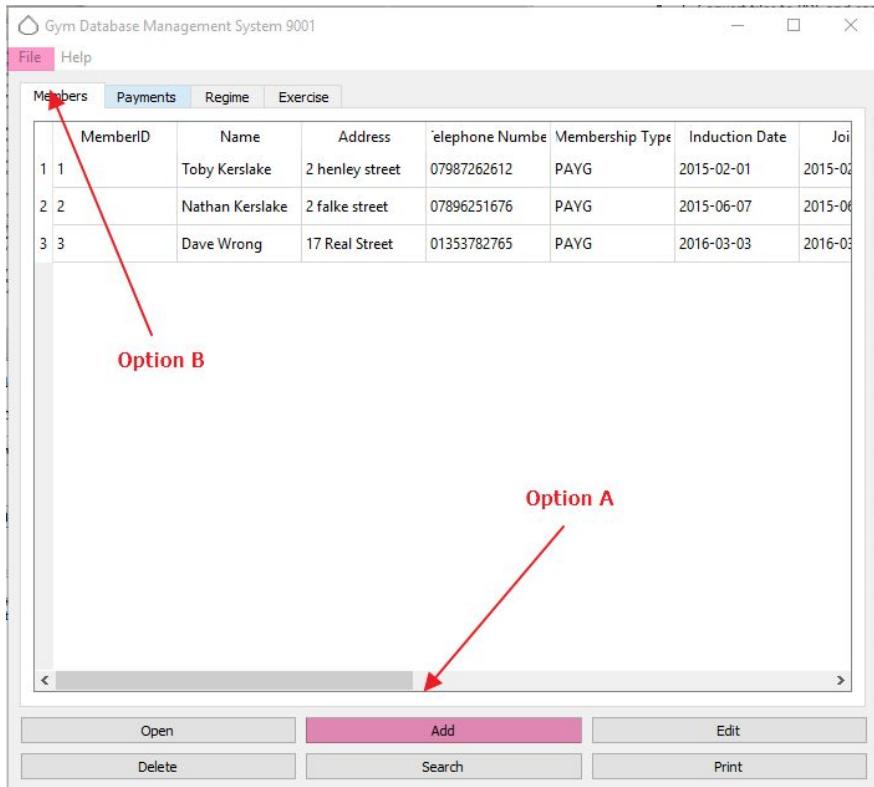
MemberID	Name	Address	Telephone Number	Membership Type	Induction Date	Join Date
1 1	Toby Kerslake	2 henley street	07987262612	PAYG	2015-02-01	2015-02-01
2 2	Nathan Kerslake	2 falke street	07896251676	PAYG	2015-06-07	2015-06-07
3 3	Dave Wrong	17 Real Street	01353782765	PAYG	2016-03-03	2016-03-03

Below the table, there is a horizontal scrollbar. At the bottom of the window, there are several buttons: "Open", "Add", "Edit", "Delete", "Search", and "Print". A red arrow points from the text "Table View containing opened database" to the "Edit" button.

Adding a member

Using this program allows you to enter and save data about a member of your gym.

1. There are 3 ways to add member data
 - a. Find the “Add” Push Button and left click it
 - b. Go to the The Tool bar and highlight the “File” menu item and scroll down to the “Add” item and left click
 - c. Press Ctrl + A



2. After starting the Add function an Add dialog will open

The screenshot shows a Windows-style dialog box titled 'Add'. At the top left is a small icon of a person with arms raised. To its right are three buttons: a question mark icon, a blue-bordered 'Add' button, and a close ('X') button. Below the title bar is a dropdown menu labeled 'Members' with a downward arrow. The main area contains twelve text input fields, each preceded by a label: MemberID, Name, Address, Telephone Number, Membership Type, Induction Date, Join Date, How Paid, Amount, Registration Fee, Registration Date, Payment Type, and Comments. The 'Add' button is highlighted with a blue border.

3. At the top of this dialog there is a combobox that lets you switch the table you want to enter info in. To add member info left click it and select the “Member” option.

This screenshot is identical to the one above, but the 'Members' dropdown menu is highlighted with a pink background, indicating it has been selected. The rest of the dialog, including the buttons and input fields, remains the same.

4. Now enter all the member information in the correctly labelled columns by left clicking on the Line edits and typing with the keyboard.

Column	Value
MemberID	4
Name	Geoffrey Smith
Address	17 Main Street
Telephone Number	01364557892
Membership Type	PAYG
Induction Date	2015-02-03
Join Date	2015-02-03
How Paid	Card
Amount	50
Registration Fee	50
Registration Date	2015-02-03
Payment Type	Monthly
Comments	(empty)

Line Edits where information
is input using the keyboard

- 5.
- The columns "Name", "Address", "Membership Type", "How Paid", "Payment Type", "Comments" must be entered as text with the characters a-Z and should not exclusively be numbers but can include all characters apart from quotation marks or double quotation marks
 - The columns "MemberID", "Telephone Number", "Amount", "Registration Fee" must be entered as any combination of integer numbers 0-9 only
 - The columns "InductionDate", "JoinDate", "RegistrationDate" must be dates in the format yyyy-mm-dd (For example 2015-05-12)

6. Click the “Add” push button to add all of the info you’ve input into the database.

Members

Add

MemberID	4
Name	Geoffrey Smith
Address	17 Main Street
Telephone Number	01364557892
Membership Type	PAYG
Induction Date	2015-02-03
Join Date	2015-02-03
How Paid	Card
Amount	50
Registration Fee	50
Registration Date	2015-02-03
Payment Type	Monthly
Comments	

Click the "Add" push button

7. Re-open the database you've added the data too.

Gym Database Management System 9001

File Help

Members Payments Regime Exercise

	MemberID	Name	Address	Telephone Number	Membership Type	Induction Date	Join Date
1	1	Toby Kerslake	2 henley street	07987262612	PAYG	2015-02-01	2015-02-01
2	2	Nathan Kerslake	2 falke street	07896251676	PAYG	2015-06-07	2015-06-07
3	3	Dave Wrong	17 Real Street	01353782765	PAYG	2016-03-03	2016-03-03

Click the "Open" push button and re-open the database

Open Add Edit
Delete Search Print

Gym Database Management System 9001

File Help

Members Payments Regime Exercise

	MemberID	Name	Address	Telephone Number	Membership Type	Induction Date	Join Date
1	1	Toby Kerslake	2 henley street	07987262612	PAYG	2015-02-01	2015-02-01
2	2	Nathan Kerslake	2 falke street	07896251676	PAYG	2015-06-07	2015-06-07
3	3	Dave Wrong	17 Real Street	01353782765	PAYG	2016-03-03	2016-03-03
4	4	Geoffrey Smith	17 Main Street	01364557892	PAYG	2015-02-03	2015-02-03

The input data has been inserted into the database

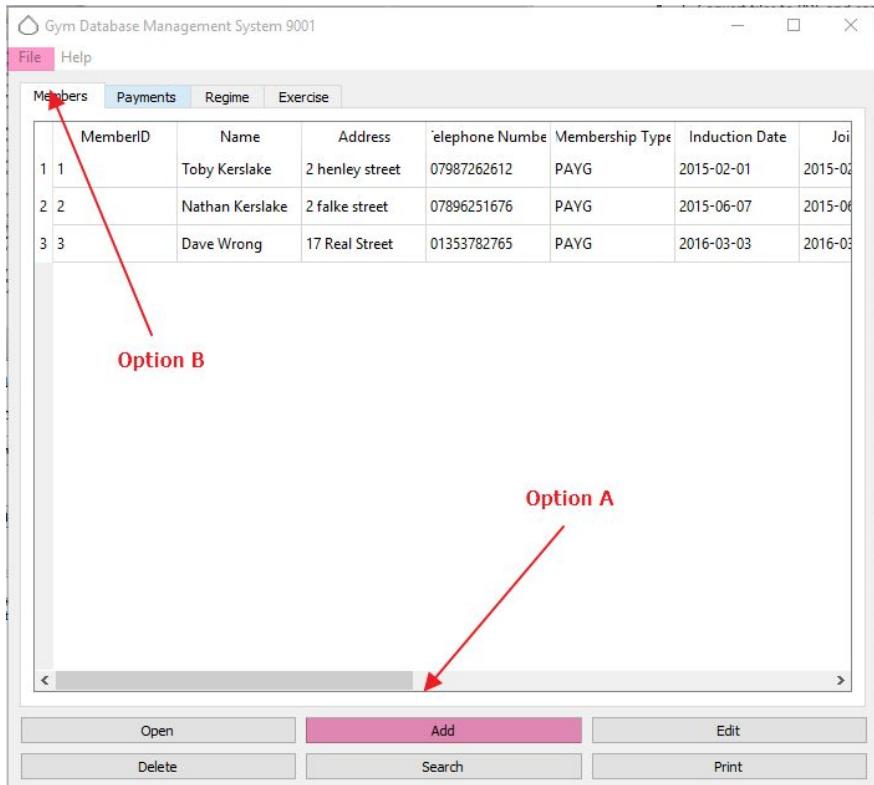
< >

Open Add Edit
Delete Search Print

Adding a payment

Using this program allows you to enter and save data about a member's payments

1. There are 3 ways to add payment data
 - a. Find the "Add" Push Button and left click it
 - b. Go to the The Tool bar and highlight the "File" menu item and scroll down to the "Add" item and left click
 - c. Press Ctrl + A



2. After starting the Add function an Add dialog will open

The screenshot shows a Windows-style dialog box titled "Add". In the top-left corner, there is a dropdown menu set to "Members". To the right of the dropdown is a blue rectangular button labeled "Add". The main area of the dialog contains ten text input fields, each preceded by a label: "MemberID", "Name", "Address", "Telephone Number", "Membership Type", "Induction Date", "Join Date", "How Paid", "Amount", "Registration Fee", "Registration Date", "Payment Type", and "Comments".

3. At the top of this dialog there is a combobox that lets you switch the table you want to enter info in. To add payment info left click it and select the "Payment" option.

This screenshot shows the same "Add" dialog box as the previous one, but with a difference: the dropdown menu at the top-left is now set to "Payments", which is highlighted with a pink background. The rest of the interface, including the "Add" button and the list of input fields below, remains identical to the first screenshot.

4. Now enter all the payment information in the correctly labelled columns by left clicking on the Line edits and typing with the keyboard.

MemberID Payment Date How Much Paid

Line Edits where information is input using the keyboard

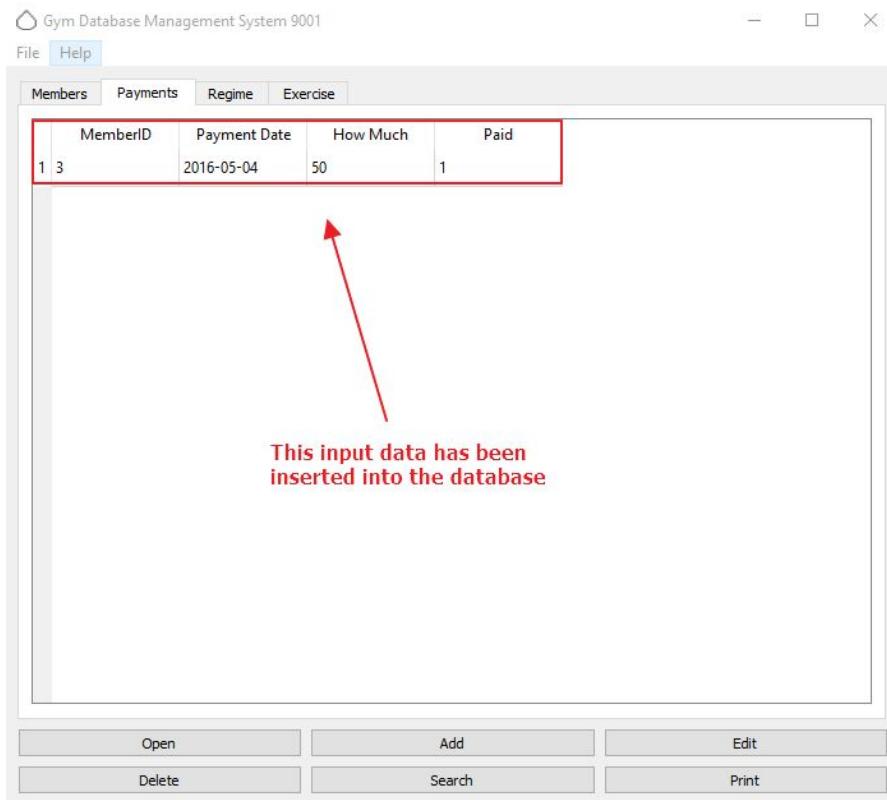
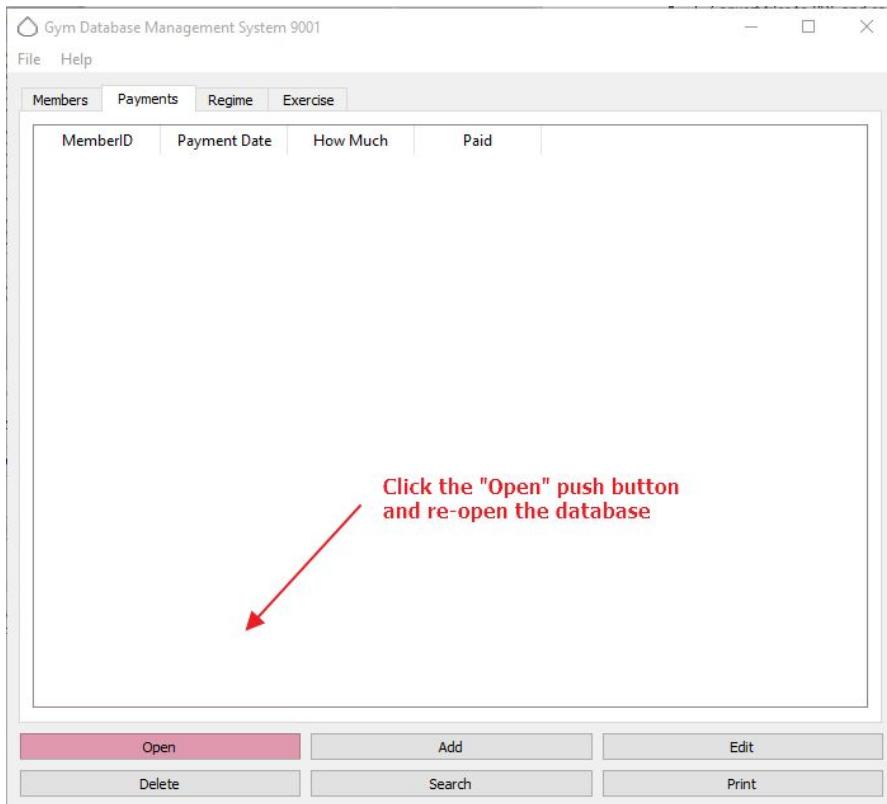
- 5.
- The column “Paid” must be entered with either the value “0” or “1”. “0” representing not paid and “1” representing paid
 - The columns “MemberID” and “How Much” must be entered as any combination of integer numbers 0-9 only
 - The column “paymentDate” must be a date in the format yyyy-mm-dd (For example 2015-05-12)

6. Click the “Add” push button to add all of the info you’ve input into the database.

MemberID Payment Date How Much Paid

Click the "Add" push button

7. Re-open the database you've added the data too.

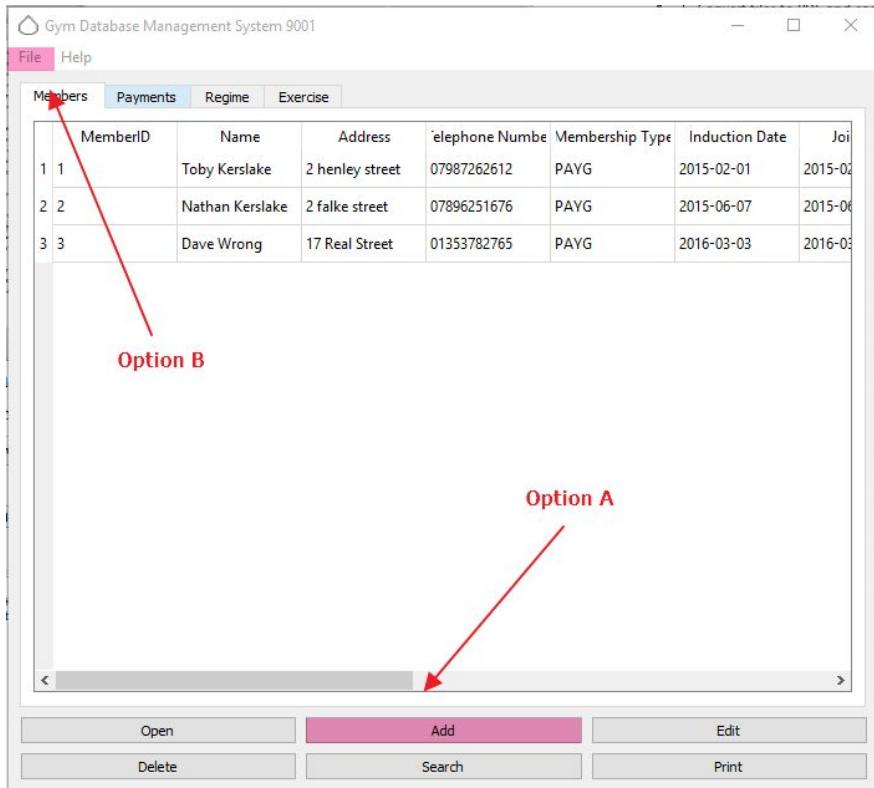


Adding a regime

Using this program allows you to enter and save data about a member's regime

1. There are 3 ways to add regime data

- a. Find the "Add" Push Button and left click it
- b. Go to the The Tool bar and highlight the "File" menu item and scroll down to the "Add" item and left click
- c. Press Ctrl + A



2. After starting the Add function an Add dialog will open

The screenshot shows a Windows-style dialog box titled "Add". In the top-left corner, there is a small icon of a person with arms raised. To the right of the icon, the word "Add" is displayed. Below the title bar, there is a dropdown menu set to "Members". On the right side of the dialog, there is a large blue rectangular button also labeled "Add". The main area of the dialog contains twelve text input fields, each preceded by a label: MemberID, Name, Address, Telephone Number, Membership Type, Induction Date, Join Date, How Paid, Amount, Registration Fee, Registration Date, Payment Type, and Comments. All these fields are currently empty.

3. At the top of this dialog there is a combobox that lets you switch the table you want to enter info in. To add regime info left click it and select the “Regime” option.

The screenshot shows the same "Add" dialog box, but the dropdown menu at the top has been changed to "Regimes". The rest of the interface is identical to the previous screenshot, featuring the large blue "Add" button and twelve text input fields for MemberID, Name, Address, Telephone Number, Membership Type, Induction Date, Join Date, How Paid, Amount, Registration Fee, Registration Date, Payment Type, and Comments.

4. Now enter all the regime information in the correctly labelled columns by left clicking on the Line edits and typing with the keyboard.

MemberID 2

Exercise ID 1

Start Date 2016-04-05

End Date

Description 50 Push - ups

Line Edits where information is input using the keybaord

- 5.
- The column “Description” must be enter as text with the characters a-Z and should not exclusively be numbers but can include all characters apart from quotation marks or double quotation marks
 - The columns “MemberID” and “ExerciseID” must be entered as any combination of the integer numbers 0-9 only
 - The columns “startDate” and “endDate” must be dates in the format yyyy-mm-dd
(For example 2015-05-12)
6. Click the “Add” push button to add all of the info you’ve input into the database.

MemberID 2

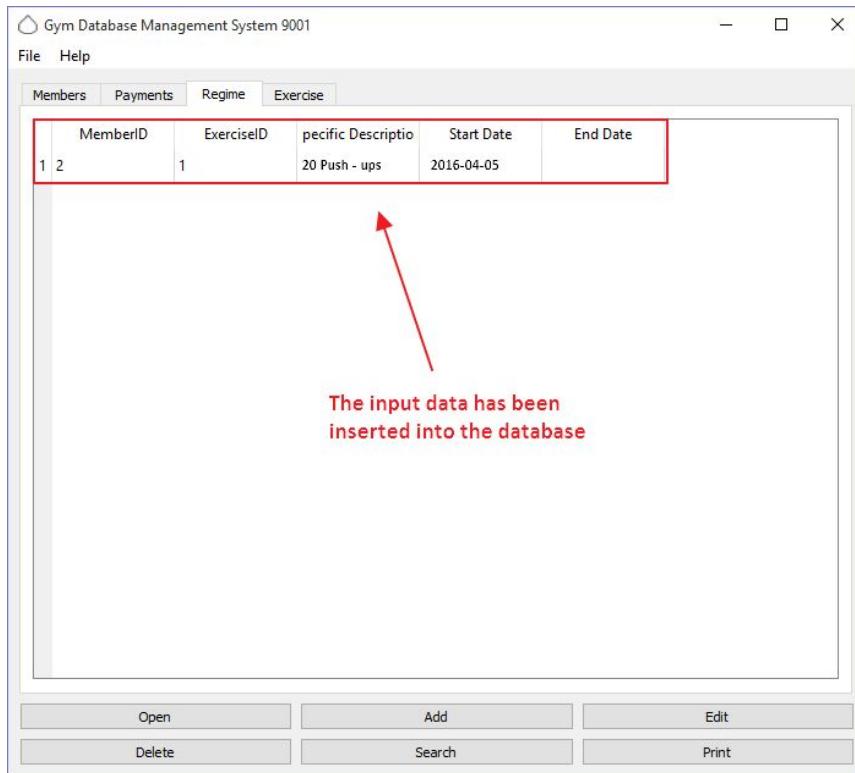
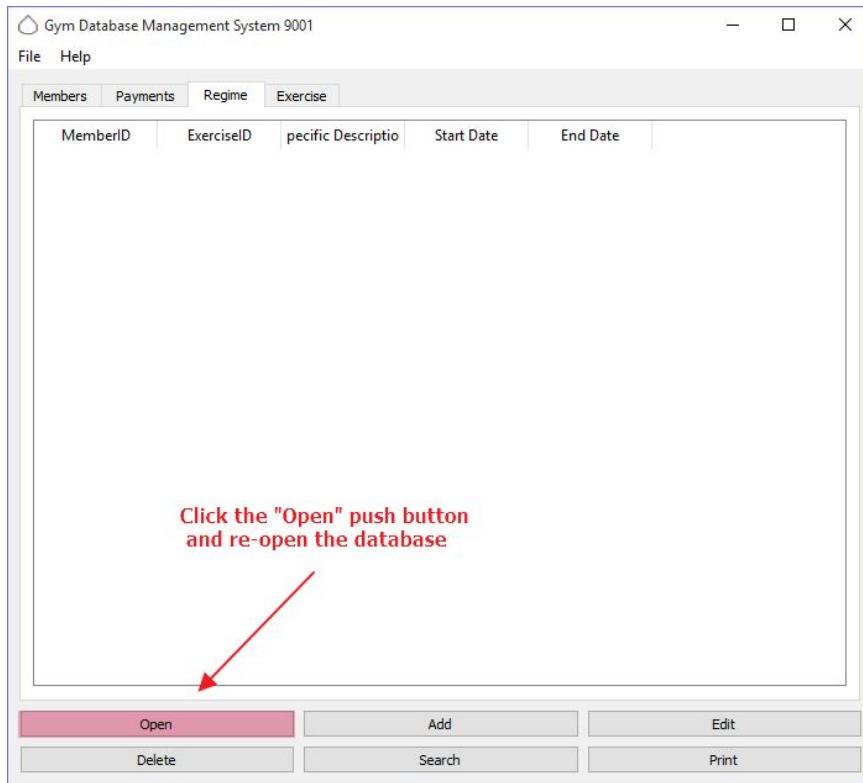
Exercise ID 1

Start Date 2016-04-05

End Date

Description 50 Push - ups

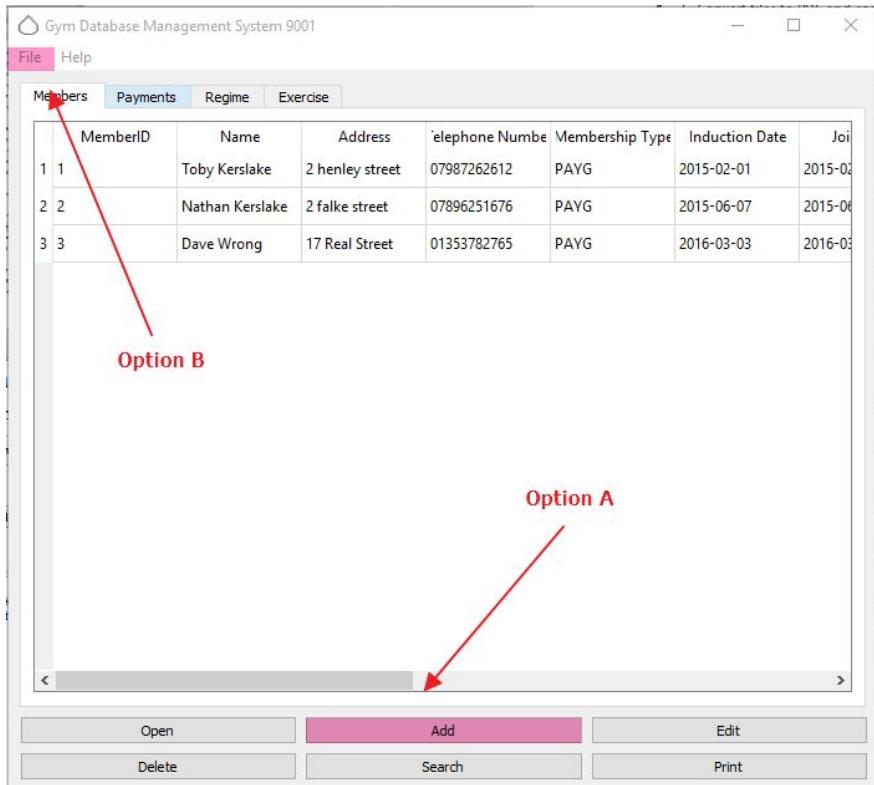
7. Re-open the database you've added the data too.



Adding an exercise

Using this program allows you to enter and save data about an exercise

1. There are 3 ways to add exercise data
 - a. Find the “Add” Push Button and left click it
 - b. Go to the The Tool bar and highlight the “File” menu item and scroll down to the “Add” item and left click
 - c. Press Ctrl + A



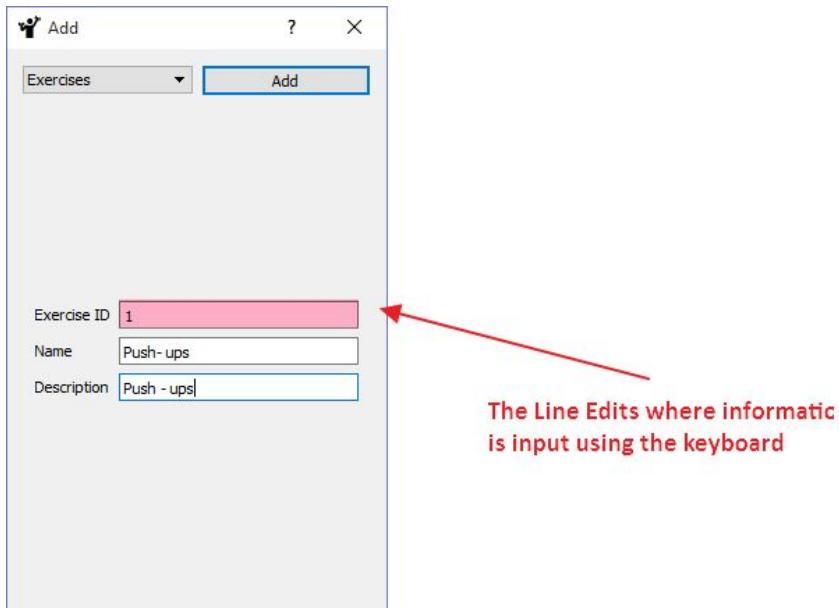
2. After starting the Add function an Add dialog will open

The screenshot shows a Windows-style dialog box titled "Add". In the top-left corner, there is a small icon of a person with arms raised. To the right of the title are three buttons: a question mark icon, a standard window control button, and a close button (X). Below the title bar is a dropdown menu labeled "Members" with a downward arrow, and a blue rectangular button labeled "Add". The main body of the dialog contains twelve text input fields, each preceded by a label: MemberID, Name, Address, Telephone Number, Membership Type, Induction Date, Join Date, How Paid, Amount, Registration Fee, Registration Date, Payment Type, and Comments. All input fields are currently empty.

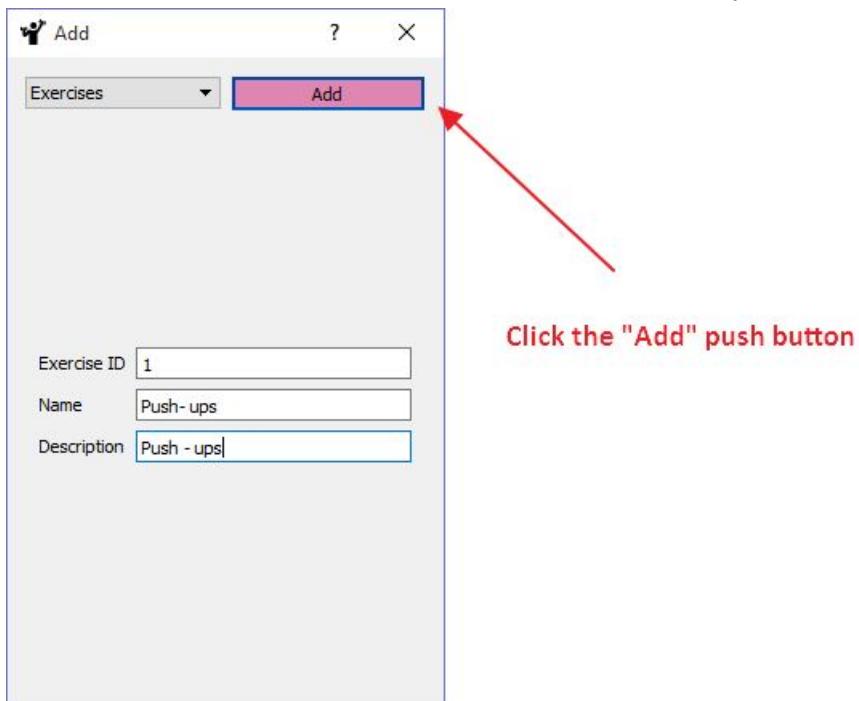
3. At the top of this dialog there is a combobox that lets you switch the table you want to enter info in. To add exercise info left click it and select the "Exercise" option.

The screenshot shows the same "Add" dialog box, but the dropdown menu at the top has been changed to "Exercises". The rest of the interface is identical to the previous screenshot, featuring the "Add" button and twelve input fields for exercise information.

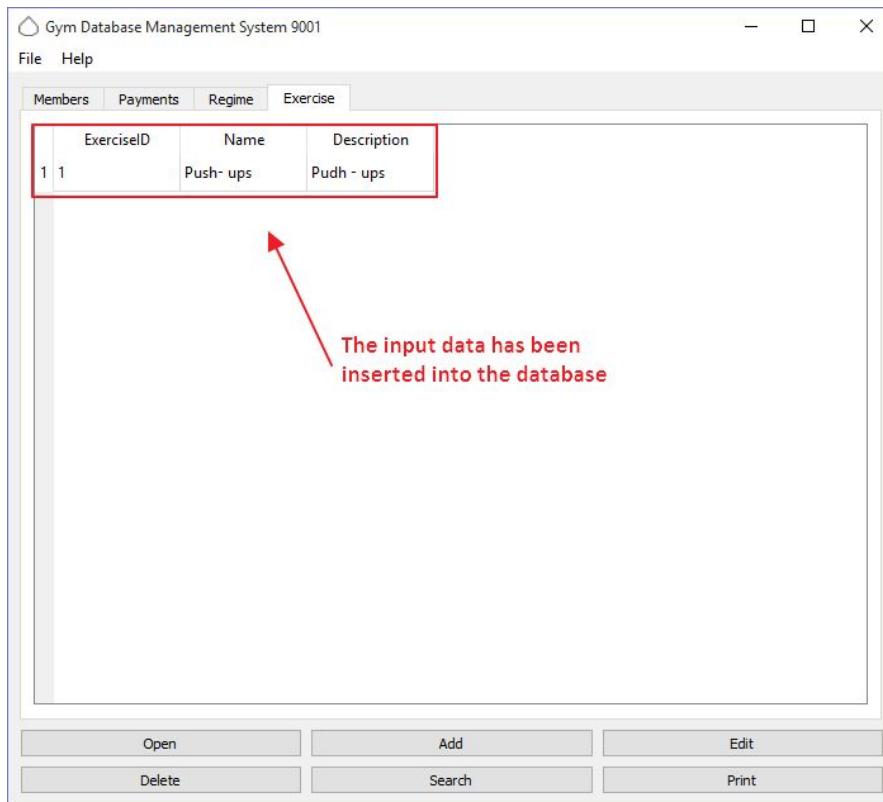
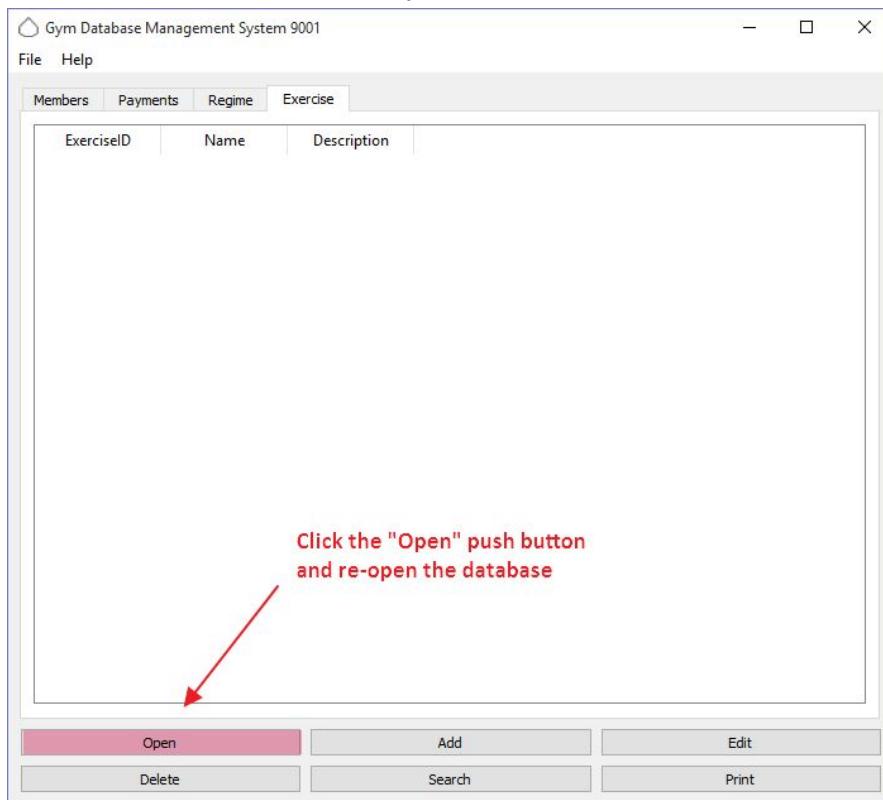
4. Now enter all the exercise information in the correctly labelled columns by left clicking on the Line edits and typing with the keyboard.



- 5.
- The columns "Description" and "Name" must be entered as text with the characters a-Z and should not exclusively be numbers but can include all characters apart from quotation marks or double quotation marks
 - The column "ExerciseID" must be entered as any combination of the integer numbers 0-9 only
6. Click the "Add" push button to add all of the info you've input into the database.



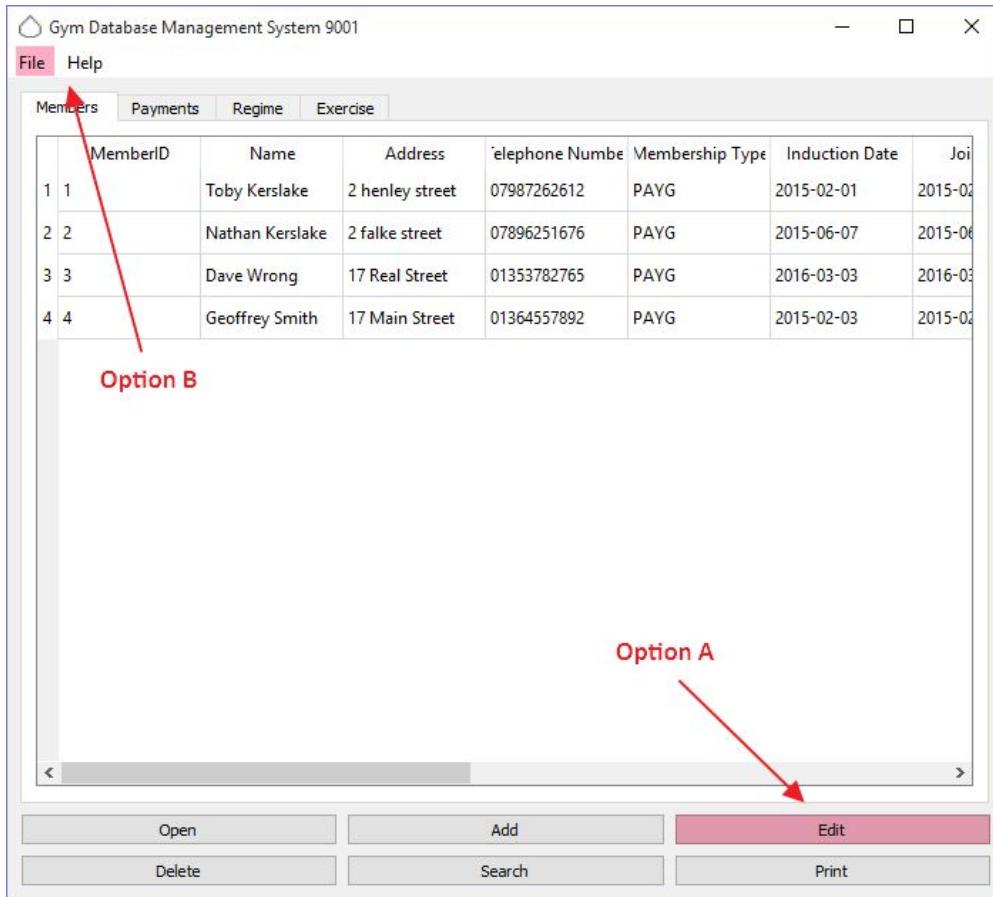
7. Re-open the database you've added the data too.



Editing a member

Using this program allows you to edit and save data about a member of your gym.

1. There are 3 ways to edit member data
 - a. Find the “Edit” Push Button and left click it
 - b. Go to the The Tool bar and highlight the “File” menu item and scroll down to the “Edit” item and left click
 - c. Press Ctrl + E



2. After starting the Edit function an Edit dialog will open

The screenshot shows a software interface titled 'Edit'. At the top left is a small icon of a person with arms raised. To its right are three buttons: a question mark icon, a blue rectangular button labeled 'Edit', and a standard close ('X') button. Below these buttons is a dropdown menu with the text 'Members' followed by a downward arrow. The main area contains twelve input fields, each preceded by a label: 'MemberID', 'Name', 'Address', 'Telephone Number', 'Membership Type', 'Induction Date', 'Join Date', 'How Paid', 'Amount', 'Registration Fee', 'Registration Date', 'Payment Type', and 'Comments'. Each label is aligned with a corresponding empty text input field.

3. At the top of this dialog there is a combobox that lets you switch the table you want to enter info in. To edit member info left click it and select the "Member" option.

This screenshot is identical to the one above, showing the 'Edit' dialog for 'Members'. However, the dropdown menu at the top left is now highlighted with a pink background, indicating it has been selected. The other elements, including the 'Edit' button and the list of input fields, remain the same.

4. Now enter all the member information in the correctly labelled columns by left clicking on the Line edits and typing with the keyboard.

Line Edits where information is input using the keyboard

MemberID	1
Name	Toby Kerslake
Address	4 main street
Telephone Number	
Membership Type	
Induction Date	
Join Date	
How Paid	
Amount	
Registration Fee	
Registration Date	
Payment Type	
Comments	

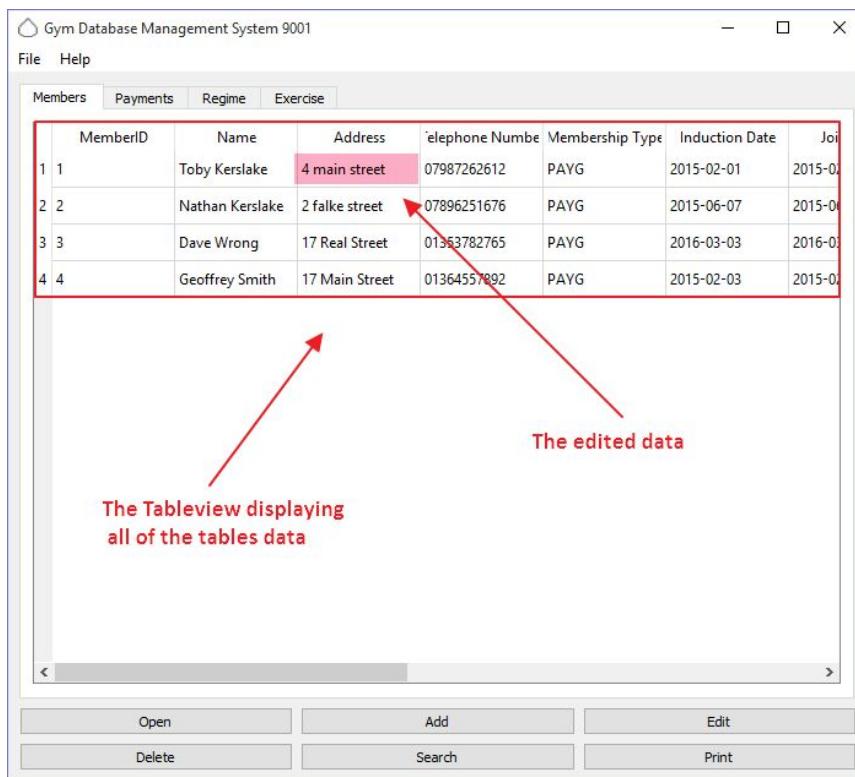
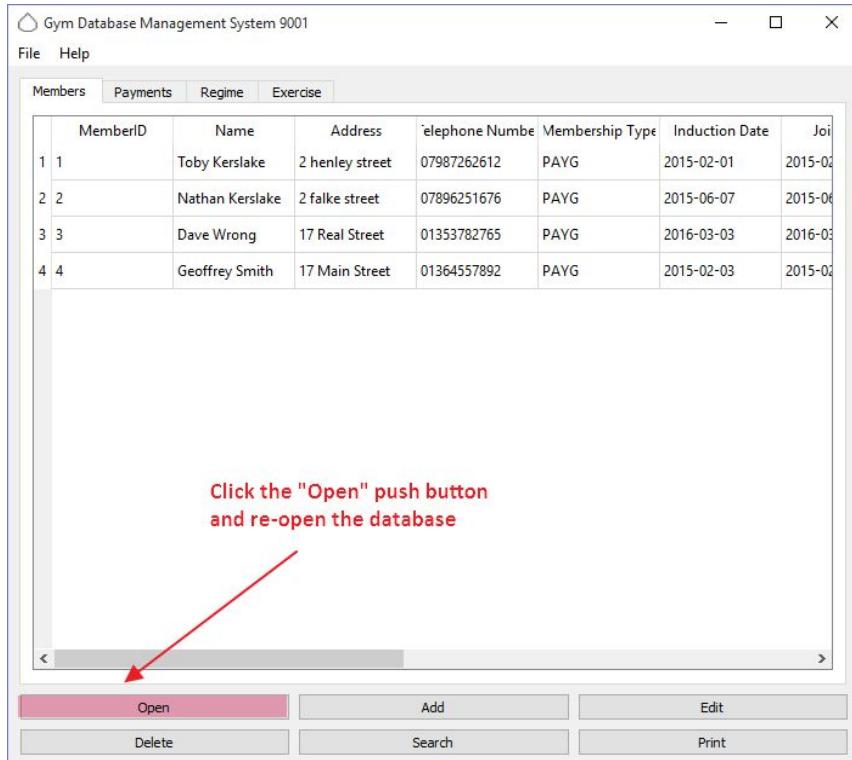
- 5.
- The columns "Name", "Address", "Membership Type", "How Paid", "Payment Type", "Comments" must be entered as text with the characters a-Z and should not exclusively be numbers but can include all characters apart from quotation marks or double quotation marks
 - The columns "MemberID", "Telephone Number", "Amount", "Registration Fee" must be entered as any combination of integer numbers 0-9 only. The column MemberID must also be an existing MemberID for the member you want to edit the information of
 - The columns "InductionDate", "JoinDate", "RegistrationDate" must be dates in the format yyyy-mm-dd (For example 2015-05-12)

6. Click the "Edit" push button to add all of the info you've input into the database.

Click the "Edit" push Button

MemberID	1
Name	Toby Kerslake
Address	4 main street
Telephone Number	
Membership Type	
Induction Date	
Join Date	
How Paid	
Amount	
Registration Fee	
Registration Date	
Payment Type	
Comments	

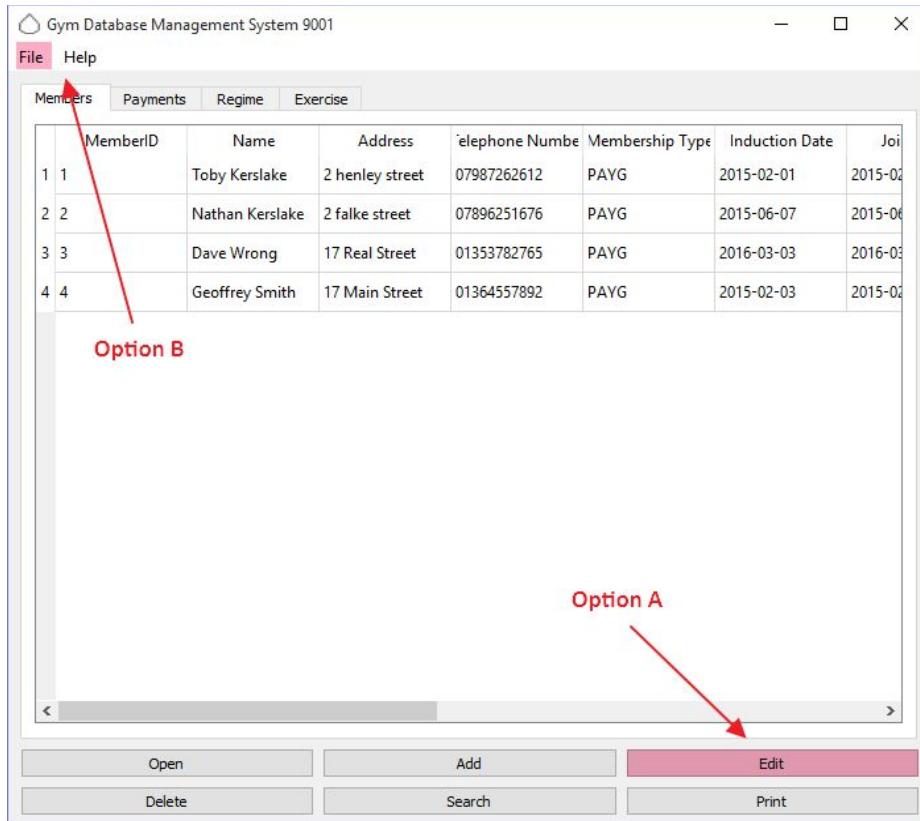
7. Re-open the database you've edited the data of.



Editing a payment

Using this program allows you to edit and save data about a member's payments

1. There are 3 ways to edit payment data
 - a. Find the “Edit” Push Button and left click it
 - b. Go to the The Tool bar and highlight the “File” menu item and scroll down to the “Edit” item and left click
 - c. Press Ctrl + E



2. After starting the Edit function an Edit dialog will open

The screenshot shows an 'Edit' dialog window titled 'Edit'. At the top left is a dropdown menu set to 'Members'. To its right are three buttons: a question mark icon, a blue 'Edit' button, and a close 'X' button. The main area contains twelve input fields, each with a label and a corresponding text box:

- MemberID
- Name
- Address
- Telephone Number
- Membership Type
- Induction Date
- Join Date
- How Paid
- Amount
- Registration Fee
- Registration Date
- Payment Type
- Comments

3. At the top of this dialog there is a combobox that lets you switch the table you want to enter info in. To add payment info left click it and select the “Payment” option.

The screenshot shows an 'Edit' dialog window titled 'Edit'. The dropdown menu at the top left is now set to 'Payments', which is highlighted with a red background. To its right are three buttons: a question mark icon, a blue 'Edit' button, and a close 'X' button. The main area contains four input fields, each with a label and a corresponding text box:

- MemberID
- Payment Date
- How Much
- Paid

4. Now enter all the payment information in the correctly labelled columns by left clicking on the Line edits and typing with the keyboard.

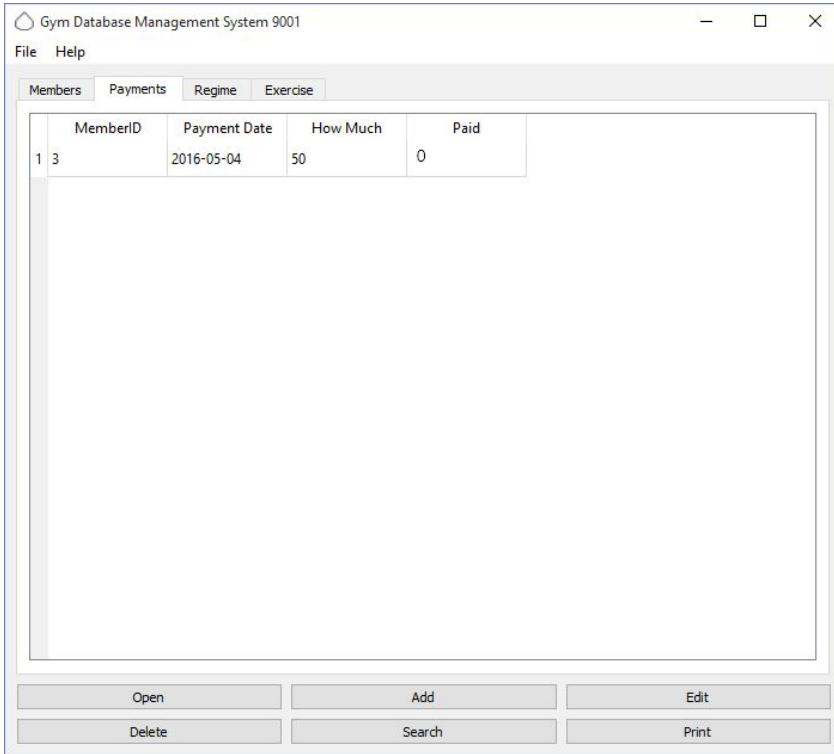
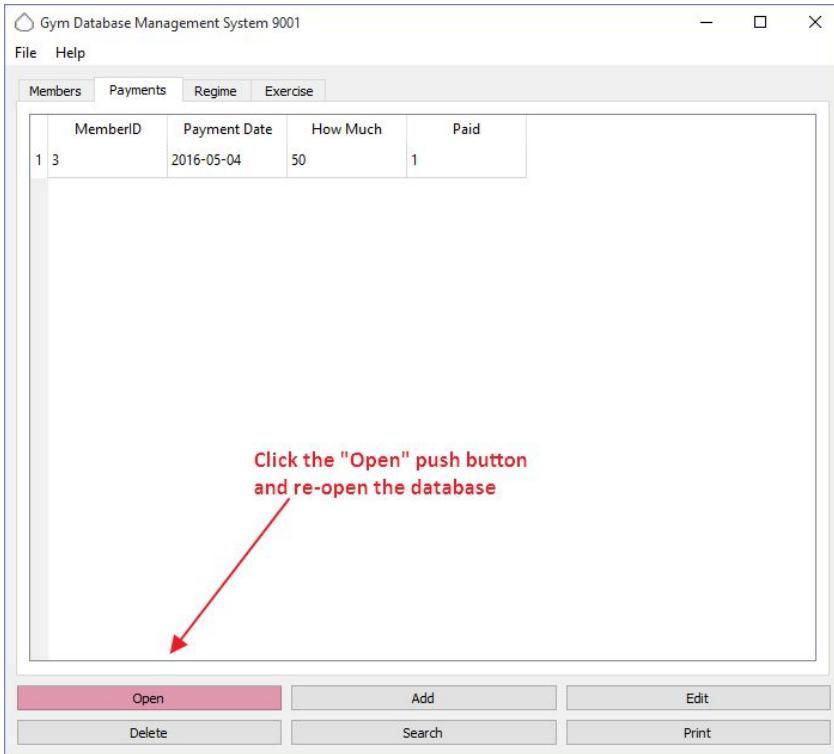
MemberID	3
Payment Date	2016-05-04
How Much	50
Paid	0

- 5.
- The column “Paid” must be entered with either the value “0” or “1”. “0” representing not paid and “1” representing paid
 - The columns “MemberID” and “How Much” must be entered as any combination of integer numbers 0-9 only
 - The column “paymentDate” must be a date in the format yyyy-mm-dd (For example 2015-05-12)
 - The columns “MemberID” and “PaymentDate” must be existing items in the database for the payment you want to edit the information of

6. Click the “Edit” push button to add all of the info you’ve input into the database.

MemberID	3
Payment Date	2016-05-04
How Much	50
Paid	0

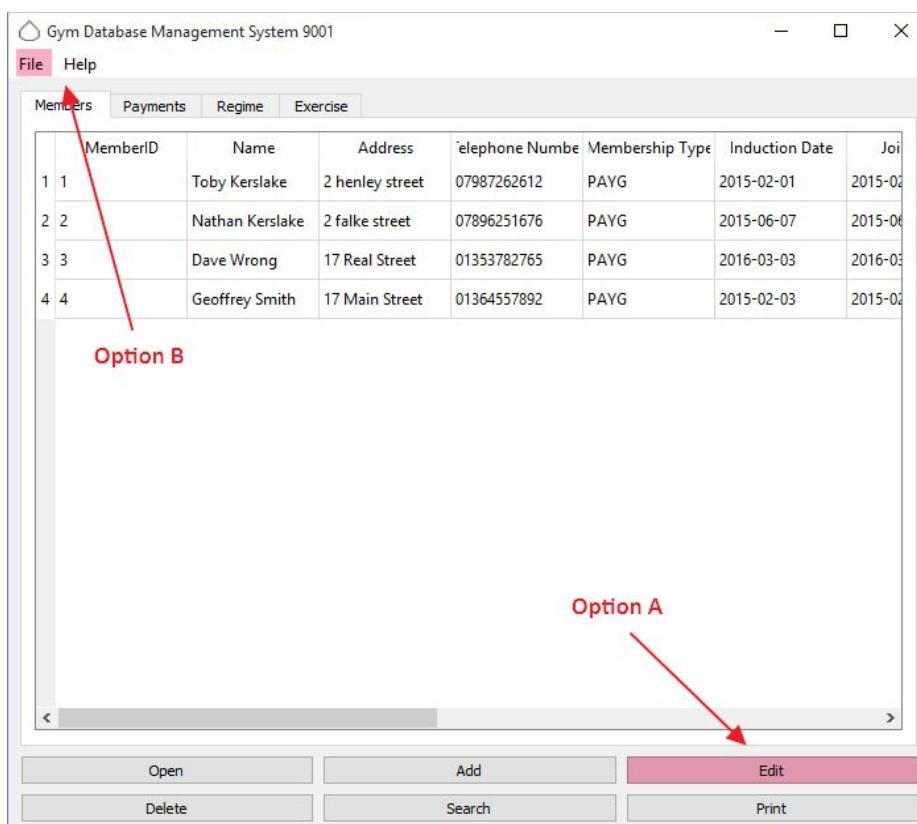
7. Re-open the database you've edited the data of.



Editing a regime

Using this program allows you to edit and save data about a member's regime

1. There are 3 ways to edit regime data
 - a. Find the "Edit" Push Button and left click it
 - b. Go to the The Tool bar and highlight the "File" menu item and scroll down to the "Edit" item and left click
 - c. Press Ctrl + E



2. After starting the Edit function an Edit dialog will open

The screenshot shows an 'Edit' dialog window titled 'Edit'. At the top left is a dropdown menu set to 'Members'. To its right are three buttons: a question mark icon, a blue 'Edit' button, and a close 'X' button. The main area contains a vertical list of fields with corresponding input boxes:

Field	Type
MemberID	Text
Name	Text
Address	Text
Telephone Number	Text
Membership Type	Text
Induction Date	Text
Join Date	Text
How Paid	Text
Amount	Text
Registration Fee	Text
Registration Date	Text
Payment Type	Text
Comments	Text

3. At the top of this dialog there is a combobox that lets you switch the table you want to enter info in. To add regime info left click it and select the "Regime" option.

The screenshot shows an 'Edit' dialog window titled 'Edit'. The dropdown menu at the top left is now set to 'Regimes', indicated by a pink background. To its right are three buttons: a question mark icon, a blue 'Edit' button, and a close 'X' button. The main area contains a vertical list of fields with corresponding input boxes:

Field	Type
MemberID	Text
Exercise ID	Text
Start Date	Text
End Date	Text
Description	Text

4. Now enter all the regime information in the correctly labelled columns by left clicking on the Line edits and typing with the keyboard.

MemberID

Exercise ID

Start Date

End Date

Description

Line Edits where the information is input with a keyboard

- 5.
- The column “Description” must be enter as text with the characters a-Z and should not exclusively be numbers but can include all characters apart from quotation marks or double quotation marks
 - The columns “MemberID” and “ExerciseID” must be entered as any combination of the integer numbers 0-9 only. The columns “MemberID” and “ExerciseID” must be existing items in the database for the regime you want to edit the information of
 - The columns “startDate” and “endDate” must be dates in the format yyyy-mm-dd (For example 2015-05-12)

6. Click the “Edit” push button to edit all of the info you’ve input into the database.

MemberID

Exercise ID

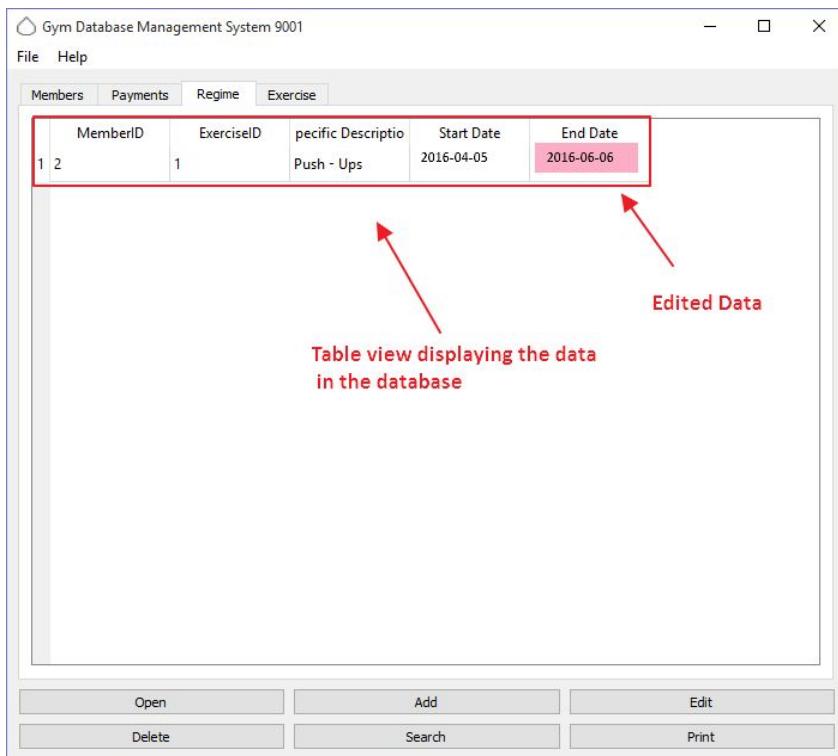
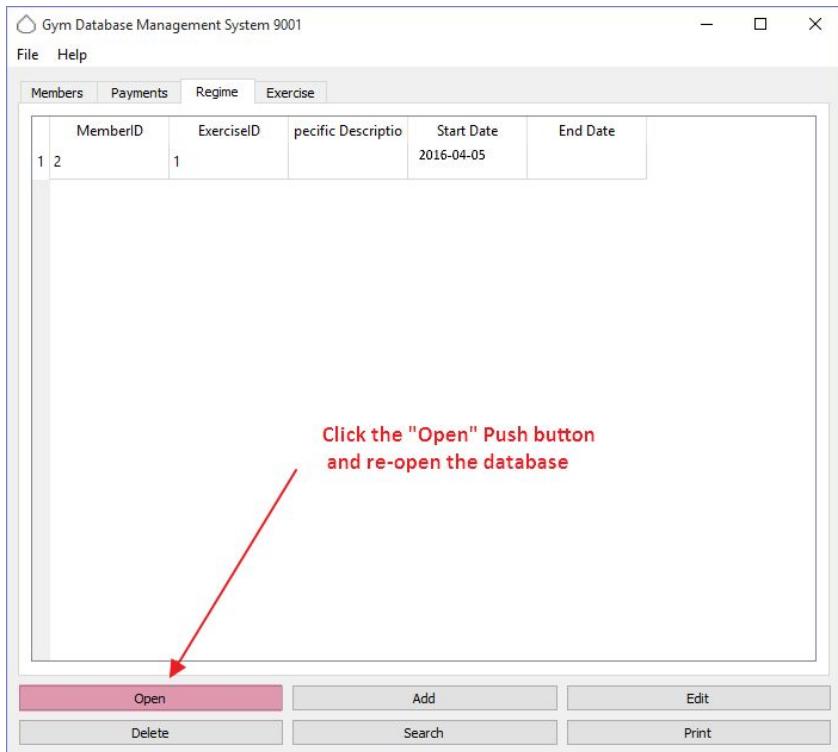
Start Date

End Date

Description

Click the "Edit" push button

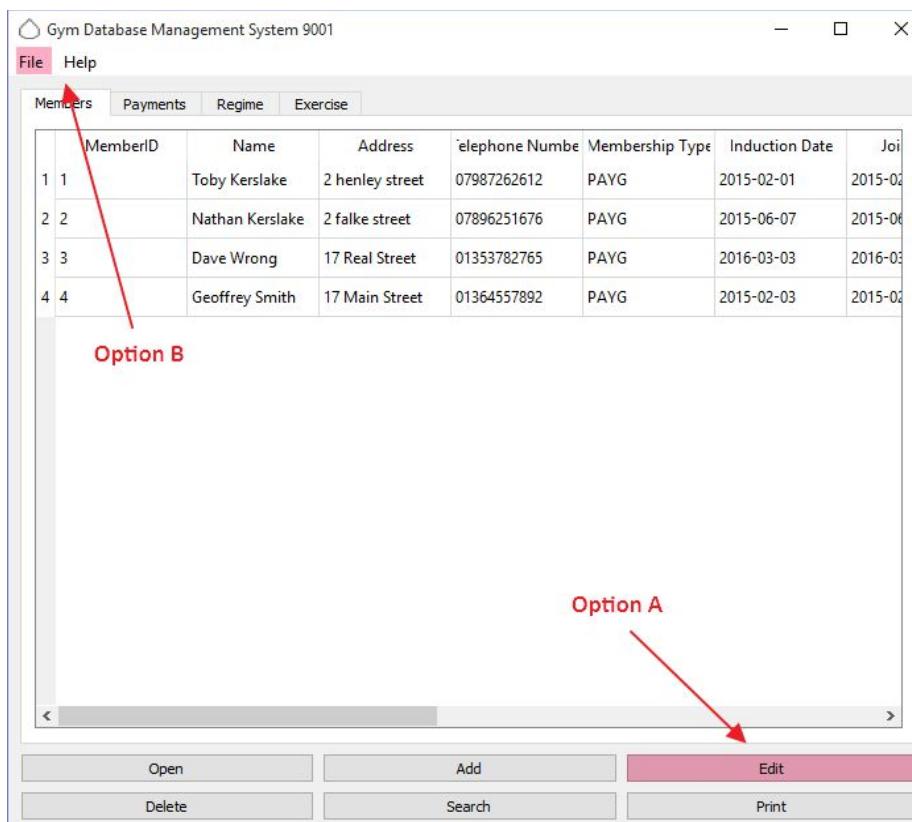
7. Re-open the database you've edited the data of.



Editing an exercise

Using this program allows you to edit and save data about an exercise

1. There are 3 ways to edit exercise data
 - a. Find the “Edit” Push Button and left click it
 - b. Go to the Tool bar and highlight the “File” menu item and scroll down to the “Edit” item and left click
 - c. Press Ctrl + E



2. After starting the Edit function an Edit dialog will open

The screenshot shows an 'Edit' dialog window titled 'Edit'. At the top left is a person icon, followed by the word 'Edit'. To the right are three buttons: a question mark icon, a blue 'Edit' button, and a close 'X' button. Below the buttons is a dropdown menu labeled 'Members' with a downward arrow. The main area contains twelve input fields, each with a label and a corresponding text box:

- MemberID
- Name
- Address
- Telephone Number
- Membership Type
- Induction Date
- Join Date
- How Paid
- Amount
- Registration Fee
- Registration Date
- Payment Type
- Comments

3. At the top of this dialog there is a combobox that lets you switch the table you want to enter info in. To edit exercise info left click it and select the "Exercise" option.

The screenshot shows an 'Edit' dialog window titled 'Edit'. At the top left is a person icon, followed by the word 'Edit'. To the right are three buttons: a question mark icon, a blue 'Edit' button, and a close 'X' button. Below the buttons is a dropdown menu labeled 'Exercises' with a pink background and a downward arrow. The main area contains three input fields:

- Exercise ID
- Name
- Description

4. Now enter all the exercise information in the correctly labelled columns by left clicking on the Line edits and typing with the keyboard.

Exercise ID Name Description

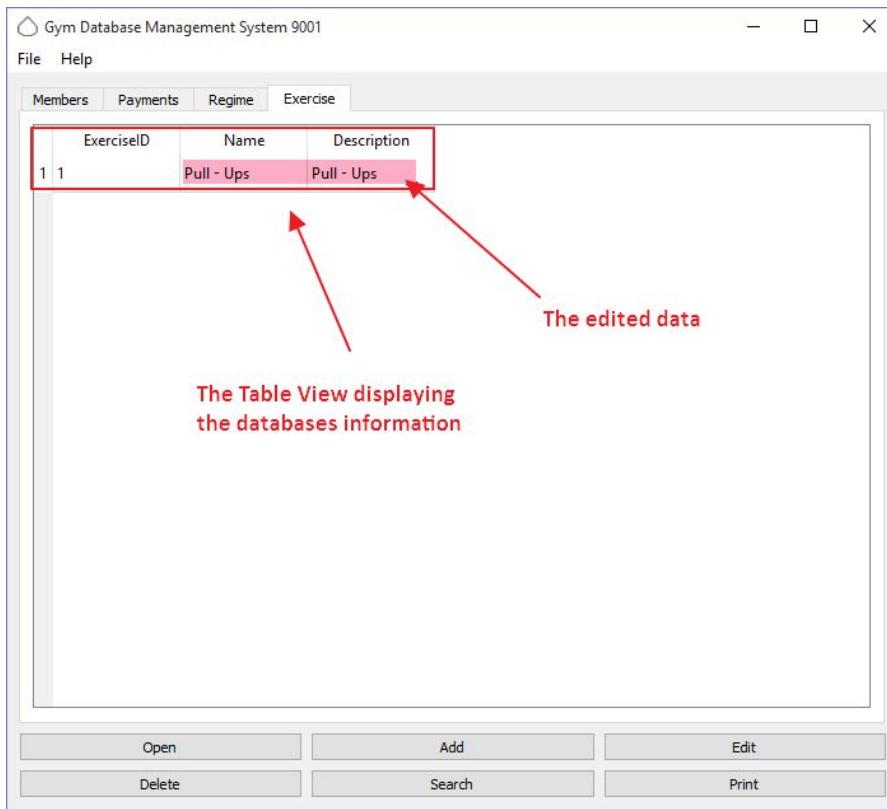
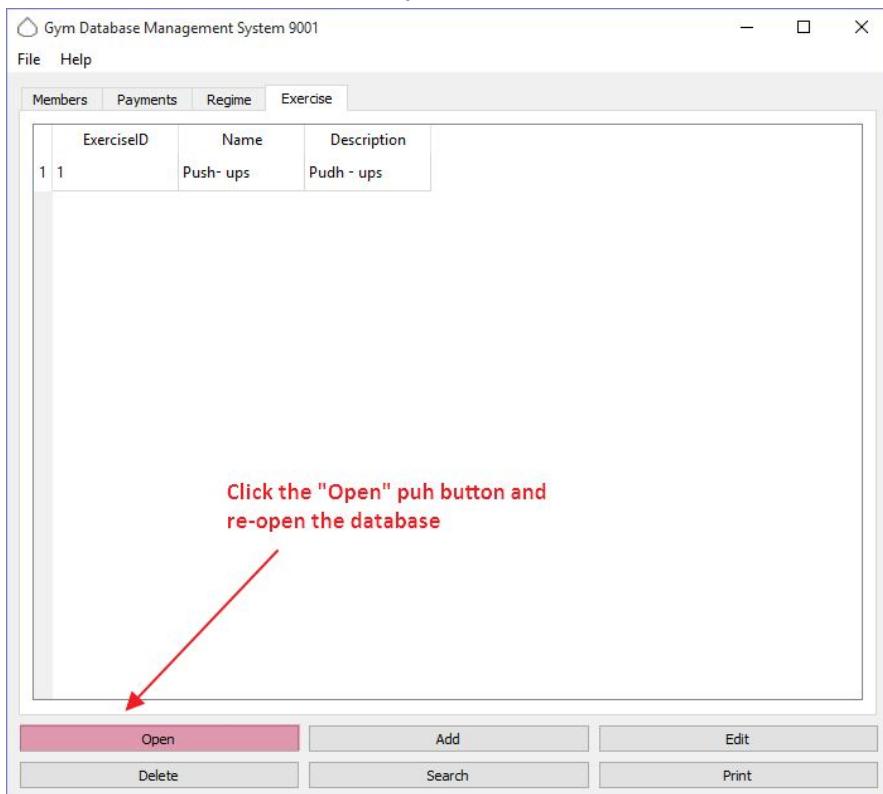
Line Edits where information is input with a keyboard

- 5.
- The columns "Description" and "Name" must be entered as text with the characters a-Z and should not exclusively be numbers but can include all characters apart from quotation marks or double quotation marks
 - The column "ExerciseID" must be entered as any combination of the integer numbers 0-9 only. The column ExerciseID must also be an existing ExerciseID for the member you want to edit the information of
6. Click the "Edit" push button to edit all of the info you've input into the database.

Exercise ID Name Description

Click the "Edit" push button

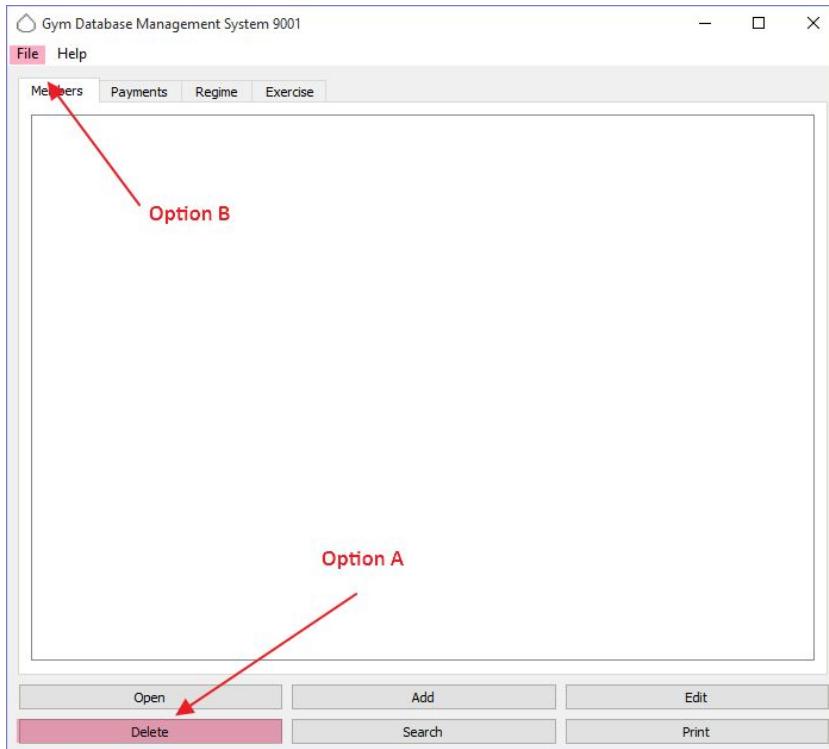
7. Re-open the database you've edited the data of.



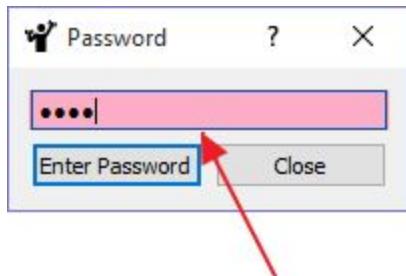
Deleting an item from a table

Using this program allows you to delete certain items from your database

1. There are 3 ways to delete data
 - a. Find the “Delete” Push Button and left click it
 - b. Go to the Tool bar and highlight the “file” menu item and scroll down to the “Delete” Item and left click
 - c. Press Ctrl + D

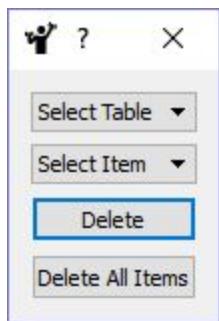


2. You will now be asked to enter the password (see tutorial : Entering Password)

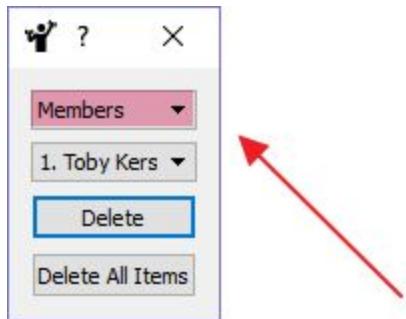


Type Password
here

3. After entering the password correctly a Delete dialog will open

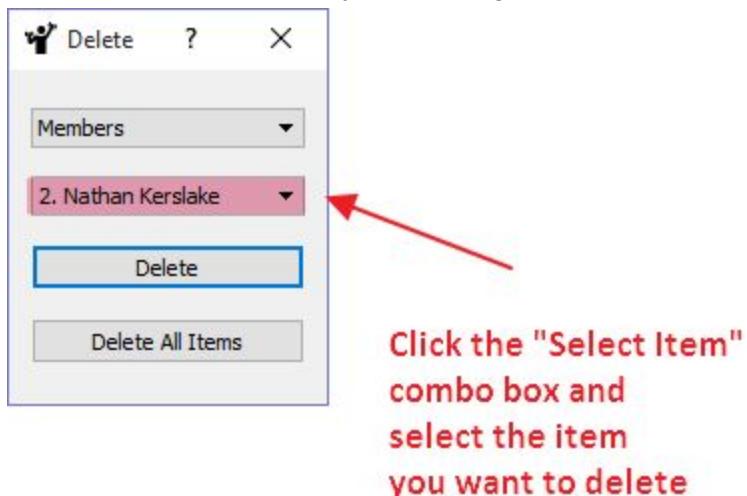


4. Left click the "Select Table" combobox and select the table you want to delete an item from from the drop down menu by left clicking

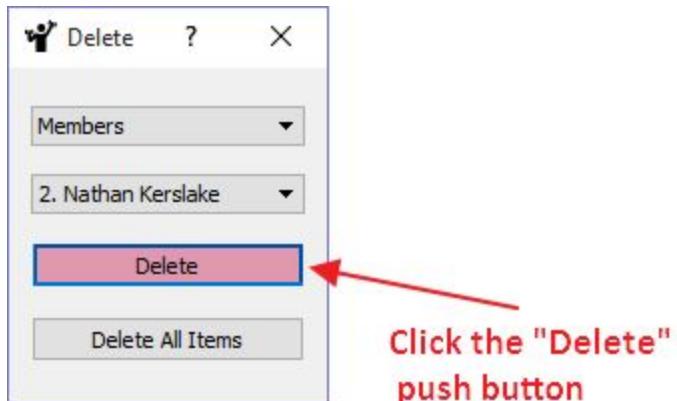


Click the "Select Table"
combo box and
select the table
you want to delete from

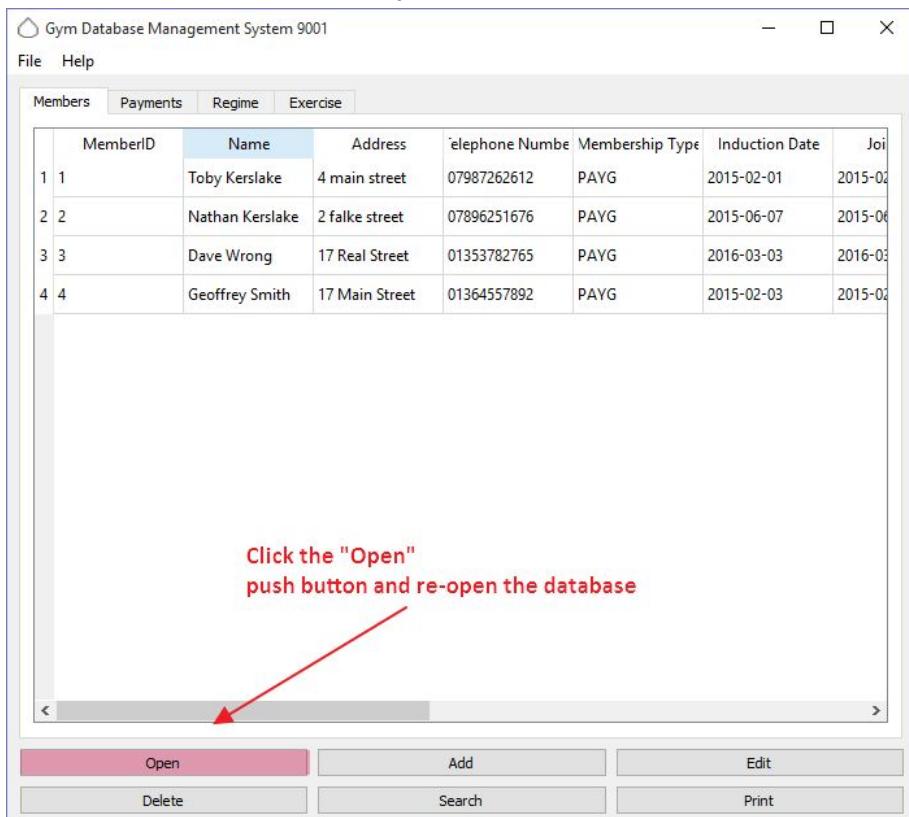
5. Left click the “Select Item” combobox and select the item you want to delete from the drop down menu by left clicking



6. Left click on the “Delete” push button to delete the item



7. Re-open the database you've deleted data from.



Gym Database Management System 9001

File Help

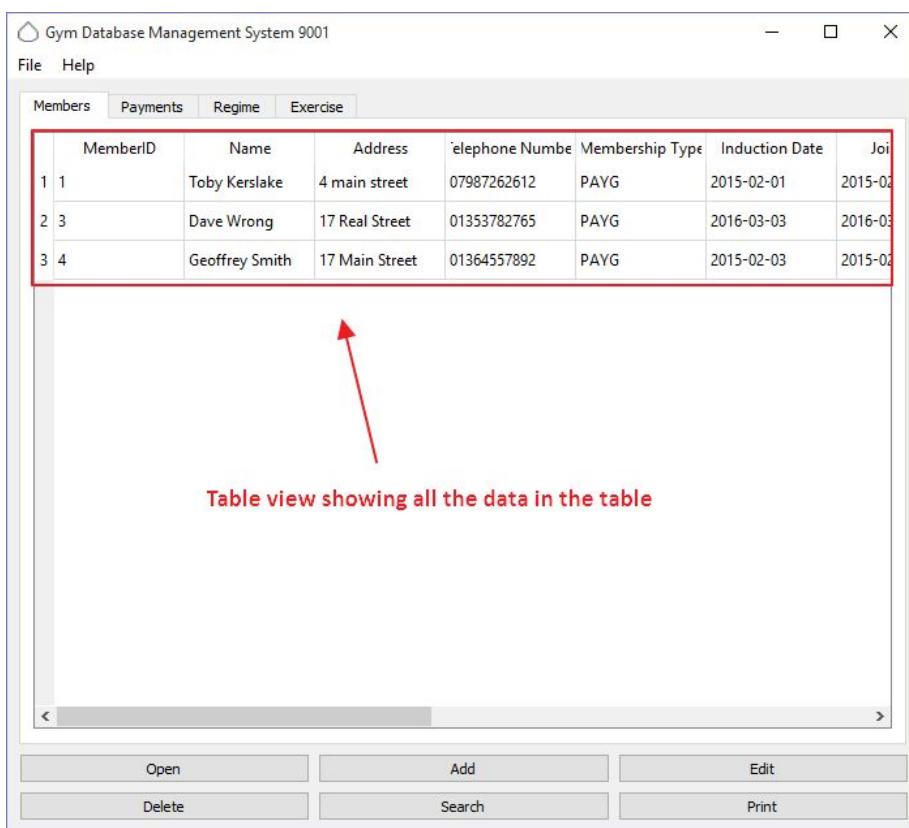
Members Payments Regime Exercise

	MemberID	Name	Address	Telephone Number	Membership Type	Induction Date	Join Date
1	1	Toby Kerslake	4 main street	07987262612	PAYG	2015-02-01	2015-02-01
2	2	Nathan Kerslake	2 falke street	07896251676	PAYG	2015-06-07	2015-06-07
3	3	Dave Wrong	17 Real Street	01353782765	PAYG	2016-03-03	2016-03-03
4	4	Geoffrey Smith	17 Main Street	01364557892	PAYG	2015-02-03	2015-02-03

Click the "Open" push button and re-open the database

< >

Open Add Edit
Delete Search Print



Gym Database Management System 9001

File Help

Members Payments Regime Exercise

	MemberID	Name	Address	Telephone Number	Membership Type	Induction Date	Join Date
1	1	Toby Kerslake	4 main street	07987262612	PAYG	2015-02-01	2015-02-01
2	3	Dave Wrong	17 Real Street	01353782765	PAYG	2016-03-03	2016-03-03
3	4	Geoffrey Smith	17 Main Street	01364557892	PAYG	2015-02-03	2015-02-03

Table view showing all the data in the table

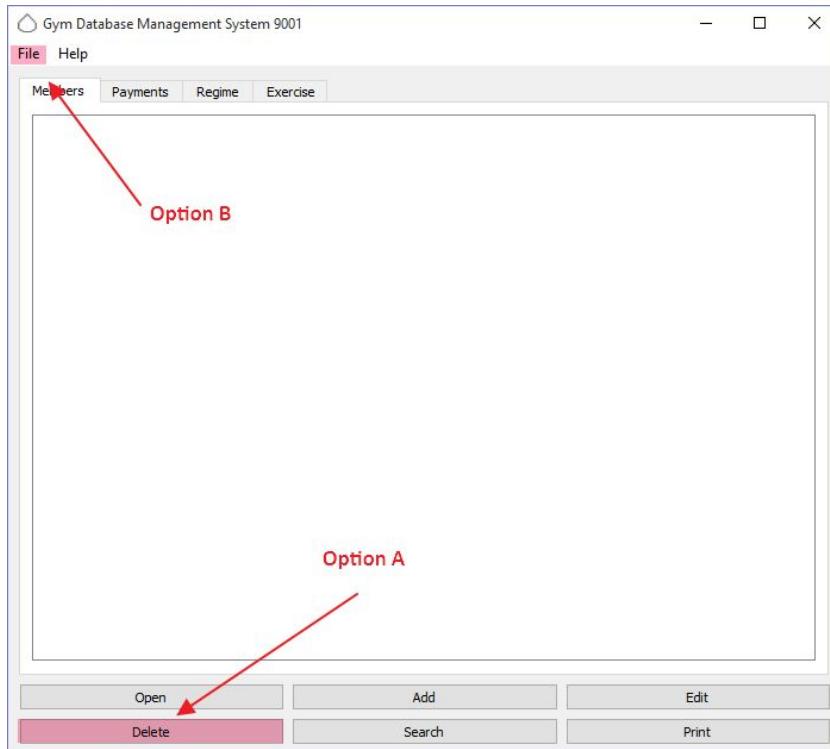
< >

Open Add Edit
Delete Search Print

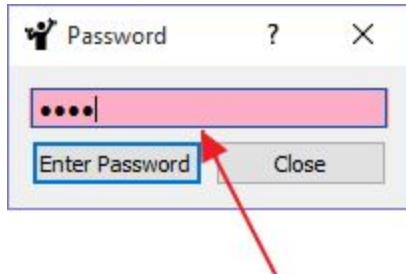
Deleting an entire Table

Using this program allows you to delete all items from a table in your database

1. There are 3 ways to delete data
 - a. Find the “Delete” Push Button and left click it
 - b. Go to the Tool bar and highlight the “file” menu item and scroll down to the “Delete” Item and left click
 - c. Press Ctrl + D



2. You will now be asked to enter the password (see tutorial : Entering Password)

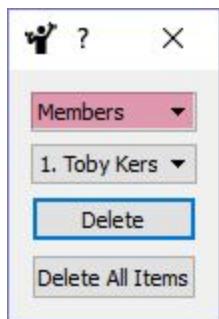


Type Password
here

3. After entering the password correctly a Delete dialog will open

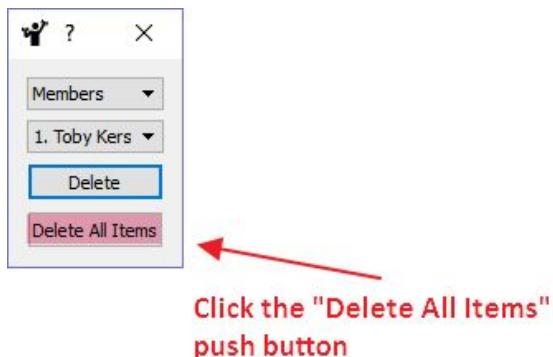


4. Left click the “Select Table” combobox and select the table you want to delete an item from from the drop down menu by left clicking on the item

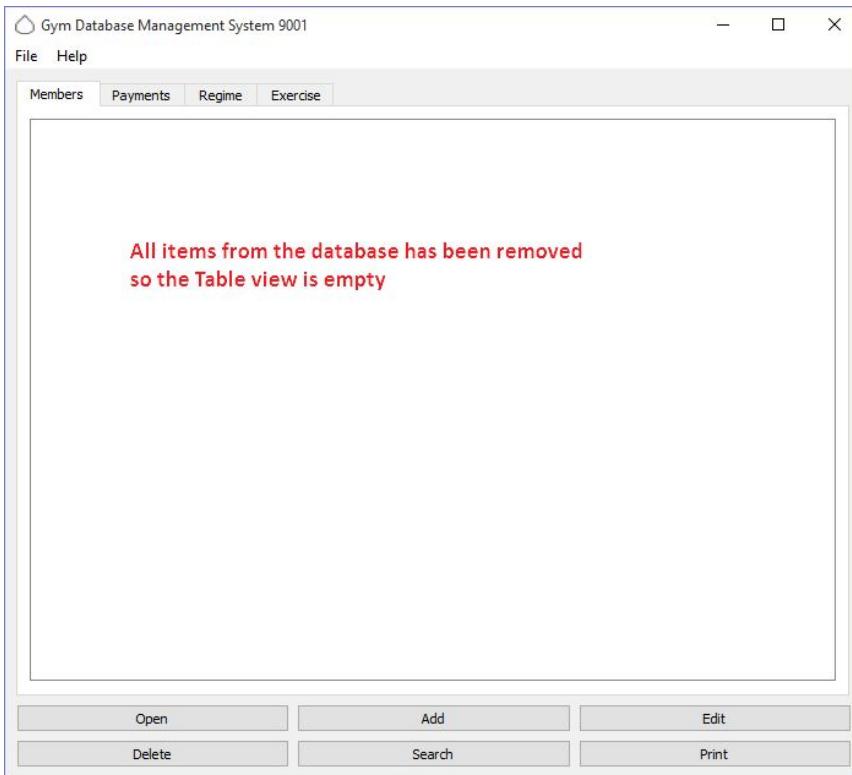
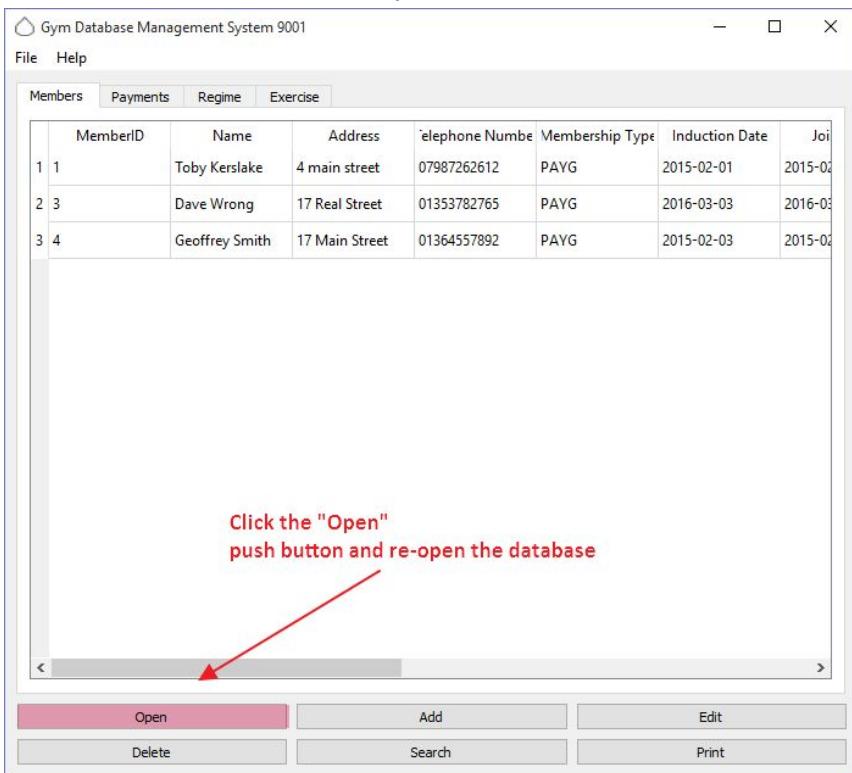


Click the "Select Table"
combo box and
select the table
you want to delete from

5. Left click on the “Delete All Items” button to delete the item



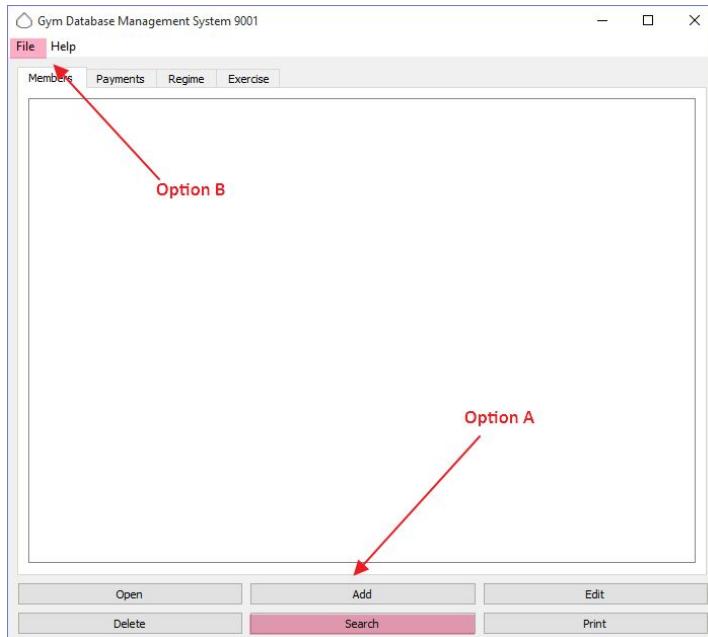
6. Re-open the database you've deleted data from.



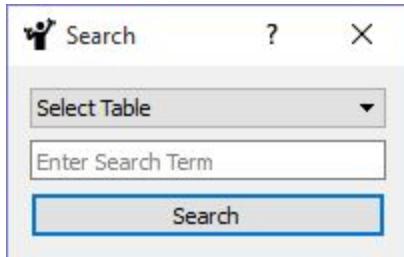
Searching through a table

Using this program allows you to search through a table in your database

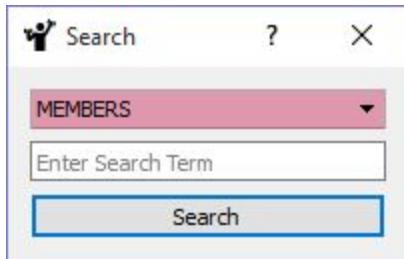
1. There are 3 ways to search through data
 - a. Find the “Search” Push Button and left click it
 - b. Go to the Tool bar and highlight the “file” menu item and scroll down to the “Search” item and left click
 - c. Press Ctrl + S



2. A Search dialog will now open

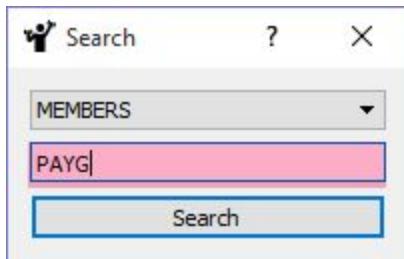


3. Left click the “Select Table” combobox and select the table you want to search through from the drop down menu by left clicking on the item



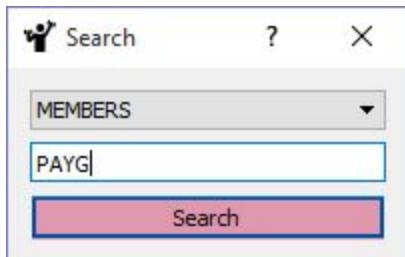
Click the "Select Table" combo box and select the table you want to search through

4. Now enter your search term (can be any datatype) in the “Enter Search Term” Line Edit by left clicking on it



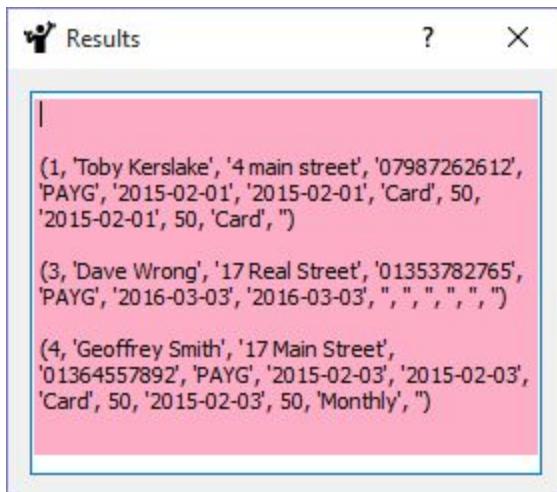
Line Edit to enter your search term with a keyboard

5. Left click the “Search” push button to search through the database



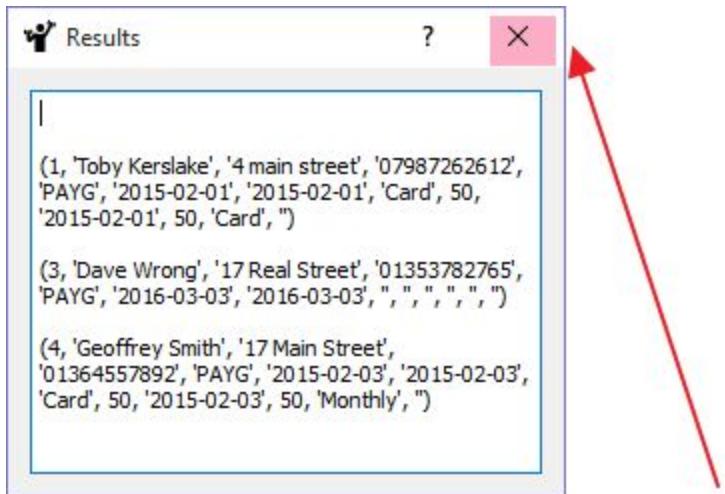
Click the "search" push button

6. A Results dialog will now appear containing your results



The Results are displayed
in a
results dialog

7. Left click the “X” in the top right corner of the Results window to close it and return to the main user interface

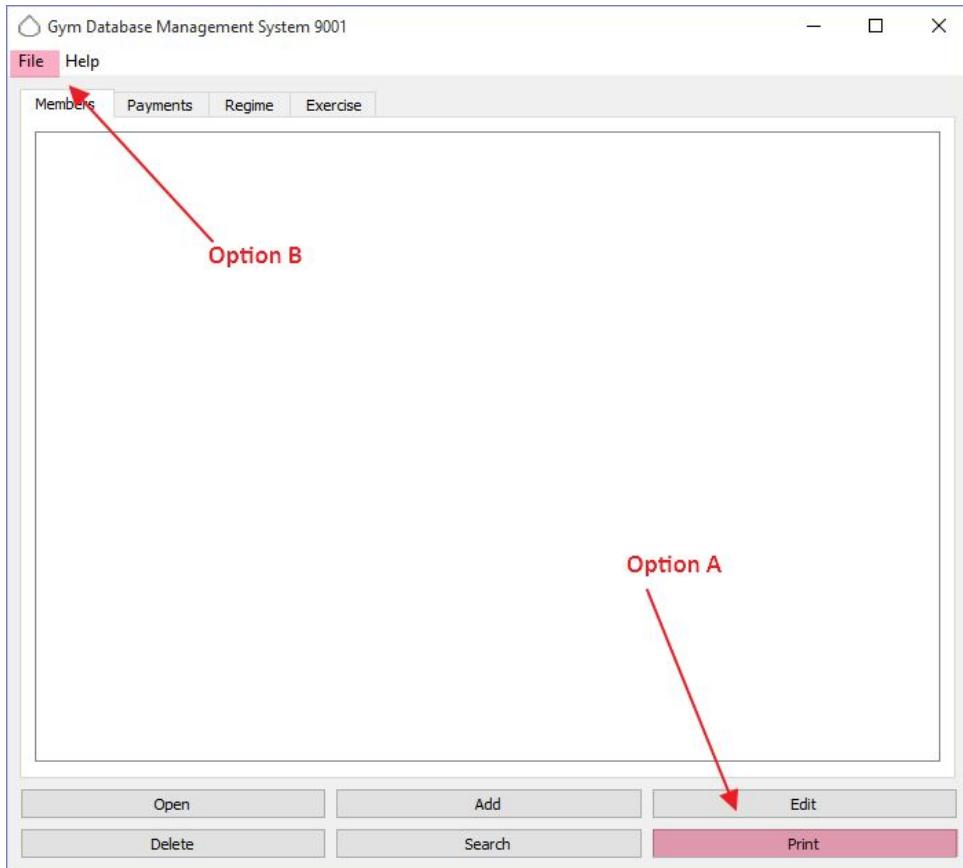


Click the "X" button in the top right corner to return to the main interface

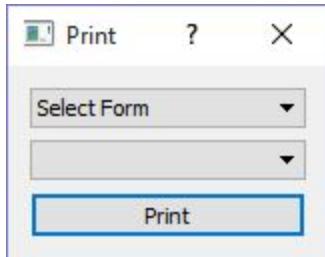
Printing member Info/Invoice/Regime Info

Using this program allows you to print forms from your database

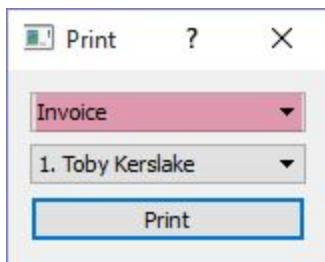
1. There are 3 ways to print data
 - a. Find the “Print” Push Button and left click it
 - b. Go to the Tool bar and highlight the “file” menu item and scroll down to the “Print” item and left click
 - c. Press Ctrl + P



2. A Print dialog will now open

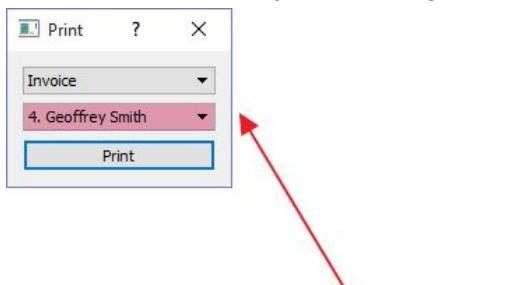


3. Left Click the "Select Form" combo box and select the form you want to print from the drop down menu by left clicking on the item



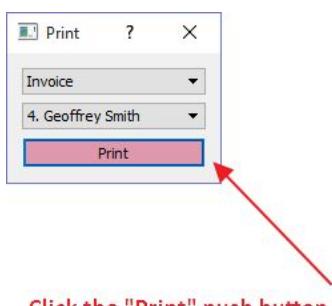
Click the "Select Form" combo box and click on the kind of form you want to print

4. Left click the “Select Item” combo box and select the item you want to print from the drop down menu by left clicking on the item



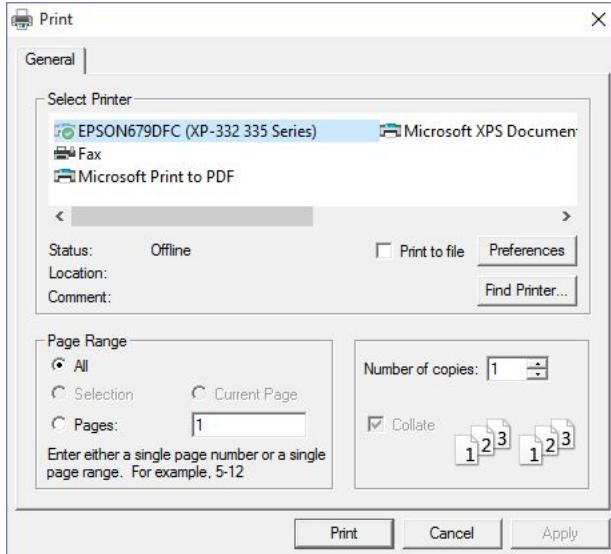
Click the "Select Item" combo box and click on the item you want to print

5. Left click the “Print” push button to select a printer

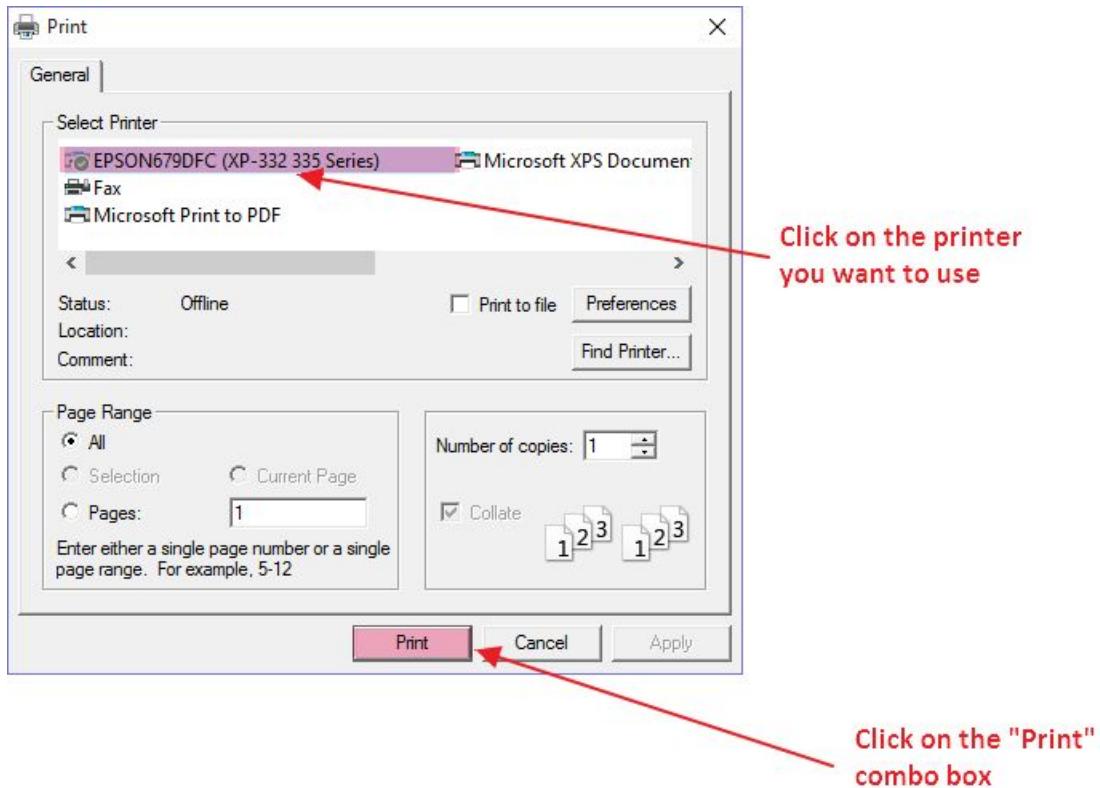


Click the "Print" push button

6. A windows Print dialog will now open



7. Select the printer you want to print form and left click the “Print” push button to print the form

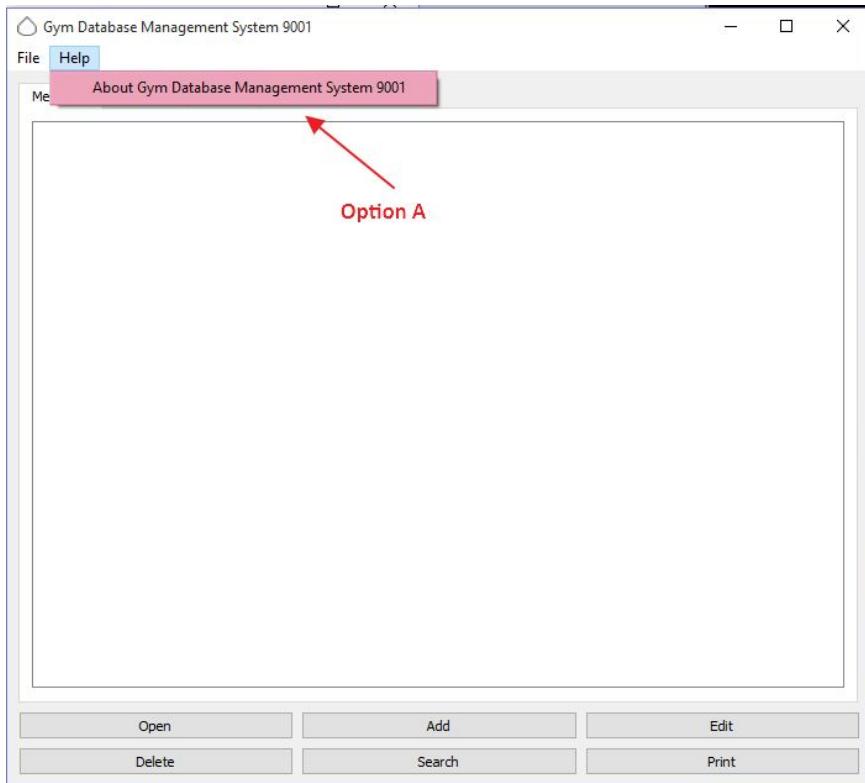


Accessing the about section

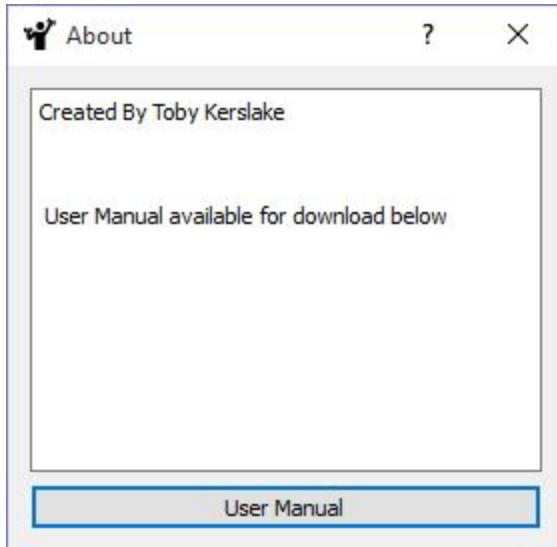
This program has an about section detailing the creator and linking a digital version of this user manual

1. There are 2 ways to open the about section

- Go to the Tool bar and highlight the “help” menu item and scroll down to the “About” item and left click
- Press Ctrl + A



2. An About dialog will now open



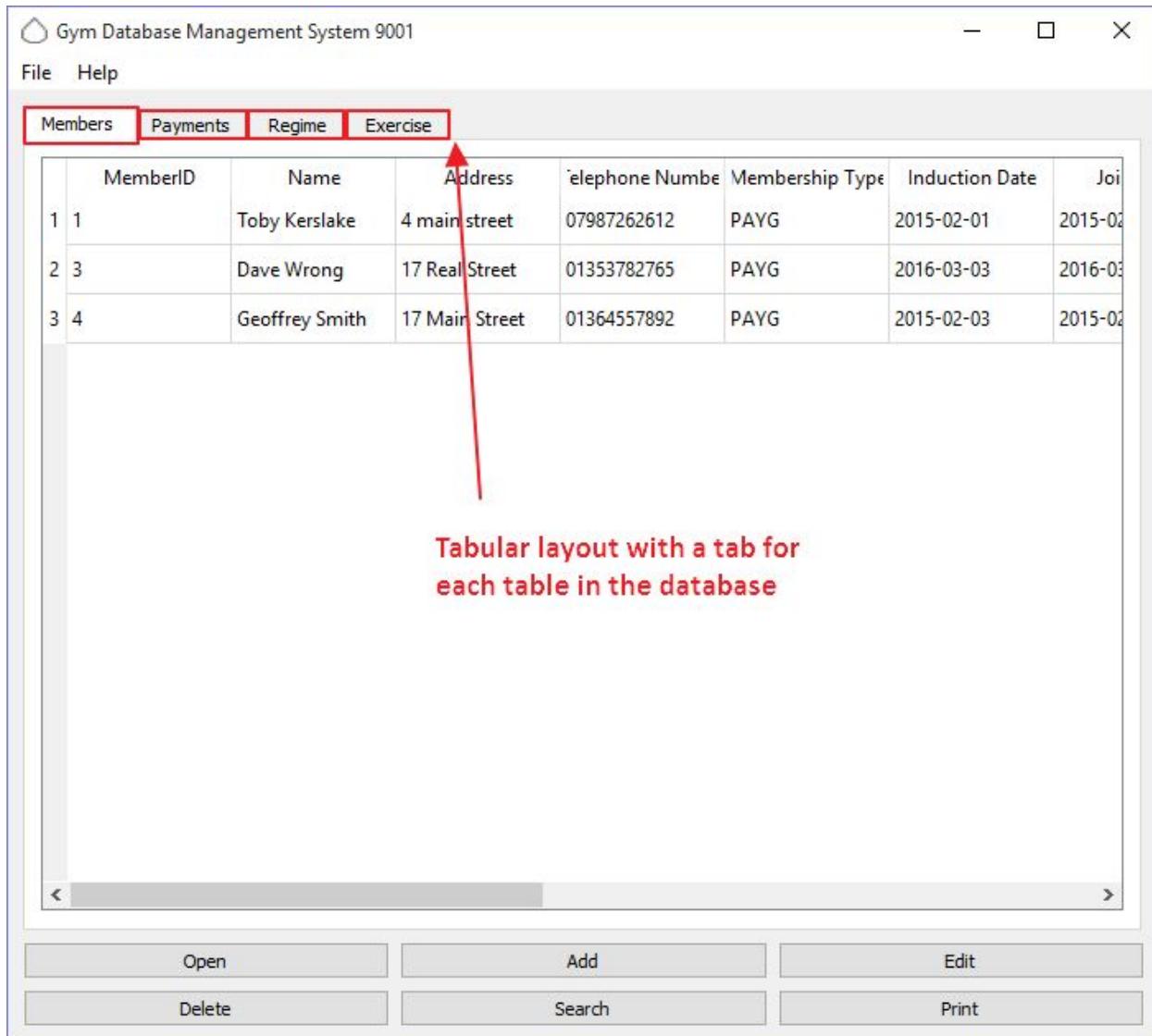
3. If you want to access a digital copy of this user manual left click on the "User Manual" push button



Navigating the database

This program uses a tabular layout for displaying the loaded database and can't be navigated easily

1. After a database is opened it will display its “Member”, “Payment”, “Regime” and “Exercise” tables will be displayed in 4 tabs selectable from a table view at the top of the table view.



The screenshot shows a window titled "Gym Database Management System 9001". At the top, there is a menu bar with "File" and "Help" options. Below the menu is a horizontal bar with four tabs: "Members", "Payments", "Regime", and "Exercise". A red arrow points upwards from the bottom of the page towards the "Members" tab. The main area displays a table with data for three members:

	MemberID	Name	Address	Telephone Number	Membership Type	Induction Date	Join Date
1	1	Toby Kerslake	4 main street	07987262612	PAYG	2015-02-01	2015-02-01
2	3	Dave Wrong	17 Real Street	01353782765	PAYG	2016-03-03	2016-03-03
3	4	Geoffrey Smith	17 Main Street	01364557892	PAYG	2015-02-03	2015-02-03

Tabular layout with a tab for each table in the database

At the bottom of the window, there are several buttons: "Open", "Add", "Edit", "Delete", "Search", and "Print".

2. To view another table just left click on the tab with its name on it

The screenshot shows a window titled "Gym Database Management System 9001". The menu bar includes "File" and "Help". Below the menu is a tab bar with four tabs: "Members", "Payments" (which is highlighted with a red border), "Regime", and "Exercise". A data grid displays a single row of data:

MemberID	Payment Date	How Much	Paid
1 3	2016-05-04	50	1

A red arrow points from the text "Click on another Tab to view another" to the "Regime" tab. At the bottom of the window are several buttons: "Open", "Add", "Edit", "Delete", "Search", and "Print".

3. To scroll down or across long tables just left click on the horizontal or vertical scrollbar and drag them left or right(for the horizontal scroll bar) or up and down (for the vertical scroll bar) while holding down left click

Gym Database Management System 9001

File Help

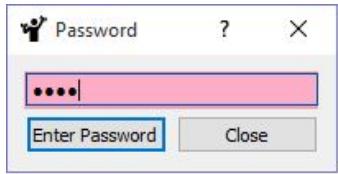
	MemberID	Name	Address	Telephone Number	Membership Type	Induction Date	
1	1	Toby Kerslake	4 main street	07987262612	PAYG	2015-02-01	2015-02-01
2	3	Dave Wrong	17 Real Street	01353782765	PAYG	2016-03-03	2016-03-03
3	4	Geoffrey Smith	17 Main Street	01364557892	PAYG	2015-02-03	2015-02-03
4	5						
5	6						
6	7						
7	8						
8	9						
9	23						
10	24						
11	25						
12	26						
13	27						
14	28						

Open Add Edit
Delete Search Print

Entering Password

This program requires the knowledge of a password to access the system and delete items

1. After the password dialog opens left click the “Enter Password” Line Edit and type the correct password using the keyboard



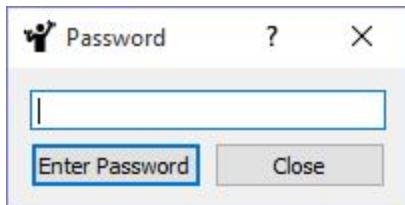
Type the correct password
in the Line Edit with
the keyboard

2. Left Click on the “Enter” button to enter your input password or simply press the “Enter” key on your keyboard



Click the "Enter Password"
push button

3. If the password dialog reopens this means the password was entered incorrectly



Errors

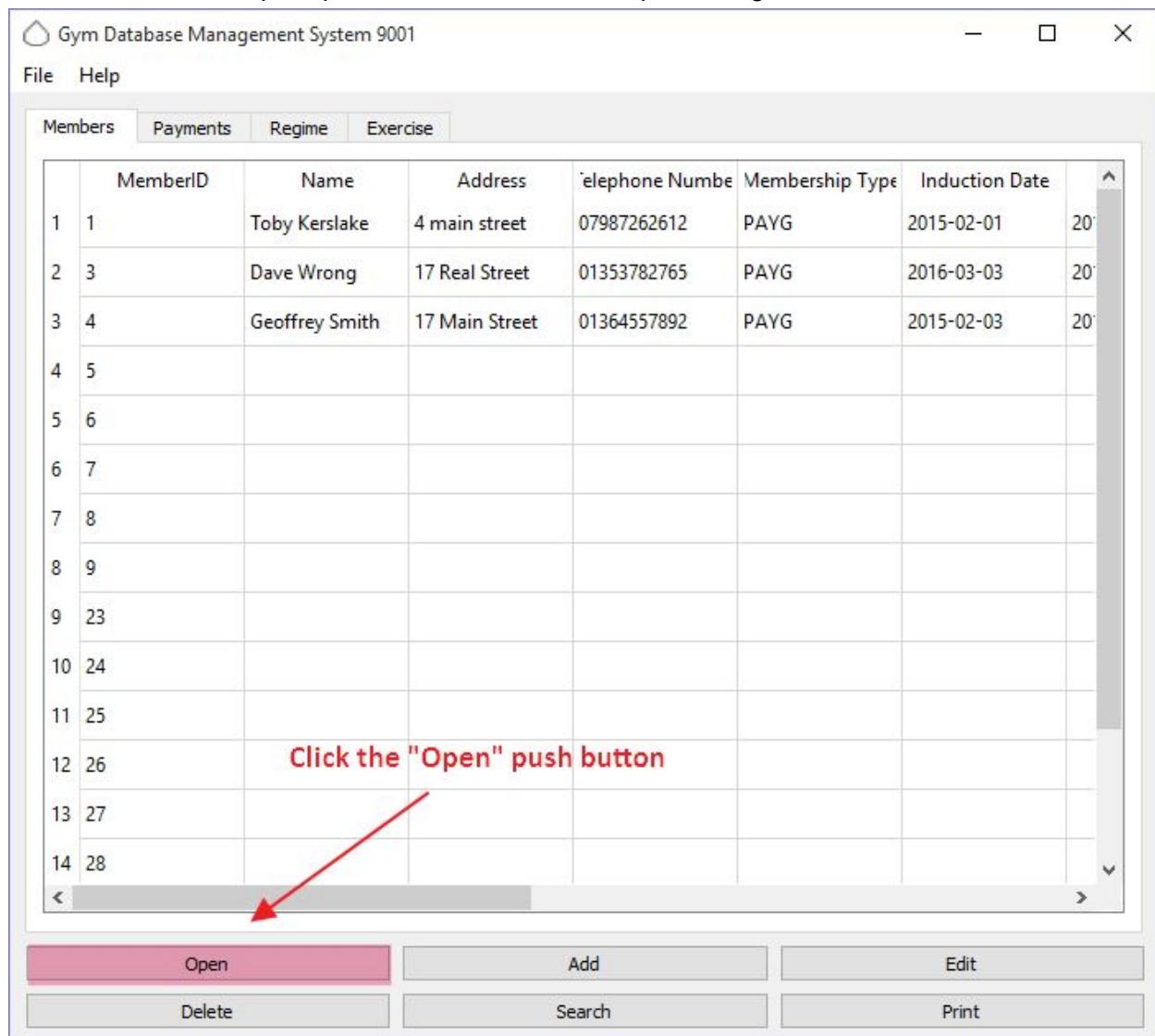
Certain sections of the system may raise an error under certain circumstances, each of which are described below in detail with tutorials on recovering and getting passed these errors.

No Functions Working

Sometimes none of the the functions will work correctly even though a database appears to be loaded in the table view. This is because if you try to open the database again or another database and then cancel the function the table view will appear as if the database is still loaded when in fact the program has closed it.

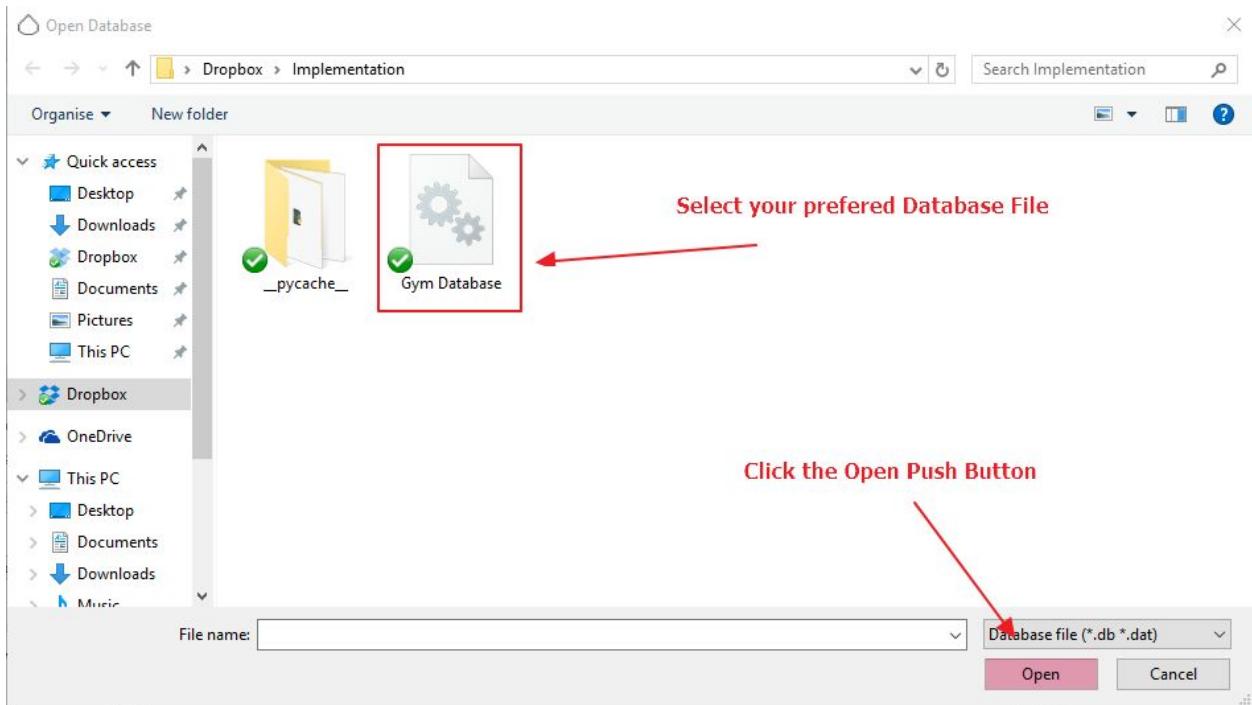
To solve the error you have to do the following:

1. Left click on the “Open” push button to initiate an open dialog



2. Re-select the correct database

3. Left click the “Open” push button to close the dialog and load the database



4. Continue with your user experience

Gym Database Management System 9001						
		Members		Payments		
		Regime		Exercise		
MemberID	Name	Address	Telephone Number	Membership Type	Induction Date	
1	Toby Kerslake	4 main street	07987262612	PAYG	2015-02-01	2015-02-01
2	Dave Wrong	17 Real Street	01353782765	PAYG	2016-03-03	2016-03-03
3	Geoffrey Smith	17 Main Street	01364557892	PAYG	2015-02-03	2015-02-03
4						
5						
6						
7						
8						
9						
23						
24						
25						
26						
27						
28						

At the bottom of the window, there are several buttons: Open, Add, Edit, Delete, Search, and Print.

Limitations

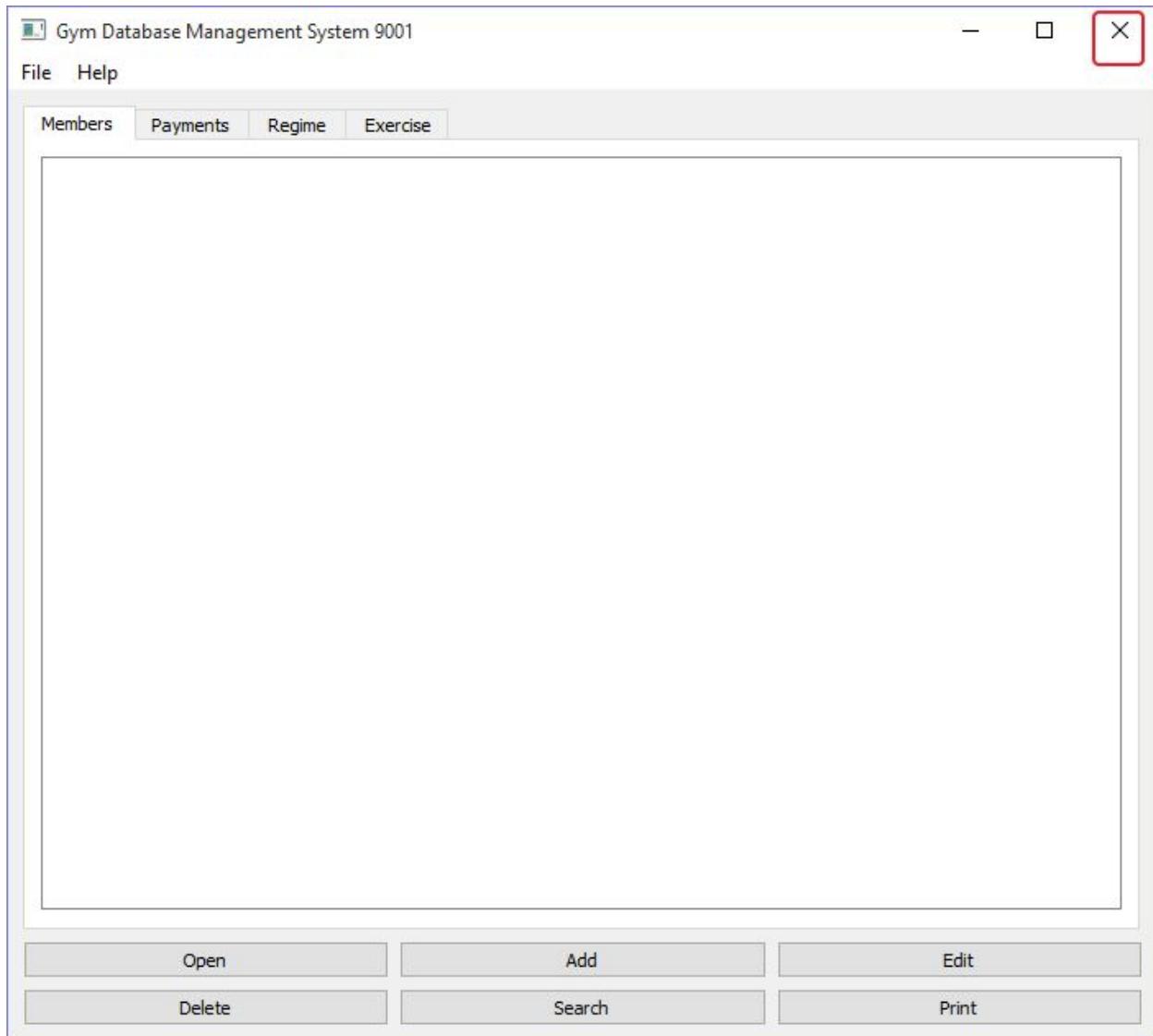
Currently the system does everything it was initially proposed to do in the design and analysis sections, with the exception of printable information input forms which couldn't be implemented due to time constraints and the fact that if needed my client can make his own forms using a word processor for his clients to fill out. The only other limitation is the error presented above which was never corrected due to time constraints and the unlikeliness of its occurrence as well as its easy solvability.

System Recovery

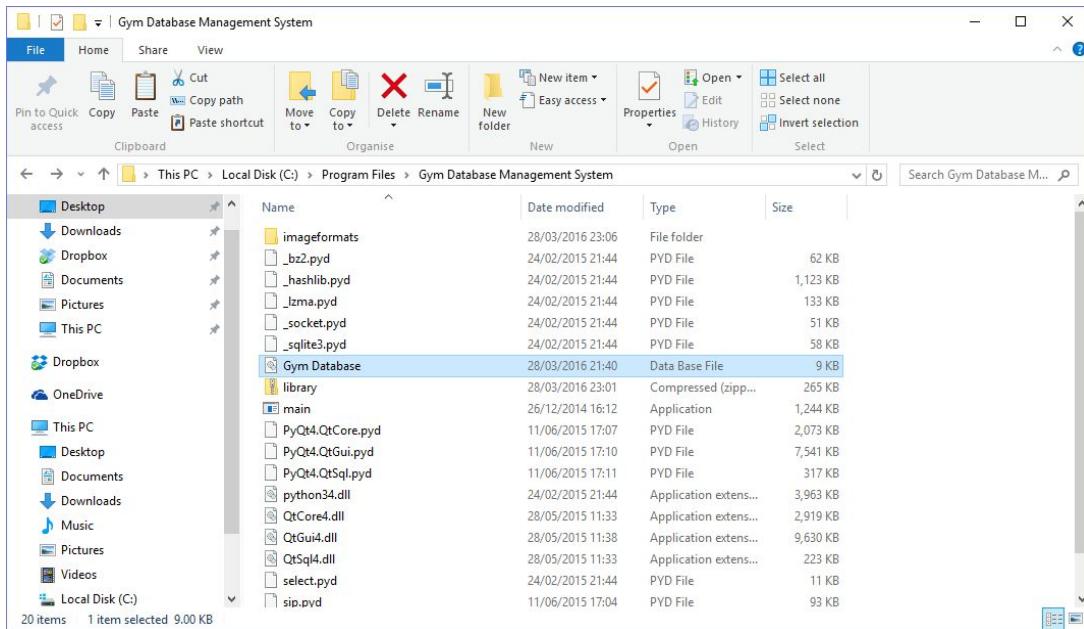
Backing Up Data

Backing up any databases is incredibly easy and can be achieved by following the steps below:

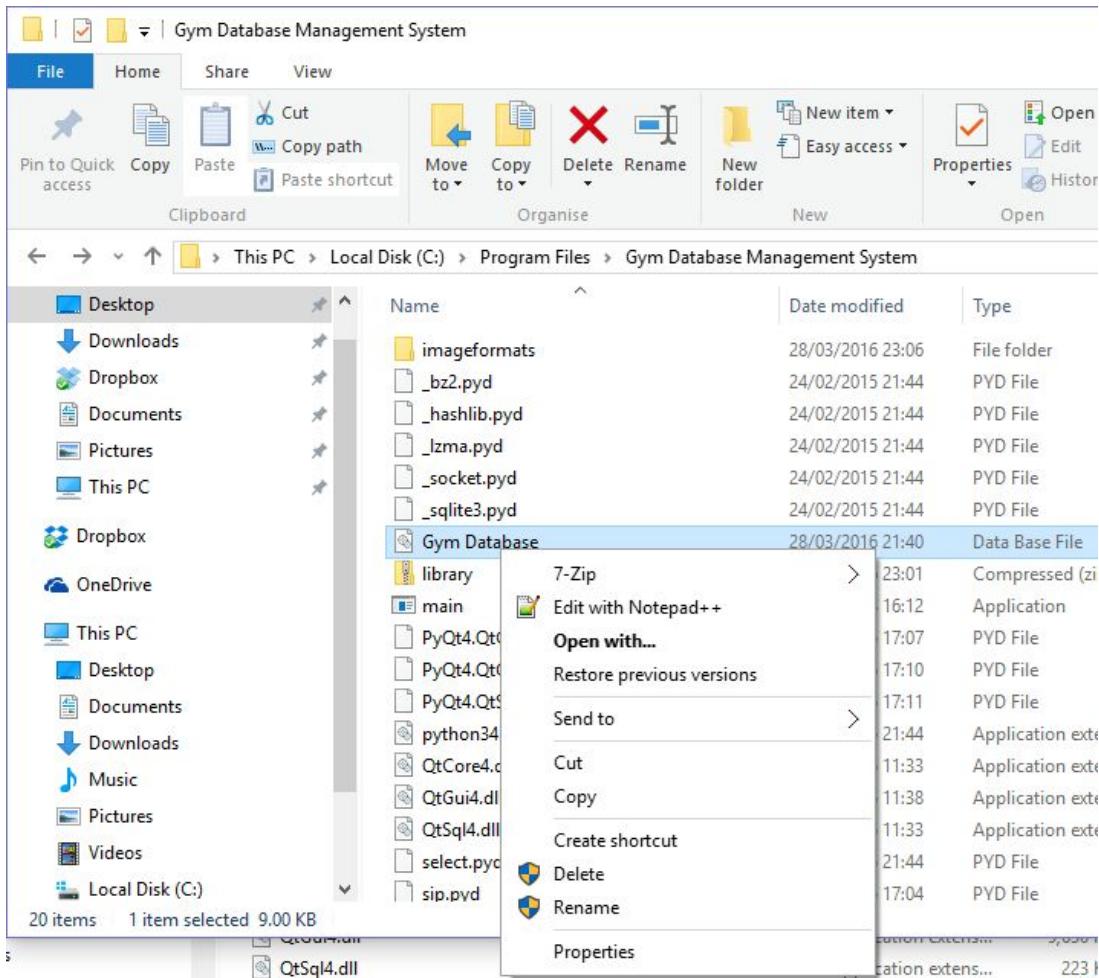
1. Close the system if it's already running



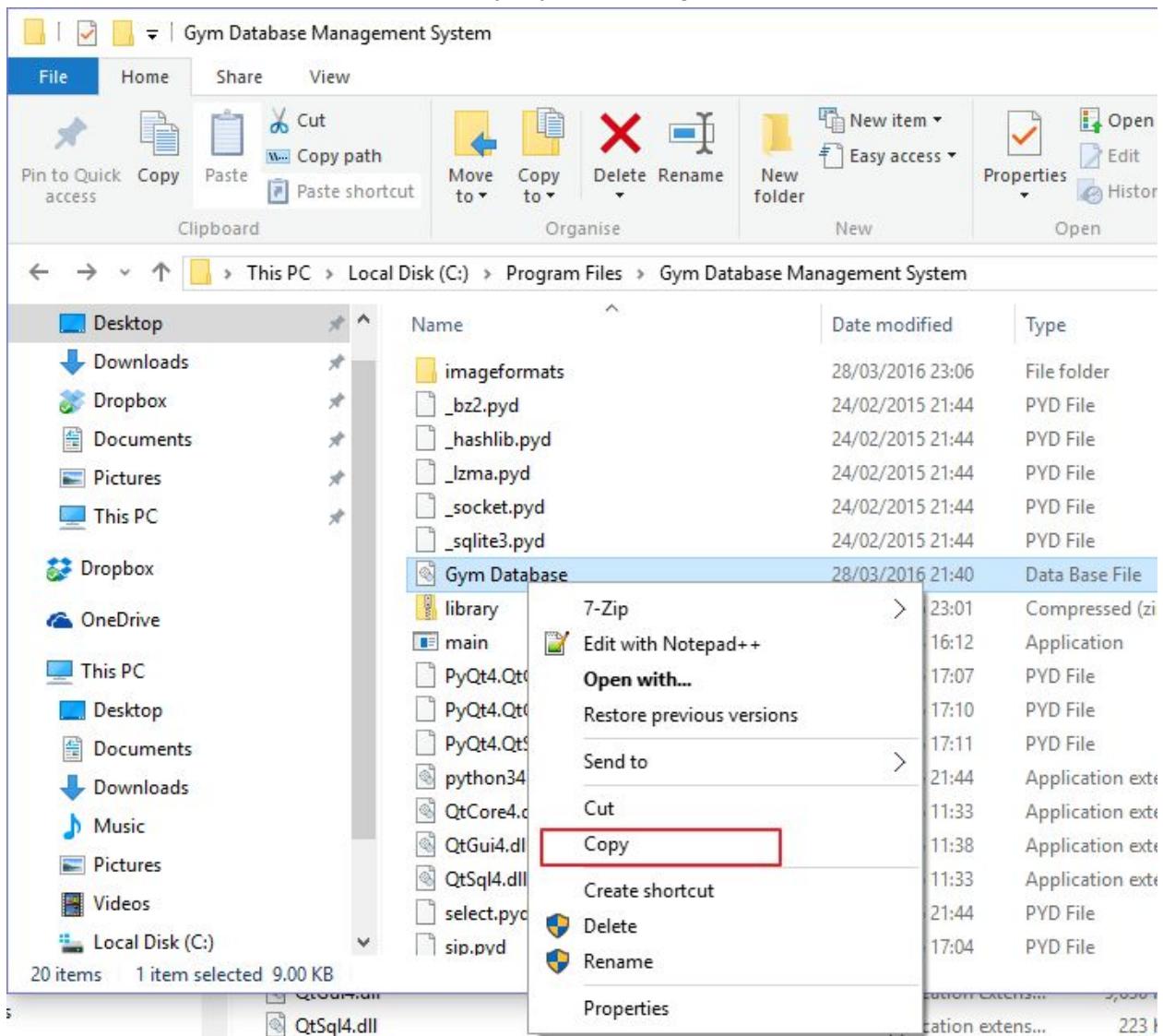
2. Navigate to the directory where the System Executable is stored



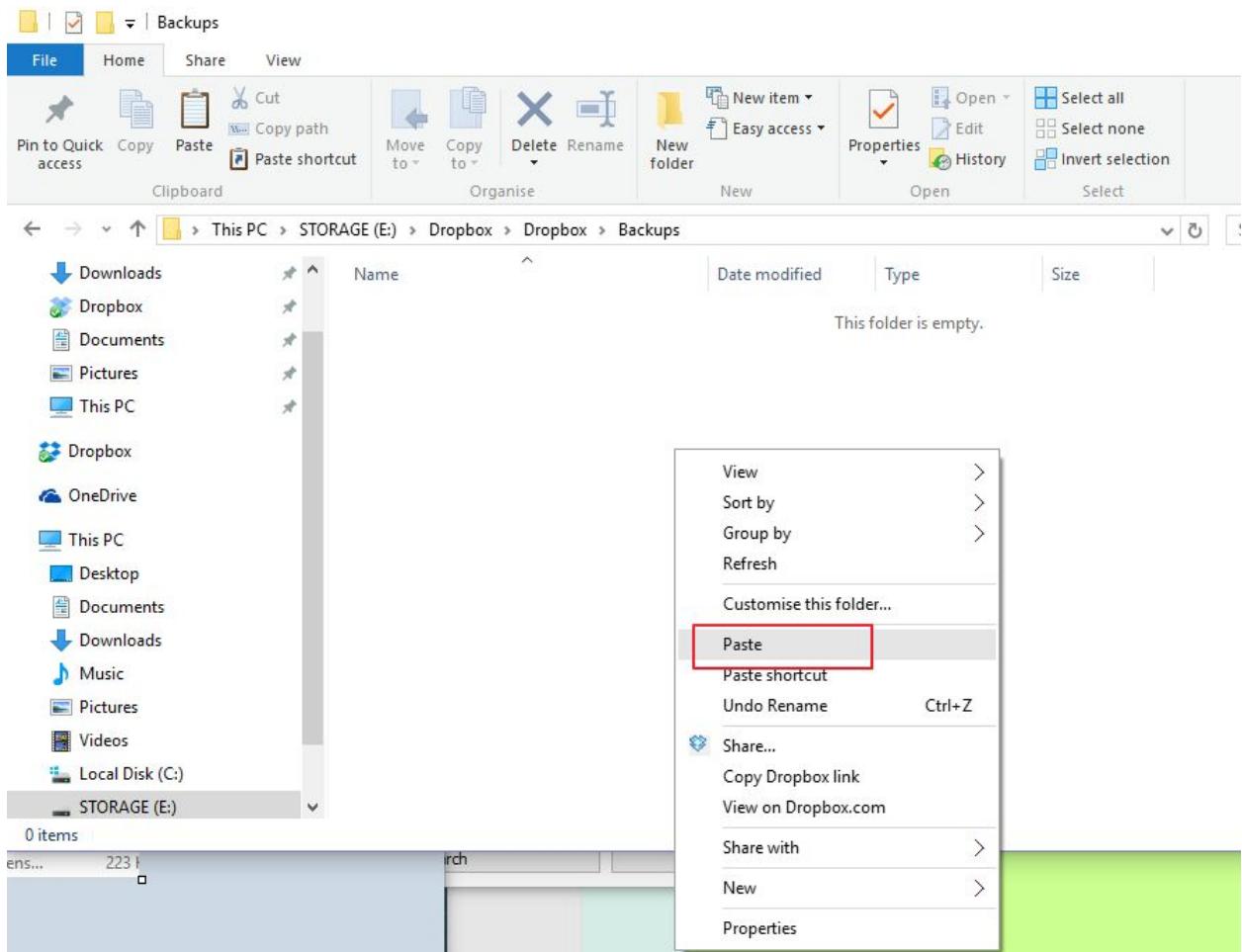
3. Find the database file, by default it's labelled "gym_database.db", and right click the file



4. From the drop down menu select “Copy” by left clicking



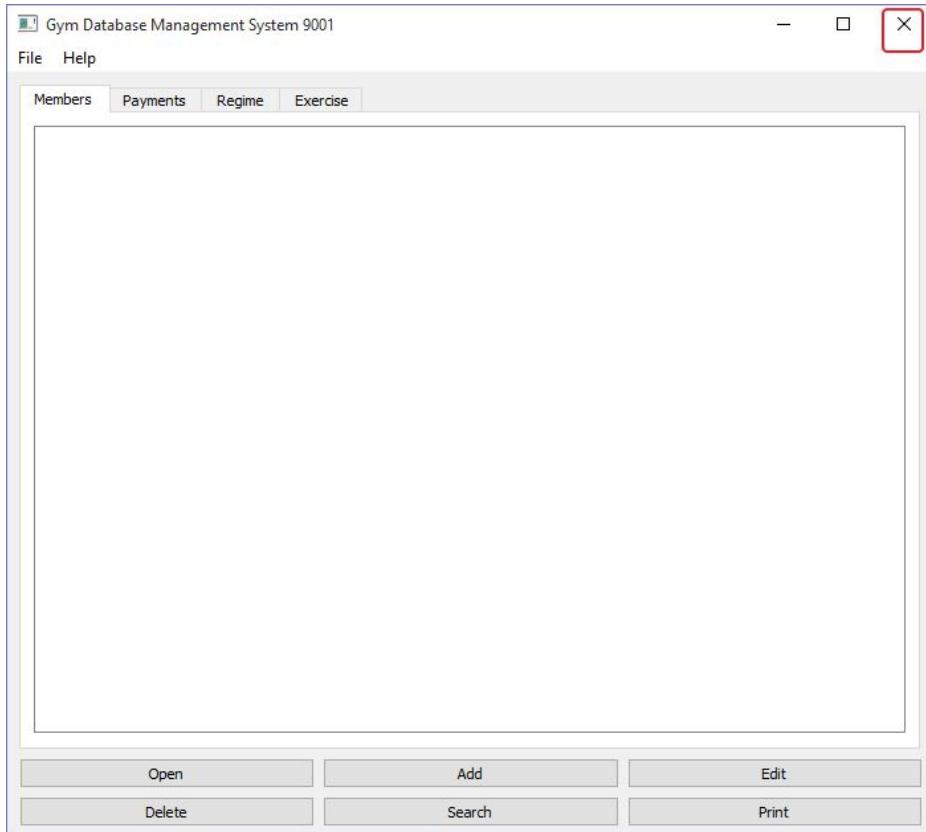
5. Right click and Select “Paste” from the drop down menu by left clicking in the location you want to store the backup.



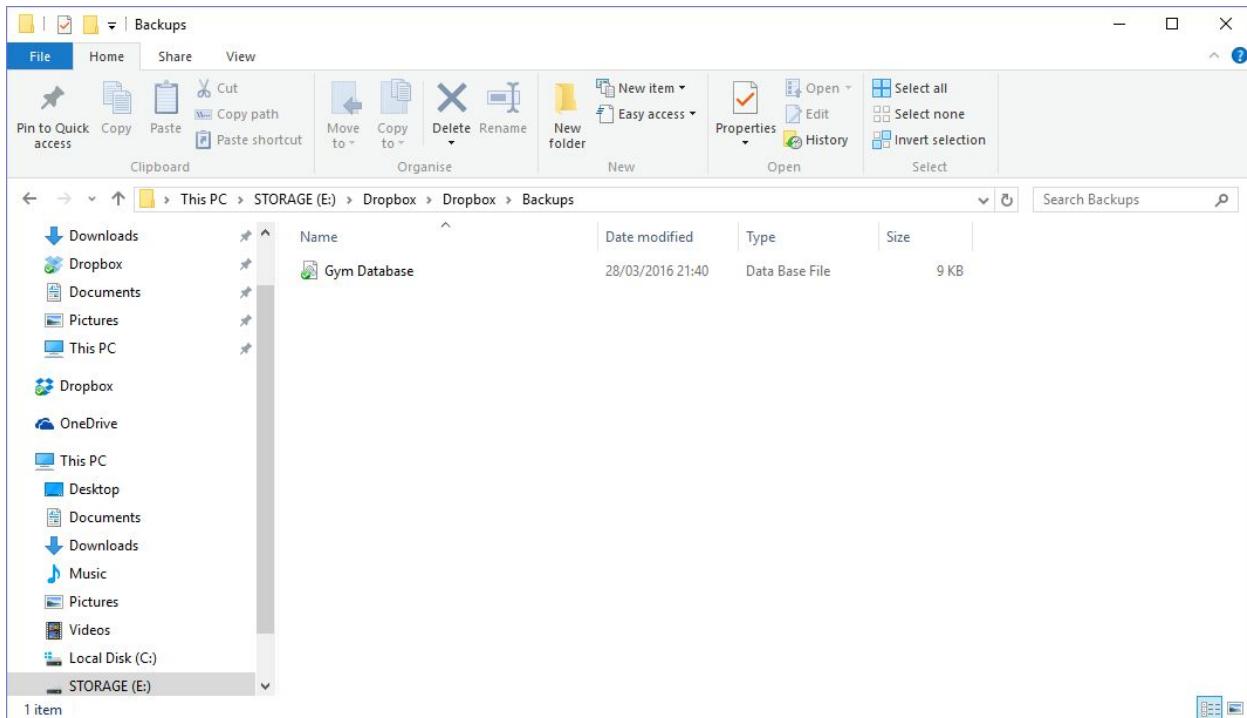
Restoring Data

In the event you need to restore data you can easily do this by doing the following:

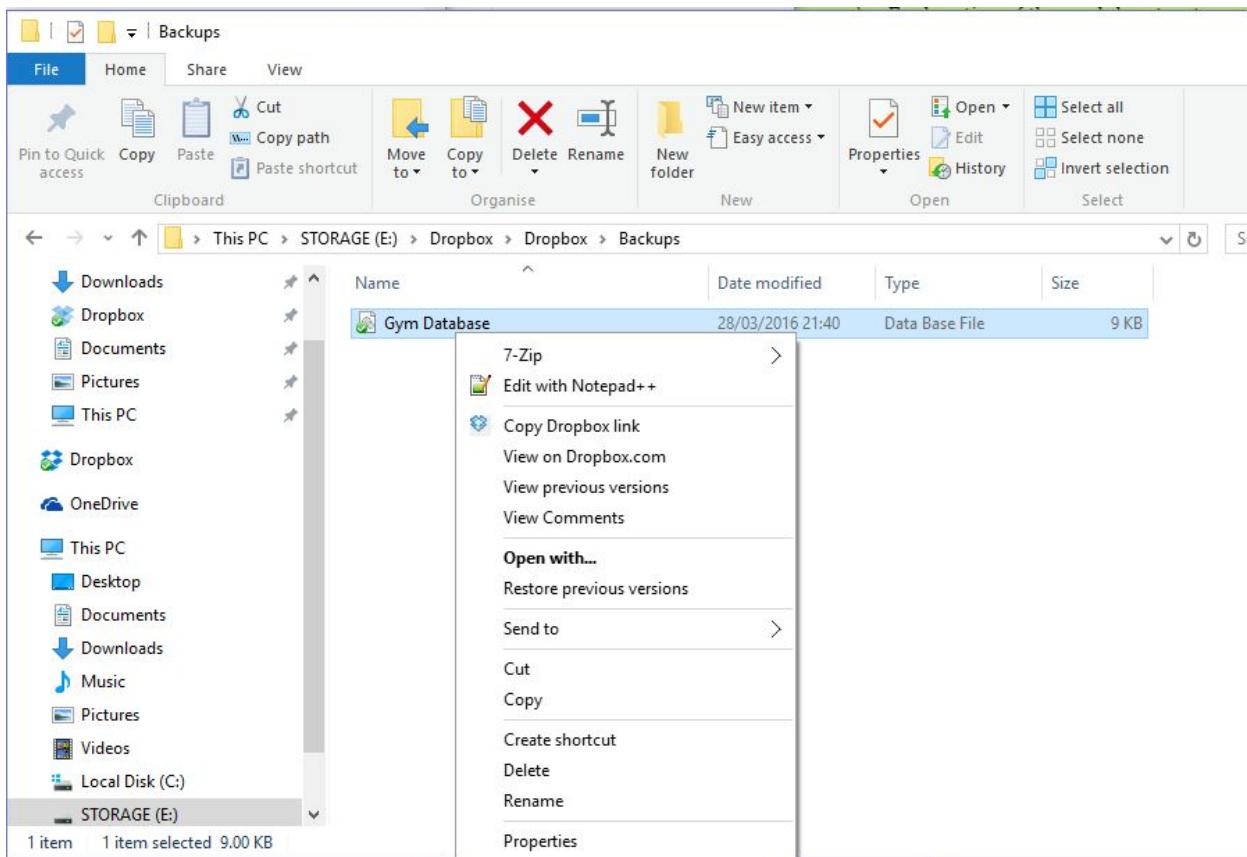
1. Close the system if it's already running



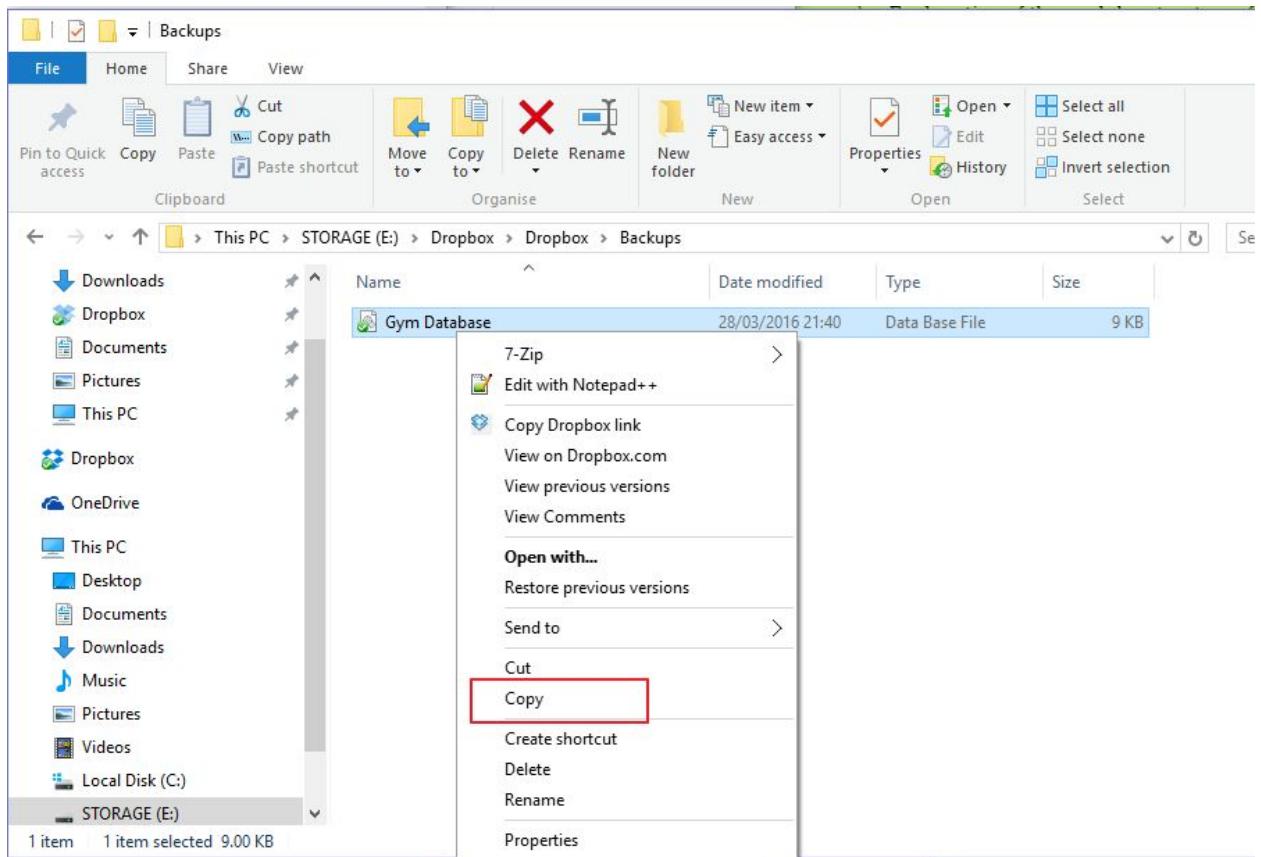
2. Navigate to the directory where the backup data is stored



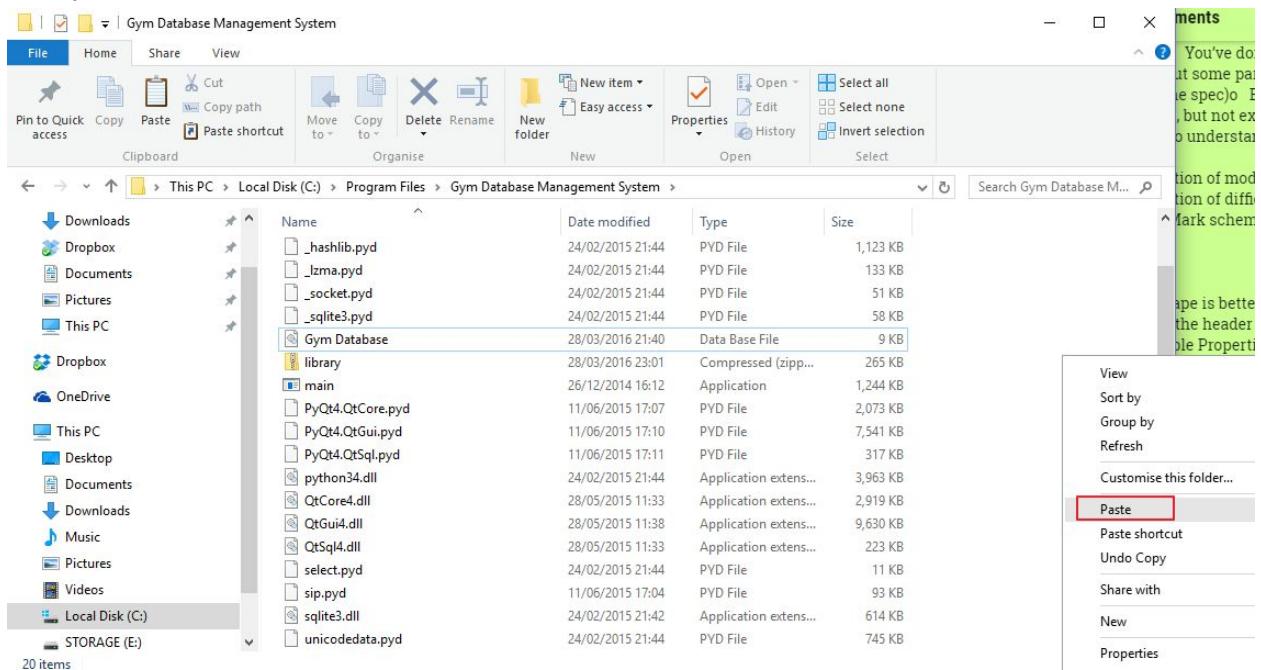
3. Find the backup database file and right click the file



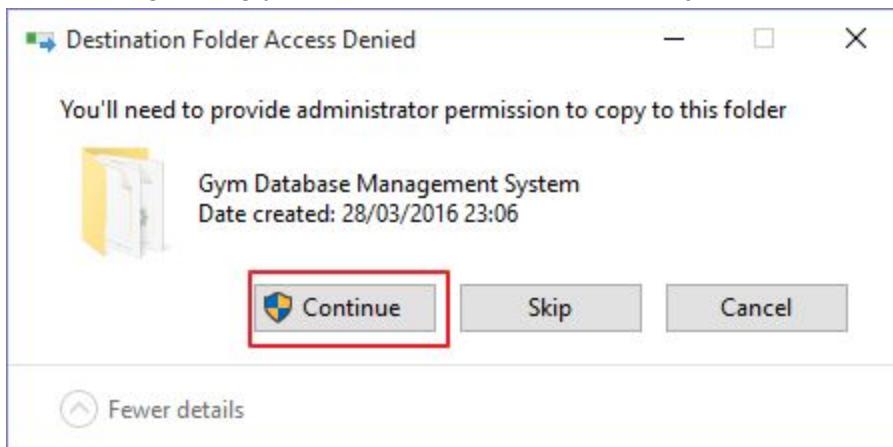
4. From the drop down menu select “Copy” by left clicking



5. Right click and Select “Paste” from the drop down menu by left clicking in the directory where the System Executable is stored.



6. If a dialog asking you for administrator permission just click on the “Continue” push button.



7. Your data is now restored!

Evaluation

Customer Requirements

In this section I will evaluate whether or not each of the objectives set in the specification has been fulfilled by my system to determine whether or not the system has met my clients requirements. If my system hasn't met a particular objective I will explain why this is the case. If the system has met an objective, evidence as to why will be provided.

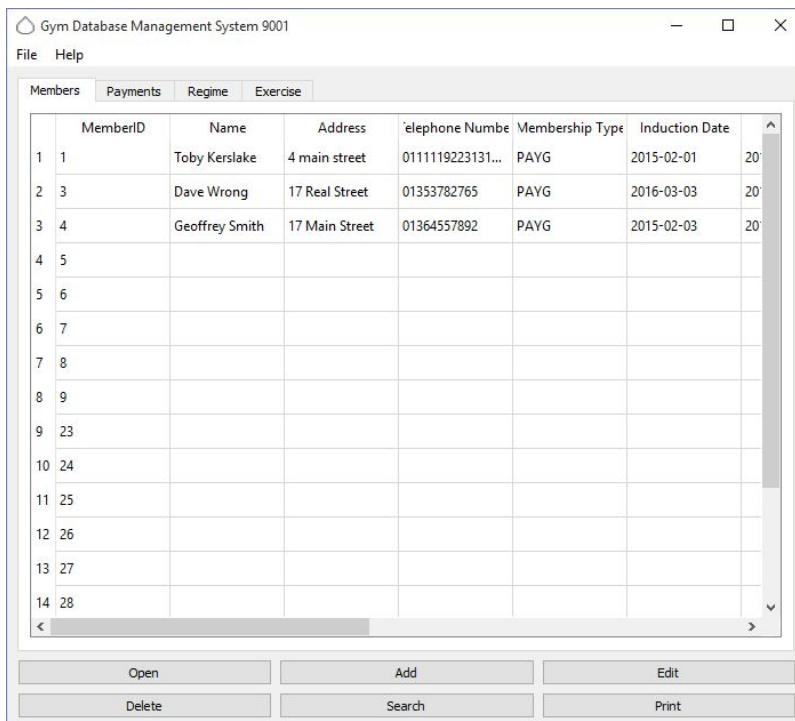
Simple Gui

Objective: Easy to use Simple Gui with clearly labelled buttons to choose an option.

Fulfilled?:

This objective has been fulfilled. I have achieved this by designing an interface that didn't have an intimidating amount of windows and used clearly labelled push buttons with plain english titles as opposed to technical jargon that can be easily accessible by users that have little prior knowledge of computer systems.

Evidence:



The screenshot shows a Windows application window titled "Gym Database Management System 9001". The menu bar includes "File" and "Help". Below the menu is a tab bar with "Members" (selected), "Payments", "Regime", and "Exercise". The main area is a grid table with columns: MemberID, Name, Address, Telephone Number, Membership Type, Induction Date, and a date field. The table contains 14 rows of data. At the bottom of the window are several buttons: Open, Add, Edit, Delete, Search, and Print.

	MemberID	Name	Address	Telephone Number	Membership Type	Induction Date	
1	1	Toby Kerslake	4 main street	011119223131...	PAYG	2015-02-01	20
2	3	Dave Wrong	17 Real Street	01353782765	PAYG	2016-03-03	20
3	4	Geoffrey Smith	17 Main Street	01364557892	PAYG	2015-02-03	20
4	5						
5	6						
6	7						
7	8						
8	9						
9	23						
10	24						
11	25						
12	26						
13	27						
14	28						

As you can see above the implementation contains no technical jargon and clearly labelled push buttons. My client agreed saying "The Interface was clearly laid out".

Gui Buttons

Objective: There should be buttons for opening a table, add to it, editing it, deleting an item, creating an invoice or physical record, and searching for something specific.

Fulfilled?: This objective has been fulfilled. I have achieved this by implemented 6 push buttons for opening a table, add to it, editing it, deleting an item, creating an invoice or physical record, and searching for something specific. These buttons have been clearly labelled with concise english terms and no technical language.

Evidence:

Open	Add	Edit
Delete	Search	Print

This has clearly been achieved by the push buttons presented above and this sentiment is agreed with by my clients saying “All of the buttons were labelled well in non_technical language”.

Dialog Windows

Objective: Each of these buttons should open up either drop down menus or entirely new windows, depending on which is more user friendly, and display further options within those commands.

Fulfilled?: This objective has been fulfilled. I have achieved this by creating 5 separate dialogs that are opened when the user click the add, edit, delete, search, or print push buttons. The Open push button goes straight to an inherited open dialog from the pyqt library. Each of these dialogs then present different input options for the user to enter.

Evidence: Each of the buttons opened up a new dialog presenting several interactable options for the user as presented by screenshots in the Testing and User Manual sections. My client also agreed saying “Upon clicking the buttons new windows would open”.

Input Interface

Objective: The interface for entering new information should be easy to use and clearly labelled for adding each attribute to the specified table.

Fulfilled?: This objective has been fulfilled. I have achieved this by creating input forms in the form of the Add and Edit dialogs that allow the user to easily enter appropriate information in a clearly labelled format.

Evidence: The easy to use interface has been presented in the testing and user manual sections and my client agreed that it is “well laid out and easy to enter information”.

Search Function

Objective: There should be an easy/efficient way to switch and search between all of the tables in the database.

Fulfilled?: This objective has been fulfilled. I have achieved this by implementing a search function that searches through a table selected by a user for any specific term they simply input into a line edit.

Evidence: The search function has been presented in the testing and user manual sections and my client agreed that “The search function works easily and has no lag”.

Security

Objective: Avoid more novice users accidentally deleting items by having a secondary password for higher level access.

Fulfilled?: This objective has been partially fulfilled. I have achieved this by implementing a secondary password dialog to be required when opening the delete dialog which requires a different password only known by higher level users. It could be improved if the user was able to reset and change the password in case of a security risk.

Evidence: The password and delete dialogs have been presented in the testing and user manual sections and my client agreed that it “works well but the system doesn’t allow me to change the password”.

Online Backups

Objective: Store backups of the database online using a system like dropbox or GitHub so that in the event of data corruption there would be recent version to backup to.

Fulfilled?: This objective has not been fulfilled. This is due to time constraints as well as the difficulty that came from trying to implement the python dropbox library.

Evidence: The Online back ups were not implemented to their is no evidence for this section

Local Backups

Objective: Storing the latest version of the spreadsheet locally also adds an extra layer

of security as the most recent version won't be easily accessible except by people with direct access to the workstation.

Fulfilled?: This objective has been partially fulfilled. This has been achieved by allowing the user to make local backups manually by simply copying and pasting the database files to a location they desire. This wasn't done automatically due to time constraints.

Evidence: There is no evidence for this objective since it was not implemented but the User Manual section details how to manually backup databases.

Reports

Objective: Easy to create reports containing all of a specific client's information in an organised manner on as little paper as legibly possible.

Fulfilled?: This has been partially fulfilled. This has been achieved by allowing the user to create a report on a member which is then easily printable via a printer by the user but the printed forms aren't presented in the best way as they are printed at a small font size.

Evidence: The report function has been presented in the testing and user manual sections but my client believes that it "works well but maybe prints to small".

Invoices

Objective: Easy to create invoices for members based off of the information stored in the tables.

Fulfilled?: This has been partially fulfilled. This has been achieved by allowing the user to create an invoice for a member which is then easily printable via a printer by the user but the printed forms aren't presented in the best way as they are printed at a small font size.

Evidence: The report function has been presented in the testing and user manual sections but my client believes that it "works well but maybe prints to small".

Database Table Printouts

Objective: Reports should include Invoices, database table printouts, member reports giving details selected from the databases.

Fulfilled?: This has been partially fulfilled. This has been achieved by allowing the user to create a report on a member, an invoice, or a regime table which is then easily printable via a printer

by the user but the printed forms aren't presented in the best way as they are printed at a small font size. It also doesn't have the option to print out entire tables due to time constraints.

Evidence: The report function has been presented in the testing and user manual sections but my client believes that it "works well but maybe prints to small".

Printable Reports

Objective: These reports should be printable so that they can be given to members and stored physically.

Fulfilled?: This has been partially fulfilled. This has been achieved by allowing the user to create a report on a member, an invoice, or a regime table which is then easily printable via a printer by the user but the printed forms aren't presented in the best way as they are printed at a small font size.

Evidence: The report function has been presented in the testing and user manual sections but my client believes that it "works well but maybe prints to small".

Digital Input Forms

Objective: Easy to fill out digital input forms for entering new data into the database tables instead of a more complicated sql query based system.

Fulfilled?: The objective has been fulfilled. This was achieved by creating the Add and Edit input forms in the AddEditDialogs. These forms have clear and concise labels and line edits for easy input by the user.

Evidence: The Add and Edit Dialogs have been presented in the testing and user manual sections and my client believes that they "work really well but it could be improved".

Printable Input Forms

Objective: Printable versions of these forms that can be created to be filled out by hand by less technically inclined clients that can then have their information manually entered into the system by a member of staff.

Fulfilled?: This objective was not fulfilled. This was due to time constraints and the fact that the feature was deemed slightly redundant as a much more presentable looking form can be designed by the user in a word processing package.

Evidence: This objective was not fulfilled therefore there is no evidence.

Summary

Overall I feel like my system has achieved most of my clients original specification, and has reasonable explanations and alternatives for the instances in which the system failed to fully or only partially met the requirements and this has been understood by my client, as presented in the evidence sections for each objective.

Effectiveness

Simple Gui

Objective: Easy to use Simple Gui with clearly labelled buttons to choose an option.

Evaluation Criteria:

- Simple Button layout that can be easily navigated
- Clear English Labels
- Easily comprehended and navigated database view

Judgement and Evidence:

The evidence for this is provided in the Testing, and User Manual sections. The screenshots presented in these sections clearly demonstrate that the gui has a simple button layout that can be easily navigated, clear english labels, and an easily comprehended and navigated database view.

Gui Buttons

Objective: There should be buttons for opening a table, add to it, editing it, deleting an item, creating an invoice or physical record, and searching for something specific.

Evaluation Criteria:

- Simple Button Layout that can be easily navigated
- Clear english labels

Judgement and Evidence:

The evidence for this is provided in the Testing, and User Manual sections. The screenshots presented in these sections clearly demonstrate that the buttons had a simple button layout that could be easily navigated and clear english labels.

Dialog Windows

Objective: Each of these buttons should open up either drop down menus or entirely new windows, depending on which is more user friendly, and display further options within those commands.

Evaluation Criteria:

- Easy to navigate new windows opened when buttons clicked
- Further options inside those windows

Judgement and Evidence:

The evidence for this is provided in the Testing, and User Manual sections. The screenshots presented in these sections clearly demonstrate that the dialog windows are easily navigated

and open when their respective buttons clicked and each of said windows contain further options.

Input Interface

Objective: The interface for entering new information should be easy to use and clearly labelled for adding each attribute to the specified table.

Evaluation Criteria:

- Clear, easy to use interface
- Clearly labelled

Judgement and Evidence:

The evidence for this is provided in the Testing, and User Manual sections. The screenshots presented in these sections clearly demonstrate that the input interface has a clear, easy to use interface, and is clearly labelled.

Search Function

Objective: There should be an easy/efficient way to switch and search between all of the tables in the database.

Evaluation Criteria:

- Easy way to search through the tables

Judgement and Evidence:

The evidence for this is provided in the Testing, and User Manual sections. The screenshots presented in these sections clearly demonstrate that the search function provides an easy way to search through tables.

Security

Objective: Avoid more novice users accidentally deleting items by having a secondary password for higher level access.

Evaluation Criteria:

- Have a secondary Password set for higher level access
- Make the password changeable in case of a security breach

Judgement and Evidence:

The evidence for this is provided in the Testing, and User Manual sections. The screenshots presented in these sections clearly demonstrate that a secondary password is required for higher level access but currently the password is not changeable by the user, only by the programmer.

Online Backups

Objective: Store backups of the database online using a system like dropbox or GitHub so that in the event of data corruption there would be recent version to backup to.

Evaluation Criteria:

- Store backups of databases online
- Stores them automatically

Judgement and Evidence:

There is no evidence for this since this objective was not implemented and thus fails to meet any of its Evaluation Criteria.

Local Backups

Objective: Storing the latest version of the spreadsheet locally also adds an extra layer of security as the most recent version won't be easily accessible except by people with direct access to the workstation.

Evaluation Criteria:

- Store local backups of databases offline
- Stores them automatically

Judgement and Evidence:

There is no evidence for this since this objective was not implemented and thus fails to meet any of its Evaluation Criteria.

Reports

Objective: Easy to create reports containing all of a specific client's information in an organised manner on as little paper as legibly possible.

Evaluation Criteria:

- Easy to create reports
- Presentable reports

Judgement and Evidence:

The evidence for this is provided in the Testing, and User Manual sections. The screenshots presented in these sections clearly demonstrate that the reports are easy to create but don't produce that presentable a final design.

Invoices

Objective: Easy to create invoices for members based off of the information stored in the tables.

Evaluation Criteria:

- Easy to create reports
- Presentable reports
-

Judgement and Evidence:

The evidence for this is provided in the Testing, and User Manual sections. The screenshots presented in these sections clearly demonstrate that the reports are easy to create but don't produce that presentable a final design.

Database Table Printouts

Objective: Reports should include Invoices, database table printouts, member reports giving details selected from the databases.

Evaluation Criteria:

- Easy to create reports
- Presentable reports

Judgement and Evidence:

The evidence for this is provided in the Testing, and User Manual sections. The screenshots presented in these sections clearly demonstrate that the reports are easy to create but don't produce that presentable a final design.

Printable Reports

Objective: These reports should be printable so that they can be given to members and stored physically.

Evaluation Criteria:

- Easy to create reports
- Presentable reports

Judgement and Evidence:

The evidence for this is provided in the Testing, and User Manual sections. The screenshots presented in these sections clearly demonstrate that the reports are easy to create but don't produce that presentable a final design.

Digital Input Forms

Objective: Easy to fill out digital input forms for entering new data into the database tables instead of a more complicated sql query based system.

Evaluation Criteria:

- Easy to fill out digital input forms

Judgement and Evidence:

The evidence for this is provided in the Testing, and User Manual sections. The screenshots presented in these sections clearly demonstrate that the digital input forms are easy to fill out.

Printable Input Forms

Objective: Printable versions of these forms that can be created to be filled out by hand by less technically inclined clients that can then have their information manually entered into the system by a member of staff

Evaluation Criteria:

- Presentable, easy to create printable input forms

Judgement and Evidence:

There is no evidence for this since this objective was not implemented and thus fails to meet any of its Evaluation Criteria.

Summary

In summary I feel that my system is fairly effective as it was able to meet the majority of the evaluation criteria and the sections that don't have acceptables reasons as to why whether it be redundancy or time constraints, and my client appears to agree, as presented in the End User Evidence Section.

Learnability

When I first consulted with my client about developing this system we discussed my clients prior knowledge and experience with computers so I could develop the system at a level that suited his needs (see Analysis Section). The client had a reasonable knowledge of how to operate a computer system but not to the point where he would be able to understand any sql or technical terms in regards to databases so I knew I would have to phrase everything with non-technical jargon substituting words like "INSERT" for "Add". I also knew I would have to have a user friendly interface that could be easily operated by him and other members of his staff as their computer knowledge may also vary. With this in mind I took the following steps when designing my system:

- I ensured the main interface (and all extra dialogs for that matter) was easily accessible to all users using large obvious buttons and a large detailed table view (as presented in the screenshot below), while still letting more experienced users use keyboard shortcuts. This was well received by my client, as demonstrated in the End User Evidence section of this Evaluation.

The screenshot shows a Windows application window titled "Gym Database Management System 9001". The menu bar includes "File" and "Help". Below the menu is a tab bar with "Members", "Payments", "Regime", and "Exercise", where "Members" is selected. The main area displays a table with columns: MemberID, Name, Address, Telephone Number, Membership Type, Induction Date, and Expire Date. The table contains 14 rows of data. At the bottom of the table are navigation arrows and a vertical scroll bar. Below the table are six buttons: Open, Add, Edit, Delete, Search, and Print.

	MemberID	Name	Address	Telephone Number	Membership Type	Induction Date	Expire Date
1	1	Toby Kerslake	4 main street	0111119223131...	PAYG	2015-02-01	2015-07-31
2	3	Dave Wrong	17 Real Street	01353782765	PAYG	2016-03-03	2016-08-31
3	4	Geoffrey Smith	17 Main Street	01364557892	PAYG	2015-02-03	2015-07-31
4	5						
5	6						
6	7						
7	8						
8	9						
9	23						
10	24						
11	25						
12	26						
13	27						
14	28						

- I ensured to use easy to understand English words, as opposed to technical database related jargon like "INSERT" and "UPDATE" which my client also approved of.
- I also ensured that the table view was similar to that of an Microsoft Excel document as that was what my client was using prior to this system

Overall I would argue that due to the design features that I implemented into my program and the positive feedback provided by my client, the system has a very friendly user interface that appears to be accessible to most people of varying levels of prior computer knowledge. However to someone who has virtually no experience using a computer system may have trouble as the range of options and amount of windows that can be open at one time could be considered daunting and intimidating as well as confusing.

Usability

In this section I will assess how easy the system I have developed is to use, with particular emphasis on the effectiveness and ease of use regarding the user interface. This will be achieved by measuring the system against a set of criteria listed below:

- Language Used
- Navigability
- Latency

Language Used

One of my main goals regarding usability was making sure the language I used was easily understandable by any user as I didn't want to confuse any users that had little experience interacting with computer systems and all the terminology that went along with that. I feel that I achieved this by labelling the main interfacing buttons terms like "Add" and "Edit" as opposed to "INSERT" and "UPDATE". This was acknowledged by and agreed upon by my client in the End User Evidence section of this Evaluation.

Navigability

Another of my goals in regards to usability was making sure that the interface is easy to navigate and quickly get from one section or function to another. Part of accomplishing this was with the Language Used as explained above. Another part of accomplishing this was implemented a small amount of buttons, organised neatly, that would supply access to all or most of the main functions of the system, that aren't intimidating or confusing the the user. I feel I achieved that with the 6 main function buttons in my main interface and my client acknowledged this and agreed.

Latency

Another important feature for this program to have high usability was latency. I didn't want the program to have a large amount of lag when being operated and I wanted the push buttons and all other interactive elements of the system to react almost instantly without a discernible delay. I believe I achieved this with efficient programming and it appears to be the case when actually in use. Although my client never really pointed out the low latency, I feel that that means I've accomplished my goal as if it had a high latency he definitely would have pointed it out. These results were found using test data and so the program hasn't actually been tested using a database with more than 1000 entries, which is most likely how it will be used, so these claims may not be reliable.

Summary

In summary, I feel that my program has achieved a great level of usability. I feel that the strengths of the system are definitely in its graphical user interface as it appears to be easy to navigate and uses appropriate language that's easily read by most individuals. I feel the programs low latency is also one of its strengths however, the system hasn't actually been stressed tested with a large amount of data on a low spec system, though with the system my clients using it should be handled incredibly well.

Maintainability

Fixing Bugs

It should prove to be relatively easy to fix any bugs in my system. Although I only found one bug in my system through testing, if any later materialise I should be able to fix it easily for the following reasons:

- The code is largely self documented and contains clear and concise comments and therefore whoever is maintaining the code in the future to easily identify and fix any problems that may arise.
- Even more documentation explaining the more difficult to understand code is in the System Maintenance section along with explanations of all the classes along with class diagrams and variable listings.
- The one bug my client found by using the program was an error that stops the main functions from working due to a database loading error but could easily be fix with a try except exception handling and can easily be fix by the user by simply reloading the database, as demonstrated in the errors section in the User Manual.

Changing Parameters

The parameters of this system may need to be changed as the user may have new requirements for his database needs and need new columns inserted into a table or a new table entirely. If these changes need to be made it should be fairly straightforward for the following reasons:

- The system almost entirely uses local variables and parameters in all its functions and methods so only the parameters need change instead of large code sections
- The code is well commented to the user can easily understand all of the functions and variables uses enabling them to easily edit them
- There is a large amount of documentation and literature provided in the System Maintenance section to further explain each variables, parameters, attributes and methods purpose, making it simple to alter the system as required.

Responding to New Requirements

The system may or may not be easy to adapt to new requirements depending on the nature of the requirements. For example if the user needed a new interface for managing the data this could be done with relative ease since the classes for the gui are separate from the functions for the actual data manipulation but If the user wanted a new table added to their database then a large number of functions and classes would have to be dramatically altered. A summary of the system's ease to respond to new requirements is detailed below:

- The software is 100% modular. Most of the functions and gui's are separate entities but most of the gui's layouts are dependant on the nature of the functions. For example if a new column is added to the add or edit function the add or edit gui would be missing an input line edit for the new column. Although this is the case the functions aren't dependant on the GUI so if the user just wants to make the program more efficient he should easily be able to do so by editing just the functions.
- The code is largely self documented with clear and concise comments and variable names that are direct and appropriate making the code easy to read which will help identify the areas which need altering by the programmer (see System Maintenance section).

- The system mostly uses local variables meaning that only parameters need to be edited for a function instead of several global variables in many places around the program.
- The code is largely documented in the System Maintenance section explaining all of the difficult to understand pieces of code and classes so the programmer can easily identify how to deal with a certain piece of code,
- I have not identified which variables are and are not private meaning any future programmers will have to read through the code to identify whether a variable can be assigned outside of the initial class, and may have to just work at their own accord. This is because the system was developed in python and although common syntax dictates that private variables should be preceded with an underscore, there is no actual way to declare a variable as public or private since public and private variables don't exist in python.

Summary of Maintainability

Overall, I would argue that my system is fairly maintainable. This is due to the fact that my code is well and consistently commented as well as having a large amount of documentation in the System Maintenance section, meaning it should be fairly easy for any programmer to understand all sections of the code and amend them how they please. This should allow for easy debugging and fixing of bugs and the changing of parameters. Although there are some problems with the variables not stating whether their public or private, and some of the classes relying on other code sections, the abundance of documentation should help any programmer to circumvent these issues.

Suggestions for Improvement

My client highlighted several shortcomings and possible improvements that could be implemented in any future versions of the system, all of which are listed and explained in this section.

Auto-Updating

Although I discussed my time constraints and how some things I felt were less essential to the functionality to the program were not implemented he made it clear that having to re-open the database after every time he makes any form of change to the database and feels like this would be an easy thing to implement in future versions which I agree with. He made this clear in an off-record discussion we had while he was testing the program.

Online Backups

One of my main objectives were to have automatic appropriate online backups of any database files involved with the system using a service like dropbox or github but this wasn't implemented due to 2 reasons. First of all python modules for automatic dropbox and github integration with python files were particularly difficult and tedious to integrate, which could have impacted the efficiency of the program. Secondly, time constraints meant I had to avert my attention to more important/essential features, especially since the files can be easily backed up manually, as detailed in the User Manual.

Improved Printable Forms

Although my program is capable of printing a variety of forms my client feels as though my program could be improved by making the format of the forms more appropriate. He feels that the text of the forms is way to small and although legible, is a waste of paper and not very presentable if he wants to give a form to a client. He also wishes that the program was capable of printing input forms that his users can fill out to hand in to him at a later date to add to the database.

End User Evidence Appendix

Questionnaire

For each question I asked my client to provide a response by writing a number from 0 to 3 to represent how the effectiveness of my solution to an objective.

The key is shown below:

- 0 - System has completely failed to meet the objective
- 1 - System has only partially met the objective
- 2 - The system has met the objective but the solution is not effective or is difficult to use
- 3 - The system has completely fulfilled the objective

My client also provided a comment for each question if he felt it necessary to provide reasoning for his score which is included in the transcript below

1. To what extent would you agree that the system has an easy to use simple GUI that has clearly labelled buttons to choose an option?

Score: 3

Comment: The interface was clearly laid out giving me multiple user friendly ways to interact with the system. Though they were overwhelming at first I soon got the hang of it.

2. To what extent would you agree that the system has appropriate buttons for opening a database, adding to it, editing it, deleting an item, creating any printable forms, and searching for something specific?

Score: 3

Comment: All of the buttons were labelled well in non_technical language that was easy to understand by me and my member of staff

3. To what extent would you agree that the system opens appropriate user friendly windows upon these button being clicked displaying more options for the user?

Score: 3

Comment: Upon clicking the buttons new windows would open allowing me to interact with the main functions of the program

4. To what extent would you agree that the interface for entering new information is easy to use and clearly labelled for adding each attribute to a specific table?

Score: 3

Comment: It was well laid out and easy to enter information as long as I consulted the user manual for the correct data types

5. To what extent would you agree that the system has an easy/efficient way to switch and search between all the tables in the database?

Score: 2

Comment: The search function works easily and has no lag but if I want to search every table I have to search each one separately

6. To what extent would you agree that the system helps avoid more novice users accidentally deleting items by having a secondary password for higher level access?

Score: 2

Comment: It works well but the system doesn't allow me to change the password so if the passwords discovered my security is compromised

7. To what extent would you agree that the system stores appropriate and reliable backups of the database online using a system like dropbox in the event of data corruption?

Score: 0

Comment: This wasn't implemented due to supposed time restraints but the system has presented other backup options

8. To what extent would you agree that the system stores a reliable backup locally to the users workstation on which the systems installed?

Score: 1

Comment: The system itself doesn't create the backups but the user manual details an easy way to do it manually

9. To what extent would you agree that the system allows you to easily create reports containing details from the database on as little paper as legibly possible?

Score: 2

Comment: The print function works well but maybe prints to small which is still legible but wastes paper

10. To what extent would you agree that the system uses easy to fill out digital input forms?

Score: 3

Comment: The input forms to add and edit data work really well but it could be improved by allowing me to select the data to edit instead of remembering the primary key

11. To what extent would you agree that the system allows for appropriately formatted printable versions of these printable forms to allow your clients to fill out for a member of staff to later input into the system?

Score: 0

Comment: This wasn't implemented but is something I could easily create myself in word.

Questionnaire

1. To what extent would you agree that the system has an easy to use simple GUI that has clearly labelled buttons to choose an option? **3**

The interface was clearly laid out with multiple user friendly ways to interact with the system. Though there were over whelming at first I soon got the hang of it.

2. To what extent would you agree that the system has appropriate buttons for opening a database, adding to it, editing it, deleting an item, creating any printable forms, and searching for something specific? **3**

All of the buttons were labelled well in non-technical language that was easy to understand by me and my friends of staff.

3. To what extent would you agree that the system opens appropriate user friendly windows upon these button being clicked displaying more options for the user? **3**

Upon clicking the buttons new windows would open allowing me to interact with the main functions of the program.

4. To what extent would you agree that the interface for entering new information is easy to use and clearly labelled for adding each attribute to a specific table? **3**

It was well laid out an easy to enter information as long as I consulted the user manual for the correct data types.

5. To what extent would you agree that the system has an easy/efficient way to switch and search between all the tables in the database? **2**

The search function works easily and has no lag but if I want to search every table, I have to search each one separately.

6. To what extent would you agree that the system helps avoid more novice users accidentally deleting items by having a secondary password for higher level access? **2**

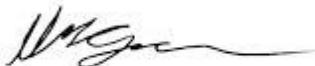
It works well but the system doesn't allow me to change the password so if that discovered my security is compromised.

7. To what extent would you agree that the system stores appropriate and reliable backups of the database online using a system like dropbox in the event of data corruption? **0**

This wasn't implemented due to supposed time restraints but the system has presented other backup options.

8. To what extent would you agree that the system stores a reliable backup locally to the users workstation on which the systems installed? 1
The system doesn't create the backups, but the user manual details an easy way to do it manually.
9. To what extent would you agree that the system allows you to easily create reports containing details from the database on as little paper as legibly possible? 2
The print function works well but wastes prints to small which is still legible but wastes paper.
10. To what extent would you agree that the system uses easy to fill out digital input forms? 3
The input forms really work well but it could be improved by allowing us to select the data to edit instead of rekeying the information.
11. To what extent would you agree that the system allows for appropriately formatted printable versions of these printable forms to allow your clients to fill out for a member of staff to later input into the system? 0
This wasn't implemented but is something I could easily rekey myself in Word.

Client signature



Neill Green

It's worth noting that my client did give me other feedback verbally as we had frequent contact as he's my brother in law, meaning I got other feedback that has no literature of any kind meaning it's not listed here.

Graph

The below graph is a graphical representation of the scores my client gave to each of my objectives in the questionnaire section above

