

<b>Version</b>	<b>Last Modification was done on</b>	<b>Modification done by</b>	<b>Brief description about document modification</b>
1.1	17 <sup>th</sup> April 2022	Veerabhadraswamy.NG	Document first created. Included basic content and python script
1.2	18 <sup>th</sup> April 2022	Parth Parmar	Front-End GUI related python script using Tkinter included
1.3	19 <sup>th</sup> April 2022	Fernando Rushen	Bank-End data included and final correction done

Unit Title:	B9CY100 Advanced Programming Techniques (B9CY100_2122_TMD2)
<b>Unit Code:</b>	<b>B9CY100</b>
Unit Lecturer:	Paul Laird
Level:	9
Assessment Title:	CA_TWO
Assessment Type:	Group Report
Restrictions on Time/Length:	N/A
Individual/Group:	Group
Assessment Weighting:	70%
Submission Date:	22 <sup>nd</sup> April 2022
Mode of Submission:	On-line via Moodle
<b>Individual Members</b>	Fernando Rushen Parth Parmar Veerabhadraswamy.NG

Document Version Control.....	1
Document Contents Page.....	2
Introduction.....	3
Purpose.....	3
Scope.....	3
Problem Scenario.....	3
Programming tools and public git selections.....	3
<b>Github Repository link.....</b>	<b>3</b>
<b>Contribution.....</b>	<b>3</b>
Selection of public GitHub for the front-end and back-end operations.....	4
Programming environment setups.....	4
Step-1: Create a background and the title of the user interface.....	5
Step-2 Create a left panel in the user interface. ....	6
Step-3 Add the right panel in the user interface.....	7
Step-4 Add the bottom panel buttons to the user interface. ....	9
Back-end database process.....	10
Critical analysis .....	16
Application and tools required. ....	16
References.....	16

**Purpose:**

Primary intent of the project is to study and understand how to solve the real-time issue by using programming skills and related libraries. In this project work, we are working on the issue to decode the data management systems in the warehouse and the detailed information collection software is built to solve the problem. This is done by utilizing the python programming language with the graphic user interface.

**Scope:**

In the project work, we have built the graphical user interface which can load the dataset, store the dataset, update and remove the dataset. For the development of the user interface, we have utilized python programming tool Tkinter and for the database operation we have used the sqlite3 database. For the database operation, we are using the SQLite3 open-source tools. The SQL query-based method is widely utilised in the field of database management, data science, machine learning and many other purposes. It is also easy to set up on any device with the Mac, Win, or Linux operating system.

**Problem Scenario:**

The Warehouse inventory is difficult to handle by most of the large production houses and many retail businesses for analyzing the available products. Also, sometimes the poor management of products in the Warehouse inventory can lead to loss, due to delay in the company's operations, and liabilities of the warehouses keeps on increasing.

To solve this issue with the warehouses inventory management system, we have developed a database that contains all the information about the products and the available quantities of the product information stored in the database

**Programming tools and public git selections**

In this project's programming works, we divided our programming task into two main parts:

- The first part included the information on the front-end user part, which involved the process of creating the graphical user interface and how it is visualised to the user end. We developed that part with the various multiple functions that are elaborated thoroughly in the report's further detail.
- The other part included the back-end operation of the programme software. This involved the functions such as the dataset storing and calling part of the programme, dataset entry, upgrade and delete operations in the back-end database operation.

**Github Repository link:** <https://github.com/Veera1987/Group-E/>

**Contribution:**

- In the "database\_project.py",
- Line 1 to 139 is configured by Veerabhadraswamy.NG
- Line 140 to 295 is configured by Parth Parmar
- Line 296 to 443 is configured by Fernando Rushen

### Selection of public GitHub for the front-end and back-end operations.

For the front-end operation of making process of the graphical user interface, we have used Python-based public GitHub library Tkinter. The link to this public git is attached in the reference part of the project work. The Tkinter is one of the well-known libraries in python. This tool of GUI is entirely open-source, and it is easy to understand, unlike other libraries in which the API document is necessary to read. Also, these tools have been utilised for the past several years, so it is available with very stable solutions. The developer made this API and library's syntax simple and easy to understand other than the coding background. It also provided the packages and templates based on the user interface grids to easily add the various things to our user interface based on the grid coordinates.

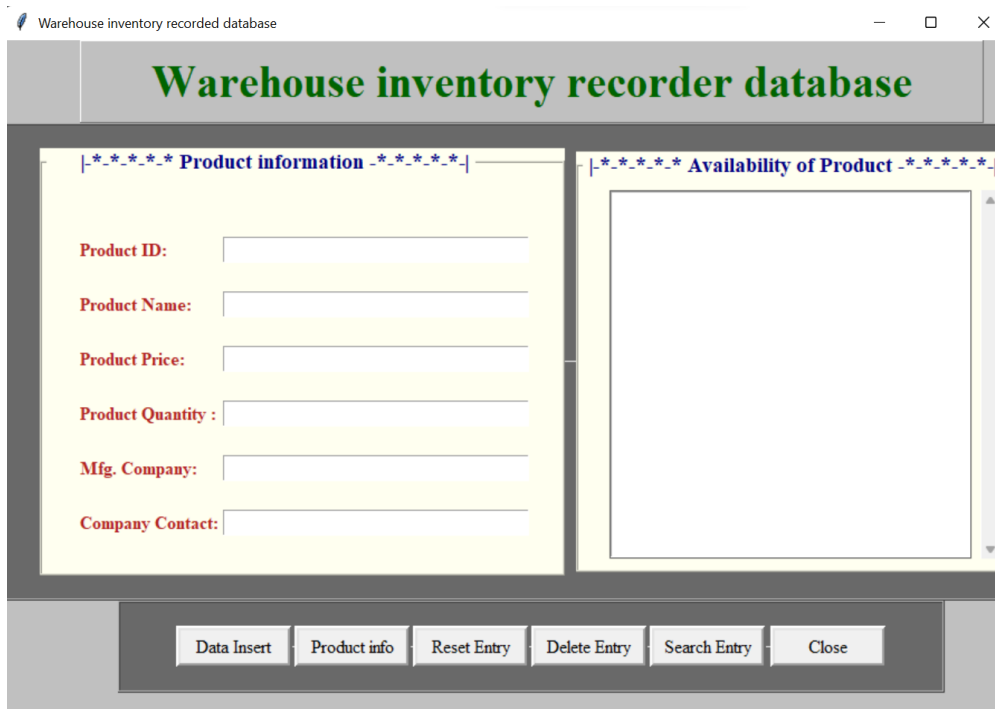
For the back-end operation of the databased is operated using the public git named sqlite3. This provided the support for the database operation across the various platforms. This database can successfully run on the system hard drives as a desktop-based application and support cloud operations. This database can store information about the financial report, media-based information, and much other categorical information stored in the database. This database has better performance accuracy for the operations such as reading and writing operations as per the SQL info operation is around 35% faster than the local file systems.

In the other section of the reported work, we elaborate on the programming work and the explanation of the user interface.

### Programming environment setups

The programming part is split into two main parts,

1. Front-end graphical user interface programming in python by using Tkinter.



The screenshot shows a Tkinter window titled "Warehouse inventory recorder database". The window has a title bar with standard minimize, maximize, and close buttons. The main content area is divided into two sections. The left section, titled "Product information", contains six input fields with labels: "Product ID:", "Product Name:", "Product Price:", "Product Quantity:", "Mfg. Company:", and "Company Contact:". The right section, titled "Availability of Product", contains a large empty text area. At the bottom of the window, there is a row of six buttons: "Data Insert", "Product info", "Reset Entry", "Delete Entry", "Search Entry", and "Close".

In the Python programming language, we first create a class named `product_ui`, which defines the operational function `__init__`, which has multiple operations functions of a button and storing the database.

```
#####
#Make a class for the user interface module
class product_ui:

    #Step 1: Create a basic root path and the background of the UI
    def __init__(self,root):

        #Create an objecte refrace of database class
        dat = op_database()
        dat.connection()

        self.root = root
        self.root.title(' Warehouse inventory recorded database ')#Name of product
        self.root.geometry("900x690")#Dimention of the UI
        self.root.config(bg='silver')#Background color of UI

        MainFrame =Frame(self.root,bg='silver')
        MainFrame.grid()

        #Step2.1 Create a frame with header
        HeadFrame = Frame(MainFrame, bd=1, padx=50,
                           pady=10, bg='silver', relief=RIDGE)
        HeadFrame.pack(side=TOP)

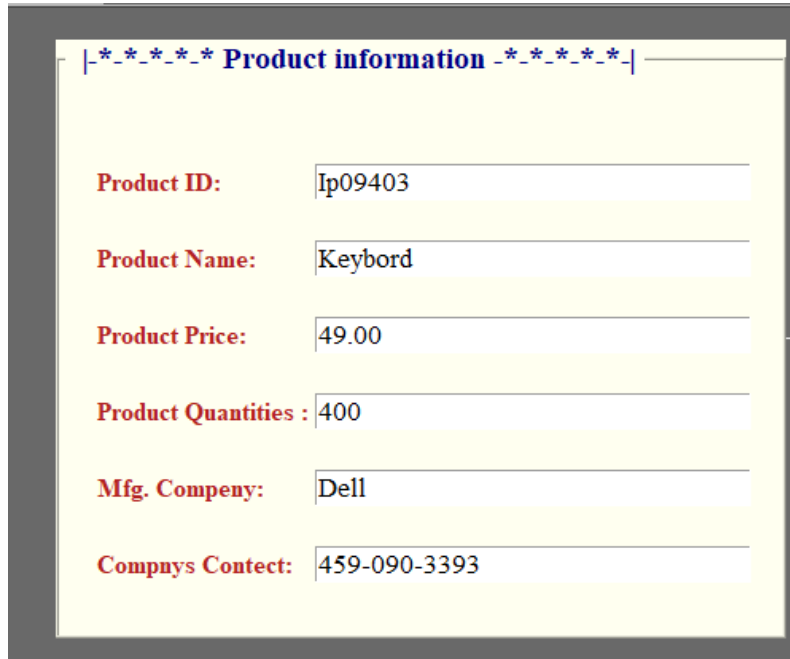
        #Step2.2 Add the title of the UI
        self.Title = Label(HeadFrame, font=('Times New Roman', 30,'bold'),fg='darkgreen',
                           text=' Warehouse inventory recorder database ',
                           bg='silver')
        self.Title.grid()

        #Step 2.3 Add the Operation body frame into the UI
        '''Setup the UI body of operation frame
        dimentions borders and padding buffers'''
        OperatingFrame = Frame(MainFrame, bd=2, width=1100, height = 50,
                               padx=50, pady=20, bg = 'dimgray', relief=RIDGE)
        OperatingFrame.pack(side=BOTTOM)
#####
```

Figure 2 Programme for the user interface background colours.

As shown in Figure 2, we added the codes and information to build the background of the user interface. We selected the background silver colour for the interface; then, we decided on the dimension of the interface, which is 900x690. Once all that information is set, the next step is to set the title of the user interface. For that, we selected the fonts of the title, which is times new roman style, with the bold and high of the fonts being 40 with the chosen colour as a dark green.

2. Create a left panel in the user interface.



The screenshot shows a Tkinter window titled "Product information" with a yellow background. It contains six labeled input fields arranged vertically:

- Product ID:** Ip09403
- Product Name:** Keyboard
- Product Price:** 49.00
- Product Quantities :** 400
- Mfg. Compenny:** Dell
- Compnys Contact:** 459-090-3393

*Figure 3 Left product information panel to load and search the dataset.*

To create this left panel, we utilised the Tkinter grid and windage template from GitHub for the creation. In the left panel's design, we want it to seem like we are given the product attributes and variables information. It must be stored in the particular variable called in the back-end operation to keep the information in the database. So for that, first of we needed to assign variables in the program:

```
##Create a variable to store the product information befor adding labels
product_id = StringVar()
product_name = StringVar()
product_Qty = StringVar()
product_price = StringVar()
product_company = StringVar()
product_contect = StringVar()
```

*Figure 4 Assign the product information storage variables.*

Now we are adding the left body panel into the user interface, and then setting such as product information panels, and the other background part is formed, which can be seen in Figure 5

```
#Add the left pannel
LeftSideFrame=LabelFrame(UiBodyFrame, bd=2, width=600,
                           height = 400, padx=20, pady=10, bg='ivory', fg='navy',
                           relief=RIDGE, font=('Times New Roman',14, 'bold'),
                           text = ' |-*--*- Product information -*-|-| ')
LeftSideFrame.pack(side= LEFT)
```

Once the essential information is added now, we are adding the left-most part with the attributes needed to load the information like the product name ID and all those information with the proper black space widgets in which the user needs to fill the data to load the dataset into a database or fill key to search about the product. As shown in Figure 3, all the information is added on the right side, which is then loaded into the product information attributes of Figure 4. All those information is then added to the programme by the following syntax:

```
#Add a product id variables label
self.labelp_id= Label(LeftSideFrame,
                       font=('Times New Roman',12, 'bold'),
                       text = 'Product ID:', padx = 2,pady=2,
                       bg = 'ivory', fg = 'firebrick')
self.labelp_id.grid(row = 2, column = 0, stick =W)

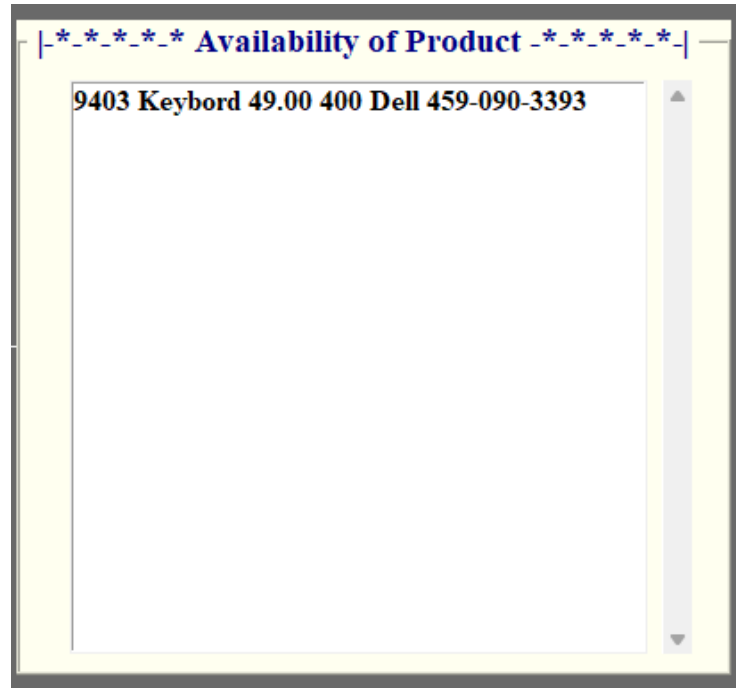
self.txtp_id = Entry(LeftSideFrame,
                     font=('Times New Roman',13),
                     textvariable = product_id, width= 30,)
self.txtp_id.grid(row=2, column = 1, sticky = W)
```

*Figure 6 Product variable loader syntax*

As shown in Figure 6, the first step of the label is utilised for the left-most title displayed in Figure 3 with red captions and the Entry command to load the information into the variables in Figure 4. Now all the part of the left panel is set next step is for the right panel part.

### 3. Add the right panel in the user interface.

Once the data points are added to the database, it's up to the right panel to display the information. As shown in Figure3, we added one product information, and it also shows the information about that product in the correct forum.



*Figure 7 Rightmost panel for the available displaying products*

We added the following syntax to construct the user interface's rightmost panel.

```
#Add the right pannel
RightSideFrame=LabelFrame(UiBodyFrame, bd=2, width=500,
                           height = 400, padx=20, pady=10, bg='ivory',fg='navy',
                           relief=RIDGE, font=('Times New Roman',14, 'bold'),
                           text = ' |-*--*- Availability of Product -*-|-| ')
RightSideFrame.pack(side=RIGHT)
```

*Figure 8 Syntax to add the background information of the rightmost panel*

In the right-side part, we need to add the blank display which shows the available products from the inventory; There are multiple products in the warehouse inventory. To add all the products to the collection, we add a scroll function to the program with the number of the item list, which shows the number of items available with the SQL query.



```
#Step 3.4 add a scroll bar in the UI
scroll_bar = Scrollbar(RightSideFrame)
scroll_bar.grid(row = 0, column = 1, sticky = 'ns')

iteamList = Listbox(RightSideFrame, width = 40, height = 16,
                    font = ('Times New Roman', 12, 'bold'),
                    yscrollcommand = scroll_bar.set)

#Above created product list from init module
iteamList.bind('<<ListboxSelect>>', product_reco)
iteamList.grid(row = 0, column = 0, padx = 8)
scroll_bar.config(command = iteamList.yview)
```

*Figure 9 Right panel command and syntax*

#### 4. Add the bottom panel buttons to the user interface.

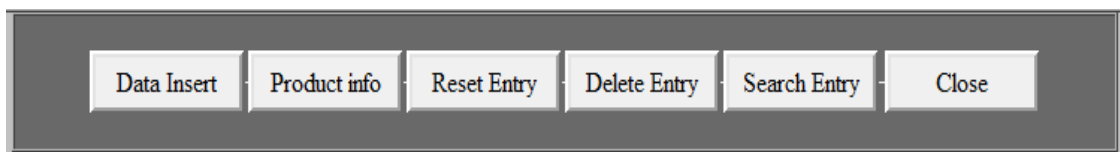
Before adding a button to the user interface, we needed to add the buttons section location to the user interface. We selected the bottom part of the user interface for the button operation, assigned by the following syntax.

```
#Step 2.3 Add the Operation body frame into the UI
'''Setup the UI body of operation frame
dimentions borders and padding buffers'''
OperatingFrame = Frame(MainFrame, bd = 2, width = 1100, height = 50,
                        padx = 50, pady = 20, bg = 'dimgray', relief = RIDGE)
OperatingFrame.pack(side = BOTTOM)
```

*Figure 10 Adding the operating button window into the programme.*

As shown in Figure 10, we add the operation frame into the user interface. Now next part is to count the number of operational buttons on the user interface.

As shown in Figure 1, we needed to add around six buttons into the interface to operate the database operation on the front and collect all the front-end command that passes through the back-end SQLite3.



*Figure 11 Operation window with operational buttons to load and upgrade the database.*

The function of the Data insert button is to load the newer dataset entry into the program. This button is connected with the back-end operation to load the dataset into the SQLite3 database. So, by adding the information with this button and the syntax for the button operation, Figure 12.

```
#Insert button
self.buttonInsert = Button(OperatingFrame, text = 'Data Insert',
                           font= ('Times New Roman',12),
                           height = 1, width= '10', bd = 4,
                           command=save_data)
self.buttonInsert.grid(row = 0, column = 0)

#Product info button
self.buttoninfo = Button(OperatingFrame, text = 'Product Info',
                         font= ('Times New Roman',12),
                         height = 1, width= '10', bd = 4,
                         command=ShowProductList)
self.buttoninfo.grid(row = 0, column = 2)
```

*Figure 12 The button syntax and operation command*

As shown in Figure 12, we added the function of the buttons with the buttons' aesthetic appearance and the button's location on the lower part of the user interface. Similarly, all the button functions and commands are added to the programme. Once all the button functions are prepared, we are almost ready with the user interface, as shown in Figure 1.

Now all the part of the front-end programme is prepared, and we are heading towards the back-end programming part, which includes the database process with the SQLite3.

### Back-end database process

In the back-end operations, we are utilising the public git of SQLite3 with the python programming language. This SQLite3 is capable of operating with any programming language, and it stores the data set dynamically with the autosave and automatic update of the database.

```
#####
#Back-end database related buttons operation class
'''In this class will store all the function
and the back-end operation of every button
and store those information into the SQL database'''

class op_database:
    '''Connection with the database'''
    def connection(self):
        print('Database: Connection needed')
        #Store the database into new variable
        conn = sqlite3.connect('inventory.db')
        #Feature and backend function of the cursor is added
        curs = conn.cursor()
        query = "create table if not exists product (pid integer primary key, \
pname text, price text, qty text, company text, contact text)"
        curs.execute(query)
        conn.commit()
        conn.close()
        print('Database : Connection process is completed\n')
```

*Figure 13 Back-end process class for the database operation*

As shown in Figure 13, we create the back-end database operation class. In the programme, the front-end back-end is two different processes, so first of we needed to connect these processes so the back-end database could adapt to the changes by the front end. To join the front-end and back-end processes, we create the connection function, which can provide the connection between the user interface. During the connection function operation first, it makes a database file, as shown in Figure 13 arrowed. Then select the SQL cursors in the programme. Once all the information is stored, we pass the SQL query to load the primary dataset informational. This database query ensures that every product ID is uniquely similar to the real-world product id that needs to be unique. So, to maintain the uniqueness of the product ID, we utilised the SQL function of the primary key, which kept the essence of the available products in the warehouse and avoided the duplication of the same product ID.

We create the database function on the following processes once the connection between the front-end GUI and the back-end database.

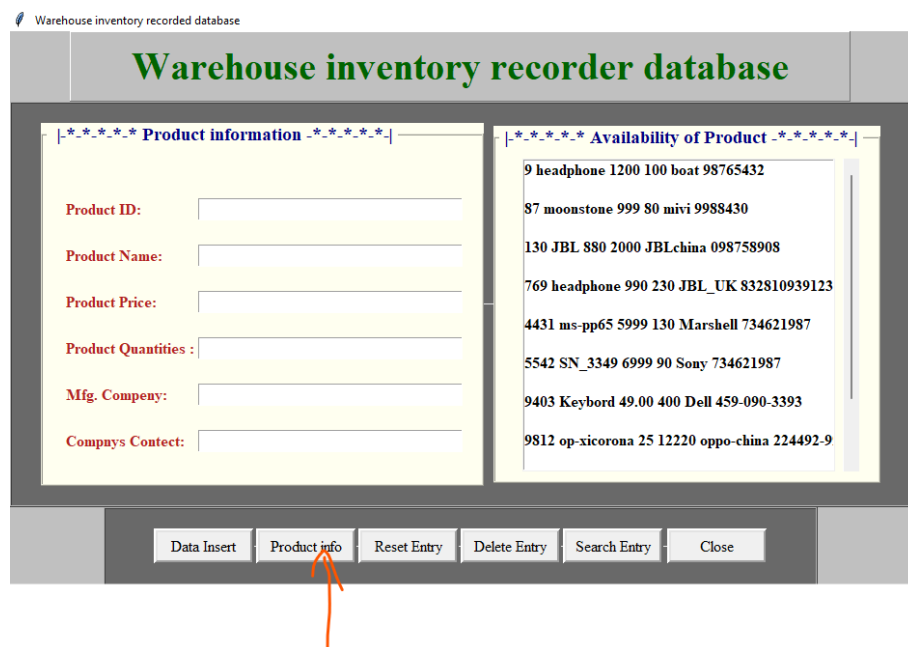
- Data inserting function: The six critical pieces of information about the product are inserted into the database. That six pieces of information are product ID, name, price, quantities, company name, and the contact number, which are stored in the

database to store all this critical information. SQL query asks the value of 6 variables by questioning.

```
#insert operation of the vrious datapoints
def insert(self,pid,name,price,qty,compny,contact):
    print("Database: Insert operation processed")
    conn = sqlite3.connect('inventory.db')
    #Feature and backend function of the curser is added
    curs = conn.cursor()
    query = "insert into product values(?, ?, ?, ?, ?, ?)"
    curs.execute(query, (pid, name, price, qty, compny, contact))
    conn.commit()
    conn.close()
    print("Database: Loading process is completed\n")
```

Figure 14 Database insert function back-end operation

- Data display or calling function: this function includes the process to load and display the available dataset to the GUI right panel display. This function is connected with the button of the product info. When that button is pressed, the button's process is called on. This function of the backend process is in operation, and it calls back all the product data entries from the SQL database.



Warehouse inventory recorder database

### Warehouse inventory recorder database

Product ID:

Product Name:

Product Price:

Product Quantities :

Mfg. Company:

Compny's Contact:

9 headphone 1200 100 boat 98765432

87 moonstone 999 80 mivi 9988430

130 JBL 880 2000 JBLchina 098758908

769 headphone 990 230 JBL\_UK 832810939123

4431 ms-pp65 5999 130 Marshall 734621987

5542 SN\_3349 6999 90 Sony 734621987

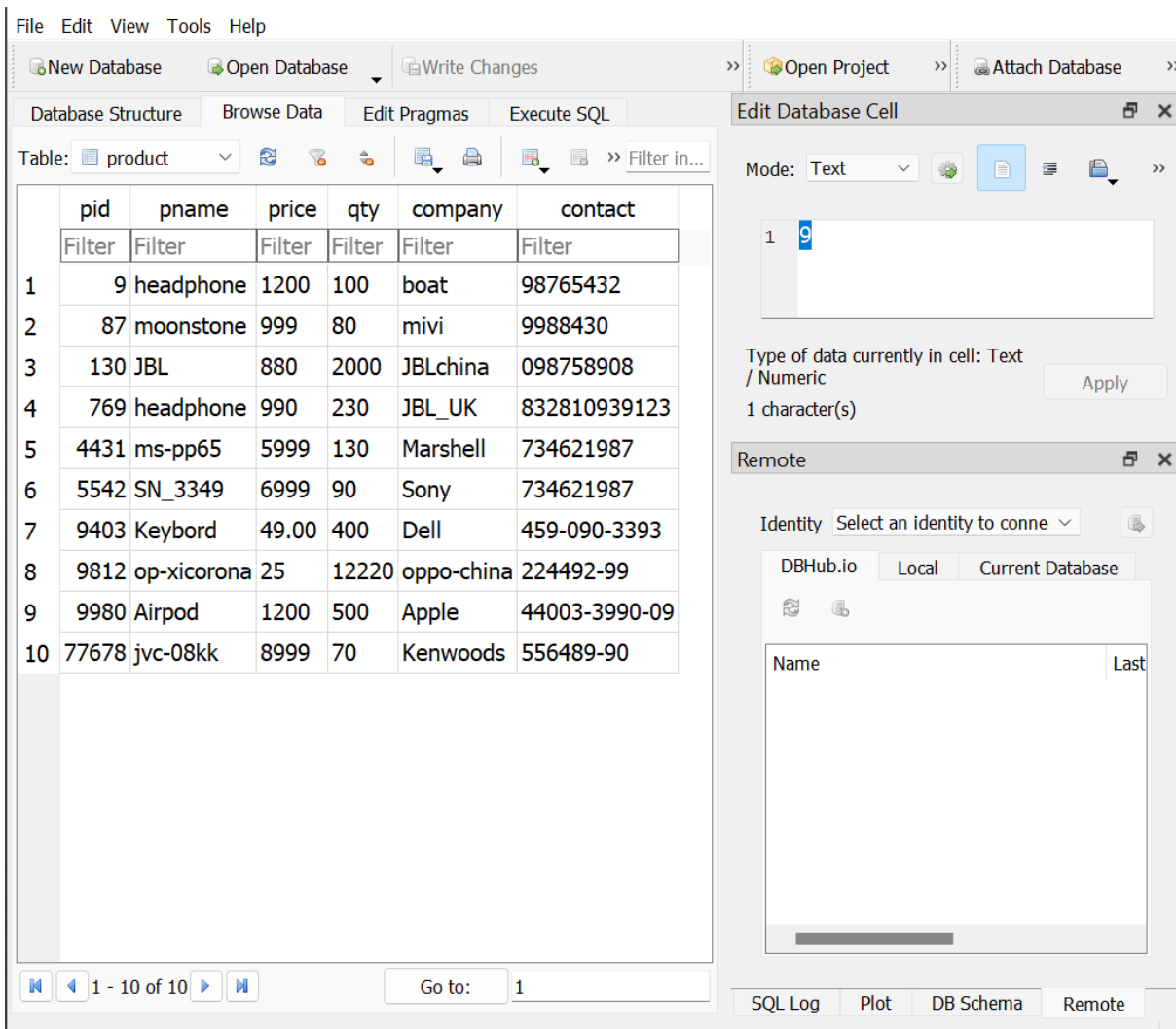
9403 Keybord 49.00 400 Dell 459-090-3393

9812 op-xicorona 25 12220 oppo-china 224492-9

Data Insert Product info Reset Entry Delete Entry Search Entry Close

Figure 15 Front-end product info is clicked

```
#function for the display the database into the system
def display(self):
    print("Database: Display the database")
    conn = sqlite3.connect('inventory.db')
    #Feature and backend function of the cursor is added
    curs = conn.cursor()
    query = "Select * from product"
    curs.execute(query)
    rows= curs.fetchall()
    conn.close()
    print('Database: Disply the database process is completed\n')
    return rows
```



The screenshot shows a database application interface. The main window displays a table of product data with columns: pid, pname, price, qty, company, and contact. The table contains 10 rows of data. The right-hand panel is titled 'Edit Database Cell' and shows a text input field with the value '9'. Below the input field, it indicates the data type is 'Text / Numeric' and the length is '1 character(s)'. There is an 'Apply' button. Below this, there is a 'Remote' section with a dropdown menu for 'Identity' and buttons for 'DBHub.io', 'Local', and 'Current Database'. At the bottom of the interface, there are tabs for 'SQL Log', 'Plot', 'DB Schema', and 'Remote'.

	pid	pname	price	qty	company	contact
1	9	headphone	1200	100	boat	98765432
2	87	moonstone	999	80	mivi	9988430
3	130	JBL	880	2000	JBLchina	098758908
4	769	headphone	990	230	JBL_UK	832810939123
5	4431	ms-pp65	5999	130	Marshell	734621987
6	5542	SN_3349	6999	90	Sony	734621987
7	9403	Keybord	49.00	400	Dell	459-090-3393
8	9812	op-xicorona	25	12220	oppo-china	224492-99
9	9980	Airpod	1200	500	Apple	44003-3990-09
10	77678	jvc-08kk	8999	70	Kenwoods	556489-90

So at the back-end, all this product information is stored in SQL, which is called by the user interface and displayed on the GUI.

- Data delete operation: This function would be a back-end operation of the delete button. If we want to delete some entries, this button identifies the entrance and

gives input to the back end function. That function determines the access and eliminates it from the database.

Warehouse inventory recorder database

## Warehouse inventory recorder database

**Product information**

Product ID:

Product Name:

Product Price:

Product Quantities :

Mfg. Company:

Compnys Conctect:

**Availability of Product**

9 headphone 1200 100 boat 98765432

87 moonstone 999 80 mivi 9988430

130 JBL 880 2000 JBLchina 098758908

769 headphone 990 230 JBL\_UK 832810939123

4431 ms-pp65 5999 130 Marshall 734621987

5542 SN\_3349 6999 90 Sony 734621987

**9403 Keyboard 49.00 400 Dell 459-090-3393**

9812 op-xicorona 25 12220 oppo-china 224492-9

Figure 18 Deleting the data entry from the user interface to the Database

Once we deleted the data entry from the user interface, it also deleted the entry from the database, which can be analysed in Figure 19

	pid	pname	price	qty	company	contact
	Filter	Filter	Filter	Filter	Filter	Filter
1	9	headphone	1200	100	boat	98765432
2	87	moonstone	999	80	mivi	9988430
3	130	JBL	880	2000	JBLchina	098758908
4	769	headphone	990	230	JBL_UK	832810939123
5	4431	ms-pp65	5999	130	Marshell	734621987
6	5542	SN_3349	6999	90	Sony	734621987
7	9812	op-xicorona	25	12220	oppo-china	224492-99
8	9980	Airpod	1200	500	Apple	44003-3990-09
9	77678	jvc-08kk	8999	70	Kenwoods	556489-90

Figure 19 Automatically updated database.

As you compare Figure 17 with Figure 19, you can observe that we have element one data point from the database. Once we give a user command, the interface automatically deletes the databased, so in that manner, we set up the delete function in the backend.

- Search of particular entity process: In this process, the search buttons backend process are connected with this function. This function aims to collate the search keyword from the user interface about the product. It can be product id, name, price-wise search, company wise search or the contact number wise search approach

```
#create a funciton for the search button
def search_buttf(self,pid='',name='', price='',qty='',
                  company='', context=''):
    print('Database: Search funciton exicution')
    conn= sqlite3.connect('inventory.db')
    curs = conn.cursor()
    curs.execute("select * from product where pid = ? or \
                  pname=? or price=? or qty=? or company=? or contact=?",
                  (pid,name,price,qty,company,context))
    rows = curs.fetchall()
    conn.close()
    print(pid,'Database: Search process is completed\n')
    return rows
```

Figure 20 The function of the search button back-end

### **Critical analysis**

In this project work, we analysed some of the critical events of the user interface operations to load the database and studied the available databased. Based on that information, we can pack the product information into the database without running the complex SQL query-based coding. However, when we are trying to add the product information and if by mistake, we said the duplicate value in the product ID, then to shows the error of the databased is crash. We have to re-run the code and database in order to load the database. This error occurs due to information leaking and the duplication of primary critical bugs in the SQLite 3 GitHub source file. However, the changes made on the previous database entry were securely stored in the database. Otherwise, if we wanted to build the user interface with a more attractive look and more interactive graphics, we have to use some paid tools of the python user interface, such as PyQt, which has more windage with better graphics colour options.

### **Application and tools required.**

We utilised the Python programming language to accomplish this project work, a completely open-source language. We used the Spyder IDE for the code building, which provided excellent support for the python codes. In the project work, we utilised two public Gits, one for the building for the user interface named Tkinter links are in the references. Then the other public git is to handle the database. To observe the database, we can directly show it on the user interface; otherwise, we can use the Database Browser, which provides the information about the tables and data points.

### **References**

1. Python SQLite3 Database public git: [cpython/sqlite3.rst at main · python/cpython \(github.com\)](https://github.com/python/cpython/blob/main/Documentation/library/sqlite3.rst)
2. Python Tkinter Graphical user interface public git: [cpython/Lib/tkinter at main · python/cpython \(github.com\)](https://github.com/python/cpython/blob/main/Lib/tkinter)
3. Wayman, W.A., 1995. Inventory accuracy through warehouse control. Production and inventory management journal, 36(2), p.17.
4. Rippel, M., Schmiester, J., Wandfluh, M. and Schönsleben, P., 2016. Building blocks for volume-oriented changeability of assets in production plants. *Procedia CIRP*, 41, pp.15-20.
5. Beniz, D. and Espindola, A., 2016. Using Tkinter of python to create graphical user interface (GUI) for scripts in LNLS. *WEOPRPO25*, 9, pp.25-28.