## 

## 

## 

GOODWILL PHARMACY

**-FOR QUALITY MEDICINAL DRUGS**

# Introduction:

This is a Medicine Billing Application by using Python-MYSQL to make this project. We have used Python Module Tkinter to Create GUI and MYSQL Connector to Connect to MYSQL Database.

Here Python-Tkinter Works as a front-end and MYSQL works as a back-end.

# Abstracts:

1. With some Alterations in the code, it can also be used in Hospitals to record Incoming and Outgoing Patients.
2. It can be used in Pharmacies to check the availability of drugs and check the retails of the day. It will help you keep track of the cash flow in your business.
3. It Gets the input medicine name from the Customer, generates a bill, and saves a copy in MYSQL DATABASE.

# Objectives:

1. To Establish a Connection between MYSQL and Python.
2. To prepare GUI for Billing Application
3. To Code for the following functions:-
   * 1. DISPLAYING older bills from Database
     2. ADDING new bills to Database
     3. MODIFY the existing bill from Database
     4. DELETING an Item from the Bill
     5. DELETING an Entire Bill from Database

**Front-End: PYTHON**

We have used Python Tkinter Module to create GUI & Python has a lot of GUI frameworks, but Tkinter is the only framework that’s built into the Python standard library. Tkinter has several strengths.

It’s cross-platform, so the same code works on Windows, macOS, and Linux. Visual elements are rendered using native operating system elements, so applications built with Tkinter look like they belong on the platform where they’re run.

Tkinter is lightweight and relatively painless to use compared to other frameworks. This makes it a compelling choice for building GUI applications in Python, especially for applications where a modern sheen is unnecessary, and the top priority is to build something functional and cross-platform quickly.

****

**Back-End: MYSQL**

It is open source, Non pirated, reliable, compatible with all major hosting providers, cost-effective, and easy to manage. Many organizations are leveraging the data security and strong transactional support offered by MySQL to secure online transactions and enhance customer interactions.

The primary factor differentiating relational databases from other digital storage lies in how data is organized at a high level. Databases like MySQL contain records in multiple, separate, and highly codified tables, as opposed to a single all-encompassing repository, or collections of semi- or unstructured documents.

This allows RDBMSs to better optimize actions like data retrieval, updating information, or more complex actions like aggregations. A logical model is defined over all of the contents of the database, describing for example the values allowed in individual columns, characteristics of tables and views, or how indices from two tables are related.



**HARDWARE REQUIREMENT:**

* **CPU compatible with your OS(Arm 64-bit, x86 64-bit, etc. Preferably quad-core processor).**
* **Minimum of 4GB of RAM. (8GB is recommended).**
* **Minimum of 16GB of free storage.**

**SOFTWARE REQUIREMENTS:**

* **OS such as Linux, MacOS, Windows, etc.**
* **Python 3.7 version or above.**
* **Python modules such as ‘Tkinter’ and ‘MySQL connector’ which must be installed using a command prompt(on Windows OS) or terminal(on Linux and MacOS).**
* **MySQL database version 5.6, 5.7, 8.0, or greater.**

# Flow chart:

# 

SOURCE CODE:

1. GUI and Python :

# importing all the required packages

from tkinter import \*

from tkinter import messagebox, ttk

from database import \*

# colors

black = "#000000"

white = "#FFFFFF"

red = "#FF0000"

green = "#00FF00"

blue = "#0066FF"

yellow = "#FFFF00"

cyan = "#00FFFF"

purple = "#A64DFF"

orange = "#FF8533"

# preping a window

root = Tk()

root.title("BILLING APP")

root.configure(background = black)

# setting an icon for the app

icon\_image = PhotoImage(file = r"pharmacy icon.png")

root.tk.call('wm', 'iconphoto', root.\_w, icon\_image)

# clearing all the widgets from the page to make room for the next page

def clear\_page():

for widget in root.grid\_slaves():

widget.destroy()

# command for the exit button

def exit():

global exit\_window

exit\_window = Tk()

exit\_window.title("EXIT")

exit\_label = Label(exit\_window, text = "DO YOU WANT TO EXIT?")

exit\_label.grid(row = 0, column = 0, padx = 2, pady = 2, columnspan = 2)

yes\_button = Button(exit\_window, text = "YES", borderwidth = 2, command = lambda : destroy\_app(1))

yes\_button.grid(row = 1, column = 0, padx = 2, pady = 2)

no\_button = Button(exit\_window, text = "NO", background = black, foreground = white, borderwidth = 2, command = lambda : destroy\_app(0))

no\_button.grid(row = 1, column = 1, padx = 2, pady = 2)

# this function destroys the eintire app by closing the all the windows and quiting

def destroy\_app(a):

global root

if a == 1:

root.destroy()

exit\_window.destroy()

quit()

else:

exit\_window.destroy()

# destroying all the widgets in the left side frame

def clear\_left\_frame():

for widgets in left\_side\_frame.winfo\_children():

widgets.destroy()

# we set the output\_box state to disabled so that the user can't change anything in the output box

# It is only used to see the output bill

# changing the state from disabled to normal so that the system can edit the content in the output box

def clear\_a\_line\_in\_text\_box(line):

output\_box.config(state = NORMAL)

output\_box.delete(float(line), float(line + 1))

output\_box.config(state = DISABLED)

def clear\_the\_entire\_text\_box():

output\_box.config(state = NORMAL)

output\_box.delete(1.0, END)

output\_box.config(state = DISABLED)

def clear\_entire\_text\_box\_from\_start\_line(line):

output\_box.config(state = NORMAL)

output\_box.delete(float(line), END)

output\_box.config(state = DISABLED)

def display\_in\_output\_box(string, position = -1):

output\_box.config(state = NORMAL)

output\_box.insert(f"end {position} chars", string)

output\_box.config(state = DISABLED)

def get\_entire\_text\_box():

return output\_box.get(1.0, END)

# login gui for the user with login button.

def login\_gui():

# entry boxes for username and password and passing the parameters to the button function

# this is for clearing the page when the login button is pressed

try:

clear\_page()

except:

pass

username\_name\_label = Label(root, text = "Username:", foreground = white, background = black)

username\_name\_label.grid(row = 0, column = 0, padx = 4, pady = 4, sticky = EW)

global username\_name\_entry

username\_name\_entry = Entry(root)

username\_name\_entry.grid(row = 0, column = 1, padx = 4, pady = 4)

password\_label = Label(root, text = "Password:", foreground = white, background = black)

password\_label.grid(row = 1, column = 0, padx = 4, pady = 4, sticky = EW)

global password\_entry

password\_entry = Entry(root, show = "⚹")

password\_entry.grid(row = 1, column = 1, padx = 4, pady = 4)

login\_button = Button(root, text = "LOGIN", foreground = black, background = green ,command = lambda : login\_button\_func(username\_name\_entry.get(), password\_entry.get()))

login\_button.grid(row = 3, column = 1, padx = 4, pady = 4, sticky = E)

# login code to execute to authenticate the username and the password entered by the user

def login\_button\_func(username\_input, password\_input):

if authentication(username\_input, password\_input):

# savind the username to know who made pa bill

global username

username = username\_name\_entry.get()

# wiping the entire page and changing the gui to the mainpage

clear\_page()

# creating two frames to sperate the output box and all the buttons and lables

global left\_side\_frame

global right\_side\_frame

left\_side\_frame = Frame(root, background = black)

right\_side\_frame = Frame(root, background = black)

left\_side\_frame.grid(row=0, column=0)

right\_side\_frame.grid(row=0, column=1)

main\_page()

else:

# error pop up,clear the entry widgets ask the user to re-enter all the creadentials

messagebox.showerror("ACCESS DENIED!!","Wrong username or password entered!")

password\_entry.delete(0, END)

username\_name\_entry.delete(0, END)

# creating main page where user can access all the commands like "new bill"

def main\_page():

clear\_left\_frame()

# clear\_the\_entire\_text\_box()

# getting data of main\_bill from database.

global database\_bill\_no

database\_bill\_no = get\_bill\_no()

global extra\_bill\_no

extra\_bill\_no = [num for num in range(1, 10001) if num not in database\_bill\_no]

global main\_bill\_raw

main\_bill\_raw = get\_main\_bill()

# LEFT FRAME

home\_label = Label(left\_side\_frame, text = "HOME", width = 30, height = 5, background = black, foreground = orange, font = (100))

home\_label.grid(row = 0, column = 0 , padx = 10 , pady = 20, sticky = EW)

show\_bill\_button = Button(left\_side\_frame, text = "SHOW A BILL", width = 25, background = green, foreground = black, command = lambda : show\_bill\_page())

show\_bill\_button.grid(row = 1, column = 0, padx = 10, pady = 20)

add\_bill\_button = Button(left\_side\_frame, text = "ADD A NEW BILL" , width = 25, background = cyan, foreground = black, command = lambda : add\_bill\_page())

add\_bill\_button.grid(row = 2, column = 0, padx = 10, pady = 20)

modify\_bill\_button = Button(left\_side\_frame, text = "MODIFY A BILL", width = 25, background = orange, foreground = black, command = lambda : modify\_bill\_page())

modify\_bill\_button.grid(row = 3, column = 0, padx = 10, pady = 20)

delete\_bill\_button = Button(left\_side\_frame, text = "DELETE A BILL", width = 25, background = red, foreground = black, command = lambda : delete\_bill\_page())

delete\_bill\_button.grid(row = 4, column = 0, padx = 10, pady = 20)

exit\_button = Button(left\_side\_frame, text = "EXIT", background = red , foreground = black, command = lambda : exit())

exit\_button.grid(row = 5, column = 0, sticky = W, padx = 10, pady = 20)

log\_out\_button = Button(left\_side\_frame, text = "LOGOUT", background = purple, foreground = black, command = lambda : login\_gui())

log\_out\_button.grid(row = 5, column = 0, sticky = E, padx = 10, pady = 20)

# RIGHT FRAME

global output\_box

output\_box = Text(right\_side\_frame, width = 95, height = 30, background = white, foreground = black, state = DISABLED)

output\_box.grid(column = 0, row = 0, padx = 5, pady = 5)

# show the main bill in the output box

text = """+-------------+---------------------------+-----------------+--------------+-----------------+

| BILL NUMBER | CUSTOMER NAME | PHONE NO. | TOTAL | BILLER NAME |

+-------------+---------------------------+-----------------+--------------+-----------------+\n"""

for items in main\_bill\_raw:

text += f"| {items[0]}{((11 - len(str(items[0]))) \* ' ')} "

text += f"| {items[1]}{((25 - len(str(items[1]))) \* ' ')} "

text += f"| {items[2]}{((15 - len(str(items[2]))) \* ' ')} "

text += f"| {items[3]}{((12 - len(str(items[3]))) \* ' ')} "

text += f"| {items[4]}{((15 - len(str(items[4]))) \* ' ')} "

text += "|\n|-------------+---------------------------+-----------------+--------------+-----------------|\n"

text = text[0:len(text)-95]

text += "+-------------+---------------------------+-----------------+--------------+-----------------+"

display\_in\_output\_box(text)

# this page is to show the desired bill entered by the user

def show\_bill\_page():

# destroying all widgets in left frame to make room for new widgets

clear\_left\_frame()

show\_bill\_page\_label = Label(left\_side\_frame, text = "SHOW BILL", background = black, foreground = orange, font = (100))

show\_bill\_page\_label.grid(row = 0, column = 0, columnspan = 3, padx = 10, pady = 20, sticky = N)

input\_label = Label(left\_side\_frame, text = "Enter the bill number to display", background = black, foreground = white)

input\_label.grid(row = 1, column = 1, padx = 5, pady = 15, sticky = EW)

global select\_number

select\_number = StringVar()

input\_bill\_no = ttk.Combobox(left\_side\_frame, textvariable = select\_number, values = database\_bill\_no, state = "readonly")

input\_bill\_no.grid(row = 2,column=1, padx = 5, pady = 15, sticky = EW)

display\_button = Button(left\_side\_frame,text = "DISPLAY BILL",background = green ,foreground = black, command = lambda : show\_bill\_in\_text\_box())

display\_button.grid(row = 3,column = 1, padx = 5, pady = 11, sticky = EW)

print\_button = Button(left\_side\_frame, text = "PRINT BILL", background = blue, foreground = black, command = lambda : print\_bill())

print\_button.grid(row = 4, column = 1, padx = 5, pady = 15 ,sticky = EW)

exit\_button = Button(left\_side\_frame, text = "EXIT", background = red , foreground = black, command = lambda : exit())

exit\_button.grid(row = 4, column = 0, padx = 5, pady = 15, sticky = W)

back\_button = Button(left\_side\_frame, text = "BACK", background = purple, foreground = black, command = lambda : main\_page())

back\_button.grid(row = 4, column = 2, padx = 5, pady = 15, sticky = E)

# validates whether the given bill is there in the database and the entered bill is an integer

def show\_bill\_in\_text\_box():

try:

global display\_bill\_no

display\_bill\_no = int(select\_number.get())

global display\_bill

display\_bill = get\_customer\_bill(display\_bill\_no)

details = get\_customer\_name\_and\_phone\_number(display\_bill\_no)

# SHOWING THE BILL

text = f"CUSTOMER NAME : {details[1]}\nPHONE NUMBER : {details[2]}\nBILL NUMBER : {display\_bill\_no}\nBILLER NAME : {details[4]}\n\n"

text += """+------+-------------------------------------+--------+-----+-----------+

| S.NO | PRODUCT NAME | PRICE | QTY | TOTAL |

+------+-------------------------------------+--------+-----+-----------+\n"""

grand\_tot = 0

for i in range(len(display\_bill)):

text += f"| {i + 1}.{((3 - len(str(i + 1))) \* ' ')} "

text += f"| {display\_bill[i][0]}{((35 - len(str(display\_bill[i][0]))) \* ' ')} "

text += f"| {display\_bill[i][1]}{((6 - len(str(display\_bill[i][1]))) \* ' ')} "

text += f"| {display\_bill[i][2]}{((3 - len(str(display\_bill[i][2]))) \* ' ')} "

text += f"| {display\_bill[i][3]}{((9 - len(str(display\_bill[i][3]))) \* ' ')} "

text += "|\n|------+-------------------------------------+--------+-----+-----------|\n"

grand\_tot += display\_bill[i][3]

text = text[0:len(text)-75]

text += f"""\n+------+-------------------------------------+--------+-----+-----------+

| GRAND TOTAL : {grand\_tot}{((10 - len(str(grand\_tot))) \* ' ')}|

+-----------------------------------------------------------------------+"""

# making room for the desired output

clear\_the\_entire\_text\_box()

# displaying the bill in the output box

display\_in\_output\_box(text)

except:

pass

def print\_bill():

file = open(f"BILL {display\_bill\_no}", "w")

# pharmacy name

# file.write(f"")

file.write(get\_entire\_text\_box())

file.close()

def add\_bill\_page():

# destroying all the widgets in the left side frame

clear\_left\_frame()

# displaying all the widgets for getting customer name, phone number, bill number

add\_bill\_page\_label = Label(left\_side\_frame, text = "ADD A NEW BILL", background = black, foreground = orange, font = (100))

add\_bill\_page\_label.grid(row = 0, column = 0, columnspan = 2, padx = 10, pady = 20, sticky = N)

label\_customer\_name = Label(left\_side\_frame, text = "Customer Name:", foreground = white, background = black)

label\_customer\_name.grid(row = 1, column = 0, padx = 10, pady = 20, sticky = NS)

global customer\_name\_entry

customer\_name\_entry = Entry(left\_side\_frame)

customer\_name\_entry.grid(row = 1, column = 1, padx = 10, pady = 20)

label\_phone\_number = Label(left\_side\_frame, text = "Phone Number:", foreground = white, background = black)

label\_phone\_number.grid(row = 2, column = 0, padx = 10, pady = 20, sticky = NS)

global phone\_number\_entry

phone\_number\_entry = Entry(left\_side\_frame)

phone\_number\_entry.grid(row = 2, column = 1, padx = 10, pady = 20)

create\_new\_bill\_button = Button(left\_side\_frame, text = "CREATE NEW BILL", background = green, foreground = black, command = lambda : add\_validation())

create\_new\_bill\_button.grid(row = 4, column = 0, columnspan = 2, padx = 10, pady = 20)

exit\_button = Button(left\_side\_frame, text = "EXIT", background = red , foreground = black, command = lambda : exit())

exit\_button.grid(row = 5, column = 0, sticky = W, padx = 10, pady = 20)

back\_button = Button(left\_side\_frame, text = "BACK", background = purple, foreground = black, command = lambda : main\_page())

back\_button.grid(row = 5, column = 1, sticky = E, padx = 10, pady = 20)

def add\_validation():

# this function is to validate whether the given credentials are correct

global customer\_name

global phone\_number

global bill\_number

customer\_name = customer\_name\_entry.get().lstrip().rstrip()

phone\_number = phone\_number\_entry.get().lstrip().rstrip()

bill\_number = extra\_bill\_no.pop(0)

message = ""

# making sure the given credentials are of correct length and data type

if str(bill\_number).isnumeric() == False or bill\_number in database\_bill\_no or len(customer\_name) > 15 or len(str(bill\_number)) > 10 or len(phone\_number) > 15 or customer\_name == "" or phone\_number == "":

if bill\_number in database\_bill\_no:

message += "# Bill no. already exist!\n"

if len(customer\_name) > 15:

message += "# Customer name has greater than 15 characters!\n"

if len(str(bill\_number)) > 10:

message += "# Bill no. has greater than 10 characters!\n"

if len(phone\_number) > 15:

message += "# Phone no. has greater than 15 characters!"

# informing user about the invalid input given

messagebox.showwarning(title = "INVALID INPUTS!!", message = message)

else:

# next operation page

customer\_name = customer\_name.title()

add\_modify\_operation\_page("add")

# this page is used by both add and modify section of code

# with few alteration we can change the code by passing different parameter

def add\_modify\_operation\_page(add\_or\_modify):

# cleaning the left frame to add new widgets and clearing the right output textbox to

# display the bill items and list them

clear\_left\_frame()

clear\_the\_entire\_text\_box()

# widgets for left frame

add\_bill\_page\_label = Label(left\_side\_frame, text = "ADD A NEW BILL", background = black, foreground = orange, font = (100))

add\_bill\_page\_label.grid(row = 0, column = 0, columnspan = 2, padx = 10, pady = 15, sticky = N)

label\_product\_name = Label(left\_side\_frame, text = "Product name:", foreground = white, background = black)

label\_product\_name.grid(row = 1, column = 0, padx = 10 , pady = 15)

global entry\_product\_name

entry\_product\_name = Entry(left\_side\_frame)

entry\_product\_name.grid(row = 1, column = 1, padx = 10 , pady = 15)

label\_product\_price = Label(left\_side\_frame, text = "Product Price:", foreground = white, background = black)

label\_product\_price.grid(row = 2, column = 0, padx = 10 , pady = 15)

global entry\_product\_price

entry\_product\_price = Entry(left\_side\_frame)

entry\_product\_price.grid(row = 2, column = 1, padx = 10 , pady = 15)

label\_product\_quantity = Label(left\_side\_frame, text = "Quantity:", foreground = white, background = black)

label\_product\_quantity.grid(row = 3, column = 0, padx = 10 , pady = 15)

global entry\_product\_quantity

entry\_product\_quantity = Entry(left\_side\_frame)

entry\_product\_quantity.grid(row = 3, column = 1, padx = 10 , pady = 15)

add\_product\_button = Button(left\_side\_frame, text = "ADD PRODUCT", foreground = black, background = green, command = lambda : add\_product\_function\_validation())

add\_product\_button.grid(row = 4, column = 0, columnspan = 2, sticky = EW, padx = 10 , pady = 15)

delete\_product\_label = Label(left\_side\_frame, text = "Enter the S.No to remove :", foreground = white, background = black)

delete\_product\_label.grid(row = 5, column = 0, columnspan = 2, sticky = W, padx = 10 , pady = 15)

global delete\_product\_entry

delete\_product\_entry = Entry(left\_side\_frame, width = 10)

delete\_product\_entry.grid(row = 5, column = 0, columnspan = 2, sticky = E, padx = 10 , pady = 15)

delete\_product\_button = Button(left\_side\_frame, text = "DELETE PRODUCT", foreground = black, background = cyan, command = lambda : delete\_an\_item(delete\_product\_entry.get()))

delete\_product\_button.grid(row = 6, column = 0, columnspan = 2, sticky = EW, padx = 10 , pady = 15)

save\_button = Button(left\_side\_frame, text = "SAVE BILL", width = 5, foreground = black, background = orange, command = lambda : save\_bill\_database())

save\_button.grid(row = 7, column = 0, columnspan = 2, padx = 10, pady = 15, sticky = EW)

exit\_button = Button(left\_side\_frame, text = "EXIT", background = red , foreground = black, command = lambda : exit())

exit\_button.grid(row = 8, column = 0, padx = 10, pady = 15, sticky = W)

back\_button = Button(left\_side\_frame, text = "BACK", background = purple, foreground = black, command = lambda : back\_button\_validation())

back\_button.grid(row = 8, column = 1, padx = 10, pady = 15, sticky = E)

# preparing for the output box

global text

text = """+------+-------------------------------------+--------+-----+-----------+

| S.NO | PRODUCT NAME | PRICE | QTY | TOTAL |

+------+-------------------------------------+--------+-----+-----------+\n"""

global serial\_no

global grand\_total

global products

global price

global quantity

global total

serial\_no, grand\_total, products, price, quantity, total = 0, 0, [], [], [], []

# by passing different parameter we can choose between modify and add functions

if add\_or\_modify == "modify":

customer\_bill = get\_customer\_bill(bill\_number)

for i in range(len(customer\_bill)):

text += f"| {i + 1}.{((3 - len(str(i + 1))) \* ' ')} "

text += f"| {customer\_bill[i][0]}{((35 - len(str(customer\_bill[i][0]))) \* ' ')} "

text += f"| {customer\_bill[i][1]}{((6 - len(str(customer\_bill[i][1]))) \* ' ')} "

text += f"| {customer\_bill[i][2]}{((3 - len(str(customer\_bill[i][2]))) \* ' ')} "

text += f"| {customer\_bill[i][3]}{((9 - len(str(customer\_bill[i][3]))) \* ' ')} "

text += "|\n|------+-------------------------------------+--------+-----+-----------|\n"

serial\_no += 1

grand\_total += customer\_bill[i][1] \* customer\_bill[i][2]

products.append(customer\_bill[i][0])

price.append(customer\_bill[i][1])

quantity.append(customer\_bill[i][2])

total.append(customer\_bill[i][3])

text = text[0:len(text)-74]

text += f"""+------+-------------------------------------+--------+-----+-----------+

| GRAND TOTAL : {grand\_total}{((10 - len(str(grand\_total))) \* ' ')}|

+-----------------------------------------------------------------------+"""

display\_in\_output\_box(text)

# this remainds the user to save the bill before leaving to home page

# there is a function in save\_bill command to implement.

global remind\_save\_bill

remind\_save\_bill = False

# validates whether the entered credentials is valid

def add\_product\_function\_validation():

global product\_name

global product\_price

global product\_quantity

product\_name = entry\_product\_name.get().lstrip().rstrip().title()

product\_price = entry\_product\_price.get().lstrip().rstrip()

product\_quantity = entry\_product\_quantity.get().lstrip().rstrip()

global grand

grand = True

text = ""

if len(str(serial\_no)) > 4 or len(product\_name) > 35 or product\_price.isnumeric() == False or len(str(product\_price)) > 6 or product\_quantity.isnumeric() == False or len(str(product\_quantity)) > 3 or (int(grand\_total) + (int(product\_price) \* (int(product\_quantity)))) > 9999999999:

if len(str(serial\_no)) > 4:

text += "# Only 9999 products can be stored in a bill.\n"

if len(product\_name) > 35:

text += f"# Reduce the name of product by {len(product\_name) - 35} characters.\n"

if product\_price.isnumeric() == False:

text += "# Price cannot take alphabets of any special characters.\n"

grand = False

else:

product\_price = int(product\_price)

if len(str(product\_price)) > 6:

text += "# Price can take value upto 9,99,999.\n"

if product\_quantity.isnumeric() == False:

text += "# Price cannot take alphabets of any special characters.\n"

grand = False

else:

product\_quantity = int(product\_quantity)

if len(str(product\_quantity)) > 3:

text += "# Price can take value upto 999.\n"

if grand\_total + (product\_price \* product\_quantity) > 9999999999:

text += "# Grand total cannot take value greater than 999,99,99,999. Click save bill and make a new bill.\n"

# informing the user about the error

messagebox.showerror("ERROR", text)

else:

add\_product\_function()# checking whether the given bill no is there in database

def add\_product\_function():

global serial\_no, grand\_total, product\_quantity, product\_price

# type casting the required variables

product\_quantity = int(product\_quantity)

product\_price = int(product\_price)

# adding all the necessary details to the list

products.append(product\_name)

price.append(product\_price)

quantity.append(product\_quantity)

total\_price = product\_price \* product\_quantity

total.append(total\_price)

serial\_no += 1

grand\_total += total\_price

# clearing the required lines to accommodate for the added item

if serial\_no > 2:

clear\_a\_line\_in\_text\_box(serial\_no \* 2 + 1)

clear\_a\_line\_in\_text\_box(serial\_no \* 2 + 1)

clear\_a\_line\_in\_text\_box(serial\_no \* 2 + 1)

else:

clear\_a\_line\_in\_text\_box(serial\_no + 3)

clear\_a\_line\_in\_text\_box(serial\_no + 3)

clear\_a\_line\_in\_text\_box(serial\_no + 3)

# making text for displaying output

text = ""

if serial\_no > 1:

text += "\n|------+-------------------------------------+--------+-----+-----------|"

text += f"\n| {serial\_no}.{((3 - len(str(serial\_no))) \* ' ')} "

text += f"| {product\_name}{((35 - len(str(product\_name))) \* ' ')} "

text += f"| {product\_price}{((6 - len(str(product\_price))) \* ' ')} "

text += f"| {product\_quantity}{((3 - len(str(product\_quantity))) \* ' ')} "

text += f"| {total\_price}{((9 - len(str(total\_price))) \* ' ')} |"

display\_in\_output\_box(text)

text = f"""\n+------+-------------------------------------+--------+-----+-----------+

| GRAND TOTAL : {grand\_total}{((10 - len(str(grand\_total))) \* ' ')}|

+-----------------------------------------------------------------------+"""

display\_in\_output\_box(text)

# changing the value to true to remind the user to save bill

global remind\_save\_bill

remind\_save\_bill = True

entry\_product\_name.delete(0, END)

entry\_product\_price.delete(0, END)

entry\_product\_quantity.delete(0, END)

def delete\_an\_item(s\_number):

global serial\_no

global grand\_total

try:

# making the entered number positive even if the entered number is negative

s\_number = int(s\_number).\_\_abs\_\_()

if s\_number <= serial\_no:

# clearing the line according to the serial no entered by the user

if s\_number < 2 :

clear\_entire\_text\_box\_from\_start\_line(3 + s\_number)

else:

clear\_entire\_text\_box\_from\_start\_line((s\_number \* 2) + 2)

# deleting the items from the lists

del products[s\_number - 1]

del price[s\_number - 1]

del quantity[s\_number - 1]

grand\_total -= total[s\_number - 1]

del total[s\_number - 1]

serial\_no -= 1

# preparing new test for output

text = "\n"

try:

for i in range(s\_number - 1, len(products)):

text += f"| {i + 1}.{((3 - len(str(i + 1))) \* ' ')} "

text += f"| {products[i]}{((35 - len(str(products[i]))) \* ' ')} "

text += f"| {price[i]}{((6 - len(str(price[i]))) \* ' ')} "

text += f"| {quantity[i]}{((3 - len(str(quantity[i]))) \* ' ')} "

text += f"| {total[i]}{((9 - len(str(total[i]))) \* ' ')} "

text += "|\n|------+-------------------------------------+--------+-----+-----------|\n"

except:

pass

text += f"""| GRAND TOTAL : {grand\_total}{((10 - len(str(grand\_total))) \* ' ')}|

+------+-------------------------------------+--------+-----+-----------+"""

display\_in\_output\_box(text)

global remind\_save\_bill

remind\_save\_bill = True

else:

messagebox.showerror("ERROR", "ENTERED NUMBER EXCEEDS SERIAL NUMBER")

delete\_product\_entry.delete(0, END)

except:

messagebox.showerror("ERROR", "CANNOT FIND THE SERIAL NUMBER TO DELETE")

delete\_product\_entry.delete(0, END)

delete\_product\_entry.delete(0, END)

# validate for back button

def back\_button\_validation():

global remind\_save\_bill

if remind\_save\_bill:

messagebox.showwarning("WARNING", "BILL IS NOT SAVED IN THE DATABASE!\nCLICK 'SAVE BILL' TO SAVE")

else:

main\_page()

def save\_bill\_database():

global database\_bill\_no

if bill\_number not in database\_bill\_no:

create\_a\_new\_bill(bill\_number, customer\_name, phone\_number, grand\_total, username)

database\_bill\_no = get\_bill\_no()

else:

delete\_records(bill\_number)

save\_bill(bill\_number, products, price, quantity, total, grand\_total)

global remind\_save\_bill

remind\_save\_bill = False

def modify\_bill\_page():

# clearing left frame for new widgets

clear\_left\_frame()

modify\_bill\_page\_label = Label(left\_side\_frame, text = "MODIFY BILL", background = black, foreground = orange, font = (100))

modify\_bill\_page\_label.grid(row = 0, column = 0, columnspan = 3, padx = 10, pady = 20, sticky = N)

modify\_input\_label = Label(left\_side\_frame, text = "Select the bill number to modify",background = black, foreground = white)

modify\_input\_label.grid(row = 1,column = 1, padx = 5,pady = 15, sticky = EW)

global modify\_number

modify\_number = StringVar()

modify\_input\_bill\_no = ttk.Combobox(left\_side\_frame, textvariable = modify\_number, values = database\_bill\_no, state = "readonly")

modify\_input\_bill\_no.grid(row = 2,column = 1, padx = 5,pady = 15, sticky = EW)

modify\_button = Button(left\_side\_frame,text = "MODIFY BILL",background = orange ,foreground = black, command = lambda : modify\_bill\_validation())

modify\_button.grid(row = 3,column = 1, padx = 5,pady = 15, sticky = EW)

exit\_button = Button(left\_side\_frame, text = "EXIT", background = red , foreground = black, command = lambda : exit())

exit\_button.grid(row = 4, column = 0, padx = 5, pady = 15, sticky = W)

back\_button = Button(left\_side\_frame, text = "BACK", background = purple, foreground = black, command = lambda : main\_page())

back\_button.grid(row = 4, column = 2, padx = 5, pady = 15, sticky = E)

# validation for modify page

def modify\_bill\_validation():

try:

modify\_bill\_number = int(modify\_number.get())

global bill\_number

bill\_number = modify\_bill\_number

add\_modify\_operation\_page("modify")

except:

pass

def delete\_bill\_page():

# clearing left frame for new widgets

clear\_left\_frame()

delete\_bill\_page\_label = Label(left\_side\_frame, text = "DELETE BILL", background = black, foreground = orange, font = (100))

delete\_bill\_page\_label.grid(row = 0, column = 0, columnspan = 3, padx = 10, pady = 20, sticky = N)

del\_input\_label = Label(left\_side\_frame, text = "Select the bill number to delete",background = black, foreground = white)

del\_input\_label.grid(row = 1, column = 1, padx = 5,pady = 15, sticky = EW)

global delete\_number

delete\_number = StringVar()

del\_input\_bill\_no = ttk.Combobox(left\_side\_frame, textvariable = delete\_number, values = database\_bill\_no, state = "readonly")

del\_input\_bill\_no.grid(row = 2, column = 1, padx = 5,pady = 15, sticky = EW)

delete\_button = Button(left\_side\_frame,text = "DELETE BILL",background = cyan ,foreground = black, command = lambda : delete\_bill\_validation())

delete\_button.grid(row = 3,column = 1, padx = 5,pady = 15, sticky = EW)

exit\_button = Button(left\_side\_frame, text = "EXIT", background = red , foreground = black, command = lambda : exit())

exit\_button.grid(row = 4, column = 0, padx = 5, pady = 15, sticky = W)

back\_button = Button(left\_side\_frame, text = "BACK", background = purple, foreground = black, command = lambda : main\_page())

back\_button.grid(row = 4, column = 2, padx = 5, pady = 15, sticky = E)

def delete\_bill\_validation():

try:

delete\_bill\_number = int(delete\_number.get())

delete\_customer\_bill(delete\_bill\_number)

# clearing the output box for main page

clear\_the\_entire\_text\_box()

main\_page()

except:

pass

login\_gui()

root.mainloop()

1. DATABASE ( MYSQL):

import mysql.connector

#authentication for the billing person

def authentication(username, password):

try:

global mysql\_con

mysql\_con = mysql.connector.connect(host = "localhost", user = username, password = password, db = "billing\_app")

global mysql\_cursor

mysql\_cursor = mysql\_con.cursor()

return True

except:

return False

def get\_bill\_no():

mysql\_cursor.execute("SELECT bill\_no FROM main\_bill")

tmp\_list = []

for number in mysql\_cursor.fetchall():

tmp\_list.append(number[0])

return tmp\_list

def get\_main\_bill():

mysql\_cursor.execute("SELECT \* FROM main\_bill")

return mysql\_cursor.fetchall()

def create\_a\_new\_bill(bill\_no, customer, phone, total, biller):

mysql\_cursor.execute(f"CREATE TABLE bill\_{bill\_no}(item\_name varchar(35), price int, quantity int, total int)")

mysql\_con.commit()

mysql\_cursor.execute(f"INSERT INTO main\_bill VALUES({bill\_no}, '{customer}', '{phone}', '{total}', '{biller}')")

mysql\_con.commit()

def get\_customer\_bill(bill\_no):

mysql\_cursor.execute(f"SELECT \* FROM bill\_{bill\_no}")

return mysql\_cursor.fetchall()

def get\_customer\_name\_and\_phone\_number(bill\_no):

mysql\_cursor.execute(f"SELECT \* FROM main\_bill where bill\_no = {bill\_no}")

return mysql\_cursor.fetchall()[0]

def delete\_customer\_bill(bill\_no):

mysql\_cursor.execute(f"DROP TABLE bill\_{bill\_no}")

mysql\_cursor.execute(f"DELETE FROM main\_bill where bill\_no = {bill\_no}")

mysql\_con.commit()

def save\_bill(bill\_no, products, price, quantity, total, grand\_total):

for i in range(len(products)):

mysql\_cursor.execute(f"INSERT INTO bill\_{bill\_no} VALUES('{products[i]}', {price[i]}, {quantity[i]}, {total[i]})")

mysql\_con.commit()

mysql\_cursor.execute(f"UPDATE main\_bill SET total = {grand\_total} WHERE bill\_no = {bill\_no}")

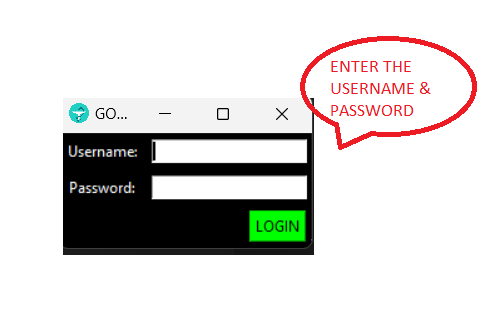
mysql\_con.commit()

def delete\_records(bill\_no):

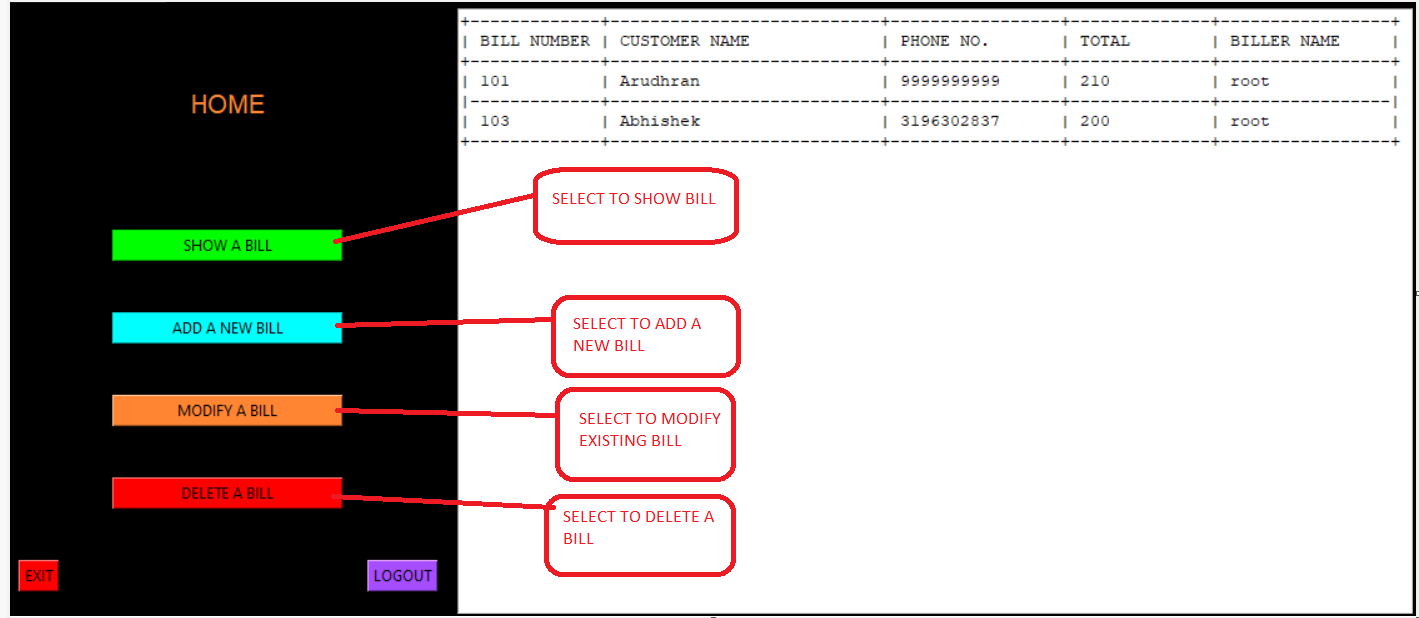
mysql\_cursor.execute(f"DELETE FROM bill\_{bill\_no}")

mysql\_con.commit()

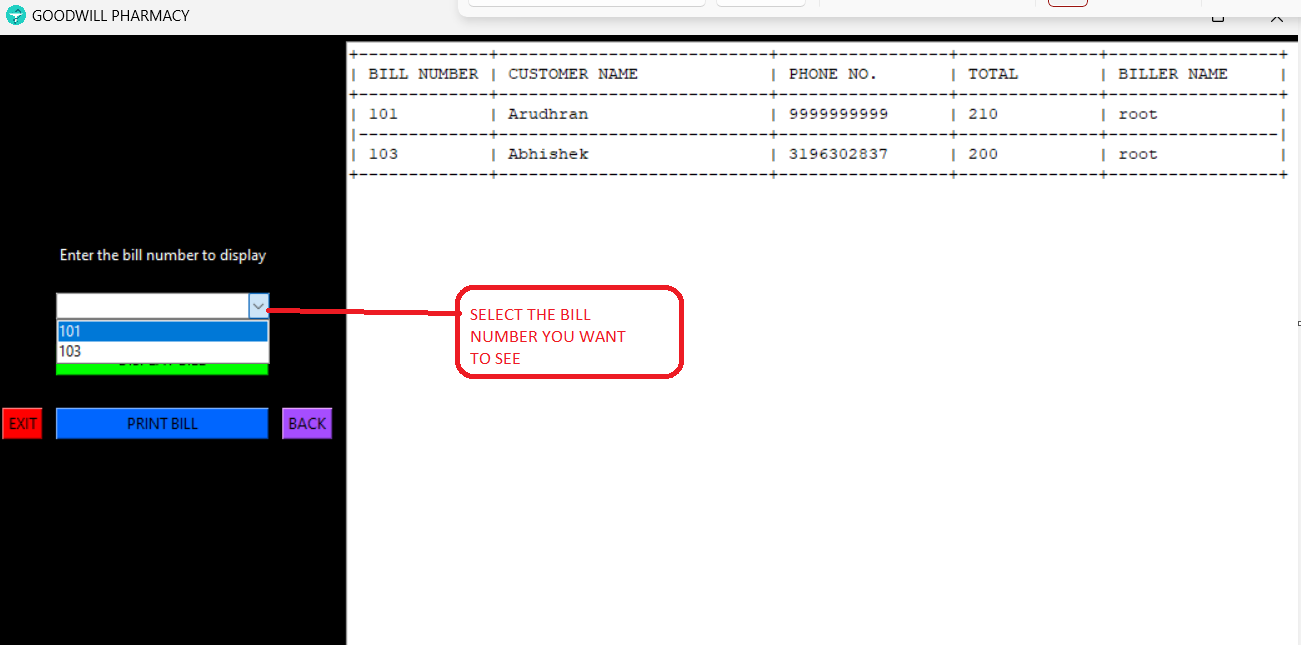
OUTPUT SCREENSHOTS:

i) Enter the Username and Password

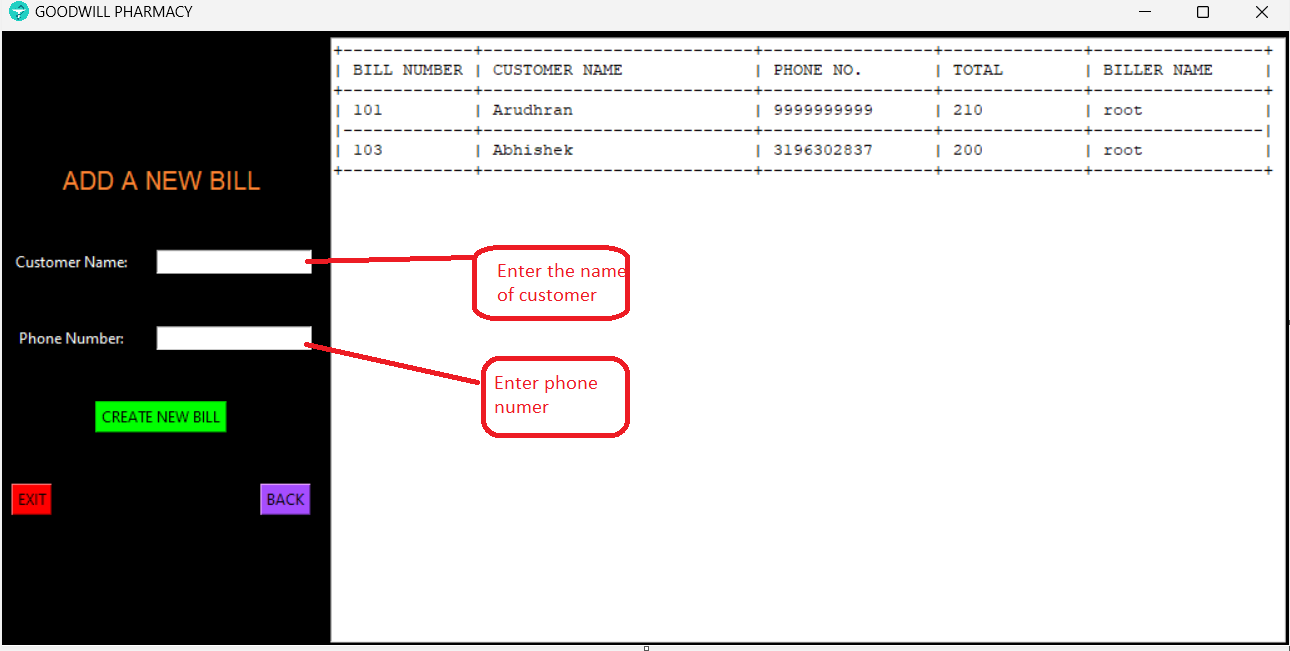
ii) The window here, is used to access the bill, i.e To DELETE, ADD, SHOW & MODIFY



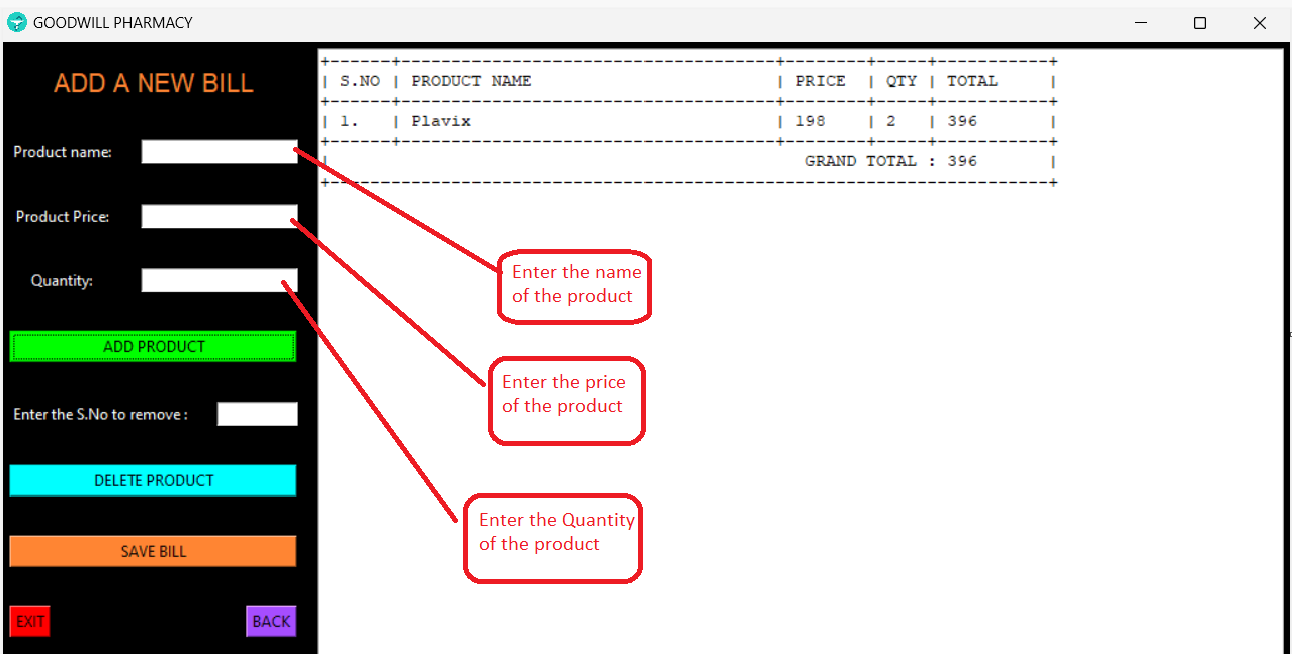
iii) To Show bills:



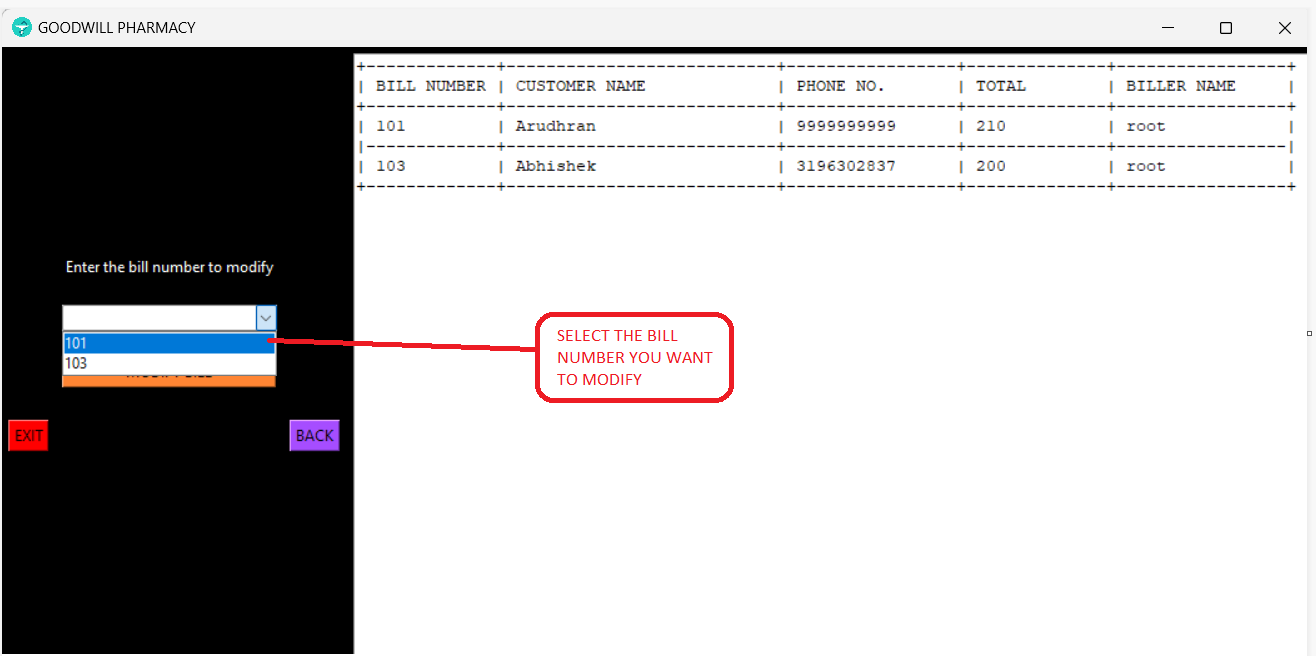
iv) a) To Add a new bill:



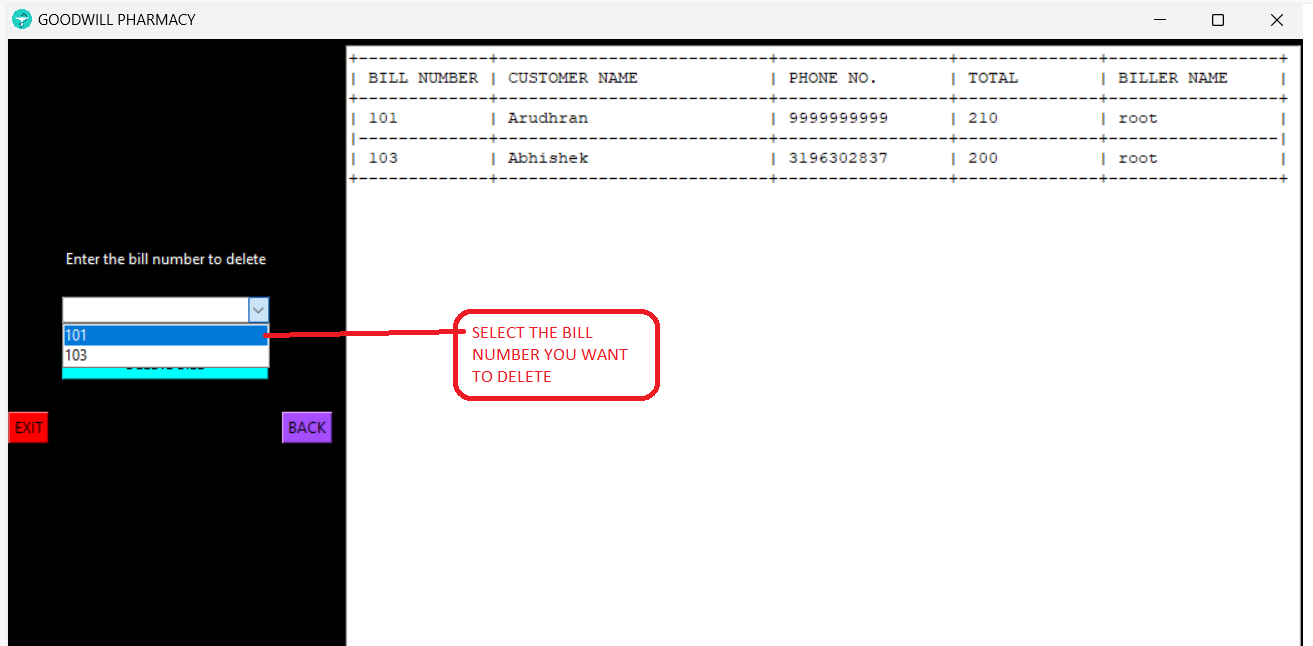
b) Add the product details:



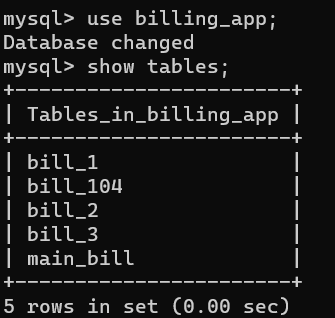
v) To Modify the Bill:

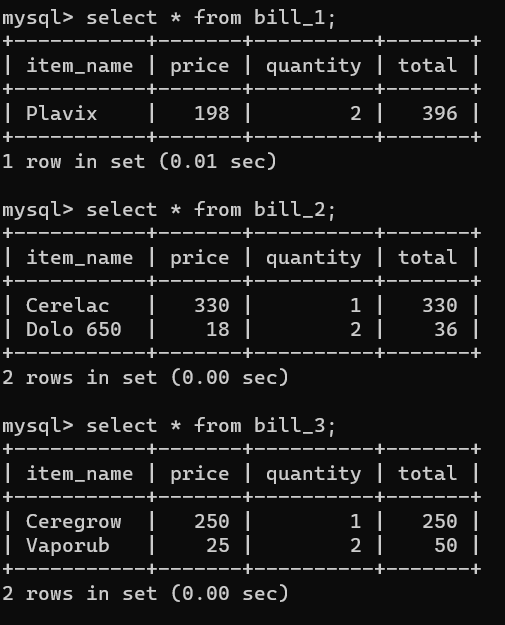


vi) To delete the bill:



DATABASE OUTPUT SCREENSHOTS:





**BIBLIOGRAPHY:**

* John Elder - (codemy.com)
* W3 Schools - (w3schools.com)
* Stack Overflow - (stackoverflow.com)
* Geeks For Geeks - (geeksforgeeks.org)

**CONCLUSION:**

The success of your revenue business relies on having the right billing system. A system meets your unique requirements and simplifies the complexities inherent to recurring revenue business models. This Pharmacy Billing System gives a User, complete Billing Management and gives a nice User Experience.

**We would like to take this opportunity to thank our teachers for their guidance and letting us accomplish this feat. We also learned the values that other members taught us in this process. Thank you!!**

