

# **MILK AND DAIRY PRODUCTS MANAGEMENT SYSTEM**



## **A PROJECT REPORT**

*Submitted by*

R.Dharshini (9517202109014)

B.Sandhiya Devi (9517202109046)

S.T.Shri Subiksha (9517202109048)

*in partial fulfillment for the award of the degree*

*of*

**Artificial Intelligence and Data Science**

*IN*

**Bachelor of Technology**

**MEPCO SCHLENK ENGINEERING COLLEGE,  
SIVAKASI-626123  
(An Autonomous Institution)**

**ANNA UNIVERSITY: CHENNAI 600 025**

**DECEMBER 2022**

**ANNA UNIVERSITY: CHENNAI 600 025**  
**BONAFIDE CERTIFICATE**

Certified that this project report “**MILK AND DAIRY MANAGEMENT SYSTEM**” is the bonafide work of “**R. Dharshini(21BAD008), B.Sandhiya Devi(21BAD022), S.T.Shri Subiksha(21BAD024)**”who carried out the project work under my supervision.

**SIGNATURE**

**Dr. A.Shenbagarajan, M.E.,Ph.D**

**Assistant Professor (SG)**

Artificial Intelligence and Data Science

Mepco Schlenk Engineering College

Sivakasi -626005

Virudhunagar District.

**SIGNATURE**

**Dr.J.Angela Jennifa Sujana,M.E.,Ph.D**

**Associate Professor (SG) & Head**

Artificial Intelligence and Data Science

Mepco Schlenk Engineering College

Sivakasi -626005

Virudhunagar District.

**ANNA UNIVERSITY: CHENNAI 600 025**  
**BONAFIDE CERTIFICATE**

Certified that this project report “**MILK AND DAIRY MANAGEMENT SYSTEM**” is the bonafide work of “**R. Dharshini(21BAD008), B.Sandhiya Devi(21BAD022), S.T.Shri Subiksha(21BAD024)**”who carried out the project work under my supervision.

**SIGNATURE**

**Mrs.L.Prasika, M.E.,(Ph.D)**

**Assistant Professor**

Artificial Intelligence and Data Science

Mepco Schlenk Engineering College

Sivakasi -626005

Virudhunagar District.

**SIGNATURE**

**Dr.J.Angela Jennifa Sujana,M.E.,Ph.D**

**Associate Professor (SG) & Head**

Artificial Intelligence and Data Science

Mepco Schlenk Engineering College

Sivakasi -626005

Virudhunagar District.

## ACKNOWLEDGEMENT

First and foremost we **praise and thank “The Almighty”**, the lord of all creations, who by his abundant grace has sustained us and helped us to work on this project successfully.

We really find unique pleasure and immense gratitude in thanking our respected management members, who is the backbone of our college.

A deep bouquet of thanks to respected Principal **Dr.S.Arivazhagan M.E.,Ph.D.**, for having provided the facilities required for our mini project.

We sincerely thank our Head of the Department **Dr. J. Angela Jennifa Sujana M.E.,Ph.D.**, Associate Professor(SG) & Head, Department of Artificial Intelligence and Data Science, for her guidance and support throughout the mini project .

We also thank our guide **Dr.A.Shenbagarajan.,M.E.,Ph.D.**, Assistant Professor(SG), **Mrs.L.Prasika., M.E(Ph.D)**, Assistant Professor Department of Artificial Intelligence and Data Science for their valuable guidance and it is great privilege to express our gratitude to them.

We extremely thank our project coordinator **Dr.A.Shenbagarajan.,M.E.,Ph.D.**, Assistant Professor(SG), **Mrs.L.Prasika., M.E(Ph.D)**, Assistant Professor Department of Artificial Intelligence and Data Science, who inspired us and supported us throughout the mini project.

We extend our heartfelt thanks and profound gratitude to all the faculty members of Artificial Intelligence and Data Science department for their kind help during our mini project work.

We also thank our parents and our friends who had been providing us with constant support during the course of the mini project work.

## TABLE OF CONTENTS

CHAPTER NO	TITLE	PAGE NO.
	ABSTRACT	7
	LIST OF FIGURES	8
	LIST OF TABLES	11
1	INTRODUCTION	12
	1.1 Overview	12
	1.2 Objective	13
2	SYSTEM REQUIREMENTS	14
	2.1 Hardware Requirements	14
	2.2 Software Requirements	14
3	FRONT END DESCRIPTION	15
	3.1 Introduction	15
	3.2 Connectivity	14
	3.2.1 PyMySQL	16
	3.2.2 Connecting to MySQL	16
	3.3 Graphical User Interface	18
	3.3.1 Tkinter module	18
	3.3.1.1 Tkinter Widget	20
	3.3.1.2 Geometry Management	24
	3.3.1.3 Standard attributes	26
	3.3.2 Pillow Library	26
4	DATABASE DESIGN	29
	4.1 Introduction	29
	4.1.1 MySQL database	29
	4.1.2 Need of database	29
	4.1.3 Need for valid data in database	30
	4.2 Schema	31
	4.3 ER Diagram	35

	4.3.1 ERD Plus	36
	4.3.2 Relationship	36
	4.4 Schema Diagram	37
	4.5 Normalization	38
	4.6 Database Creation	43
	4.6.1 Table Creation	44
	4.6.2 Source Code	45
	4.7 Database Queries	50
5	IMPLEMENTATION	57
	5.1 Flow Diagram	57
	5.2 Source Code	59
6	RESULTS	113
7	CONCLUSION	119
	REFERENCES	120

## **ABSTRACT**

The Dairy management system is developed taking into considerations the failures of the current system which is done manually and aims at converting the failures to success. The system includes number of quantities of dairy products for each day, the livestock system which provides the raw-materials, the administrator details who maintains the system, the user details who buy the dairy products and worker details who works in the system for maintenance and effective diagnosis of disease for the livestock. The system aids in automating the manual work of writing by the administrator and all the related work, by integrating all the work through a centralized computer. This helps in eliminating all the manual work of recording all the data in the dairy farm by the administrator to a large extent hence a timely and speedier recording of information can be done by the administrator. The system also aims at computerizing the process of selling the dairy farm products e. g. milk, butter etc. so that it is done online. This helps in eliminating the process of manual calculation for billing up the products cost and the updation of products to display to the user who buys the products. The Dairy Management System allows easy access to key information on each worker in the farm, or the whole worker thereby fostering an increase in efficiency. The Dairy Management Software was developed in consultation with dairy customers (users), owner(admin) and workers.

## LIST OF FIGURES

Fig No	Description	Pg No
3.1	Connection to MySQL database	16
3.2	Creating cursor object	17
3.3	Illustrating the use of execute() method	18
3.4	Closing the connection to the MySQL database	18
3.5	Tkinter module for creating GUI	19
3.6	Importing Tkinter modules in the source code	19
3.7	Main window creation code	20
3.8	Login options page	20
3.9	Tree View widget Code	23
3.10	Frame created with tree widget	23
3.11	Illustrating the use of pack() method	24
3.12	Illustrating the use of grid() method	24
3.13	Illustrating the use of place() method	25
3.14	Illustrating the use of font attribute	25
3.15	Illustrating the use of Relief Styles	26
3.16	Window created with geometry management and use of standard attributes.	26
3.17	Importing PIL library	27
3.18	Illustrating opening an image using open() method	27



3.19	Background image using PIL library	27
4.1	Need for database	30
4.2	Tables in Milk-Management	34
4.3	ER diagram	35
4.4	Schema Diagram	37
4.5	Milk Management Database	43
4.6	Table creation sample	44
4.7	Table description in Milk Management database	49
4.8	Insert query sample code	51
4.9	Inserted data records sample in database	52
4.10	Update query sample code	52
4.11	Delete query sample code	53
4.12	Select query sample code	54
4.13	Join query sample code	55
5.1	Flow Diagram	57
6.1	Login page options	113
6.2	Admin login window	114
6.3	Admin window	114
6.4	Tree view in admin window	115
6.5	User login page	115
6.6	User window	116

6.7	Bill	116
6.8	Worker login page	117
6.9	Worker window	117

## **LIST OF TABLES**

Table no	Description	Page No
3.1	Tkinter Widgets	22

# **CHAPTER 1**

## **INTRODUCTION**

### **1.1 OVERVIEW**

To develop a Milk and Dairy Management system for the aid of Milk and Dairy product sellers and customers. The Milk and Dairy Management system will provide a best user interface to maintain the admin, user, product details. Just by filing details to the form provided by this aid, the system will be able to store all details of the milk and dairy products, the admin details, the user details, the worker details and when a worker, admin, user wants to leave the job, the entire details of the particular input given can be removed from the available database. And the admin, user, product details can be fetched easily just by clicking the view details button. This system will also help in monitoring and calculating the bill of the products bought by the customer and the product generated. This interface helps in providing a better application and gives a overall good system to maintain the details about the people involved in the milk and dairy management system and also reduces the manual work of seller by generating bill and updating the product status.

## **1.2 OBJECTIVES**

- Automate the milk collection and pricing system of the Dairy Co-operatives.
- Unify the accounting and management system of the Dairy Co-operatives.
- Creating a network between admins, users and workers for efficient and effective management.
- Increase Rural Employment Opportunities through Entrepreneurship.
- Unify the accounting system of the Dairy Co-operatives.
- Profit maximization from dairy sector.
- Socio-economic stability of dairy farmers of the state.

## **CHAPTER 2**

### **SYSTEM REQUIREMENTS**

#### **2.1 HARDWARE REQUIREMENTS**

- AMD Ryzen 5 5625U Processor
- 16 GB RAM
- Mouse
- Keyboard

#### **2.2 SOFTWARE REQUIREMENTS**

- OS – Windows 11
- Python Interpreter
- Visual Studio Code
- MySQL 8.0 Command Line Client
- MySQL Database Connector

## **CHAPTER 3**

### **FRONT END DESCRIPTION**

#### **3.1 INTRODUCTION**

Python is a widely used general-purpose, high level programming language. It was initially designed by Guido van Rossum in 1991 and developed by Python Software Foundation. It was mainly developed for emphasis on code readability, and its syntax allows programmers to express concepts in fewer lines of code.

Python is a programming language that lets you work quickly and integrate systems more efficiently. Python supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is often described as a “batteries included” language due to its comprehensive standard library. Such standard libraries and modules in python are used to develop Graphical user interface in this project.

#### **3.2 CONNECTIVITY**

In order to connect Python to a database, a driver is needed, which is a library used to interact with the database. For MySQL database 3 driver choices are available.

- MySQL/connector
- MySQLdb
- PyMySQL

In this project PyMySQL is used to interact with MySql database.

### 3.2.1 PyMySQL

PyMySQL is a library that connects to MySQL from Python and it's a pure python library. PyMySQL is an opensource project. The goal of PyMySQL is to be a drop-in replacement for MySQLdb. This module does not come built-in with python, You have to install it externally. To install the module 'pip install PyMySQL' command is used.

### 3.2.2 CONNECTING TO MySQL

To connect to MySQL import PyMySQL library in your python program. The proper way to get an instance of the class is to call connect() method. This method establishes a connection to the MySQL database and accepts several arguments. The syntax of the connect() method is

```
<connection_object>=pymysql.connect(host="",user="",password="",  
database="")
```

The parameters of the method are host, user, password, database, port.

- host – Host where the database server is located
- user – Username to log in as
- password – Password to log in
- database – Database to use
- port – MySQL port to use (default: 3306)

```
import pymysql  
##### creating connection to mysql database #####  
con=pymysql.connect(host="localhost",  
                    user="root",  
                    password="dharshini1109",  
                    database="MilkManagement")  
##### creating cursor object #####
```

Fig 3.1: Connection to MySQL database



## Cursor Object

A cursor is an object which helps to execute the query and fetch the records from the database. This object is created after giving connection to the database. The cursor object is created by cursor() method.

```
##### creating cursor object using con object #####
cursor=con.cursor()
```

Fig 3.2: Creating cursor object

## The execute() method

The execute() method helps to execute the query and return records according to the query. The syntax of the execute function is

<cursor\_object>.execute(query,args=None)

- query – Query should be a string type.
- Arguments – By default,the arguments are None because sometimes we can pass only a query like SELECT query which fetches the records and does not require any values. But if values are needed to be passed in case of INSERT query, the arguments must be a tuple, list, or dict only.

This method returns the count of the number of rows affected during the query. The return type will be an integer.

## The fetchall() method

The method fetches all rows of a query result set and returns a list of tuples. If no more rows are available, it returns an empty list. The syntax of the fetchall() method is

<variable\_name>=<cursor\_object>.fetchall()

```
##### selecting admin password from database to check the password validity #####
cursor.execute("SELECT admin_password FROM Admin WHERE Admin_username='"+username_entry.get()+"'")
rows=cursor.fetchall()
cursor.execute("commit")
for i in rows:
```

Fig 3.3: Illustrating the use of execute() method

To disconnect Database connection close() method is used. This method tries to send a quit command and close the socket. It raises no exceptions. It is important to commit the transactions before closing the connection to the database. The syntax of the method is

<connection\_object>.close()

```
544 ##### closing the connection to the database #####
545 con.close()
```

Fig 3.4: Closing the connection to the MySQL database

### 3.3 Graphical User Interface

A Graphical user interface is an interface through which a user interacts with electronic devices via visual indicator representations. Graphical user interfaces would become the standard of user-centered design in software application programming, providing users the capability to intuitively operate computers and other electronic devices through the direct manipulation of graphical icons such as buttons, scroll bars, windows, tabs, menus, cursor,.. etc. Graphical user interface can be designed using python language. Tkinter is the standard GUI library for python.

#### 3.3.1 Tkinter module

The Tkinter module in python allows to build GUI applications in Python. Tkinter provides a variety of common GUI elements that can be used to build interfaces. These elements include buttons, menus, and various kinds of entry fields and display area. Tkinter provides an object-oriented interface to the Tk

GUI toolkit. Other available GUI development platforms include wPython and JPython.



Fig 3.5: Tkinter module for creating GUI

Compared to other available options, Tkinter is lightweight and easy to use. Therefore, it is the go-to the choice for quickly building applications that can operate cross-platform and do not require a modern look.

### Importing Tkinter and creating GUI in Python source code

```
1  from tkinter import *
2  import tkinter.font as f
3  from tkinter import ttk
4  from tkinter import messagebox
```

Fig 3.6: Importing Tkinter modules in the source code

Creating a GUI application using Tkinter is an easy task. The steps to create GUI application are

- Import the Tkinter module.
- Create the GUI application main window.
- Add one or more of the above-mentioned widgets to the GUI application as desired by the user.
- Enter the main event loop to take action against each event triggered by the user.

```

##### main window #####
root=Tk()
root.title("Login Page")
root.minsize(640,400)
##### setting background image #####
bg_frame=Image.open("C:\\Users\\maha9\\OneDrive\\Documents\\Dbms&py_miniproj\\bg1.jpg")
photo=ImageTk.PhotoImage(bg_frame)
bg_panel=Label(root,image=photo)
bg_panel.image=photo
bg_panel.pack(fill='both',expand='yes')
##### giving a heading for a login frame #####
txt=' DAIRY PRODUCTS MANAGEMENT SYSTEM'
heading=Label(bg_panel,text=txt,font=('Helvetica',20,'italic'))
heading.place(x=0,y=10,width=650,height=30)
##### creating admin login button #####
btn_font=f.Font(family='Helvetica',size=16,weight='normal',slant='roman')
btn1=Button(root,text='Admin Login',bd='5',command=openAdminLogin,font=btn_font)
btn1.place(x=250,y=75)
##### creating user login button #####
btn2=Button(root,text='User Login ',bd='5',command=openUserLogin,font=btn_font)
btn2.place(x=250,y=175)
##### creating worker login button #####
btn3=Button(root,text='Worker Login',bd='5',command=openWorkerLogin,font=btn_font)
btn3.place(x=250,y=275)
#####
root.mainloop()
##### closing the connection to the database #####
con.close()

```

Fig 3.7: Main window creation code

The above code would create a following window-



Fig 3.8: Login options page

### 3.3.1.1 Tkinter Widgets

Tkinter provides various controls, such as buttons, labels and text boxes used in a GUI application. These controls are commonly called widgets. There

are currently 19 types of widgets in Tkinter. The widgets are explained briefly in the below table.

S.No	Widget	Description
1	Button	The button widget is used to display buttons in the application.
2	Canvas	The Canvas widget is used to draw shapes, such as lines, ovals, polygons and rectangles, in the application.
3	Checkbutton	The Checkbutton widget is used to display a number of options as checkboxes. The user can select multiple options at a time.
4	Entry	The Entry widget is used to display a single-line text field for accepting values from a user.
5	Frame	The Frame widget is used as a container widget to organize other widgets.
6	Label	The Label widget is used to provide a single-line Caption for other widgets. It can also contain images.
7	Listbox	The Listbox widget is used to provide a list of options to a user.
8	Menubutton	The Menubutton widget is used to display menus in the application.
9	Menu	The Menu widget is used to provide various commands to a user. These commands are contained inside Menubutton.
10	Message	The Message widget is used to display multiline text fields for accepting values from the user.

11	Radiobutton	The Radiobutton widget is used to display a number of options as radio buttons. The user can select only one option at a time.
12	Scale	The Scale widget is used to provide a slider widget.
13	Scrollbar	The Scrollbar widget is used to add scrolling capability to various widgets, such as list boxes.
14	Text	The Text widget is used to display text in multiple lines.
15	Toplevel	The Toplevel widget is used to provide a separate window container.
16	Spinbox	The Spinbox widget is a variant of the standard Tkinter Entry widget, which can be used to select from a fixed number of values.
17	PanedWindow	A PanedWindow is a container widget that may contain any number of panes, arranged horizontally or vertically.
18	LabelFrame	A labelframe is a simple container widget. Its primary purpose is to act as a spacer or container for complex window layouts.
19	tkMessageBox	This module is used to display message boxes in your applications.

Table 3.1: Tkinter Widgets

Some of the above-mentioned widgets are used in the project to create a GUI application.

## TREEVIEW WIDGET IN Tkinter

Tkinter Treeview widget is used to display the data in a hierarchical structure. A Treeview widget holds a list of items. Each item has one or more columns. In short, Tkinter Treeview is a tabular representation of the data as it has all the properties of the table. Treeview widget can be created using `Treeview(parent, columns)` constructor to build the table. Treeview widget is also used in this project to display the database records.

```
def ViewLivestockDiagnosisDetailsFrame():  
    ##### creating frame #####  
    ViewLivestock=Frame(workerwindow,width=1030,height=475,bg="#3B3B3B",bd=10,relief=RIDGE)  
    ViewLivestock.place(x=225,y=300)  
    ##### instance of style widget #####  
    style = ttk.Style()  
    style.theme_use('default')  
    ##### creating tree frame to display livestock diagnosis #####  
    tree =ttk.Treeview(ViewLivestock, column=("c1", "c2","c3","c4","c5"), show='headings', height=20)  
    ##### aligning columns #####  
    tree.column("# 1", anchor=CENTER)  
    tree.heading("# 1", text="Livestock ID")  
    tree.column("# 2", anchor=CENTER)  
    tree.heading("# 2", text="Disease")  
    tree.column("# 3", anchor=CENTER)  
    tree.heading("# 3", text="DiseaseStatus")  
    tree.column("# 4", anchor=CENTER)  
    tree.heading("# 4", text="VaccineStatus")  
    tree.column("# 5", anchor=CENTER)  
    tree.heading("# 5", text="Expenses")  
    tree.place(x=5,y=0)  
    cursor.execute("SELECT * FROM LiveStockDiagnosis")  
    res=cursor.fetchall()  
    for i in res:  
        tree.insert(parent='',index='end',values=(i[1],i[2],i[3],i[5],i[4]))  
    cursor.execute("commit")
```

Fig 3.9: Tree View widget Code

The above code would create the following window-

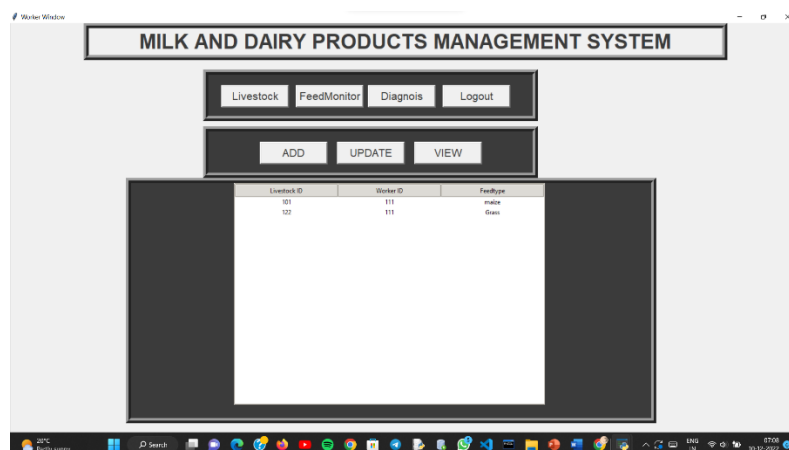


Fig 3.10: Frame created with tree widget

### 3.3.1.2 Geometry Management

All Tkinter widgets have access to specific geometry management methods, which have the purpose of organizing widgets throughout the parent widget area. Tkinter exposes the following manager classes:

- The pack() method
- The grid() method
- The place() method

#### The pack() method

The pack() method is one of the geometry manager organizes widgets in blocks before placing them in the parent widget. This method is also used in the project to create good looking GUI.

```
workerwindow.state( 'zoomed' )
#####Top frame that holds the heading of the admin window #####
topFrame=Frame(workerwindow,bd=10,relief=RIDGE,bg='#3B3B3B')#Gray23
topFrame.pack(side=TOP)
```

Fig 3.11: Illustrating the use of pack() method

#### The grid() method

The grid() method is one of the geometry manager organizes widgets in a table-like structure in the parent widget. This method is also used in the project to create good looking GUI.

```
topFrame.pack(side=TOP)
##### adding label to the top frame created #####
labelTitle=Label(topFrame,text='MILK AND DAIRY PRODUCTS MANAGEMENT SYSTEM')
labelTitle.grid(row=0,column=0)
```

Fig 3.12: Illustrating the use of grid() method



## The place() method

The grid() method is one of the geometry manager organizes widgets by placing them in a specific position in the parent widget. This method is also used in the project to create good looking GUI.

```
##### creating label for user convience #####
txt='Add LiveStock Details'
heading=Label(add_buttonFrame,text=txt,font=('Helvetica',
heading.place(x=55,y=20,width=500,height=30)
```

Fig 3.13: Illustrating the use of place() method

### 3.3.1.3 Standard attributes

Some the standard attributes in python such as size, colour and fonts are used to project the GUI in a more beautiful and in an enhanced manner. Some of such attributes are listed below,

- Dimensions
- Colours
- Fonts
- Anchors
- Relief styles
- Bitmaps
- Cursors

These attributes are also used in the project to create more attractive GUI.

```
openAdminLogin(),
btn_font=f.Font(family='Helvetica',size=16,weight='normal',slant='roman')
##### creating label for user convience #####
```

Fig 3.14: Illustrating the use of font attribute

```
##### creating a frame to add buttons #####
buttonFrame=Frame(adminwindow,bd=10,bg='lavender',width=1500,height=100,relief=RIDGE)
buttonFrame.place(x=25,y=90)
##### creating frames for the buttons correspondingly #####
```

Fig 3.15: Illustrating the use of Relief Styles

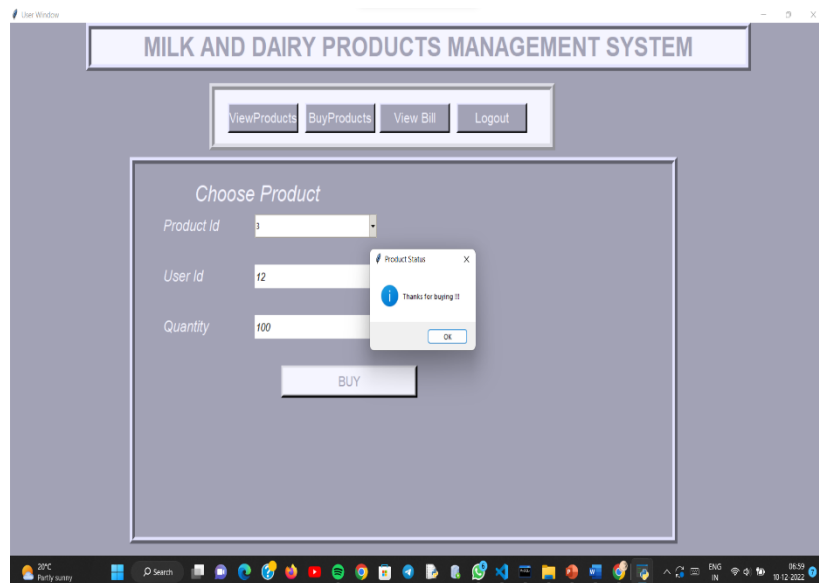


Fig 3.16: Window created with geometry management and use of standard attributes

### 3.3.2 PILLOW LIBRARY

Python Imaging Library (expansion of PIL) is the de facto image processing package for python language. It incorporates lightweight image processing tools that aids in editing, creating and saving images. Support for python imaging library got discontinued in 2011, but a project named pillow forked the original PIL project and added Python 3.x support to it. Pillow was announced as a replacement for PIL for future usage. Pillow supports a large number of image file formats including BMP, PNG, JPEG, and TIFF. The library encourages adding support for newer formats in the library by creating new file decoders.

This module is not preloaded with Python. To install ‘pip install pillow’ command should be executed. The GUI application can be made more interactive by placing an image in the background according to our desire.

```
from tkinter import messagebox
from PIL import ImageTk, Image #pip install pillow
import pymysql
```

Fig 3.17: Importing PIL library

## Opening an image using open()

The PIL.Image.Image class represents the image object. This class provides the open() method that is used to open image.

```
root.minSize(640,400)
##### setting background image #####
bg_frame=Image.open("C:\\Users\\maha9\\OneDrive\\Documents\\Dbms&py_miniproj\\bg1.jpg")
photo=ImageTk.PhotoImage(bg_frame)
bg_panel=Label(root,image=photo)
bg_panel.image=photo
bg_panel.pack(fill='both',expand='yes')
```

Fig 3.18: Illustrating opening an image using open() method

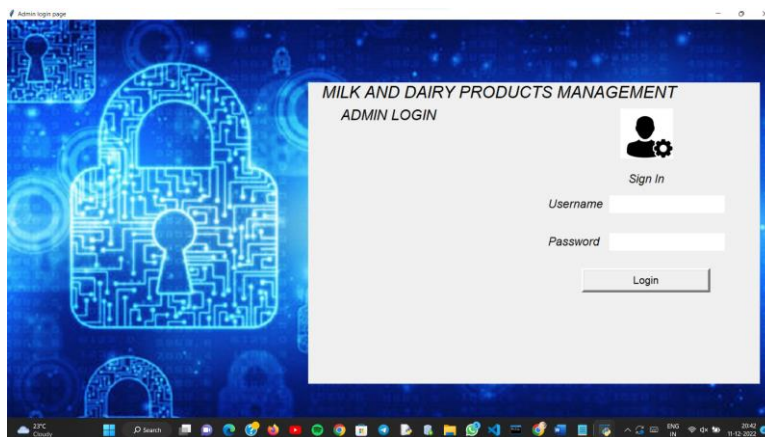


Fig 3.19: Background image using the PIL library

Using these facilities provided by Tkinter enhances the GUI application to look more attractive.

Some advantages of Tkinter module are

- Layered approach

- Accessibility
- Portability
- Availability

Hence Tkinter very easy for learning, it can be used to create a more attractive and interactive GUI application and so GUI applications for this project are designed with help Tkinter module.

## **CHAPTER 4**

### **DATABASE DESIGN**

#### **4.1 INTRODUCTION**

A database is an organized collection of structured information, or data, typically stored electronically in a computer system. A database is usually controlled by a database management system. Together the data and the DBMS, along with the applications that are associated with them, are referred to as a database system, often shortened to just database.

##### **4.1.1 MySQL Database**

MySQL is an opensource relational database management system based on SQL. It was designed and optimized for web applications and can run on any platform. As new and different requirements emerged with the internet, MySQL became the platform of choice for web developers and web-based applications. Because it's designed to process millions of queries and thousands of transactions, MySQL is a popular choice for ecommerce businesses that need to manage multiple money transfers. On-demand flexibility is the primary feature of MySQL.

MySQL is the DBMS behind some of the top websites and web-based applications in the world, including Airbnb, Uber, LinkedIn, Facebook, Twitter, and YouTube.

##### **4.1.2 Need for Database**

A good database is crucial to any company or organisation. This is because the database stores all the pertinent details about the company such as employee records, transactional records, salary details etc.

Databases let us work with large amounts of data efficiently. Some notable advantages are

- Minimum data redundancy
- Improved data security
- Increased consistency
- Reduced cost of data entry, data storage, data retrieval
- Higher data integrity from application programs

The below diagram describes the essential need for a database.

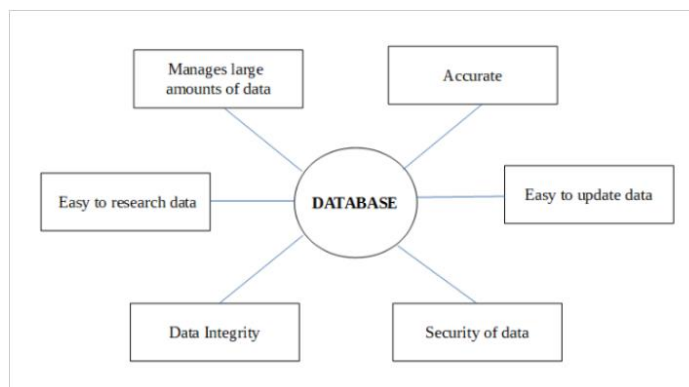


Fig 4.1: Need for database

### 4.1.3 Need for valid data in database

Data validation is crucial for every business as it ensures that you can completely trust the data, they use to be accurate, clean and helpful at all times. Making sure the data you use is correct is a proactive way to safeguard the demand-generating assets. So, it is very important to ensure the validity of the data.

## 4.2 Schema

The term “schema “ refers to the organization of data as a blueprint of how the data base is constructed. In brief schema is the definition for the entire database, so it includes tables, views, stored procedures, indexes, primary and foreign key, etc.

In this project to store the details of admin, users, workers, products and livestock certain tables are designed in the database. Table is a data set in which the data is organized in to a set of vertical columns and horizontal rows. Tables are also called as relations. The list of relations used in our project is

- Admin
- User
- Worker
- Product
- Bill
- LiveStock
- LiveStock\_Diagnosis
- Feed Monitoring

## Table Design

A properly designed table in database provides easy access to update, accurate information. A correct design of table is essential to achieve goals in working with database. The design of our tables for our database in the project is explained clearly below

### RELATION -1: Admin

Attributes
<u>Admin_id</u> (Primary Key)
Username
Password

### RELATION -2: Product

Attributes
<u>Product_id</u> (PrimaryKey)
Product_name
Quantity_avlb
Price

### RELATION -3: Livestock

Attributes
<u>Livestock_id</u> (Primary Key)
Livestock_type
Capacity_of_milk



#### RELATION -4:Livestock\_diagnosis

Attributes
<u>Worker_id</u> (Composite Key,ForeignKey)
<u>Livestock_id</u> (Composite Key,ForeignKey)
Disease
Disease_status
Expenses
Vaccine_status

#### RELATION -5:\_Worker

Attributes
<u>Worker_id</u> (Primary Key)
Worker_name
Worker_type
Salary
Mail_id
Address(Multi-Valued Attribute)
Password
Phn_num

#### RELATION -6: User

Attributes
<u>User_id</u> (Primary Key)
User_name
Password
Phn_num

## RELATION -7: **Bill**

Attributes
<u>Product_id</u> (Composite Key,Foreign Key)
<u>User_id</u> (Composite Key,Foreign Key)
Quantity_saled
Date

## RELATION -8: **Feed\_monitoring**

Attributes
<u>Worker_id</u> (Composite Key,Foreign Key)
<u>Livestock_id</u> (Composite Key,Foreign Key)
Feed_type

```
mysql> USE MilkManagement;
Database changed
mysql> SHOW TABLES;
+-----+
| Tables_in_milkmanagement |
+-----+
| admin                     |
| bill                      |
| feed_monitoring           |
| livestock                 |
| livestockdiagonsis        |
| product                   |
| user                      |
| worker                    |
+-----+
8 rows in set (0.00 sec)
```

Fig 4.2: Tables in milk management

### 4.3 ER Diagram

ER Diagram stands for Entity Relationship Diagram, displays the relationship of entity sets stored in the database. ER Diagram helps to explain the logical structure of databases.

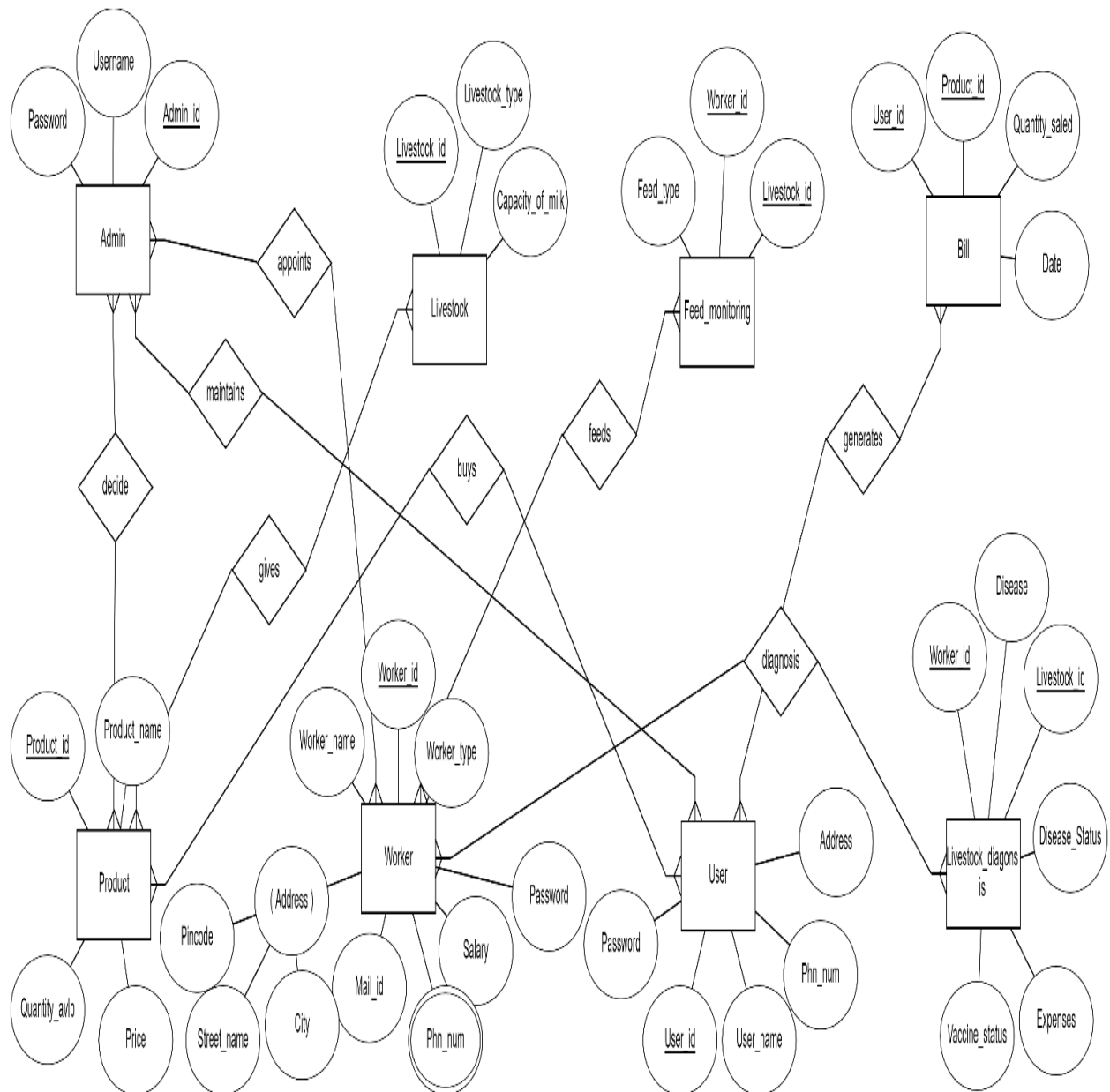


Fig 4.3: ER Diagram

### **4.3.1.ERD Plus**

ERD Plus is a web-based database modelling tool for creating entity relationship diagrams, relational schemas, star schemas, and SQL DDL statements

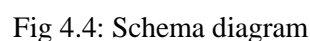
### **4.3.2.RELATIONSHIPS**

The relationship of entities in our design are as follows

**(Many to Many)**

- Decide (Admin → Product)
- Maintains (Admin → User)
- Buys (User → Product)
- Gives (Livestock → Product)
- Feeds (Worker → Feed\_monitoring)
- Appoints (Admin → Worker)
- Generates (Worker → Bill)
- Diagnosis (Worker → Livestock\_diagnosis)

A schema diagram is a diagram which contains entities and the attributes that will define that schema. A schema diagram only shows us the database design. It does not show the actual data of database. Schema can be a single table or it can have more than one table which is related. The schema diagram of our database designed is below



## 4.5 NORMALIZATION

Normalization is a database design technique that reduces data redundancy and eliminates undesirable characteristics like Insertion, Update and Deletion Anomalies. Normalization rules divides larger tables into smaller tables and links them using relationships.

The purpose of Normalisation in SQL is to eliminate redundant (repetitive) data and ensure data is stored logically. The inventor of the relational model Edgar Codd proposed the theory of normalization of data with the introduction of the First Normal Form, and he continued to extend theory with Second and Third Normal Form. Later he joined Raymond F. Boyce to develop the theory of Boyce-Codd Normal Form. Here the steps we done in normalizing our tables for our database are explained clearly.

TABLE – 1 ADMIN

<u>Admin_id</u>	Admin_username	Admin_password
-----------------	----------------	----------------

- The table satisfies 1 NF as there are no multi-values.
- The table satisfies 2 NF as the Admin\_id is the only primary key in the table and hence there is no proper set of prime attributes and no way for the presence of partial functional dependencies.
- Transitive Functional Dependency  
    { Admin\_id  $\rightarrow$  Admin\_username }  
    { Admin\_id  $\rightarrow$  Admin\_password }  
    { Admin\_username  $\rightarrow$  Admin\_password }  
     $\rightarrow$  {Non-prime attribute  $\rightarrow$  Non-prime attribute }
- Hence the admin\_username is altered to be unique and so the table statisfies 3 NF.
- Admin\_id is the Super key, Hence the table satisfies BCNF.

- The table satisfies 4 NF as there no multivalued data describing a specific entity.

TABLE – 2 USER

Initial Table

User_id	User_name	password	Phn_num	address	Product_id
---------	-----------	----------	---------	---------	------------

- The table satisfies 1 NF as there are no multi-values.
- The table satisfies 2 NF as the User\_id is the only primary key in the table and hence there is no proper set of prime attributes and no way for the presence of partial functional dependencies.
- Transitive Functional Dependency
  - { User\_id  $\rightarrow$  User\_name }
  - { User\_id  $\rightarrow$  password }
  - { user\_name  $\rightarrow$  password }
  - $\rightarrow$  {Non-prime attribute  $\rightarrow$  Non-prime attribute }
- Hence the user\_name is altered to be unique and so the table satisfies 3 NF.
- User\_id is the Super key, Hence the table satisfies BCNF.
- Independent entities
  - Password
  - User\_name
  - Address
  - Phn num
- Product\_id repeats again and again for the same user\_id if the user buys more than one products leads to redundancy in data.
- Hence the tables are decomposed as follows  
USER (Final table )

<u>User_id</u>	password	User_name	Phn_num	address
----------------	----------	-----------	---------	---------

USER\_2

<u>User_id</u>	<u>Product_id</u>
----------------	-------------------

TABLE – 3 LIVESTOCK

<u>Livestock_id</u>	Livestock_type	Capacityof milk
---------------------	----------------	-----------------

- The table satisfies 1 NF as there are no multi-values.
- The table satisfies 2 NF as the Livstock\_id is the only primary key in the table and hence there is no proper set of prime attributes and no way for the presence of partial functional dependencies.
- { Livestock\_id  $\rightarrow$  Livestock\_type }
- { Livestock\_id  $\rightarrow$  Capacity\_of\_milk }
- No Transitive Functional Dependency, hence the table satisfies 3 NF.
- livestock\_id is the Super key, Hence the table satisfies BCNF.
- The table satisfies 4 NF as there no multivalued data describing a specific entity.

TABLE – 4 PRODUCT

Initail Table

<u>Product_id</u>	Product_name	Quantityavlb	Price	Quantitiesold	Date_sold
-------------------	--------------	--------------	-------	---------------	-----------

- The table satisfies 1 NF as there are no multi-values.
- Product\_id is the only prime attribute so there no possibilities for proper set of prime attributes and partial functional dependencies.



- { Product\_id → Product\_name }
- { Product\_id → Price }
- { Product\_id → Quantityavl }  
b
- { Product\_name → Quantityavl }
- No Transitive Functional Dependency, hence the table satisfies 3 NF.
- product\_id is the only Super key , Hence the table satisfies BCNF.
- Quantitiesold, datesold have repeated values. Hence the table will not satisfies 4 NF. So the table can be decomposed as below

New Product table (final)

<u>Product_id</u>	Product_name	Price	Quantityavl
-------------------	--------------	-------	-------------

- Quantitiesold and date sold are moved to USER-2 table and renamed as BILL table to save the details of the product sold

BILL

<u>Product_id</u>	<u>User_id</u>	Quantitiesold	Datesold
-------------------	----------------	---------------	----------

Here product\_id and user\_id are Composite primary key.

- Now the New Product table satisfies 4 NF.

TABLE - 5 WORKER

<u>Workerid</u>	name	password	worktype	salary	address	phnnum	livestockid
-----------------	------	----------	----------	--------	---------	--------	-------------

- Multi values are present in the table i.e one worker can maintain more than one Livestock
- Hence the tables are decomposed further to satisfy 1 NF. The decomposed tables are as follows

## WORKER – 1

<u>Workerid</u>	name	password	worktype	salary	Phnnum	address
-----------------	------	----------	----------	--------	--------	---------

## WORKER – 2

<u>Worker_id</u>	<u>Livestock_id</u>
------------------	---------------------

- In this table worker\_id and livestock\_id are composite primary key.
- Now the Worker – 1 table satisfies 1 NF.
- Worker\_id is the only primary key, so there is no chance for partial functional dependencies so as the proper set of prime attributes.
- {Worker\_id  $\rightarrow$  name}
- {Worker\_id  $\rightarrow$  password}
- {Worker\_id  $\rightarrow$  worktype}
- {Worker\_id  $\rightarrow$  salary}
- {name  $\rightarrow$  password}
- No transitive functional dependencies
- Hence the table satisfies 3 NF.
- Worker\_id is the only superkey. Hence it satisfies BCNF.
- No Multivalued attributes, hence the table satisfies 4 NF.
- In Milk and Dairy products management system LiveStock feedmonitoring is essential, for convenience Worker – 2 is renamed as FeedMonitoring and Feedtype attribute is added to the table.

### Feed Monitoring

<u>Worker_id</u>	<u>LiveStock_id</u>	Feedtype
------------------	---------------------	----------

- As monitoring of the feed of the livestock is important , Diagnosis and vaccine status should also be concerned and so the attributes such as disease, disease\_status, expenses and vaccine\_status are added and renamed as LiveStock diagnosis table.

## LivestockDiagnosis

<u>Livestock_id</u>	<u>Worker_id</u>	Disease	Disease_status	Expenses	Vaccine_status
---------------------	------------------	---------	----------------	----------	----------------

And thereby the conclusion is the tables are normalized in such a manner they satisfy 4 NF.

### 4.6 Database Creation

MySQL implements a database as a directory that stores all files in the form of a table. It allows us to create a database mainly in two ways

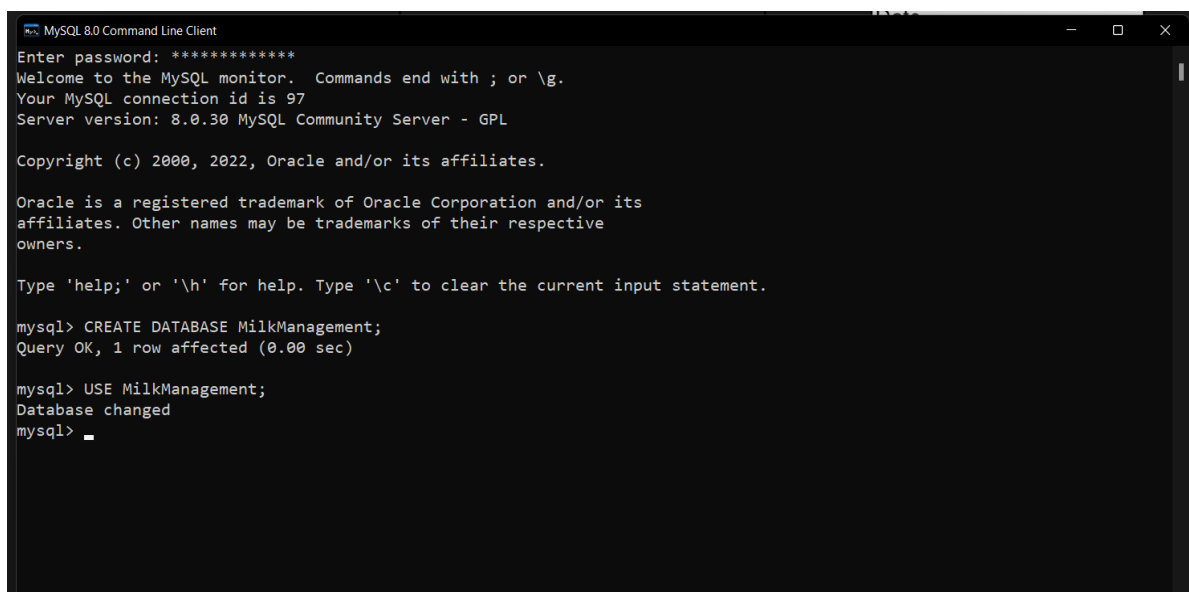
- MySQL Command Line Client
- MySQL Workbench

MySQL Command Line Client is used in the project. A new database can be created using create database statement. The syntax is

CREATE DATABASE <database name>;

The syntax to use the created database is

USE <database name>;



```
MySQL 8.0 Command Line Client
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 97
Server version: 8.0.30 MySQL Community Server - GPL

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> CREATE DATABASE MilkManagement;
Query OK, 1 row affected (0.00 sec)

mysql> USE MilkManagement;
Database changed
mysql> _
```

Fig 4.5: Milk Management database

### 4.6.1 Table Creation

The syntax to create table in the database is

```
CREATE TABLE <tablename>
( <column 1> datatype,
  <column 2> datatype,
  <column 3> datatype,...);
```

- The column parameters specify the names of the columns of the table.
- The datatype parameter specifies the type of data the column can hold (e.g. varchar, integer, date, etc.)
- Some constraints can also be included in creation of tables (e.g. NULL, DEFAULT, NOT NULL, etc.)
- A PRIMARY KEY is used to ensure data in the specific column is unique.
- A FOREIGN KEY is a column or group of columns in a relational database table that provides a link between data in two tables
- ON DELETE CASCADE and ON UPDATE CASCADE clause in MySQL is used to automatically remove and update the matching records from the child table when we delete and update the rows in the parent table.

```
mysql> CREATE TABLE LiveStockDiagnosis(
  -> Worker_id INT,
  -> LiveStock_id INT,
  -> disease VARCHAR(40),
  -> disease_status VARCHAR(40),
  -> expenses INT,
  -> Vaccine_status VARCHAR(40),
  -> PRIMARY KEY(Worker_id,LiveStock_id),
  -> FOREIGN KEY(Worker_id) REFERENCES Worker(Worker_id) ON DELETE CASCADE ON UPDATE CASCADE,
  -> FOREIGN KEY(LiveStock_id) REFERENCES LiveStock(LiveStock_id) ON DELETE CASCADE ON UPDATE CASCADE);
Query OK, 0 rows affected (0.03 sec)
```

Fig 4.6: Table creation sample

## 4.6.2 SOURCE CODE

--Creating MilkManagement Database

```
mysql> CREATE DATABASE MilkManagement;
```

Query OK, 1 row affected (0.00 sec)

```
mysql> USE MilkManagement;
```

Database changed

--Creating Table Admin

```
mysql> CREATE TABLE Admin(
```

-> Admin\_id INT PRIMARY KEY,

-> Admin\_Username VARCHAR(40) UNIQUE,

-> Admin\_Password VARCHAR(40));

Query OK, 0 rows affected (0.02 sec)

--Creating Table User

```
mysql> CREATE TABLE User(
```

-> User\_id INT PRIMARY KEY,

-> User\_name VARCHAR(40),

-> User\_Password VARCHAR(40),

-> Phn\_num NUMERIC(10) NOT NULL);

Query OK, 0 rows affected (0.01 sec)

```
mysql> ALTER TABLE User MODIFY COLUMN User_name VARCHAR(40) UNIQUE;
```

Query OK, 0 rows affected (0.07 sec)

Records: 0 Duplicates: 0 Warnings: 0

```
mysql> ALTER TABLE User ADD COLUMN address VARCHAR(50);
```

Query OK, 0 rows affected (0.06 sec)

Records: 0 Duplicates: 0 Warnings: 0

--Creating Table Worker

```
mysql> CREATE TABLE Worker(
```

```
-> Worker_id INT PRIMARY KEY,
```

```
-> Worker_name VARCHAR(40),
```

```
-> Worker_Password VARCHAR(40),
```

```
-> salary INT,
```

```
-> mail_id VARCHAR(100),
```

```
-> Worker_type VARCHAR(40),
```

```
-> address VARCHAR(60),
```

```
-> Phn_num NUMERIC(10));
```

Query OK, 0 rows affected (0.02 sec)

```
mysql> ALTER TABLE Worker MODIFY COLUMN Worker_name VARCHAR(40)
UNIQUE;
```

Query OK, 0 rows affected (0.03 sec)

Records: 0 Duplicates: 0 Warnings: 0

--Creating Table Product

```
mysql> CREATE TABLE Product(
```

```
-> Product_id INT PRIMARY KEY,
```

```
-> Product_name VARCHAR(20),
```

```
-> Quantity_avlb INT,
```

-> Price DECIMAL(10,2));

Query OK, 0 rows affected (0.02 sec)

--Creating Table LiveStock

mysql> CREATE TABLE LiveStock(

-> LiveStock\_id INT PRIMARY KEY,

-> LiveSTOCK\_type VARCHAR(50),

-> Capacity\_of\_Milk INT);

Query OK, 0 rows affected (0.01 sec)

--Creating Table Feed\_monitoring

mysql> CREATE TABLE Feed\_monitoring(

-> Worker\_id INT ,

-> LiveStock\_id INT,

-> Feed\_type VARCHAR(40),

-> PRIMARY KEY(Worker\_id,LiveStock\_id),

-> FOREIGN KEY(Worker\_id) REFERENCES Worker(Worker\_id) ON DELETE CASCADE ON UPDATE CASCADE,

-> FOREIGN KEY(LiveStock\_id) REFERENCES LiveStock(LiveStock\_id) ON DELETE CASCADE ON UPDATE CASCADE);

Query OK, 0 rows affected (0.03 sec)

--Creating Table LiveStockDiagnosis

mysql> CREATE TABLE LiveStockDiagnosis(

-> Worker\_id INT,

-> LiveStock\_id INT,

-> disease VARCHAR(40),

```

-> disease_status VARCHAR(40),

-> expenses INT,

-> Vaccine_status VARCHAR(40),

-> PRIMARY KEY(Worker_id,LiveStock_id),

-> FOREIGN KEY(Worker_id) REFERENCES Worker(Worker_id) ON DELETE
CASCADE ON UPDATE CASCADE,

-> FOREIGN KEY(LiveStock_id) REFERENCES LiveStock(LiveStock_id) ON DELETE
CASCADE ON UPDATE CASCADE);

```

Query OK, 0 rows affected (0.03 sec)

--Creating TABLE Bill

```
mysql> CREATE TABLE Bill(
```

```

-> Product_id INT,

-> User_id INT,

-> Quantity_saled INT ,

-> Date_sold DATE,

-> PRIMARY KEY(Product_id,User_id),

-> FOREIGN KEY(Product_id) REFERENCES Product(Product_id) ON DELETE
CASCADE ON UPDATE CASCADE,

-> FOREIGN KEY(User_id) REFERENCES User(User_id) ON DELETE CASCADE ON
UPDATE CASCADE);

```

Query OK, 0 rows affected (0.02 sec)

```
mysql> INSERT INTO Admin VALUES(1,'Sandhiya','Sandhiya@1708');
```

Query OK, 1 row affected (0.01 sec)

```
mysql> INSERT INTO Admin VALUES(2,'Shri','Shri@1705');
```

Query OK, 1 row affected (0.01 sec)



```
mysql> INSERT INTO Admin VALUES(3,'Dharshini','Dharshini@1109');
```

Query OK, 1 row affected (0.01 sec)

```
mysql> commit;
```

Query OK, 0 rows affected (0.00 sec)

```
mysql> DESC Admin;
+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+
| Admin_id | int | NO | PRI | NULL | |
| Admin_username | varchar(40) | YES | UNI | NULL | |
| Admin_password | varchar(40) | YES | | NULL | |
+-----+
3 rows in set (0.01 sec)

mysql> SELECT * FROM Admin;
+-----+
| Admin_id | Admin_username | Admin_password |
+-----+
| 1 | Sandhiya | Sandhiya@1708 |
| 2 | Shri | Shri@1705 |
| 3 | Dharshini | Dharshini@1109 |
+-----+
3 rows in set (0.00 sec)

mysql> DESC Product;
+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+
| Product_id | int | NO | PRI | NULL | |
| Product_name | varchar(20) | YES | | NULL | |
| Quantity_avlb | int | YES | | NULL | |
| Price | decimal(10,2) | YES | | NULL | |
+-----+
4 rows in set (0.00 sec)
```

```
mysql> DESC User;
+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+
| User_id | int | NO | PRI | NULL | |
| User_name | varchar(40) | YES | UNI | NULL | |
| User_password | varchar(40) | YES | | NULL | |
| Phn_num | decimal(10,0) | NO | | NULL | |
| address | varchar(50) | YES | | NULL | |
+-----+
5 rows in set (0.00 sec)

mysql> DESC Worker;
+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+
| Worker_id | int | NO | PRI | NULL | |
| Worker_name | varchar(40) | YES | UNI | NULL | |
| Worker_password | varchar(40) | YES | | NULL | |
| salary | int | YES | | NULL | |
| mail_id | varchar(100) | YES | | NULL | |
| Worker_type | varchar(40) | YES | | NULL | |
| address | varchar(60) | YES | | NULL | |
| Phn_num | decimal(10,0) | YES | | NULL | |
+-----+
8 rows in set (0.00 sec)
```

```
mysql> DESC Worker;
+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+
| Worker_id | int | NO | PRI | NULL | |
| Worker_name | varchar(40) | YES | UNI | NULL | |
| Worker_password | varchar(40) | YES | | NULL | |
| salary | int | YES | | NULL | |
| mail_id | varchar(100) | YES | | NULL | |
| Worker_type | varchar(40) | YES | | NULL | |
| address | varchar(60) | YES | | NULL | |
| Phn_num | decimal(10,0) | YES | | NULL | |
+-----+
8 rows in set (0.00 sec)

mysql> DESC LiveStock
-> ;
+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+
| LiveStock_id | int | NO | PRI | NULL | |
| LiveSTOCK_type | varchar(50) | YES | | NULL | |
| Capacity_of_Milk | int | YES | | NULL | |
+-----+
3 rows in set (0.00 sec)
```

```
mysql> DESC LiveStockdiagnosis;
+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+
| Worker_id | int | NO | PRI | NULL | |
| LiveStock_id | int | NO | PRI | NULL | |
| disease | varchar(40) | YES | | NULL | |
| disease_status | varchar(40) | YES | | NULL | |
| expenses | int | YES | | NULL | |
| Vaccine_status | varchar(40) | YES | | NULL | |
+-----+
6 rows in set (0.00 sec)

mysql> DESC feed_monitoring
-> ;
+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+
| Worker_id | int | NO | PRI | NULL | |
| LiveStock_id | int | NO | PRI | NULL | |
| Feed_type | varchar(40) | YES | | NULL | |
+-----+
3 rows in set (0.00 sec)

mysql> DESC Bill;
+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+
| Product_id | int | NO | PRI | NULL | |
| User_id | int | NO | PRI | NULL | |
+-----+
2 rows in set (0.00 sec)
```

Fig 4.7 Tables description in Milk Management database

## 4.7 Database Queries

In relational database management systems, a query is any command used to retrieve data from a table. MySQL can be used for querying the data, filtering data, storing data, joining the tables, grouping data, modifying data.

### MySQL Queries

- DDL (Data Definition Language)
- DML (Data Manipulation Language)
- DCL (Data Control Language)
- TCL (Transaction Control Language)

DDL and DML queries are used in the project to define and manipulate the database. DDL queries are used to create and modify the structure of the various objects within the database. The creation of database and tables are by the help of DDL queries which are already discussed above. Further, in this section we will discuss about DML queries and how they are used in our project.

### DML QUERIES

- INSERT
- UPDATE
- DELETE
- SELECT

### INSERT QUERY

INSERT query is used to insert the data records in the table. The syntax of the query is

```
INSERT INTO <table name> VALUES (<value1>,<value2>,,,<value n>);
```

```

def AddUserFrame():
    ##### creating frame #####
    add_user_Frame=Frame(adminwindow,width=1030,height=575,bg='#9898F5',bd=10,relief=RIDGE)
    add_user_Frame.place(x=225,y=200)

    ##### function to Insert user #####
    def adduser():
        ##### getting input from users #####
        user_id=user_id_entry.get()
        user_name=user_name_entry.get()
        phn_num=phn_num_entry.get()
        password=password_entry.get()
        address=address_entry.get()
        ##### checking constraints for password #####
        checknum=0
        checkalpha=0
        checklower=0
        checkcapital=0
        for i in password:
            if i.isnumeric():
                checknum=1
        for i in password:
            if i.isalpha():
                checkalpha=1
        for i in password:
            if i.islower():
                checklower=1
        for i in password:
            if i.isupper():
                checkcapital=1
        checkspl_char=0
        l=['@','!','#','$','%','^','&', '*', '_']
        for i in password:
            if i in l:
                checkspl_char=1
        ##### checking for the user to input all the fields #####
        if user_id=="" or user_name=="" or phn_num=="" or password=="":

```

Fig 4.8.1: INSERT query sample code

```

        if i in l:
            checkspl_char=1
        ##### checking for the user to input all the fields #####
        if user_id=="" or user_name=="" or phn_num=="" or password=="":
            messagebox.showinfo("Insert Status","All fields are required for adding a user")
        else:
            ##### checking for valid phone number #####
            if len(phn_num)!=10:
                messagebox.showinfo("Insert Status","Enter valid phone number")
            ##### checking for valid username #####
            elif (user_name.isalpha()==False):
                messagebox.showinfo("Insert Status","Enter valid username")
            ##### checking for strong password #####
            elif ((len(password)<8) or (checknum==0) or (checkspl_char==0) or (checkcapital==0) or (checkalpha==0) or (checklower==0)):
                messagebox.showinfo("Insert Status","Passwords must be atleast 8 characters,should contain atleast a special character,capital letter")
            ##### checking for valid user id #####
            elif user_id.isnumeric()==False:
                messagebox.showinfo("Insert Status","Enter valid user id")
            ##### Inserting if all values are correct #####
            else:
                cursor.execute("Insert INTO User VALUES('"+user_id+"','"+user_name+"','"+password+"','"+phn_num+"','"+address+"')")
                cursor.execute("commit")
                ##### clearing the value in the entry field for next user entry #####
                user_id_entry.delete(0,END)
                user_name_entry.delete(0,END)
                phn_num_entry.delete(0,END)
                password_entry.delete(0,END)
                address_entry.delete(0,END)
                messagebox.showinfo("Insert Status","User Added !!!")

```

Fig 4.8.2: INSERT query sample code

In the above figure, INSERT query is used to insert the data records in the table. In the figure certain constraints such as strong passwords, phone number validity, username validity..., are validated so as to ensure data records that are recorded in the database are valid. After the validation if all data is valid, then they are inserted into the database.

```
mysql> select * from user;
```

User_id	User_name	User_Password	Phn_num	address
12	Dharshi	Dfaa@1109	9894903888	sivakasi
101	Dharshini	Dharshini@1109	9952127789	Sivakasi

2 rows in set (0.00 sec)

Fig 4.9: Inserted data record sample in database

The above figure illustrates the sample records inserted in the database.

## UPDATE QUERY

Update query is used to update a particular or group of records in the database. The records can be updated based upon conditions. The syntax of UPDATE query is

UPDATE <table name> SET <column value>=<new column value> WHERE <condition>;

```
##### creating frame to update livestock details #####
def updateLivestock_buttonFrame():
    ##### creating frame #####
    update_buttonFrame=Frame(workerwindow,bd=10,bg='#3B3B3B',width=1030,height=475,relief=RIDGE)
    update_buttonFrame.place(x=225,y=300)

    ##### function to update livestock details #####
    def update():
        livestock_id=livestock_id_entry.get()
        dstatus=Disease_status_entry.get()
        vstatus=Vaccine_status_entry.get()
        Expenses=Expenses_entry.get()
        if livestock_id=="" or dstatus=="" or vstatus=="" or Expenses=="":
            messagebox.showinfo("Update status","All fields are required for Updating !!!!")
        if livestock_id.isdigit()==False:
            messagebox.showinfo("Update status","Enter valid id")
            return
        cursor.execute("SELECT COUNT(*) FROM LivestockDiagnosis WHERE LiveStock_id="+livestock_id_entry.get())
        rows=cursor.fetchall()
        cursor.execute("commit")
        if rows[0][0]==0:
            messagebox.showinfo("Update Status","Livestock id Not Found")
        else:
            cursor.execute("UPDATE LivestockDiagnosis SET disease_status='"+dstatus+"'WHERE LiveStock_id='"+livestock_id+"'")
            cursor.execute("UPDATE LivestockDiagnosis SET expenses='"+Expenses+"'WHERE LiveStock_id='"+livestock_id+"'")
            cursor.execute("UPDATE LivestockDiagnosis SET vaccine_status='"+vstatus+"'WHERE LiveStock_id='"+livestock_id+"'")
            cursor.execute("commit")
            messagebox.showinfo("Update status","Livestock Updated !!!")
        if dstatus=="Death":
            cursor.execute("DELETE FROM Livestock WHERE LiveStock_id='"+livestock_id+"'")
            cursor.execute("commit")
            Livestock_id_entry.delete(0,END)
            Disease_status_entry.delete(0,END)
            Vaccine_status_entry.delete(0,END)
            Expenses_entry.delete(0,END)
```

Fig 4.10 UPDATE query sample code

In the above figure, UPDATE query is used to update the data in the table. Here also the validity of the data that is going to be entered in the database is evaluated and to be updated the database record should be already recorded in the database, so the code also checks for the presence of record in the database. Here the python code illustrates the updation in the Livestock Diagnosis and hence DELETE query is used to automatically delete the records of the livestock whose disease status is death to maintain the consistency of the database. The DELETE query will be explained further.

## DELETE QUERY

The DELETE Query is used to delete a particular or group of records in the database . Like UPDATE query DELETE query also can be used based on conditions. The syntax of the DELETE query is

DELETE FROM <table name>;

DELETE <column name> FROM <table name> WHERE <condition>;

```
def DelUserFrame():
    ##### creating frame #####
    del_user_Frame=Frame(adminwindow,width=1030,height=575,bg='#9898F5',bd=10,relief=RIDGE)
    del_user_Frame.place(x=225,y=200)

    ##### function to delete user #####
    def deluser():
        ##### input from user #####
        user_id=user_id_entry.get()
        if user_id.isnumeric()==False:
            messagebox.showinfo("Delete Status","Enter valid user Id!!!")
            return
        ##### to check if the user is present or not in the records #####
        cursor.execute("SELECT COUNT(*) FROM User WHERE User_id="+user_id_entry.get())
        rows=cursor.fetchall()
        cursor.execute("commit")
        ##### checking if the user id is entered #####
        if user_id=="":
            messagebox.showinfo("Delete Status","Enter user Id")
        ##### deleting valid user id #####
        elif(user_id):
            if(rows[0][0]==0):
                messagebox.showinfo("Delete Status","User Not Found")
            else:
                cursor.execute("DELETE FROM User WHERE User_id= "+user_id)
                cursor.execute("commit")
                ##### clearing the value in the entry field for next user entry #####
                user_id_entry.delete(0,END)
                messagebox.showinfo("Delete Status","User Deleted !!!")

    ##### placing label in the frame for user convience #####
    txt='Delete User'
    heading=Label(del_user_Frame,text=txt,font=('Helvetica',20,'italic'),bg='#9898F5',fg='lavender')
    heading.place(x=50,y=20,width=200,height=30)
```

Fig 4.11: DELETE query sample code

The above figure illustrates the use of DELETE query. Here also the code checks for the presence of the record in the database and deletes the record according to the query.

## SELECT QUERY

The SELECT query is used to select particular records from the database based on some conditions. The syntax of the SELECT query is

**SELECT \* FROM < table name > WHERE <condition>;**

```
##### function to buy product #####
def buy():
    qty=qtyentry.get()
    productid=productchoosen.get()
    userid=buy_user_entry.get()
    ##### checking for valid user id #####
    if userid.isnumeric()==False:
        messagebox.showinfo("Product Status","Enter valid User id")
        return
    ##### check if the user is present in the record #####
    cursor.execute("SELECT COUNT(*) FROM User WHERE User_id="+buy_user_entry.get())
    rows=cursor.fetchall()
    cursor.execute("commit")

    ##### checking for the user to input all the fields #####
    if userid==" " or productid==" " or qty==" ":
        messagebox.showinfo("Product Status","All fields are required to buy a product ")
    elif(rows[0][0]==0):
        messagebox.showinfo("Product Status","User Not Found")
    else:
        cursor.execute("SELECT Quantity_avlb FROM Product WHERE Product_id='"+productid+"'")
        rs=cursor.fetchall()
        if int(qty)>rs[0][0]:
            messagebox.showinfo("Product Status ", "Sorry,Product Unavailable !!!")
        else:
            updateqty=rs[0][0]-int(qty)
            r=str(updateqty)
            cursor.execute("UPDATE Product SET Quantity_avlb='"+r+"'WHERE Product_id='"+productid+"'")
            cursor.execute("commit")
            messagebox.showinfo("Product Status ", "Thanks for buying !!!")
            qtyentry.delete(0,END)
            productchoosen.delete(0,END)
            buy_user_entry.delete(0,END)
            cursor.execute("INSERT INTO Bill VALUES('"+productid+"','"+userid+"','"+qty+"',curdate())")
            cursor.execute("commit")
```

Fig 4.12: SELECT query sample code

Here in this above figure SELECT query is illustrated clearly. In this block of code is used to maintain the records consistently when a user buys a product. The logic behind the code is when the user prompts to buy a product, the code checks for the availability of the product from the product table by accessing the product id using the select query “SELECT quantity\_avlb FROM Product WHERE product\_id=’’+productid+’’. If the quantity is not sufficient the code displays the message as “Sorry, Product unavailable”. If sufficient quantity is available then the corresponding details are inserted into the Bill Table by means

of INSERT query and the respective quantity of products sold is deduced from the available quantity in the product table. This maintains the records in consistent way and ensures the effective functionality of the system.

## JOIN clause

Join query is used to combine rows from two or more tables, based on a related column between them.

### Types of Joins in MySQL

- Inner Join
- Left Join
- Right Join
- Cross Join

The syntax for the join query is

<table1> INNER JOIN <table2> ON <condition>

```
buy_btn.place(x=275,y=300)

def ViewBillFrame():
    ##### creating frame #####
    viewbillFrame=Frame(userwindow,width=1030,height=575,bg='#A2A2B5',bd=10,relief=RIDGE)
    viewbillFrame.place(x=225,y=200)
    def view():
        #####input from user#####
        userid=userentry.get()
        ##### fetching values from the database and displaying in the frame #####
        cursor.execute("SELECT p.Product_id,p.Price,b.Quantity_saled,b.date_sold FROM Product p INNER JOIN Bill b ON p.Product_id=b.Product_id WHERE b.User_id="+userid)

        res=cursor.fetchall()
        cursor.execute("commit")
        for i in res:
            tree.insert(parent='',index='end',values=(i[0],i[1],i[2],i[3]))

        ##### calculating total #####
        total=0
        for i in res:
            total=total+(i[1]*i[2])
        toatlentry.insert(END,total)
        cursor.execute("SELECT address FROM User WHERE user_id='"+userid+"'")
        rs1=cursor.fetchall()
        cursor.execute("commit")
        messagebox.showinfo("Product Status ","Product will be shipped to "+rs1[0][0]+" in 7 days !! ")
        userentry.delete(0,END)
```

Fig 4.13: JOIN query sample code

In the above figure the bill calculation logic is explained clearly. Here product\_id, prize from the Product Table are retrieved along with quantity\_saled and date\_sold from the Bill Table using join query.

The query is

```
SELECT p.Product_id, p.Price, b.Quantity_saled, b.date_sold  
FROM Product p  
INNER JOIN  
Bill b ON  
p.Product_id=b.Product_id  
WHERE b.user_id=1;
```

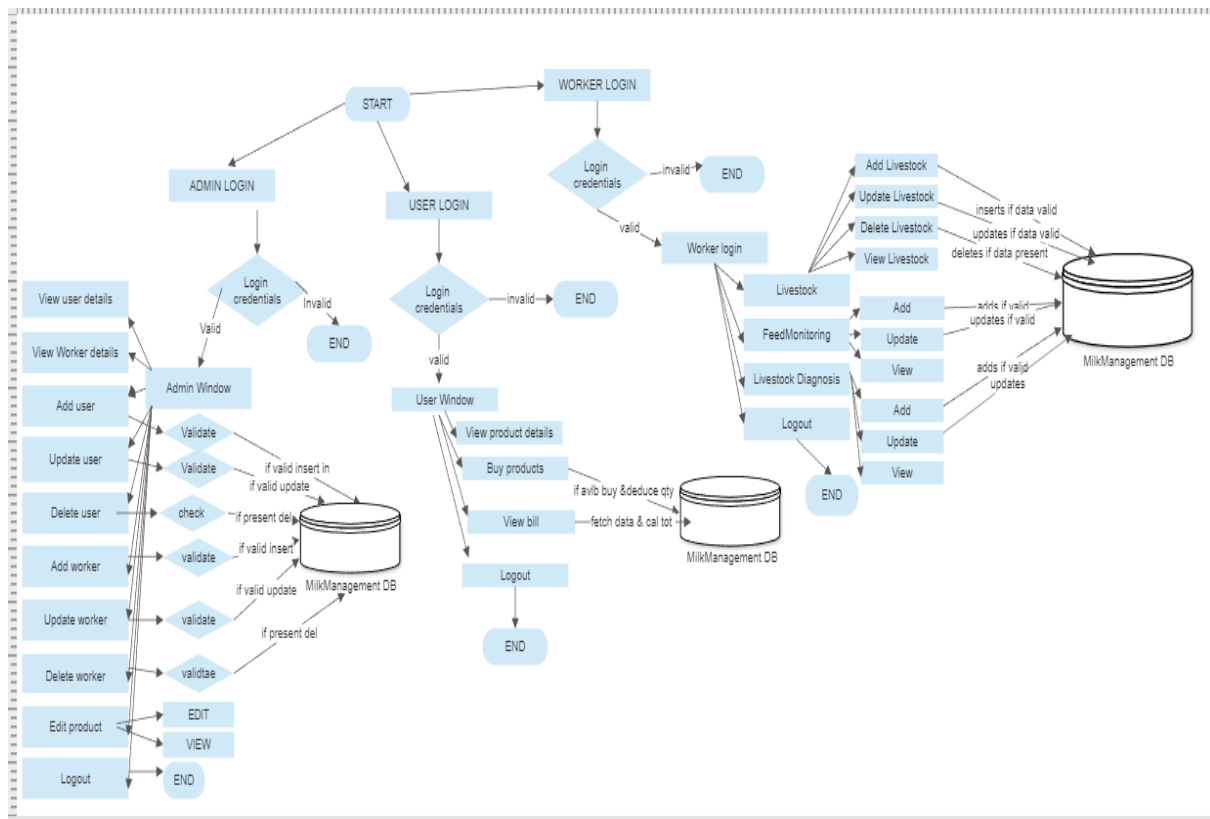
The total amount of the bill is calculated on basis of the records are retrieved and displayed to the user.



## CHAPTER 5

### IMPLEMENTATION

#### 5.1 FLOW DIAGRAM



**Fig: 5.1 Flow Diagram**

The Python Front end is designed in such a way that it is interactive and ensures the validity of the records in the database. The python source code is decomposed into several modules. At the start of the code a window with three options namely admin login, user login, worker login. Then according to the option selected corresponding login window appears.

If Admin login is selected admin login window will be displayed. After entering the valid credentials the admin can log in into the admin window and manipulate the details of user, worker and Product respectively. The code checks for the validity of the data by several constraints as mentioned in the source code 5.2 and updates DB according to the validity.

If User login is selected User login window will be displayed. After entering the valid credentials the user can log in into the User window and can view the products available and also can purchase desire products and generate the bill correspondingly. Once the user buys the products the quantity of the available products are managed in a well organized manner by the code.

If Worker login is selected worker login window will be displayed. After entering the valid credentials the worker can log in into the worker window and manipulate the details of livestock, feed monitoring and livestock diagnosis respectively. The code checks for the validity of the data by several constraints as mentioned in the source code 5.2 and updates DB according to the validity and also the code deletes the details of the livestock once the disease status of the livestock is updated to be 'death'.

And the user can logout from the window using logout button. If invalid credentials are entered the program automatically gets terminated ensuring the security of the data. The complete source code of this project is discussed in 5.2.

## 5.2 SOURCE CODE

```
from tkinter import *
import tkinter.font as f
from tkinter import ttk
from tkinter import messagebox
from PIL import ImageTk,Image #pip install pillow
import pymysql
##### creating connection to mysql database #####
con=pymysql.connect(host="localhost",user="root",password="dharshini1109",database="MilkManagement")
##### creating cursor object #####
cursor=con.cursor()
#####ADMIN WINDOW #####
def openAdminLogin():
    btn_font=f.Font(family='Helvetica',size=16,weight='normal',slant='roman')
    ##### Admin window after logging in #####
    def openAdminWindow():
        #####checking if admin in database #####
        cursor.execute("SELECT COUNT(*) FROM Admin WHERE
admin_username='"+username_entry.get()+"'")
        row=cursor.fetchall()
        cursor.execute("commit")
        if row[0][0]==0:
            messagebox.showinfo("Message","Admin not found !!!")
            return
        ##### checking if the user enters the password for login #####
        if password_entry.get()=="":
            messagebox.showinfo("Message","Enter your password")
        else:
            ##### selecting admin password from database to check the password validity #####
            cursor.execute("SELECT admin_password FROM Admin WHERE
Admin_username='"+username_entry.get()+"'")
            rows=cursor.fetchall()
            cursor.execute("commit")
            for i in rows:
```

```

##### checking password #####
if(i[0]==password_entry.get()):
    ##### to clear the entered values in the login page for the next time #####
    username_entry.delete(0,END)
    password_entry.delete(0,END)
    messagebox.showinfo("Message","Login Successful")
    ##### opening admin window if the password is correct #####
    adminwindow=Toplevel(adminlogin)
    adminwindow.title("Admin Window")
    adminwindow.minsize(1000,900)
    adminwindow.state('zoomed')
    adminwindow.config(bg='#9898F5')#CadetBlue1
    #####Top frame that holds the heading of the admin window #####
    topFrame=Frame(adminwindow,bd=10,relief=RIDGE,bg='#9898F5')
    topFrame.pack(side=TOP)
    ##### adding label to the top frame created #####
    labelTitle=Label(topFrame,text='MILK AND DAIRY PRODUCTS MANAGEMENT
SYSTEM',font=('arial',30,'bold'),bg='lavender',fg='#9898F5',width=51)
    labelTitle.grid(row=0,column=0)
    ##### creating a frame to add buttons #####

buttonFrame=Frame(adminwindow,bd=10,bg='lavender',width=1500,height=100,relief=RIDGE)
    buttonFrame.place(x=25,y=90)
    ##### creating frames for the buttons corespondingly #####
    ##### frame for add user button #####
    def AddUserFrame():
        ##### creating frame #####

add_user_Frame=Frame(adminwindow,width=1030,height=575,bg='#9898F5',bd=10,relief=RIDGE)
    add_user_Frame.place(x=225,y=200)
    ##### function to Insert user #####
    def adduser():
        ##### getting input from users #####
        user_id=user_id_entry.get()
        user_name=user_name_entry.get()

```

```

phn_num=phn_num_entry.get()
password=password_entry.get()
address=address_entry.get()
##### checking constraints for password #####
checknum=0
checkalpha=0
checklower=0
checkcapital=0
for i in password:
    if i.isnumeric():
        checknum=1
for i in password:
    if i.isalpha():
        checkalpha=1
for i in password:
    if i.islower():
        checklower=1
for i in password:
    if i.isupper():
        checkcapital=1
checkspl_char=0
l=['@','!','#','$','%','^','&','*','_']
for i in password:
    if i in l:
        checkspl_char=1
##### checking for the user to input all the fields #####
if user_id=="" or user_name=="" or phn_num=="" or password=="":
    messagebox.showinfo("Insert Status","All fields are required for adding a user")
else:
    ##### checking for valid phone number #####
    if len(phn_num)!=10:
        messagebox.showinfo("Insert Status","Enter valid phone number")
    ##### checking for valid username #####
    elif (user_name.isalpha()==False):

```

```

        messagebox.showinfo("Insert Status","Enter valid username")

##### checking for strong password #####

elif ((len(password)<8) or(checknum==0) or (checkspl_char==0) or
(checkcapital==0) or (checkalpha==0)or(checklower==0)):

        messagebox.showinfo("Insert Status","Passwords must be at least 8
characters,should contain atleast a special character,capital letter and a number")

##### checking for valid user id #####

elif user_id.isnumeric()==False:

        messagebox.showinfo("Insert Status","Enter valid user id")

##### Inserting if all values are correct #####

else:

        cursor.execute("Insert INTO User
VALUES("+user_id+", '"+user_name+"', '"+password+"', '"+phn_num+"', '"+address+"')")

        cursor.execute("commit")

##### clearing the value in the entry field for next user entry #####

        user_id_entry.delete(0,END)

        user_name_entry.delete(0,END)

        phn_num_entry.delete(0,END)

        password_entry.delete(0,END)

        address_entry.delete(0,END)

        messagebox.showinfo("Insert Status","User Added !!!")

##### placing label in the frame for user convience #####

txt='Add User'

heading=Label(add_user_Frame,text=txt,font=('Helvetica',20,'italic'),bg='#9898F5',fg='lavender')

        heading.place(x=50,y=20,width=200,height=30)

##### creating label and entry for user id #####

        user_id_label=Label(add_user_Frame,text='User Id
',font=('Helvetica',17,'italic'),bg='#9898F5',fg='lavender')

        user_id_label.place(x=50,y=75)

        user_id_entry=Entry(add_user_Frame,highlightthickness=0,relief=FLAT,fg="black",font=('Helvetica'
,12,'italic'))

        user_id_entry.place(x=200,y=75,width=230,height=35)

##### creating label and entry for user name #####

        user_name_label=Label(add_user_Frame,text='User Name
',font=('Helvetica',17,'italic'),bg='#9898F5',fg='lavender')

```

```

user_name_label.place(x=50,y=130)

user_name_entry=Entry(add_user_Frame,highlightthickness=0,relief=FLAT,fg="black",font=('Helvetica',12,'italic'))

user_name_entry.place(x=200,y=130,width=230,height=35)

##### creating label and entry for phone number #####

phn_num_label=Label(add_user_Frame,text='Phone Num',font=('Helvetica',17,'italic'),bg='#9898F5',fg='lavender')

phn_num_label.place(x=50,y=185)

phn_num_entry=Entry(add_user_Frame,highlightthickness=0,relief=FLAT,fg="black",font=('Helvetica',12,'italic'))

phn_num_entry.place(x=200,y=185,width=230,height=35)

##### creating label and entry for password #####

password_label=Label(add_user_Frame,text='Password',font=('Helvetica',17,'italic'),bg='#9898F5',fg='lavender')

password_label.place(x=50,y=240)

password_entry=Entry(add_user_Frame,highlightthickness=0,relief=FLAT,fg="black",font=('Helvetica',12,'italic'))

password_entry.place(x=200,y=240,width=230,height=35)

##### creating label and entry for address #####

address_label=Label(add_user_Frame,text='Address',font=('Helvetica',17,'italic'),bg='#9898F5',fg='lavender')

address_label.place(x=50,y=295)

address_entry=Entry(add_user_Frame,highlightthickness=0,relief=FLAT,fg="black",font=('Helvetica',12,'italic'))

address_entry.place(x=200,y=295,width=230,height=35)

##### creating button for add operation #####

add_btn=Button(add_user_Frame,text='ADD',bd='5',cursor='hand2',command=adduser,font=btn_font,width=20,bg='lavender',fg='#9898F5')

add_btn.place(x=300,y=350)

##### frame for delete user button #####

def DelUserFrame():

    ##### creating frame #####

del_user_Frame=Frame(adminwindow,width=1030,height=575,bg='#9898F5',bd=10,relief=RIDGE)

```

```

del_user_Frame.place(x=225,y=200)

##### function to delete user #####

def deluser():

    ##### input from user #####

    user_id=user_id_entry.get()

    if user_id.isnumeric()==False:

        messagebox.showinfo("Delete Status","Enter valid user Id!!!")

        return

    ##### to check if the user is present or not in the records #####

    cursor.execute("SELECT      COUNT(*)      FROM      User      WHERE
User_id="+user_id_entry.get())

    rows=cursor.fetchall()

    cursor.execute("commit")

    ##### checking if the user id is entered #####

    if user_id=="":

        messagebox.showinfo("Delete Status","Enter user Id")

    ##### deleting valid user id #####

    elif(user_id):

        if(rows[0][0]==0):

            messagebox.showinfo("Delete Status","User Not Found")

        else:

            cursor.execute("DELETE FROM User WHERE User_id= "+user_id)

            cursor.execute("commit")

            ##### clearing the value in the entry field for next user entry #####

            user_id_entry.delete(0,END)

            messagebox.showinfo("Delete Status","User Deleted !!!")

    ##### placing label in the frame for user convience #####

    txt='Delete User'

heading=Label(del_user_Frame,text=txt,font=('Helvetica',20,'italic'),bg='#9898F5',fg='lavender')

heading.place(x=50,y=20,width=200,height=30)

##### creating label and entry for user id to be deleted #####

user_id_label=Label(del_user_Frame,text='User                                     Id
',font=('Helvetica',17,'italic'),bg='#9898F5',fg='lavender')

user_id_label.place(x=50,y=75)

```



```

user_id_entry=Entry(del_user_Frame,highlightthickness=0,relief=FLAT,fg="black",font=('Helvetica',
12,'italic'))

user_id_entry.place(x=200,y=75,width=230,height=35)

##### creating button for delete operation #####

del_btn=Button(del_user_Frame,text='DELETE',bd='5',cursor='hand2',command=deluser,font=btn_font,width=20,bg='lavender',fg='#9898F5')

del_btn.place(x=200,y=200)

##### frame for update user button #####

def UpdateUserFrame():

    ##### creating frame #####

update_user_Frame=Frame(adminwindow,width=1030,height=575,bg='#9898F5',bd=10,relief=RIDGE)

update_user_Frame.place(x=225,y=200)

##### function to update user #####

def updateuser():

    ##### getting input from users #####

    user_id=user_id_entry.get()

    password=new_password_entry.get()

    ##### checking for valid user id #####

    if user_id.isnumeric()==False:

        messagebox.showinfo("Update Status","Enter valid user id")

        return

    ##### check if the users is present in the record #####

    cursor.execute("SELECT COUNT(*) FROM User WHERE
User_id="+user_id_entry.get())

    rows=cursor.fetchall()

    cursor.execute("commit")

    ##### checking constraints for password #####

    checknum=0

    checkalpha=0

    checklower=0

    checkcapital=0

    for i in password:

```

```

        if i.isnumeric():
            checknum=1
    for i in password:
        if i.isalpha():
            checkalpha=1
    for i in password:
        if i.islower():
            checklower=1
    for i in password:
        if i.isupper():
            checkcapital=1
    checkspl_char=0
    l=['@','!','#','$','%','^','&', '*', '_']
    for i in password:
        if i in l:
            checkspl_char=1

##### checking for the user to input all the fields #####
if user_id=="" or password=="":
    messagebox.showinfo("Update Status","All fields are required to update a user")
else:
    if(rows[0][0]==0):
        messagebox.showinfo("Update Status","User Not Found")

##### checking for strong password #####
    elif ((len(password)<8) or(checknum==0) or (checkspl_char==0) or
(checkcapital==0) or (checkalpha==0)or(checklower==0)):
        messagebox.showinfo("Update Status","Passwords must be atleast 8
characters,should contain atleast a special character,capital letter and a number")

##### Updating if all values are correct #####
    else:
        cursor.execute("UPDATE User SET User_Password='"+password+"'WHERE
User_id="+user_id)
        cursor.execute("commit")

##### clearing the value in the entry field for next user entry #####
        user_id_entry.delete(0,END)
        new_password_entry.delete(0,END)

```

```

        messagebox.showinfo("Update Status","User Updated !!!")

##### placing label in the frame for user convience #####
txt='Update User'

heading=Label(update_user_Frame,text=txt,font=('Helvetica',20,'italic'),bg='#9898F5',fg='lavender')
    heading.place(x=50,y=20,width=200,height=30)

##### creating label and entry field for user_id to be updated #####
    user_id_label=Label(update_user_Frame,text='User Id
',font=('Helvetica',17,'italic'),bg='#9898F5',fg='lavender')
        user_id_label.place(x=50,y=75)

user_id_entry=Entry(update_user_Frame,highlightthickness=0,relief=FLAT,fg="black",font=('Helvet
ica',12,'italic'))
        user_id_entry.place(x=225,y=75,width=230,height=35)

##### creating label and entry field for entering new password #####
        new_password_label=Label(update_user_Frame,text='New Password
',font=('Helvetica',17,'italic'),bg='#9898F5',fg='lavender')
            new_password_label.place(x=50,y=130)

new_password_entry=Entry(update_user_Frame,highlightthickness=0,relief=FLAT,fg="black",font=(
'Helvetica',12,'italic'))
        new_password_entry.place(x=225,y=130,width=230,height=35)

##### creating button for update operation #####

update_btn=Button(update_user_Frame,text='UPDATE',bd='5',cursor='hand2',command=updateuser,
font=btn_font,width=20,bg='lavender',fg='#9898F5')
        update_btn.place(x=300,y=300)

##### frame for add worker button #####
def AddWorkerFrame():
    ##### creating frame #####

add_worker_Frame=Frame(adminwindow,width=1030,height=575,bg='#9898F5',bd=10,relief=RIDG
E)

        add_worker_Frame.place(x=225,y=200)

##### function to add worker #####
def addworker():
    ##### user input #####
        id=worker_id_entry.get()

```

```

name=worker_name_entry.get()
phnnum=phn_num_entry.get()
mail_id=mail_id_entry.get()
password=password_entry.get()
salary=salary_entry.get()
worktype=workchoosen.get()
address=address_entry.get()
##### checking constraints for password #####
checknum=0
checkalpha=0
checklower=0
checkcapital=0
for i in password:
    if i.isnumeric():
        checknum=1
for i in password:
    if i.isalpha():
        checkalpha=1
for i in password:
    if i.islower():
        checklower=1
for i in password:
    if i.isupper():
        checkcapital=1
checkspl_char=0
l=['@','!','#','$','%','^','&', '*', '_']
for i in password:
    if i in l:
        checkspl_char=1
##### checking valid inputs #####
if id=="" or name=="" or password=="" or worktype=="" or phnnum=="" or
mail_id=="" or salary=="" or address=="":
    messagebox.showinfo("Insert Status","All fields are required for insertion")
elif id.isnumeric()==False:

```

```

        messagebox.showinfo("Insert Status","Enter valid user id")
elif name.isalpha()==False:
    messagebox.showinfo("Insert Status","Enter valid user name")
elif len(phnnum)!=10:
    messagebox.showinfo("Insert Status","Enter valid phn number")
elif((mail_id.find('.')== -1)or(mail_id.find('@')== -1):
    messagebox.showinfo("Insert Status","Enter valid mail id")
    elif ((len(password)<8) or(checknum==0) or (checkspl_char==0) or
(checkcapital==0) or (checkalpha==0)or(checklower==0)):
        messagebox.showinfo("Insert Status","Passwords must be at least 8
characters,should contain atleast a special character,capital letter and a number")
    else:
        cursor.execute("Insert INTO Worker
VALUES("+id+", '"+name+"', '"+password+"', '"+salary+'','"+mail_id+"', '"+worktype+'','"+address+"', "
+phnnum+"')")

        cursor.execute("commit")

        ##### freeing entry field for next entry #####
        worker_id_entry.delete(0,END)
        worker_name_entry.delete(0,END)
        phn_num_entry.delete(0,END)
        mail_id_entry.delete(0,END)
        password_entry.delete(0,END)
        salary_entry.delete(0,END)
        workchoosen.delete(0,END)
        address_entry.delete(0,END)

        messagebox.showinfo("Insert Status","Worker Added!!!")

        ##### placing label in the frame for user convience #####
        txt='Add Worker'

heading=Label(add_worker_Frame,text=txt,font=('Helvetica',20,'italic'),bg='#9898F5',fg='lavender')

        heading.place(x=50,y=20,width=200,height=30)

        ##### creating label and entry for worker id #####

        worker_id_label=Label(add_worker_Frame,text='Worker Id
',font=('Helvetica',17,'italic'),bg='#9898F5',fg='lavender')

        worker_id_label.place(x=50,y=75)

```

```

worker_id_entry=Entry(add_worker_Frame,highlightthickness=0,relief=FLAT,fg="black",font=('Hel
vetica',12,'italic'))

worker_id_entry.place(x=200,y=75,width=230,height=35)

##### creating label and entry for worker name #####

worker_name_label=Label(add_worker_Frame,text='Worker Name
',font=('Helvetica',17,'italic'),bg='#9898F5',fg='lavender')

worker_name_label.place(x=50,y=130)

worker_name_entry=Entry(add_worker_Frame,highlightthickness=0,relief=FLAT,fg="black",font=('
Helvetica',12,'italic'))

worker_name_entry.place(x=200,y=130,width=230,height=35)

##### creating label and entry for phone number #####

phn_num_label=Label(add_worker_Frame,text='Phone Num
',font=('Helvetica',17,'italic'),bg='#9898F5',fg='lavender')

phn_num_label.place(x=50,y=185)

phn_num_entry=Entry(add_worker_Frame,highlightthickness=0,relief=FLAT,fg="black",font=('Helv
etica',12,'italic'))

phn_num_entry.place(x=200,y=185,width=230,height=35)

##### creating label and entry for password #####

password_label=Label(add_worker_Frame,text='Password
',font=('Helvetica',17,'italic'),bg='#9898F5',fg='lavender')

password_label.place(x=50,y=240)

password_entry=Entry(add_worker_Frame,highlightthickness=0,relief=FLAT,fg="black",font=('Helv
etica',12,'italic'))

password_entry.place(x=200,y=240,width=230,height=35)

##### creating label and entry for salary #####

salary_label=Label(add_worker_Frame,text='Salary
',font=('Helvetica',17,'italic'),bg='#9898F5',fg='lavender')

salary_label.place(x=50,y=295)

salary_entry=Entry(add_worker_Frame,highlightthickness=0,relief=FLAT,fg="black",font=('Helvetic
a',12,'italic'))

salary_entry.place(x=200,y=295,width=230,height=35)

##### creating label and entry for worker type #####

worker_type_label=Label(add_worker_Frame,text='Worker
type',font=('Helvetica',17,'italic'),bg='#9898F5',fg='lavender')

```

```

worker_type_label.place(x=50,y=350)

##### creating combobox #####
n =StringVar()
workchoosen = ttk.Combobox(add_worker_Frame, width = 27, textvariable = n)
# Adding combobox drop down list
workchoosen['values'] = (' FeedMonitor',' CareTaker',)
workchoosen.place(x=200,y=350,width=230,height=35)

##### creating label and entry for address #####
address_label=Label(add_worker_Frame,text='Address
',font=('Helvetica',17,'italic'),bg='#9898F5',fg='lavender')
address_label.place(x=50,y=405)

address_entry=Entry(add_worker_Frame,highlightthickness=0,relief=FLAT,fg="black",font=('Helvet
ica',12,'italic'))
address_entry.place(x=200,y=405,width=230,height=35)

##### creating label and entry for mail id #####
mail_id_label=Label(add_worker_Frame,text='Mail id
',font=('Helvetica',17,'italic'),bg='#9898F5',fg='lavender')
mail_id_label.place(x=50,y=460)

mail_id_entry=Entry(add_worker_Frame,highlightthickness=0,relief=FLAT,fg="black",font=('Helvet
ica',12,'italic'))
mail_id_entry.place(x=200,y=460,width=230,height=35)

##### creating add button for add operation #####

add_btn=Button(add_worker_Frame,text='ADD',bd='5',cursor='hand2',command=addworker,font=bt
n_font,width=20,bg='lavender',fg='#9898F5')
add_btn.place(x=400,y=500)

##### frame for delete worker button #####
def DelWorkerFrame():
    ##### creating frame #####

del_worker_Frame=Frame(adminwindow,width=1030,height=575,bg='#9898F5',bd=10,relief=RIDG
E)

del_worker_Frame.place(x=225,y=200)

##### function to delete worker #####
def delworker():
    ##### input from user #####

```

```

worker=worker_id_entry.get()
if worker.isnumeric()==False:
    messagebox.showinfo("Delete Status","Enter valid Worker Id!!!")
    return
##### to check if the worker is present or not in the records #####
cursor.execute("SELECT COUNT(*) FROM Worker WHERE
Worker_id="+worker_id_entry.get())
rows=cursor.fetchall()
cursor.execute("commit")
##### checking if the worker id is entered #####
if worker=="":
    messagebox.showinfo("Delete Status","Enter Worker Id")
##### deleting valid worker id #####
elif(worker):
    if(rows[0][0]==0):
        messagebox.showinfo("Delete Status","Worker Not Found")
    else:
        cursor.execute("DELETE FROM Worker WHERE Worker_id= "+worker)
        cursor.execute("commit")
        ##### clearing the value in the entry field for next user entry #####
        worker_id_entry.delete(0,END)
        messagebox.showinfo("Delete Status","Worker Deleted !!!")
##### placing label in the frame for user convience #####
txt='Delete Worker'

heading=Label(del_worker_Frame,text=txt,font=('Helvetica',20,'italic'),bg='#9898F5',fg='lavender')
heading.place(x=50,y=20,width=200,height=30)
##### creating label and entry for worker id to be deleted #####
worker_id_label=Label(del_worker_Frame,text='Worker Id
',font=('Helvetica',17,'italic'),bg='#9898F5',fg='lavender')
worker_id_label.place(x=50,y=75)

worker_id_entry=Entry(del_worker_Frame,highlightthickness=0,relief=FLAT,fg="black",font=('Helv
etica',12,'italic'))
worker_id_entry.place(x=200,y=75,width=230,height=35)
##### creating delete button for delete operation #####

```



```

del_btn=Button(del_worker_Frame,text='DELETE',bd='5',cursor='hand2',command=delworker,font=
btn_font,width=20,bg='lavender',fg='#9898F5')

del_btn.place(x=200,y=200)

##### frame for update worker button #####

def UpdateWorkerFrame():

update_worker_Frame=Frame(adminwindow,width=1030,height=575,bg='#9898F5',bd=10,relief=RI
DGE)

update_worker_Frame.place(x=225,y=200)

##### function to update worker #####

def updateworker():

##### getting input from users #####

worker_id=worker_id_entry.get()

salary=salary_entry.get()

##### checking for valid Worker id #####

if worker_id.isnumeric()==False:

    messagebox.showinfo("Update Status","Enter valid Worker id")

    return

##### check if the Worker is present in the record #####

cursor.execute("SELECT      COUNT(*)      FROM      Worker      WHERE
Worker_id="+worker_id_entry.get())

rows=cursor.fetchall()

cursor.execute("commit")

##### checking for the user to input all the fields #####

if worker_id=="" or salary=="":

    messagebox.showinfo("Update Status","All fields are required to update a user")

else:

    if(rows[0][0]==0):

        messagebox.showinfo("Update Status","Worker Not Found")

        ##### Updating if all values are correct #####

        else:

            cursor.execute("UPDATE      Worker      SET      salary='"+salary+"WHERE
Worker_id="+worker_id)

            cursor.execute("commit")

            ##### clearing the value in the entry field for next user entry #####

```

```

        worker_id_entry.delete(0,END)

        salary_entry.delete(0,END)

        messagebox.showinfo("Update Status","Worker Updated !!!")

        ##### placing label in the frame for user convience #####

        txt='Update Worker'

heading=Label(update_worker_Frame,text=txt,font=('Helvetica',20,'italic'),bg='#9898F5',fg='lavender')

        heading.place(x=50,y=20,width=200,height=30)

        ##### creating label and entry for worker id to be updated #####

        worker_id_label=Label(update_worker_Frame,text='WorkerId',font=('Helvetica',17,'italic'),bg='#9898F5',fg='lavender')

        worker_id_label.place(x=50,y=75)

worker_id_entry=Entry(update_worker_Frame,highlightthickness=0,relief=FLAT,fg="black",font=('Helvetica',12,'italic'))

        worker_id_entry.place(x=225,y=75,width=230,height=35)

        ##### creating label and entry for salary #####

        salary_label=Label(update_worker_Frame,text='Salary',font=('Helvetica',17,'italic'),bg='#9898F5',fg='lavender')

        salary_label.place(x=50,y=130)

salary_entry=Entry(update_worker_Frame,highlightthickness=0,relief=FLAT,fg="black",font=('Helvetica',12,'italic'))

        salary_entry.place(x=225,y=130,width=230,height=35)

        ##### creating update button for update operation #####

update_btn=Button(update_worker_Frame,text='UPDATE',bd='5',cursor='hand2',command=updateworker,font=btn_font,width=20,bg='lavender',fg='#9898F5')

        update_btn.place(x=300,y=300)

        ##### frame for edit product button #####

def EditProductFrame():

        ##### creating frame #####

edit_product_Frame=Frame(adminwindow,width=1030,height=575,bg='#9898F5',bd=10,relief=RIDGE)

        edit_product_Frame.place(x=225,y=200)

        ##### function to edit product #####

```

```

def editproduct():
    ##### input from admin #####
    p=productchoosen.get()
    qty=qty_entry.get()
    price=price_entry.get()
    ##### updating in database #####
    cursor.execute("UPDATE    Product    SET    Quantity_avlb='"+qty+"'WHERE
Product_id="+p)
    cursor.execute("UPDATE Product SET Price='"+price+"'WHERE Product_id="+p)
    cursor.execute("COMMIT")
    ##### clearing the entry field for next entry #####
    productchoosen.delete(0,END)
    qty_entry.delete(0,END)
    price_entry.delete(0,END)
    messagebox.showinfo("Update Status","Product Edited !!!")
    ##### function to view product #####
def viewproduct():
    ##### creating frame #####

view_product_Frame=Frame(adminwindow,width=1030,height=575,bg='#9898F5',bd=10,relief=RID
GE)

    view_product_Frame.place(x=225,y=200)
    ##### instance of style widget #####
    style = ttk.Style()
    style.theme_use('clam')
    ##### creating tree frame to display user details #####
    tree    =ttk.Treeview(view_product_Frame,    column=("c1",    "c2","c3","c4"),
show='headings', height=26)
    ##### aligning columns #####
    tree.column("# 1", anchor=CENTER)
    tree.heading("# 1", text="Product ID")
    tree.column("# 2", anchor=CENTER)
    tree.heading("# 2", text="Product Name")
    tree.column("# 3", anchor=CENTER)
    tree.heading("# 3", text="Quantity avlb")

```

```

tree.column("# 4", anchor=CENTER)

tree.heading("# 4", text="Price ")

tree.place(x=100,y=0)

##### fetching values from the database and displaying in the frame #####

cursor.execute("SELECT * FROM Product")

res=cursor.fetchall()

for i in res:

    tree.insert(parent="",index='end',values=(i[0],i[1],i[2],i[3]))

cursor.execute("commit")

##### listing the products available #####

Product_id_label=Label(edit_product_Frame,text='ProductId',font=('Helvetica',17,'italic'),bg='#9898F5',fg='lavender')

Product_id_label.place(x=550,y=75)

Product_name_label=Label(edit_product_Frame,text='ProductName',font=('Helvetica',17,'italic'),bg='#9898F5',fg='lavender')

Product_name_label.place(x=725,y=75)

Product_id1_label=Label(edit_product_Frame,text='1',font=('Helvetica',17,'italic'),bg='#9898F5',fg='lavender')

Product_id1_label.place(x=550,y=130)

Product_name1_label=Label(edit_product_Frame,text='Milk',font=('Helvetica',17,'italic'),bg='#9898F5',fg='lavender')

Product_name1_label.place(x=725,y=130)

Product_id2_label=Label(edit_product_Frame,text='2',font=('Helvetica',17,'italic'),bg='#9898F5',fg='lavender')

Product_id2_label.place(x=550,y=185)

Product_name2_label=Label(edit_product_Frame,text='Curd',font=('Helvetica',17,'italic'),bg='#9898F5',fg='lavender')

Product_name2_label.place(x=725,y=185)

Product_id3_label=Label(edit_product_Frame,text='3',font=('Helvetica',17,'italic'),bg='#9898F5',fg='lavender')

Product_id3_label.place(x=550,y=240)

Product_name3_label=Label(edit_product_Frame,text='Butter',font=('Helvetica',17,'italic'),bg='#9898F5',fg='lavender')

Product_name3_label.place(x=725,y=240)

Product_id4_label=Label(edit_product_Frame,text='4',font=('Helvetica',17,'italic'),bg='#9898F5',fg='lavender')

Product_id4_label.place(x=550,y=295)

```

```

        Product_name4_label=Label(edit_product_Frame,text='Cheese
',font=('Helvetica',17,'italic'),bg='#9898F5',fg='lavender')

        Product_name4_label.place(x=725,y=295)

        Product_id5_label=Label(edit_product_Frame,text='5
',font=('Helvetica',17,'italic'),bg='#9898F5',fg='lavender')

        Product_id5_label.place(x=550,y=350)

        Product_name5_label=Label(edit_product_Frame,text='Paneer
',font=('Helvetica',17,'italic'),bg='#9898F5',fg='lavender')

        Product_name5_label.place(x=725,y=350)

        Product_id6_label=Label(edit_product_Frame,text='6
',font=('Helvetica',17,'italic'),bg='#9898F5',fg='lavender')

        Product_id6_label.place(x=550,y=405)

        Product_name6_label=Label(edit_product_Frame,text='Ghee
',font=('Helvetica',17,'italic'),bg='#9898F5',fg='lavender')

        Product_name6_label.place(x=725,y=405)

        ##### placing label in the frame for user convience #####
        txt='Edit Product'

heading=Label(edit_product_Frame,text=txt,font=('Helvetica',20,'italic'),bg='#9898F5',fg='lavender')

        heading.place(x=50,y=20,width=200,height=30)

        ##### creating label and entry field for entering product id to be edited #####

        edit_product_label=Label(edit_product_Frame,text='ProductId
',font=('Helvetica',17,'italic'),bg='#9898F5',fg='lavender')

        edit_product_label.place(x=50,y=75)

        # Combobox creation

        n = int()

        productchoosen = ttk.Combobox(edit_product_Frame, width = 27, textvariable = n)

        # Adding combobox drop down list

        productchoosen['values'] = ('1','2','3','4','5','6')

        productchoosen.place(x=225,y=75,width=230,height=35)

        ##### creating label and entry field for entering price #####

        price_label=Label(edit_product_Frame,text='Price
',font=('Helvetica',17,'italic'),bg='#9898F5',fg='lavender')

        price_label.place(x=50,y=130)

        price_entry=Entry(edit_product_Frame,highlightthickness=0,relief=FLAT,fg="black",font=('Helvetic
a',12,'italic'))

        price_entry.place(x=225,y=130,width=230,height=35)

```

```

##### creating label and entry field for entering quantity #####

qty_label=Label(edit_product_Frame,text='Quantity
',font=('Helvetica',17,'italic'),bg='#9898F5',fg='lavender')

qty_label.place(x=50,y=185)

qty_entry=Entry(edit_product_Frame,highlightthickness=0,relief=FLAT,fg="black",font=('Helvetica',
12,'italic'))

qty_entry.place(x=225,y=185,width=230,height=35)

##### creating edit button for edit operations #####

edit_btn=Button(edit_product_Frame,text='EDIT',bd='5',cursor='hand2',command=editproduct,font=b
tn_font,width=20,bg='lavender',fg='#9898F5')

edit_btn.place(x=250,y=300)

##### creating edit button for edit operations #####

view_btn=Button(edit_product_Frame,text='VIEW',bd='5',cursor='hand2',command=viewproduct,fon
t=btn_font,width=20,bg='lavender',fg='#9898F5')

view_btn.place(x=250,y=370)

def ViewUserFrame():

##### creating frame #####

viewuserFrame=Frame(adminwindow,width=1030,height=575,bg='#9898F5',bd=10,relief=RIDGE)

viewuserFrame.place(x=225,y=200)

##### instance of style widget #####

style = ttk.Style()

style.theme_use('clam')

##### creating tree frame to display user details #####

tree =ttk.Treeview(viewuserFrame, column=("c1", "c2","c3","c4"), show='headings',
height=26)

##### aligning columns #####

tree.column("# 1", anchor=CENTER)

tree.heading("# 1", text="User ID")

tree.column("# 2", anchor=CENTER)

tree.heading("# 2", text="User Name")

tree.column("# 3", anchor=CENTER)

tree.heading("# 3", text="Password ")

tree.column("# 4", anchor=CENTER)

```

```

tree.heading("# 4", text="Phone number ")
tree.place(x=100,y=0)
##### fetching values from the database and displaying in the frame #####
cursor.execute("SELECT * FROM User")
res=cursor.fetchall()
for i in res:
    tree.insert(parent="",index='end',values=(i[0],i[1],i[2],i[3]))
cursor.execute("commit")
def ViewWorkerFrame():
    ##### creating frame #####

viewworkerFrame=Frame(adminwindow,width=1030,height=575,bg='#9898F5',bd=10,relief=RIDGE
)

viewworkerFrame.place(x=225,y=200)
##### instance of style widget #####
style = ttk.Style()
style.theme_use('clam')
##### creating tree frame to display worker details #####
tree =ttk.Treeview(viewworkerFrame,      column=("c1",      "c2","c3","c4","c5"),
show='headings', height=26)
##### aligning columns #####
tree.column("# 1", anchor=CENTER)
tree.heading("# 1", text="Worker ID")
tree.column("# 2", anchor=CENTER)
tree.heading("# 2", text="Worker Name")
tree.column("# 3", anchor=CENTER)
tree.heading("# 3", text="Worker Type")
tree.column("# 4", anchor=CENTER)
tree.heading("# 4", text="Password ")
tree.column("# 5", anchor=CENTER)
tree.heading("# 5", text="Salary")
tree.place(x=0,y=0)
##### fetching values from the database and displaying in the frame #####
cursor.execute("SELECT * FROM Worker")
res=cursor.fetchall()

```

```

        for i in res:

            tree.insert(parent="",index='end',values=(i[0],i[1],i[5],i[2],i[3]))

            cursor.execute("commit")

            ##### creating and adding buttons to frame #####

            ##### add button to add user #####

            add_user_btn=Button(buttonFrame,text='Add
User',bd='3',cursor='hand2',command=AddUserFrame,font=btn_font,width=10,bg='#9898F5',fg='lavender')

            add_user_btn.place(x=5,y=20)

            ##### delete button to delete user #####

            del_user_btn=Button(buttonFrame,text='Delete
User',bd='3',cursor='hand2',command=DelUserFrame,font=btn_font,width=10,bg='#9898F5',fg='lavender')

            del_user_btn.place(x=150,y=20)

            ##### update button to update user #####

            update_user_btn=Button(buttonFrame,text='Update
User',bd='3',cursor='hand2',command=UpdateUserFrame,font=btn_font,width=10,bg='#9898F5',fg='lavender')

            update_user_btn.place(x=295,y=20)

            ##### add button to add worker #####

            add_worker_btn=Button(buttonFrame,text='Add
Worker',bd='3',cursor='hand2',command=AddWorkerFrame,font=btn_font,width=10,bg='#9898F5',fg='lavender')

            add_worker_btn.place(x=440,y=20)

            ##### delete button to delete worker #####

            del_worker_btn=Button(buttonFrame,text='DeleteWorker',bd='3',cursor='hand2',command=DelWorkerFrame,font=btn_font,width=10,bg='#9898F5',fg='lavender')

            del_worker_btn.place(x=585,y=20)

            ##### update button to update worker #####

            update_worker_btn=Button(buttonFrame,text='UpdateWorker',bd='3',cursor='hand2',command=UpdateWorkerFrame,font=btn_font,width=10,bg='#9898F5',fg='lavender')

            update_worker_btn.place(x=730,y=20)

            ##### edit button to edit worker #####

            edit_product_btn=Button(buttonFrame,text='Edit
Product',bd='3',cursor='hand2',command=EditProductFrame,font=btn_font,width=10,bg='#9898F5',fg='lavender')

            edit_product_btn.place(x=875,y=20)

```



```

##### button to view user details #####

user_details_btn=Button(buttonFrame,text='UserDetails',bd='3',cursor='hand2',command=ViewUserF
rame,font=btn_font,width=10,bg='#9898F5',fg='lavender')

user_details_btn.place(x=1020,y=20)

##### button to view worker details #####

worker_details_btn=Button(buttonFrame,text='WorkerDetails',bd='3',cursor='hand2',command=View
WorkerFrame,font=btn_font,width=10,bg='#9898F5',fg='lavender')

worker_details_btn.place(x=1165,y=20)

##### button to logout #####

logout_btn=Button(buttonFrame,text='Logout',bd='3',cursor='hand2',command=adminwindow.destro
y,font=btn_font,width=10,bg='#9898F5',fg='lavender')

logout_btn.place(x=1310,y=20)

break

else:

##### showing warning message for invalid password #####

messagebox.showwarning("Message","Invalid password")

##### admin login window #####

adminlogin = Toplevel(root)

adminlogin.title("Admin login page")

adminlogin.minsize(1000,900)

adminlogin.state('zoomed')

##### background image #####

bg_frame=Image.open("C:\\Users\\maha9\\OneDrive\\Documents\\Dbms&py_miniproj\\bg5.jpg")

photo=ImageTk.PhotoImage(bg_frame)

bg_panel=Label(adminlogin,image=photo)

bg_panel.image=photo

bg_panel.pack(fill='both',expand='yes')

##### login frame #####

##### creating a login frame #####

login_frame=Frame(adminlogin,width='900',height=600)

login_frame.place(x=600,y=125)

##### giving a heading for a login frame #####

txt='ADMIN LOGIN'

```

```

txt1='MILK AND DAIRY PRODUCTS MANAGEMENT'
heading=Label(login_frame,text=txt,font=('Helvetica',22,'italic'),fg='black')
heading1=Label(login_frame,text=txt1,font=('Helvetica',25,'italic'),fg='black')
heading1.place(x=5,y=5,width=750,height=30)
heading.place(x=10,y=50,width=300,height=30)
##### placing a image in the login frame #####

sign_in_image=Image.open("C:\\Users\\maha9\\OneDrive\\Documents\\Dbms&py_miniproj\\s1.jpg")
photo=ImageTk.PhotoImage(sign_in_image)
sign_in_image_label=Label(login_frame,image=photo)
sign_in_image_label.image=photo
sign_in_image_label.place(x=620,y=50)
#####placing 'sign in 'text #####
sign_in_label=Label(login_frame,text='Sign In',fg='black',font=('Helvetica',17,'italic'))
sign_in_label.place(x=635,y=175)
##### creating username label #####
username_label=Label(login_frame,text='Username',font=('Helvetica',17,'italic'),fg='black')
username_label.place(x=475,y=225)
##### creating textfield for username entry #####

username_entry=Entry(login_frame,highlightthickness=0,relief=FLAT,fg="black",font=('Helvetica',17,'italic'))
username_entry.place(x=600,y=225,width=230,height=35)
##### creating password label #####
password_label=Label(login_frame,text='Password',font=('Helvetica',17,'italic'),fg='black')
password_label.place(x=475,y=300)
##### creating textfield for password entry #####

password_entry=Entry(login_frame,highlightthickness=0,relief=FLAT,fg="black",font=('Helvetica',17,'italic'))
password_entry.place(x=600,y=300,width=230,height=35)
##### login button creation #####

login_btn=Button(login_frame,text='Login',bd=5,cursor='hand2',command=openAdminWindow,font=btn_font,width=20)
login_btn.place(x=545,y=370)

```

```

#####USER LOGIN#####

def openUserLogin():
    btn_font=f.Font(family='Helvetica',size=16,weight='normal',slant='roman')

    ##### creating user window #####

    def openUserWindow():
        #####checking if user in database #####

        cursor.execute("SELECT          COUNT(*)          FROM          User          WHERE
user_name='"+username_entry.get()+"'")

        row=cursor.fetchall()

        cursor.execute("commit")

        if row[0][0]==0:
            messagebox.showinfo("Message","User not found !!!")

            return

        ##### checking if the user enters the password for login #####

        if password_entry.get()=="":
            messagebox.showinfo("Message","Enter your password")

        else:
            ##### selecting admin password from database to check the password validity #####

            cursor.execute("SELECT          User_password          FROM          User          WHERE
User_name='"+username_entry.get()+"'")

            rows=cursor.fetchall()

            print(rows[0][0])

            cursor.execute("commit")

            ##### checking password #####

            if(rows[0][0]==password_entry.get()):
                ##### to clear the entered values in the login page for the next time #####

                messagebox.showinfo("Message","Login Successful")

                username_entry.delete(0,END)

                password_entry.delete(0,END)

                userwindow=Toplevel(userlogin)

                userwindow.title("User Window")

                userwindow.minsize(1000,900)

                userwindow.state('zoomed')

                userwindow.config(bg='#A2A2B5')#LightSteelBlue3

```

```

#####Top frame that holds the heading of the admin window #####
topFrame=Frame(userwindow,bd=10,relief=RIDGE,bg='#A2A2B5')
topFrame.pack(side=TOP)

##### adding label to the top frame created #####
labelTitle=Label(topFrame,text='MILK AND DAIRY PRODUCTS MANAGEMENT
SYSTEM',font=('arial',30,'bold'),bg='#F5F5FF',fg='#A2A2B5',width=51)##"#F5F5FF"--"mint cream"
labelTitle.grid(row=0,column=0)

##### creating a frame to add buttons #####

buttonFrame=Frame(userwindow,bd=10,bg='#F5F5FF',width=650,height=100,relief=RIDGE)
buttonFrame.place(x=375,y=90)

##### creating frames correspondingly to the buttons #####
def ViewProductFrame():
    ##### creating frame #####

viewproductFrame=Frame(userwindow,width=1030,height=575,bg='#A2A2B5',bd=10,relief=RIDGE
)

    viewproductFrame.place(x=225,y=200)

    ##### giving a heading for user convience #####
    txt='PRODUCTS AVAILABLE '

    heading=Label(viewproductFrame,text=txt,font=('Helvetica',25,'italic'),fg='#F5F5FF',bg='#A2A2B5')
    heading.place(x=80,y=30,width=500,height=30)

    ##### listing the products available #####

    Product_id_label=Label(viewproductFrame,text='Product                                Id
',font=('Helvetica',17,'italic'),bg='#A2A2B5',fg='#F5F5FF')
    Product_id_label.place(x=200,y=75)

    Product_name_label=Label(viewproductFrame,text='Product                                Name
',font=('Helvetica',17,'italic'),bg='#A2A2B5',fg='#F5F5FF')
    Product_name_label.place(x=400,y=75)

    Product_id1_label=Label(viewproductFrame,text='1
',font=('Helvetica',17,'italic'),bg='#A2A2B5',fg='#F5F5FF')
    Product_id1_label.place(x=200,y=130)

    Product_name1_label=Label(viewproductFrame,text='Milk
',font=('Helvetica',17,'italic'),bg='#A2A2B5',fg='#F5F5FF')
    Product_name1_label.place(x=400,y=130)

    Product_id2_label=Label(viewproductFrame,text='2
',font=('Helvetica',17,'italic'),bg='#A2A2B5',fg='#F5F5FF')

```

```

Product_id2_label.place(x=200,y=185)

Product_name2_label=Label(viewproductFrame,text='Curd
',font=('Helvetica',17,'italic'),bg='#A2A2B5',fg='#F5F5FF')

Product_name2_label.place(x=400,y=185)

Product_id3_label=Label(viewproductFrame,text='3
',font=('Helvetica',17,'italic'),bg='#A2A2B5',fg='#F5F5FF')

Product_id3_label.place(x=200,y=240)

Product_name3_label=Label(viewproductFrame,text='Butter
',font=('Helvetica',17,'italic'),bg='#A2A2B5',fg='#F5F5FF')

Product_name3_label.place(x=400,y=240)

Product_id4_label=Label(viewproductFrame,text='4
',font=('Helvetica',17,'italic'),bg='#A2A2B5',fg='#F5F5FF')

Product_id4_label.place(x=200,y=295)

Product_name4_label=Label(viewproductFrame,text='Cheese
',font=('Helvetica',17,'italic'),bg='#A2A2B5',fg='#F5F5FF')

Product_name4_label.place(x=400,y=295)

Product_id5_label=Label(viewproductFrame,text='5
',font=('Helvetica',17,'italic'),bg='#A2A2B5',fg='#F5F5FF')

Product_id5_label.place(x=200,y=350)

Product_name5_label=Label(viewproductFrame,text='Paneer
',font=('Helvetica',17,'italic'),bg='#A2A2B5',fg='#F5F5FF')

Product_name5_label.place(x=400,y=350)

Product_id6_label=Label(viewproductFrame,text='6
',font=('Helvetica',17,'italic'),bg='#A2A2B5',fg='#F5F5FF')

Product_id6_label.place(x=200,y=405)

Product_name6_label=Label(viewproductFrame,text='Ghee
',font=('Helvetica',17,'italic'),bg='#A2A2B5',fg='#F5F5FF')

Product_name6_label.place(x=400,y=405)

def BuyProductFrame():

    ##### creating frame #####

buyproductFrame=Frame(userwindow,width=1030,height=575,bg='#A2A2B5',bd=10,relief=RIDGE)

buyproductFrame.place(x=225,y=200)

##### function to buy product #####

def buy():

    qty=qtyentry.get()

    productid=productchoosen.get()

    userid=buy_user_entry.get()

```

```

##### checking for valid user id #####
if userid.isnumeric()==False:
    messagebox.showinfo("Product Status","Enter valid User id")
    return
##### check if the user is present in the record #####
cursor.execute("SELECT      COUNT(*)      FROM      User      WHERE
User_id="+buy_user_entry.get())
rows=cursor.fetchall()
cursor.execute("commit")
##### checking for the user to input all the fields #####
if userid==" " or productid==" " or qty==" ":
    messagebox.showinfo("Product Status","All fields are required to buy a product ")
elif(rows[0][0]==0):
    messagebox.showinfo("Product Status","User Not Found")
else:
    cursor.execute("SELECT      Quantity_avlb      FROM      Product      WHERE
Product_id="+productid+"")
rs=cursor.fetchall()
if int(qty)>=rs[0][0]:
    messagebox.showinfo("Product Status ", "Sorry,Product Unavailable !!!")
else:
    updateqty=rs[0][0]-int(qty)
    r=str(updateqty)
    cursor.execute("UPDATE      Product      SET      Quantity_avlb="+r+"WHERE
Product_id="+productid+"")
    cursor.execute("commit")
    messagebox.showinfo("Product Status ", "Thanks for buying !!!")
    qtyentry.delete(0,END)
    productchoosen.delete(0,END)
    buy_user_entry.delete(0,END)
    cursor.execute("INSERT      INTO      Bill
VALUES("+productid+", "+userid+", "+qty+",curdate())")
    cursor.execute("commit")
##### giving a heading for user convience #####
txt='Choose Product'

```

```

heading=Label(buyproductFrame,text=txt,font=('Helvetica',25,'italic'),fg='#F5F5FF',bg='#A2A2B5')

    heading.place(x=80,y=30,width=300,height=30)

    ##### creating label and entry field for entering product id to be bought #####

    buy_product_label=Label(buyproductFrame,text='Product                                     Id
',font=('Helvetica',17,'italic'),bg='#A2A2B5',fg='#F5F5FF')

    buy_product_label.place(x=50,y=75)

    # Combobox creation

    n = int()

    productchoosen = ttk.Combobox(buyproductFrame, width = 27, textvariable = n)

    # Adding combobox drop down list

    productchoosen['values'] = ('1','2','3','4','5','6')

    productchoosen.place(x=225,y=75,width=230,height=35)

    ##### creating label and entry field for entering product id #####

    buy_user_label=Label(buyproductFrame,text='User                                     Id
',font=('Helvetica',17,'italic'),bg='#A2A2B5',fg='#F5F5FF')

    buy_user_label.place(x=50,y=150)

buy_user_entry=Entry(buyproductFrame,highlightthickness=0,relief=FLAT,fg="black",font=('Helvet
ica',12,'italic'))

    buy_user_entry.place(x=225,y=150,width=230,height=35)

    ##### creating label and entry field for entering product id #####

    qtylabel=Label(buyproductFrame,text='Quantity
',font=('Helvetica',17,'italic'),bg='#A2A2B5',fg='#F5F5FF')

    qtylabel.place(x=50,y=225)

qtyentry=Entry(buyproductFrame,highlightthickness=0,relief=FLAT,fg="black",font=('Helvetica',12,'
italic'))

    qtyentry.place(x=225,y=225,width=230,height=35)

    ##### creating buy button #####

buy_btn=Button(buyproductFrame,text='BUY',bd='5',cursor='hand2',command=buy,font=btn_font,wi
dth=20,bg='#F5F5FF',fg='#A2A2B5')

    buy_btn.place(x=275,y=300)

def ViewBillFrame():

    ##### creating frame #####

viewbillFrame=Frame(userwindow,width=1030,height=575,bg='#A2A2B5',bd=10,relief=RIDGE)

```

```

viewbillFrame.place(x=225,y=200)
def view():
    #####input from user#####
    userid=userentry.get()
    ##### fetching values from the database and displaying in the frame #####
    cursor.execute("SELECT  p.Product_id,p.Price,b.Quantity_saled,b.date_sold  FROM
Product p INNER JOIN Bill b ON p.Product_id=b.Product_id WHERE b.User_id="+userid)
    res=cursor.fetchall()
    cursor.execute("commit")
    for i in res:
        tree.insert(parent="",index='end',values=(i[0],i[1],i[2],i[3]))
    ##### calculating total #####
    total=0
    for i in res:
        total=total+(i[1]*i[2])
    toatlentry.insert(END,total)
    cursor.execute("SELECT address FROM User WHERE user_id="+userentry.get()+"")
    rs1=cursor.fetchall()
    cursor.execute("commit")
    messagebox.showinfo("Product Status ","Product will be shipped to "+rs1[0][0]+" in 1
hour !! ")
    userentry.delete(0,END)
    ##### instance of style widget #####
    style = ttk.Style()
    style.theme_use('clam')
    ##### creating tree frame to display bill #####
    tree =ttk.Treeview(viewbillFrame, column=("c1", "c2","c3","c4"), show='headings',
height=20)
    ##### aligning columns #####
    tree.column("# 1", anchor=CENTER)
    tree.heading("# 1", text="Product ID")
    tree.column("# 2", anchor=CENTER)
    tree.heading("# 2", text="Price")
    tree.column("# 3", anchor=CENTER)
    tree.heading("# 3", text="Quantity")

```



```

tree.column("# 4", anchor=CENTER)

tree.heading("# 4", text="Date")

tree.place(x=5,y=0)

##### creating label and text field for displaying total #####

totallabel=Label(viewbillFrame,text='Total
',font=('Helvetica',17,'italic'),bg='#A2A2B5',fg='#F5F5FF')

totallabel.place(x=50,y=500)

toatlentry=Entry(viewbillFrame,highlightthickness=0,relief=FLAT,fg="black",font=('Helvetica',12,'it
alic'))

toatlentry.place(x=225,y=500,width=230,height=35)

##### creating label and entry field for user to enter userid #####

userlabel=Label(viewbillFrame,text='User                                     id
',font=('Helvetica',17,'italic'),bg='#A2A2B5',fg='#F5F5FF')

userlabel.place(x=50,y=450)

userentry=Entry(viewbillFrame,highlightthickness=0,relief=FLAT,fg="black",font=('Helvetica',12,'ita
lic'))

userentry.place(x=225,y=450,width=230,height=35)

##### creating buy button #####

buy_btn=Button(viewbillFrame,text='VIEW',bd='5',cursor='hand2',command=view,font=btn_font,wi
dth=20,bg='#F5F5FF',fg='#A2A2B5')

buy_btn.place(x=470,y=475)

##### button to check products available #####

check_products_btn=Button(buttonFrame,text='ViewProducts',bd='3',cursor='hand2',command=View
ProductFrame,font=btn_font,width=10,bg='#A2A2B5',fg='#F5F5FF')

check_products_btn.place(x=25,y=20)

##### button to buy products #####

buy_product_btn=Button(buttonFrame,text='BuyProducts',bd='3',cursor='hand2',command=BuyProdu
ctFrame,font=btn_font,width=10,bg='#A2A2B5',fg='#F5F5FF')

buy_product_btn.place(x=170,y=20)

##### button to view bill #####

view_bill_btn=Button(buttonFrame,text='View
Bill',bd='3',cursor='hand2',command=ViewBillFrame,font=btn_font,width=10,bg='#A2A2B5',fg='#F5
F5FF')

view_bill_btn.place(x=310,y=20)

```

```

##### button to logout #####

logout_btn=Button(buttonFrame,text='Logout',bd='3',cursor='hand2',command=userwindow.destroy,font=btn_font,width=10,bg='#A2A2B5',fg='#F5F5FF')

logout_btn.place(x=455,y=20)

else:

    ##### showing warning message for invalid password #####

    messagebox.showwarning("Message","Invalid password")

##### creating user login page #####

userlogin = Toplevel(root)

userlogin.title("User login page")

userlogin.minsize(1000,900)

userlogin.state('zoomed')

##### setting a image in background #####

bg_frame=Image.open("C:\\Users\\maha9\\OneDrive\\Documents\\Dbms&py_miniproj\\bg3.jpg")

photo=ImageTk.PhotoImage(bg_frame)

bg_panel=Label(userlogin,image=photo)

bg_panel.image=photo

bg_panel.pack(fill='both',expand='yes')

##### login frame #####

##### creating a login frame #####

login_frame=Frame(userlogin,width='900',height=600)

login_frame.place(x=50,y=125)

##### giving a heading for a login frame #####

txt='USER LOGIN'

txt1='MILK AND DAIRY PRODUCTS MANAGEMENT'

heading=Label(login_frame,text=txt,font=('Helvetica',22,'italic'),fg='black')

heading1=Label(login_frame,text=txt1,font=('Helvetica',25,'italic'),fg='black')

heading1.place(x=5,y=5,width=750,height=30)

heading.place(x=10,y=50,width=300,height=30)

##### placing a image in the login frame #####

sign_in_image=Image.open("C:\\Users\\maha9\\OneDrive\\Documents\\Dbms&py_miniproj\\usericonlogin.png")

photo=ImageTk.PhotoImage(sign_in_image)

```

```

sign_in_image_label=Label(login_frame,image=photo)
sign_in_image_label.image=photo
sign_in_image_label.place(x=670,y=50)
#####placing 'sign in 'text #####
sign_in_label=Label(login_frame,text='Sign In',fg='black',font=('Helvetica',17,'italic'))
sign_in_label.place(x=635,y=175)
##### creating username label #####
username_label=Label(login_frame,text='Username',font=('Helvetica',17,'italic'),fg='black')
username_label.place(x=475,y=225)
##### creating textfield for username entry #####

username_entry=Entry(login_frame,highlightthickness=0,relief=FLAT,fg="black",font=('Helvetica',1
2,'italic'))
username_entry.place(x=600,y=225,width=230,height=35)
##### creating password label #####
password_label=Label(login_frame,text='Password',font=('Helvetica',17,'italic'),fg='black')
password_label.place(x=475,y=300)
##### creating textfield for password entry #####

password_entry=Entry(login_frame,highlightthickness=0,relief=FLAT,fg="black",font=('Helvetica',1
2,'italic'))
password_entry.place(x=600,y=300,width=230,height=35)
##### login button creation #####

login_btn=Button(login_frame,text='Login',bd='5',cursor='hand2',command=openUserWindow,font=
btn_font,width=20)
login_btn.place(x=545,y=370)
#####WORKER LOGIN#####
def openWorkerLogin():
    btn_font=f.Font(family='Helvetica',size=16,weight='normal',slant='roman')
    ##### creating worker window #####
    def openWorkerWindow():
        #####checking if admin in database #####
        cursor.execute("SELECT          COUNT(*)          FROM          Worker          WHERE
worker_name='"+username_entry.get()+"'")
        row=cursor.fetchall()

```

```

cursor.execute("commit")
if row[0][0]==0:
    messagebox.showinfo("Message","Worker not found !!!")
    return
if password_entry.get()=="":
    messagebox.showinfo("Message","Enter your password")
else:
    ##### selecting admin password from database to check the password validity #####
    cursor.execute("SELECT      Worker_password      FROM      Worker      WHERE
Worker_name='"+username_entry.get()+"'")
    rows=cursor.fetchall()
    cursor.execute("commit")
    ##### checking password #####
    if(rows[0][0]==password_entry.get()):
        messagebox.showinfo("Message","Login successfull")
        password_entry.delete(0,END)
        username_entry.delete(0,END)
        workerwindow=Toplevel(workerlogin)
        workerwindow.title("Worker Window")
        workerwindow.minsize(1000,900)
        workerwindow.state('zoomed')
        #####Top frame that holds the heading of the admin window #####
        topFrame=Frame(workerwindow,bd=10,relief=RIDGE,bg='#3B3B3B')#Gray23
        topFrame.pack(side=TOP)
        ##### adding label to the top frame created #####
        labelTitle=Label(topFrame,text='MILK  AND  DAIRY  PRODUCTS  MANAGEMENT
SYSTEM',font=('arial',30,'bold'),fg='#3B3B3B',width=51)
        labelTitle.grid(row=0,column=0)
        ##### creating a frame to add buttons #####

buttonFrame=Frame(workerwindow,bd=10,bg='#3B3B3B',width=650,height=100,relief=RIDGE)
        buttonFrame.place(x=375,y=90)
        ##### creating frames for the button correspondingly #####
        def LiveStock_btnFrame():
            ##### creating frame #####

```

```
livestock_buttonFrame=Frame(workerwindow,bd=10,bg='#3B3B3B',width=650,height=100,relief=RIDGE)
```

```
    livestock_buttonFrame.place(x=375,y=200)
```

```
##### creating frame for buttons in LiveStock_btnFrame #####
```

```
##### frame for adding livestock details #####
```

```
def AddLiveStock_btnFrame():
```

```
    ##### creating frame #####
```

```
add_buttonFrame=Frame(workerwindow,bd=10,bg='#3B3B3B',width=1030,height=475,relief=RIDGE)
```

```
    add_buttonFrame.place(x=225,y=300)
```

```
def add():
```

```
    livestock_id=livestock_id_entry.get()
```

```
    livestocktype=livestock_type_entry.get()
```

```
    capacity=capacity_entry.get()
```

```
    if livestock_id=="" or livestocktype=="" or capacity=="":
```

```
        messagebox.showinfo("Insert status","All fields are required for adding !!!")
```

```
    elif livestock_id.isnumeric()==False:
```

```
        messagebox.showinfo("Insert status","Enter valid id")
```

```
    else:
```

```
        cursor.execute("INSERT INTO LiveStock  
VALUES("+livestock_id+", "+livestocktype+", "+capacity+")")
```

```
        cursor.execute("commit")
```

```
        messagebox.showinfo("Insert status","Livestock added !!!")
```

```
        livestock_id_entry.delete(0,END)
```

```
        livestock_type_entry.delete(0,END)
```

```
        capacity_entry.delete(0,END)
```

```
##### creating label for user convience #####
```

```
txt='Add LiveStock Details'
```

```
heading=Label(add_buttonFrame,text=txt,font=('Helvetica',20,'italic'),bg='#3B3B3B',fg='white')
```

```
    heading.place(x=55,y=20,width=500,height=30)
```

```
##### creating label and entry field for entering Livestock id #####
```

```
    livestock_id_label=Label(add_buttonFrame,text='Livestock Id',font=('Helvetica',17,'italic'),bg='#3B3B3B',fg='white')
```

```

        livestock_id_label.place(x=50,y=75)

livestock_id_entry=Entry(add_buttonFrame,highlightthickness=0,relief=FLAT,font=('Helvetica',12,'italic'))

        livestock_id_entry.place(x=225,y=75,width=230,height=35)

        ##### creating label and entry field for entering LiveStock type #####

        livestock_type_label=Label(add_buttonFrame,text='LiveStock                                Type',font=('Helvetica',17,'italic'),bg='#3B3B3B',fg='white')

        livestock_type_label.place(x=50,y=130)

livestock_type_entry=Entry(add_buttonFrame,highlightthickness=0,relief=FLAT,font=('Helvetica',12,'italic'))

        livestock_type_entry.place(x=225,y=130,width=230,height=35)

        ##### creating label and entry field for entering capacity of milk #####

        capacity_label=Label(add_buttonFrame,text='Capacity                                of                                milk',font=('Helvetica',17,'italic'),bg='#3B3B3B',fg='white')

        capacity_label.place(x=50,y=185)

capacity_entry=Entry(add_buttonFrame,highlightthickness=0,relief=FLAT,font=('Helvetica',12,'italic'))

        capacity_entry.place(x=225,y=185,width=230,height=35)

        ##### creating add button #####

add_btn=Button(add_buttonFrame,text='ADD',bd='5',command=add,cursor='hand2',font=btn_font,width=20,fg='#3B3B3B')

        add_btn.place(x=300,y=350)

        ##### creating frame to update livestock details #####

def updateLivestock_buttonFrame():

    ##### creating frame #####

update_buttonFrame=Frame(workerwindow,bd=10,bg='#3B3B3B',width=1030,height=475,relief=RIDGE)

        update_buttonFrame.place(x=225,y=300)

def update():

    livestock_id=livestock_id_entry.get()

    capacity=capacity_entry.get()

    if livestock_id.isnumeric()==False:

        messagebox.showinfo("Update status","Enter valid id")

```

```

        return

        cursor.execute("SELECT COUNT(*) FROM LiveStock WHERE
LiveStock_id="+livestock_id_entry.get())

        rows=cursor.fetchall()

        cursor.execute("commit")

        if livestock_id=="" or capacity=="":

            messagebox.showinfo("Update status","All fields are required for Updating !!!!")

        elif(rows[0][0]==0):

            messagebox.showinfo("Update Status","Livestock id Not Found")

        else:

            cursor.execute("UPDATE LiveStock SET
Capacity_of_Milk="+capacity+"WHERE LiveStock_id="+livestock_id+"")

            cursor.execute("commit")

            messagebox.showinfo("Update status","Livestock Updated !!!")

            livestock_id_entry.delete(0,END)

            capacity_entry.delete(0,END)

            ##### creating label for user convience #####

            txt='Update LiveStock Details'

            heading=Label(update_buttonFrame,text=txt,font=('Helvetica',20,'italic'),bg='#3B3B3B',fg='white')

            heading.place(x=75,y=20,width=500,height=30)

            ##### creating label and entry field for entering LiveStock Id to be updated #####

            livestock_id_label=Label(update_buttonFrame,text='LiveStock Id
',font=('Helvetica',17,'italic'),bg='#3B3B3B',fg='white')

            livestock_id_label.place(x=50,y=75)

            livestock_id_entry=Entry(update_buttonFrame,highlightthickness=0,relief=FLAT,font=('Helvetica',1
2,'italic'))

            livestock_id_entry.place(x=225,y=75,width=230,height=35)

            ##### creating label and entry field for entering capacity #####

            capacity_label=Label(update_buttonFrame,text='Capacity of milk
',font=('Helvetica',17,'italic'),bg='#3B3B3B',fg='white')

            capacity_label.place(x=50,y=130)

            capacity_entry=Entry(update_buttonFrame,highlightthickness=0,relief=FLAT,font=('Helvetica',12,'ita
lic'))

            capacity_entry.place(x=225,y=130,width=230,height=35)

```

```

##### creating UPDATE button #####

update_btn=Button(update_buttonFrame,text='UPDATE',bd='5',command=update,cursor='hand2',font=btn_font,width=20,fg='#3B3B3B')

update_btn.place(x=305,y=350)

##### creating frame to delete livestock details #####

def deleteLivestock_buttonFrame():

    ##### creating frame #####

del_buttonFrame=Frame(workerwindow,bd=10,bg='#3B3B3B',width=1030,height=475,relief=RIDGE)

del_buttonFrame.place(x=225,y=300)

def delete():

    livestock_id=livestock_id_entry.get()

    if livestock_id.isnumeric()==False:

        messagebox.showinfo("Delete status","Enter valid id")

        return

    cursor.execute("SELECT COUNT(*) FROM LiveStock WHERE LiveStock_id="+livestock_id_entry.get())

    rows=cursor.fetchall()

    cursor.execute("commit")

    if livestock_id=="":

        messagebox.showinfo("Delete status","All fields are required for Deleting !!!!")

    elif(rows[0][0]==0):

        messagebox.showinfo("Delete Status","Livestock id Not Found")

    else:

        cursor.execute("DELETE FROM LiveStock WHERE LiveStock_id="+livestock_id+"")

        cursor.execute("commit")

        messagebox.showinfo("Delete status","Livestock Deleted !!!!")

        livestock_id_entry.delete(0,END)

    ##### creating label for user convience #####

    txt='Delete LiveStock Details'

heading=Label(del_buttonFrame,text=txt,font=('Helvetica',20,'italic'),bg='#3B3B3B',fg='white')

heading.place(x=75,y=20,width=500,height=30)

```



```

##### creating label and entry field for entering LiveStock Id to be deleted #####
livestock_id_label=Label(del_buttonFrame,text='LiveStock Id
',font=('Helvetica',17,'italic'),bg='#3B3B3B',fg='white')

livestock_id_label.place(x=50,y=75)

livestock_id_entry=Entry(del_buttonFrame,highlightthickness=0,relief=FLAT,font=('Helvetica',12,'it
alic'))

livestock_id_entry.place(x=225,y=75,width=230,height=35)

##### creating DELETE button #####

del_btn=Button(del_buttonFrame,text='DELETE',bd='5',cursor='hand2',command=delete,font=btn_fo
nt,width=20,fg='#3B3B3B')

del_btn.place(x=225,y=350)

##### creating frame to display livestock details #####
def ViewLivestockDetailsFrame():
    ##### creating frame #####

ViewLivestock=Frame(workerwindow,width=1030,height=475,bg='#3B3B3B',bd=10,relief=RIDGE)

ViewLivestock.place(x=225,y=300)

##### instance of style widget #####
style = ttk.Style()

style.theme_use('clam')

##### creating tree frame to display livestock details #####
tree =ttk.Treeview(ViewLivestock, column=("c1", "c2","c3"), show='headings',
height=20)

##### aligning columns #####
tree.column("# 1", anchor=CENTER)
tree.heading("# 1", text="Livestock ID")
tree.column("# 2", anchor=CENTER)
tree.heading("# 2", text="Livestock type")
tree.column("# 3", anchor=CENTER)
tree.heading("# 3", text="Capacity of milk")
tree.place(x=200,y=0)

##### fetching values from the database and displaying in the frame #####
cursor.execute("SELECT * FROM LiveStock")
res=cursor.fetchall()

```

```

        for i in res:
            tree.insert(parent="",index='end',values=(i[0],i[1],i[2]))
        cursor.execute("commit")

##### button for adding livestock #####

add_btn=Button(livestock_buttonFrame,text='ADD',bd='3',cursor='hand2',command=AddLiveStock_
btnFrame,font=btn_font,width=10,fg='#3B3B3B')

add_btn.place(x=5,y=20)

##### button for updating livestock #####

update_btn=Button(livestock_buttonFrame,text='UPDATE',bd='3',cursor='hand2',command=updateLi
vestock_buttonFrame,font=btn_font,width=10,fg='#3B3B3B')

update_btn.place(x=145,y=20)

##### button for deleting livestock #####

del_btn=Button(livestock_buttonFrame,text='DELETE',bd='3',cursor='hand2',command=deleteLivest
ock_buttonFrame,font=btn_font,width=10,fg='#3B3B3B')

del_btn.place(x=290,y=20)

##### button for viewing livestock details #####

view_btn=Button(livestock_buttonFrame,text='VIEW',bd='3',cursor='hand2',command=ViewLivestoc
kDetailsFrame,font=btn_font,width=10,fg='#3B3B3B')

view_btn.place(x=435,y=20)

def FeedMonitor_btnFrame():

##### creating frame #####

FeedMonitor_buttonFrame=Frame(workerwindow,bd=10,bg='#3B3B3B',width=650,height=100,relie
f=RIDGE)

FeedMonitor_buttonFrame.place(x=375,y=200)

##### creating frame for buttons in LiveStockDiagnosis_btnFrame #####

##### frame for adding livestock feedmonitor details #####

def AddLiveStock_btnFrame():

##### creating frame #####

add_buttonFrame=Frame(workerwindow,bd=10,bg='#3B3B3B',width=1030,height=475,relief=RIDG
E)

add_buttonFrame.place(x=225,y=300)

def add():

```

```

livestock_id=livestock_id_entry.get()
worker_id=Worker_id_entry.get()
feedtype=feedtype_entry.get()
if livestock_id=="" or worker_id=="" or feedtype=="":
    messagebox.showinfo("Insert status","All fields are required for adding !!!!")
elif livestock_id.isnumeric()==False:
    messagebox.showinfo("Insert status","Enter valid livestock id")
elif worker_id.isnumeric()==False:
    messagebox.showinfo("Insert status","Enter valid worker id")
    cursor.execute("SELECT COUNT(*) FROM LiveStock WHERE
LiveStock_id="+livestock_id_entry.get())
    rows=cursor.fetchall()
    cursor.execute("commit")
    cursor.execute("SELECT COUNT(*) FROM Worker WHERE
Worker_id="+Worker_id_entry.get())
    row=cursor.fetchall()
    cursor.execute("commit")
    if(rows[0][0]==0):
        messagebox.showinfo("Insert Status","Livestock id Not Found")
    elif(row[0][0]==0):
        messagebox.showinfo("Insert Status","Worker id Not Found")
    else:
        cursor.execute("INSERT INTO Feed_monitoring
VALUES("+worker_id+"",""+livestock_id+"",""+feedtype+"")")
        cursor.execute("commit")
        messagebox.showinfo("Insert status","Feedtype added !!!")
        livestock_id_entry.delete(0,END)
        Worker_id_entry.delete(0,END)
        feedtype_entry.delete(0,END)
##### creating label for user convience #####
txt='Add LiveStock FeedMonitoring Details'

heading=Label(add_buttonFrame,text=txt,font=('Helvetica',20,'italic'),bg='#3B3B3B',fg='white')
heading.place(x=55,y=20,width=500,height=30)
##### creating label and entry field for entering Livestock id #####

```

```

        livestock_id_label=Label(add_buttonFrame,text='Livestock
',font=('Helvetica',17,'italic'),bg='#3B3B3B',fg='white')
        livestock_id_label.place(x=50,y=75)

livestock_id_entry=Entry(add_buttonFrame,highlightthickness=0,relief=FLAT,font=('Helvetica',12,'it
alic'))

        livestock_id_entry.place(x=225,y=75,width=230,height=35)
        ##### creating label and entry field for entering Worker ID #####

        Worker_id_label=Label(add_buttonFrame,text='Worker
',font=('Helvetica',17,'italic'),bg='#3B3B3B',fg='white')
        Worker_id_label.place(x=50,y=130)

Worker_id_entry=Entry(add_buttonFrame,highlightthickness=0,relief=FLAT,font=('Helvetica',12,'ita
lic'))

        Worker_id_entry.place(x=225,y=130,width=230,height=35)
        ##### creating label and entry field for entering feed type #####

        feedtype_label=Label(add_buttonFrame,text='Feedtype
',font=('Helvetica',17,'italic'),bg='#3B3B3B',fg='white')
        feedtype_label.place(x=50,y=185)

feedtype_entry=Entry(add_buttonFrame,highlightthickness=0,relief=FLAT,font=('Helvetica',12,'italic
'))

        feedtype_entry.place(x=225,y=185,width=230,height=35)
        ##### creating add button #####

add_btn=Button(add_buttonFrame,text='ADD',bd='5',command=add,cursor='hand2',font=btn_font,wi
dth=20,fg='#3B3B3B')

        add_btn.place(x=300,y=350)
        ##### creating frame to update livestock feed monitor details #####

def updateLivestock_buttonFrame():
    ##### creating frame #####

update_buttonFrame=Frame(workerwindow,bd=10,bg='#3B3B3B',width=1030,height=475,relief=RI
DGE)

        update_buttonFrame.place(x=225,y=300)
        def update():
            livestock_id=livestock_id_entry.get()
            feedtype=feedtype_entry.get()
            if livestock_id=="" or feedtype=="":

```

```

        messagebox.showinfo("Update status","All fields are required for Updating !!!!")
    if livestock_id.isnumeric()==False:
        messagebox.showinfo("Update status","Enter valid id")
        return

    cursor.execute("SELECT COUNT(*) FROM Feed_monitoring WHERE
LiveStock_id="+livestock_id_entry.get())

    rows=cursor.fetchall()
    cursor.execute("commit")
    if(rows[0][0]==0):
        messagebox.showinfo("Update Status","Livestock id Not Found")
    else:
        cursor.execute("UPDATE Feed_monitoring SET
Feed_type='"+feedtype+" WHERE LiveStock_id='"+livestock_id+"'")
        cursor.execute("commit")
        messagebox.showinfo("Update status","Feedtype Updated !!!")
        livestock_id_entry.delete(0,END)
        feedtype_entry.delete(0,END)

        ##### creating label for user convience #####
        txt='Update LiveStock FeedMonitoring Details'

        heading=Label(update_buttonFrame,text=txt,font=('Helvetica',20,'italic'),bg='#3B3B3B',fg='white')
        heading.place(x=75,y=20,width=500,height=30)

        ##### creating label and entry field for entering LiveStock Id to be updated #####
        livestock_id_label=Label(update_buttonFrame,text='LiveStock Id
',font=('Helvetica',17,'italic'),bg='#3B3B3B',fg='white')
        livestock_id_label.place(x=50,y=75)

        livestock_id_entry=Entry(update_buttonFrame,highlightthickness=0,relief=FLAT,font=('Helvetica',1
2,'italic'))
        livestock_id_entry.place(x=225,y=75,width=230,height=35)

        ##### creating label and entry field for entering feedtype #####
        feedtype_label=Label(update_buttonFrame,text='Feedtype
',font=('Helvetica',17,'italic'),bg='#3B3B3B',fg='white')
        feedtype_label.place(x=50,y=130)

        feedtype_entry=Entry(update_buttonFrame,highlightthickness=0,relief=FLAT,font=('Helvetica',12,'it
alic'))

```

```

feedtype_entry.place(x=225,y=130,width=230,height=35)

##### creating UPDATE button #####

update_btn=Button(update_buttonFrame,text='UPDATE',bd='5',command=update,cursor='hand2',font=btn_font,width=20,fg='#3B3B3B')

update_btn.place(x=305,y=350)

##### creating frame to display livestock feedmonitor details #####

def ViewLivestockFeedMonitorDetailsFrame():

    ##### creating frame #####

ViewLivestock=Frame(workerwindow,width=1030,height=475,bg='#3B3B3B',bd=10,relief=RIDGE)

ViewLivestock.place(x=225,y=300)

##### instance of style widget #####

style = ttk.Style()

style.theme_use('clam')

##### creating tree frame to display livestock diagonsis #####

tree =ttk.Treeview(ViewLivestock, column=("c1", "c2","c3"), show='headings',
height=20)

##### aligning columns #####

tree.column("# 1", anchor=CENTER)

tree.heading("# 1", text="Worker ID")

tree.column("# 2", anchor=CENTER)

tree.heading("# 2", text="Livestock ID")

tree.column("# 3", anchor=CENTER)

tree.heading("# 3", text="Feedtype")

tree.place(x=200,y=0)

##### fetching values from the database and displaying in the frame #####

cursor.execute("SELECT * FROM Feed_monitoring")

res=cursor.fetchall()

for i in res:

    tree.insert(parent="",index='end',values=(i[0],i[1],i[2]))

cursor.execute("commit")

##### button for adding new livestock feed monitoring #####

add_btn=Button(FeedMonitor_buttonFrame,text='ADD',bd='3',cursor='hand2',command=AddLiveStock_btnFrame,font=btn_font,width=10,fg='#3B3B3B')

```

```

add_btn.place(x=100,y=20)

##### button for updating livestock feedtype #####

update_btn=Button(FeedMonitor_buttonFrame,text='UPDATE',bd='3',cursor='hand2',command=updateLivestock_buttonFrame,font=btn_font,width=10,fg='#3B3B3B')

update_btn.place(x=250,y=20)

##### button for viewing livestock feedtype details #####

view_btn=Button(FeedMonitor_buttonFrame,text='VIEW',bd='3',cursor='hand2',command=ViewLivestockFeedMonitorDetailsFrame,font=btn_font,width=10,fg='#3B3B3B')

view_btn.place(x=400,y=20)

def LiveStockDiagnosis_btnFrame():

    ##### creating frame #####

LiveStockDiagnosis_buttonFrame=Frame(workerwindow,bd=10,bg='#3B3B3B',width=650,height=100,relief=RIDGE)

LiveStockDiagnosis_buttonFrame.place(x=375,y=200)

##### creating frame for buttons in LiveStockDiagnosis_btnFrame #####

##### frame for adding livestock diagnosis details #####

def AddLiveStock_btnFrame():

    ##### creating frame #####

add_buttonFrame=Frame(workerwindow,bd=10,bg='#3B3B3B',width=1030,height=475,relief=RIDGE)

add_buttonFrame.place(x=225,y=300)

def add():

    livestock_id=livestock_id_entry.get()

    worker_id=worker_id_entry.get()

    disease=disease_entry.get()

    dstatus=Disease_status_entry.get()

    vstatus=Vaccine_status_entry.get()

    expenses=Expenses_entry.get()

    if livestock_id=="" or worker_id=="" or disease=="" or dstatus=="" or vstatus=="" or expenses=="":

        messagebox.showinfo("Insert status","All fields are required for adding !!!")

    elif livestock_id.isnumeric()==False:

        messagebox.showinfo("Insert status","Enter valid livestock id")

```

```

else:

    cursor.execute("INSERT INTO LiveStockDiagnosis
VALUES('"+worker_id+"','"+livestock_id+"','"+disease+"','"+dstatus+"','"+expenses+"','"+vstatus+"')")

    cursor.execute("commit")

    messagebox.showinfo("Insert status","LiveStock details added !!!")

    livestock_id_entry.delete(0,END)

    worker_id_entry.delete(0,END)

    disease_entry.delete(0,END)

    Disease_status_entry.delete(0,END)

    Vaccine_status_entry.delete(0,END)

    Expenses_entry.delete(0,END)

    ##### creating label for user convience #####

    txt='Add LiveStock Diagnosis Details'

heading=Label(add_buttonFrame,text=txt,font=('Helvetica',20,'italic'),bg='#3B3B3B',fg='white')

    heading.place(x=50,y=20,width=400,height=30)

    ##### creating label and entry field for entering Worker id #####

    worker_id_label=Label(add_buttonFrame,text='Worker Id
',font=('Helvetica',17,'italic'),bg='#3B3B3B',fg='white')

    worker_id_label.place(x=50,y=75)

worker_id_entry=Entry(add_buttonFrame,highlightthickness=0,relief=FLAT,font=('Helvetica',12,'italic'))

    worker_id_entry.place(x=225,y=75,width=230,height=35)

    ##### creating label and entry field for entering LiveStock Id #####

    livestock_id_label=Label(add_buttonFrame,text='LiveStock Id
',font=('Helvetica',17,'italic'),bg='#3B3B3B',fg='white')

    livestock_id_label.place(x=50,y=130)

livestock_id_entry=Entry(add_buttonFrame,highlightthickness=0,relief=FLAT,font=('Helvetica',12,'italic'))

    livestock_id_entry.place(x=225,y=130,width=230,height=35)

    ##### creating label and entry field for entering Disease #####

    disease_label=Label(add_buttonFrame,text='Disease
',font=('Helvetica',17,'italic'),bg='#3B3B3B',fg='white')

    disease_label.place(x=50,y=185)

```



```

disease_entry=Entry(add_buttonFrame,highlightthickness=0,relief=FLAT,font=('Helvetica',12,'italic')
)

disease_entry.place(x=225,y=185,width=230,height=35)

##### creating label and entry field for entering Disease Status #####

Disease_status_label=Label(add_buttonFrame,text='Disease Status',font=('Helvetica',17,'italic'),bg='#3B3B3B',fg='white')

Disease_status_label.place(x=50,y=240)

Disease_status_entry=Entry(add_buttonFrame,highlightthickness=0,relief=FLAT,font=('Helvetica',12,'italic'))

Disease_status_entry.place(x=225,y=240,width=230,height=35)

##### creating label and entry field for entering Vaccine Status #####

Vaccine_status_label=Label(add_buttonFrame,text='Vaccine Status',font=('Helvetica',17,'italic'),bg='#3B3B3B',fg='white')

Vaccine_status_label.place(x=50,y=295)

Vaccine_status_entry=Entry(add_buttonFrame,highlightthickness=0,relief=FLAT,font=('Helvetica',12,'italic'))

Vaccine_status_entry.place(x=225,y=295,width=230,height=35)

##### creating label and entry field for entering Expenses #####

Expenses_label=Label(add_buttonFrame,text='Expense',font=('Helvetica',17,'italic'),bg='#3B3B3B',fg='white')

Expenses_label.place(x=50,y=350)

Expenses_entry=Entry(add_buttonFrame,highlightthickness=0,relief=FLAT,font=('Helvetica',12,'italic'))

Expenses_entry.place(x=225,y=350,width=230,height=35)

##### creating add button #####

add_btn=Button(add_buttonFrame,text='ADD',bd=5,command=add,cursor='hand2',font=btn_font,width=20,fg='#3B3B3B')

add_btn.place(x=375,y=400)

##### creating frame to update livestock details #####

def updateLivestock_buttonFrame():

    ##### creating frame #####

update_buttonFrame=Frame(workerwindow,bd=10,bg='#3B3B3B',width=1030,height=475,relief=RIDGE)

```

```

update_buttonFrame.place(x=225,y=300)

##### function to update livestock details #####

def update():

    livestock_id=livestock_id_entry.get()

    dstatus=Disease_status_entry.get()

    vstatus=Vaccine_status_entry.get()

    Expenses=Expenses_entry.get()

    if livestock_id=="" or dstatus=="" or vstatus=="" or Expenses=="":

        messagebox.showinfo("Update status","All fields are required for Updating !!!!")

    if livestock_id.isdigit()==False:

        messagebox.showinfo("Update status","Enter valid id")

        return

    cursor.execute("SELECT COUNT(*) FROM LivestockDiagnosis WHERE
LiveStock_id="+livestock_id_entry.get())

    rows=cursor.fetchall()

    cursor.execute("commit")

    if(rows[0][0]==0):

        messagebox.showinfo("Update Status","Livestock id Not Found")

    else:

        cursor.execute("UPDATE LivestockDiagnosis SET
disease_status="+dstatus+"WHERE LiveStock_id="+livestock_id+"")

        cursor.execute("UPDATE LivestockDiagnosis SET
expenses="+Expenses+"WHERE LiveStock_id="+livestock_id+"")

        cursor.execute("UPDATE LivestockDiagnosis SET
vaccine_status="+vstatus+"WHERE LiveStock_id="+livestock_id+"")

        cursor.execute("commit")

        messagebox.showinfo("Update status","Livestock Updated !!!")

        if dstatus=="Death":

            cursor.execute("DELETE FROM Livestock WHERE
LiveStock_id="+livestock_id+"")

            cursor.execute("commit")

            livestock_id_entry.delete(0,END)

            Disease_status_entry.delete(0,END)

            Vaccine_status_entry.delete(0,END)

            Expenses_entry.delete(0,END)

##### creating label for user convience #####

```

```

txt='Update LiveStock Diagnosis Details'

heading=Label(update_buttonFrame,text=txt,font=('Helvetica',20,'italic'),bg='#3B3B3B',fg='white')
    heading.place(x=75,y=20,width=500,height=30)

##### creating label and entry field for entering LiveStock Id to be updated #####

livestock_id_label=Label(update_buttonFrame,text='LiveStock Id',font=('Helvetica',17,'italic'),bg='#3B3B3B',fg='white')
    livestock_id_label.place(x=50,y=75)

livestock_id_entry=Entry(update_buttonFrame,highlightthickness=0,relief=FLAT,font=('Helvetica',12,'italic'))
    livestock_id_entry.place(x=225,y=75,width=230,height=35)

##### creating label and entry field for entering Disease Status #####

Disease_status_label=Label(update_buttonFrame,text='Disease Status',font=('Helvetica',17,'italic'),bg='#3B3B3B',fg='white')
    Disease_status_label.place(x=50,y=130)

Disease_status_entry=Entry(update_buttonFrame,highlightthickness=0,relief=FLAT,font=('Helvetica',12,'italic'))
    Disease_status_entry.place(x=225,y=130,width=230,height=35)

##### creating label and entry field for entering Vaccine Status #####

Vaccine_status_label=Label(update_buttonFrame,text='Vaccine Status',font=('Helvetica',17,'italic'),bg='#3B3B3B',fg='white')
    Vaccine_status_label.place(x=50,y=185)

Vaccine_status_entry=Entry(update_buttonFrame,highlightthickness=0,relief=FLAT,font=('Helvetica',12,'italic'))
    Vaccine_status_entry.place(x=225,y=185,width=230,height=35)

##### creating label and entry field for entering Expenses #####

Expenses_label=Label(update_buttonFrame,text='Expense',font=('Helvetica',17,'italic'),bg='#3B3B3B',fg='white')
    Expenses_label.place(x=50,y=240)

Expenses_entry=Entry(update_buttonFrame,highlightthickness=0,relief=FLAT,font=('Helvetica',12,'italic'))
    Expenses_entry.place(x=225,y=240,width=230,height=35)

##### creating UPDATE button #####

```

```

update_btn=Button(update_buttonFrame,text='UPDATE',bd='5',command=update,cursor='hand2',font=btn_font,width=20,fg='#3B3B3B')

    update_btn.place(x=375,y=350)

##### creating frame to display livestock diagonsis details #####

def ViewLivestockDiagonsisDetailsFrame():

    ##### creating frame #####

ViewLivestock=Frame(workerwindow,width=1030,height=475,bg='#3B3B3B',bd=10,relief=RIDGE)

    ViewLivestock.place(x=225,y=300)

    ##### instance of style widget #####

    style = ttk.Style()

    style.theme_use('default')

    ##### creating tree frame to display livestock diagonsis #####

    tree =ttk.Treeview(ViewLivestock,      column=("c1",      "c2","c3","c4","c5"),
show='headings', height=20)

    ##### aligning columns #####

    tree.column("# 1", anchor=CENTER)

    tree.heading("# 1", text="Livestock ID")

    tree.column("# 2", anchor=CENTER)

    tree.heading("# 2", text="Disease")

    tree.column("# 3", anchor=CENTER)

    tree.heading("# 3", text="DiseaseStatus")

    tree.column("# 4", anchor=CENTER)

    tree.heading("# 4", text="VaccineStatus")

    tree.column("# 5", anchor=CENTER)

    tree.heading("# 5", text="Expenses")

    tree.place(x=5,y=0)

    cursor.execute("SELECT * FROM LiveStockDiagonsis")

    res=cursor.fetchall()

    for i in res:

        tree.insert(parent="",index='end',values=(i[1],i[2],i[3],i[5],i[4]))

    cursor.execute("commit")

##### button for adding new livestock diagonsis #####

```

```
add_btn=Button(LiveStockDiagonisis_buttonFrame,text='ADD',bd='3',cursor='hand2',command=Add
LiveStock_btnFrame,font=btn_font,width=10,fg='#3B3B3B')
```

```
add_btn.place(x=100,y=20)
```

```
##### button for updating livestock diagonsis #####
```

```
update_btn=Button(LiveStockDiagonisis_buttonFrame,text='UPDATE',bd='3',cursor='hand2',comma
nd=updateLivestock_btnFrame,font=btn_font,width=10,fg='#3B3B3B')
```

```
update_btn.place(x=250,y=20)
```

```
##### button for viewing livestock diagonsis details #####
```

```
view_btn=Button(LiveStockDiagonisis_buttonFrame,text='VIEW',bd='3',cursor='hand2',command=V
iewLivestockDiagonisisDetailsFrame,font=btn_font,width=10,fg='#3B3B3B')
```

```
view_btn.place(x=400,y=20)
```

```
##### button for livestock #####
```

```
livestock_btn=Button(buttonFrame,text='Livestock',bd='3',cursor='hand2',command=LiveStock_btnF
rame,font=btn_font,width=10,fg='#3B3B3B')
```

```
livestock_btn.place(x=25,y=20)
```

```
##### button for feedmonitor #####
```

```
feed_monitor_btn=Button(buttonFrame,text='FeedMonitor',bd='3',cursor='hand2',command=FeedMo
nitor_btnFrame,font=btn_font,width=10,fg='#3B3B3B')
```

```
feed_monitor_btn.place(x=170,y=20)
```

```
##### button to livestock diagonis check #####
```

```
diagonsis_btn=Button(buttonFrame,text='Diagnois',bd='3',cursor='hand2',command=LiveStockDiago
nisis_btnFrame,font=btn_font,width=10,fg='#3B3B3B')
```

```
diagonsis_btn.place(x=310,y=20)
```

```
##### button to logout #####
```

```
logout_btn=Button(buttonFrame,text='Logout',bd='3',cursor='hand2',command=workerwindow.destro
y,font=btn_font,width=10,fg='#3B3B3B')
```

```
logout_btn.place(x=455,y=20)
```

```
else:
```

```
##### showing warning message for invalid password #####
```

```
messagebox.showwarning("Message","Invalid password")
```

```
##### creating worker login page #####
```

```
workerlogin = Toplevel(root)
```

```

workerlogin.title("Worker login page")
workerlogin.minsize(1000,900)
workerlogin.state('zoomed')
##### setting background image #####
bg_frame=Image.open("C:\\Users\\maha9\\OneDrive\\Documents\\Dbms&py_miniproj\\bg4.jpg")
photo=ImageTk.PhotoImage(bg_frame)
bg_panel=Label(workerlogin,image=photo)
bg_panel.image=photo
bg_panel.pack(fill='both',expand='yes')
##### login frame #####
##### creating a login frame #####
login_frame=Frame(workerlogin,width='900',height=600)
login_frame.place(x=600,y=100)
##### giving a heading for a login frame #####
txt='WORKER LOGIN'
txt1='MILK AND DAIRY PRODUCTS MANAGEMENT'
heading=Label(login_frame,text=txt,font=('Helvetica',22,'italic'),fg='black')
heading1=Label(login_frame,text=txt1,font=('Helvetica',25,'italic'),fg='black')
heading1.place(x=5,y=5,width=750,height=30)
heading.place(x=10,y=50,width=300,height=30)
##### placing a image in the login frame #####

sign_in_image=Image.open("C:\\Users\\maha9\\OneDrive\\Documents\\Dbms&py_miniproj\\workeri
con.jpeg")
photo=ImageTk.PhotoImage(sign_in_image)
sign_in_image_label=Label(login_frame,image=photo)
sign_in_image_label.image=photo
sign_in_image_label.place(x=620,y=50)
#####placing 'sign in 'text #####
sign_in_label=Label(login_frame,text='Sign In',fg='black',font=('Helvetica',17,'italic'))
sign_in_label.place(x=635,y=175)
##### creating username label #####
username_label=Label(login_frame,text='Username',font=('Helvetica',17,'italic'),fg='black')
username_label.place(x=475,y=225)

```

```

##### creating textfield for username entry #####

username_entry=Entry(login_frame,highlightthickness=0,relief=FLAT,fg="black",font=('Helvetica',12,'italic'))

username_entry.place(x=600,y=225,width=230,height=35)

##### creating password label #####

password_label=Label(login_frame,text='Password',font=('Helvetica',17,'italic'),fg='black')

password_label.place(x=475,y=300)

##### creating textfield for password entry #####

password_entry=Entry(login_frame,highlightthickness=0,relief=FLAT,fg="black",font=('Helvetica',12,'italic'))

password_entry.place(x=600,y=300,width=230,height=35)

##### login button creation #####

login_btn=Button(login_frame,text='Login',bd='5',cursor='hand2',command=openWorkerWindow,font=btn_font,width=20)

login_btn.place(x=545,y=370)

##### login options #####

##### main window #####

root=Tk()

root.title("Login Page")

root.minsize(640,400)

##### setting background image #####

bg_frame=Image.open("C:\\Users\\maha9\\OneDrive\\Documents\\Dbms&py_miniproj\\bg1.jpg")

photo=ImageTk.PhotoImage(bg_frame)

bg_panel=Label(root,image=photo)

bg_panel.image=photo

bg_panel.pack(fill='both',expand='yes')

##### giving a heading for a login frame #####

txt=' DAIRY PRODUCTS MANAGEMENT SYSTEM'

heading=Label(bg_panel,text=txt,font=('Helvetica',20,'italic'))

heading.place(x=0,y=10,width=650,height=30)

##### creating admin login button #####

btn_font=f.Font(family='Helvetica',size=16,weight='normal',slant='roman')

btn1=Button(root,text='Admin Login',bd='5',command=openAdminLogin,font=btn_font)

```

```
btn1.place(x=250,y=75)
##### creating user login button #####
btn2=Button(root,text='User Login ',bd='5',command=openUserLogin,font=btn_font)
btn2.place(x=250,y=175)
##### creating worker login button #####
btn3=Button(root,text='Worker Login',bd='5',command=openWorkerLogin,font=btn_font)
btn3.place(x=250,y=275)
root.mainloop()
##### closing the connection to the database #####
con.close()
```



## CHAPTER 6

### RESULTS

#### Output Screenshots and Explanation

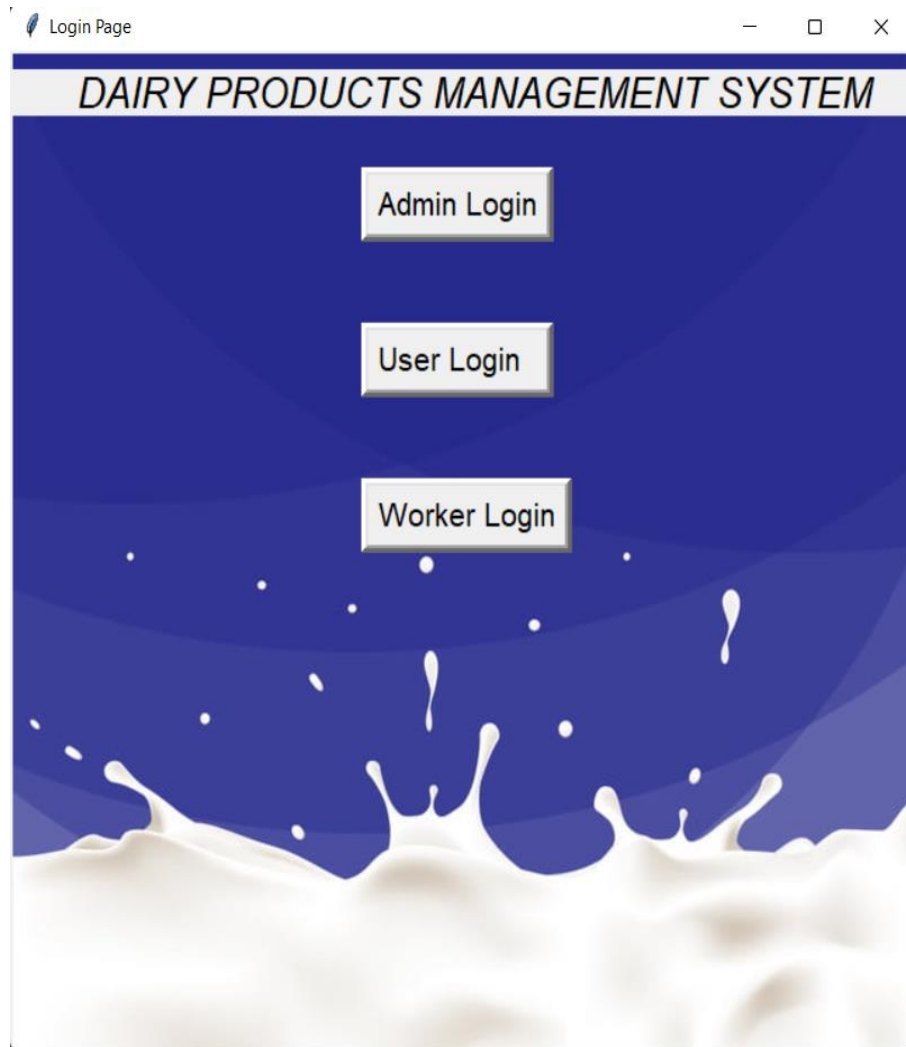
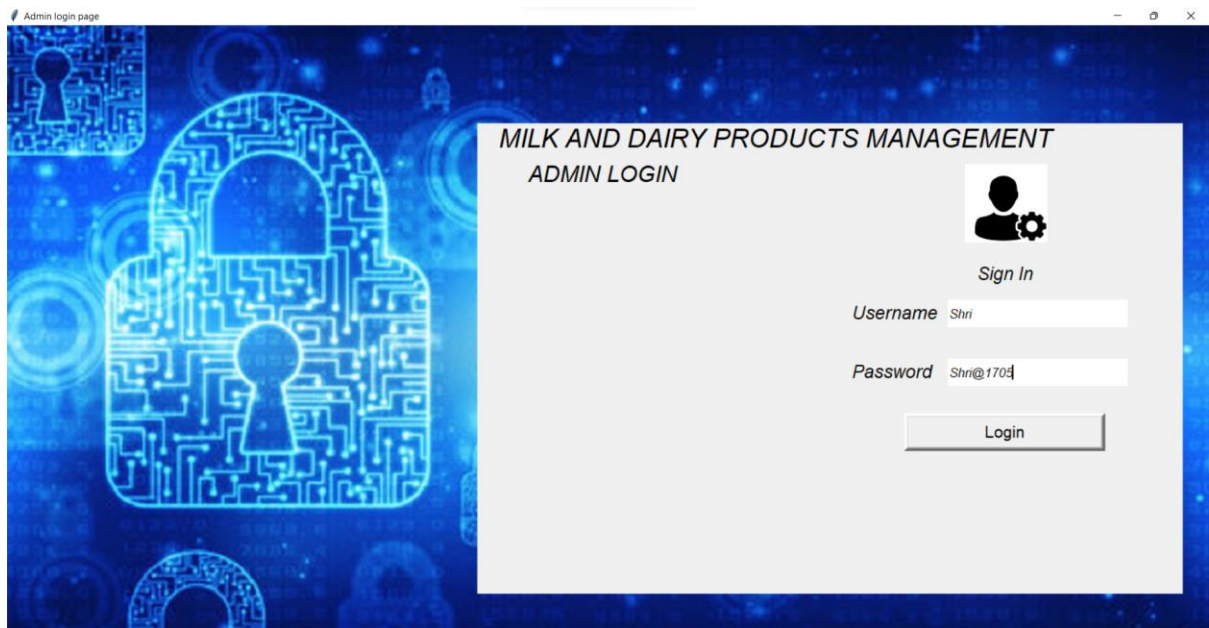


Fig 6.1: Login page options

Login options page displayed in the Fig 6.1. is the main window of the project. This window has the choices for admin, user and worker login.



**Fig 6.2: Admin login window**

If the admin login button is selected in the login page options, then this admin login window is displayed which is illustrated in Fig 6.2.



**Fig 6.3: Admin window**

If the admin logs in with correct credentials, the admin window displayed in the Fig 6.3 will be displayed. In this window the admin can add a user, delete a user, update a user, add a worker, delete a work, update a worker and edit the

product on daily basis based on the quantity available and up-to-date price.



Fig 6.4: Tree view in Admin window

And the admin can view the overall data of the products, users and the workers via tree view in the window which is illustrated in the Fig 6.4.

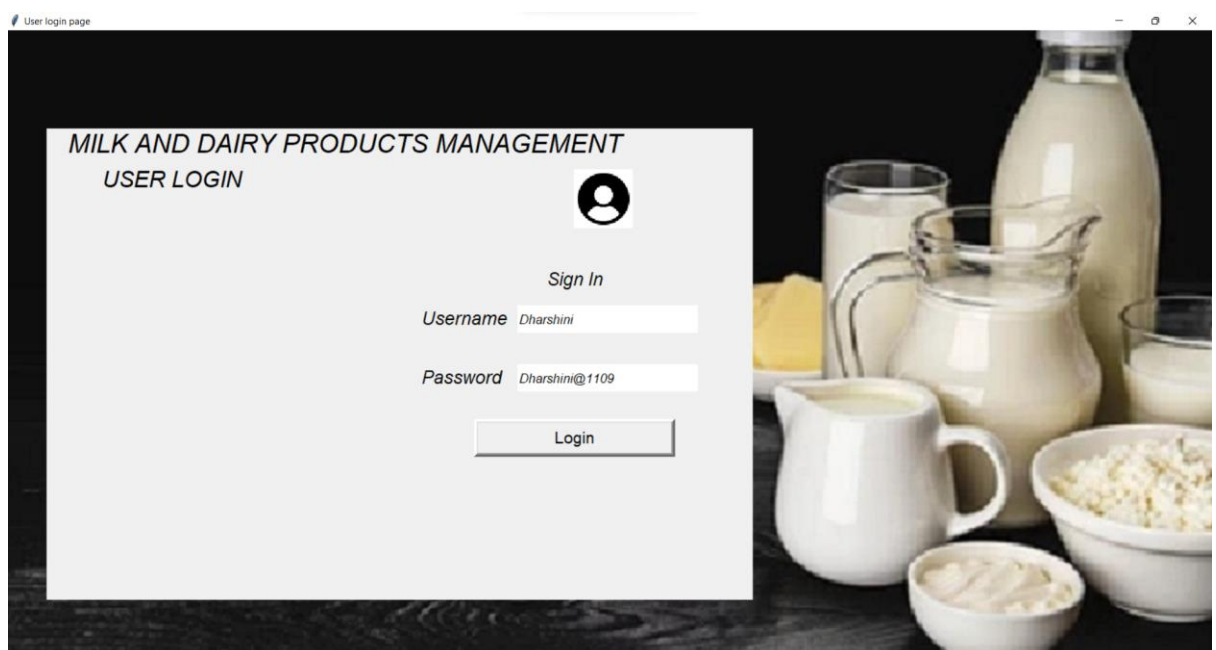


Fig 6.5: User Login page

If the User login button is selected in the login page options, then this user login window is displayed which is illustrated in Fig 6.5.

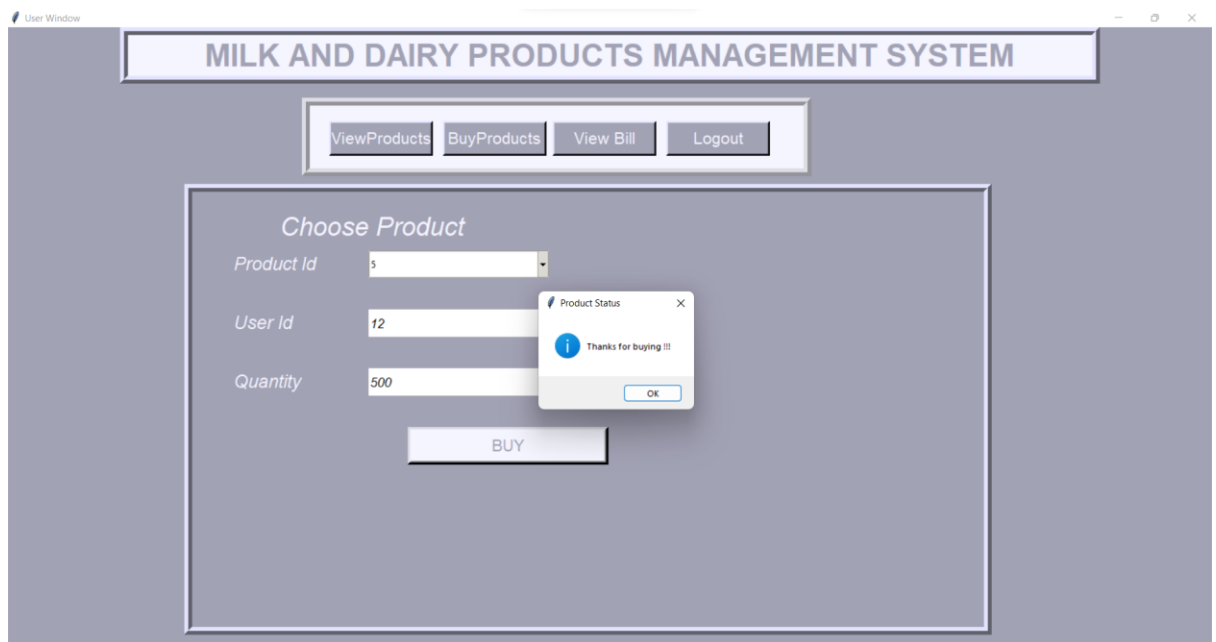


Fig 6.6: User Window

If the user logs in with correct credentials, the user window illustrated in the Fig 6.6 will be displayed. In this window the user can view the details regarding the products available, can continue to buy products.

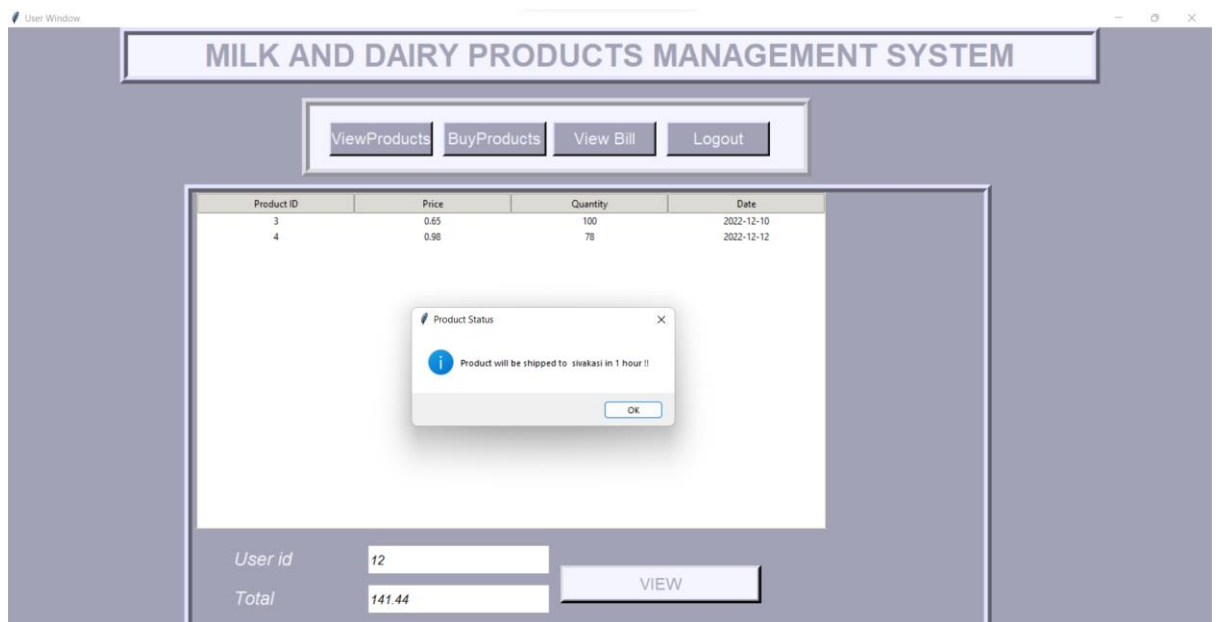


Fig 6.7: Bill

Also the user can view the bill and total amount for the purchased products.

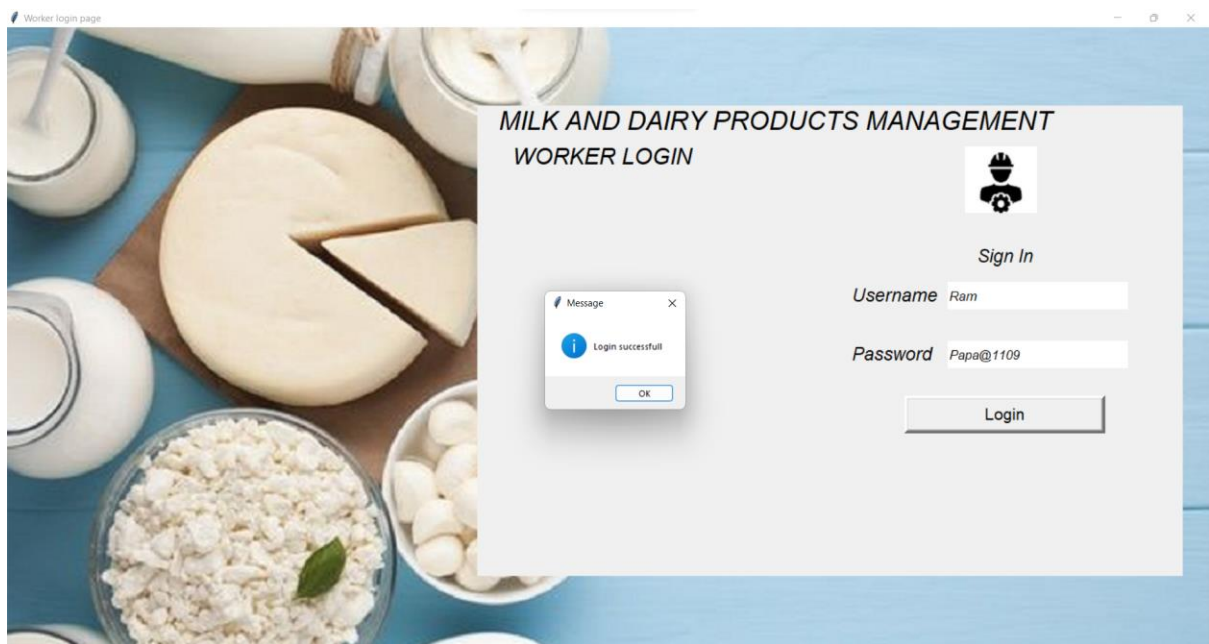


Fig 6.8: Worker login page

If the Worker login button is selected in the login page options, then this Worker login window is displayed which is illustrated in Fig 6.6.

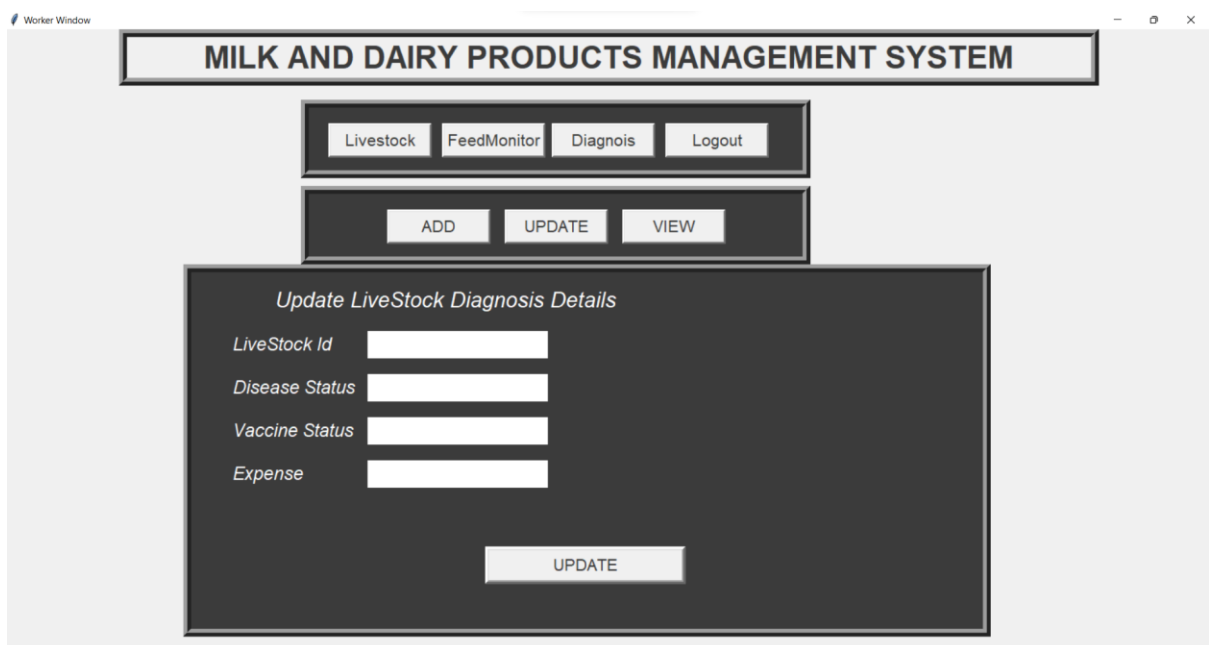


Fig 6.9: Worker window

If the worker logs in with correct credentials, the worker window illustrated in the Fig 6.9 will be displayed. In this window the admin can add a livestock,

delete a livestock, update a livestock, add a feed monitoring detail, delete a feed monitoring detail, update a feed monitoring detail and the same for livestock diagnosis, if a livestock's disease status is updated to be death, then the particular livestock will automatically deleted from the database record.

## **CHAPTER 7**

### **CONCLUSION**

- The web application has a user-friendly interface and is easy to use.
- It can be accessed by dairy managers and admin at any time in any Platform
- Keeps the track of transaction done and updates the quantity of product once a product is sold.
- Reduces manual work and time consumption.
- Provides data security.
- Data loss and misuse of data is avoided.
- Only administrators can control user and workers addition and deletion.
- Dairy Administrators can add or delete products and update their availability.

Our milk and dairy management system helps in easy storing and accessing of data especially for the people in rural areas. The administrator can add his partner administrators if he wants to, he can add and view the user details and update and delete the user records whenever he wants to. The administrator can also add and delete worker based on his requirements. He can also make the dairy products fixed and list to the user so that the user can view the products available along with the cost and buy the required products according to their need. Once the user buys the products total bill will be automatically generated and displayed to the user, once the products are sold the product quantity is automatically updated. The user can also view the address status. So, if you are running a dairy business in your city, this dairy management system will definitely help you to manage all your work within 1 page to reduce manual work.

## REFERENCES

- <https://dev.mysql.com/doc/>
- <https://docs.python.org/3/library/tk.html>
- <https://dev.mysql.com/doc/connector-python/en/connector-python-example-connecting.html>
- <https://www.youtube.com/watch?v=yQSEXcf6s2I>