Project Report on

Factory Management System

By
Rushang Phira(E011), Tarran Pidugu(E012)and
Shaurya Rawat(E018)

Under the Guidance of

Pankti Doshi
Department of Computer Engineering
Mukesh Patel School of Technology, Management, and
Engineering

MUKESH PATEL SCHOOL OF TECHNOLOGY
MANAGEMENT &ENGINEERING
SVKM's

NARSEE MONJEE INSTITUTE OF MANAGEMENT STUDIES (Declared as Deemed-to-be University Under Section 3 of the UGC Act, 1956)

V. L. Mehta Road, Vile Parle (West)

MUMBAI -400056

March 2020

Subject: Database Management Systems
Semester II, Year: II

Academic Year: 2019-2020

INDEX

Abstract

The report is based on our DBMS project which is a factory management system. The user can access the system as a manager. The user can perform several operations such as searching and deleting records, checking the inventory, manufacture a product, check orders, place an order, add customers, sell the products and check the employee details as well as the customer details. These operations reflect the overall functioning of an actual factory.

List of figures

The Following are the Figures in the Report:

List of Tables

List of Acronyms

1. ER/ERD- Entity Relation Diagram

2. NF1: First Normal Form

3. NF2: Second Normal Form

4. NF3: Third Normal Form

Chapter 1: Introduction to the System

1.1 Introduction

Database Management System (DBMS) is a software for storing and retrieving users' data while considering appropriate security measures. It consists of a group of programs which manipulate the database. The DBMS accepts the request for data from an application and instructs the operating system to provide the specific data. In large systems, a DBMS helps users and other third-party software to store and retrieve data.

DBMS allows users to create their own databases as per their requirement. The term “DBMS” includes the user of the database and other application programs. It provides an interface between the data and the software application.

Certain characteristics of DBMS are:

- Provides security and removes redundancy
- Self-describing nature of a database system
- Insulation between programs and data abstraction
- Support of multiple views of the data
- Sharing of data and multiuser transaction processing

Some sectors which require the use of DBMS are banking, university, hospital, factory and so on.

For the project, we have designed a factory management system. This idea was conceived after some brainstorming and the acceptance from all the group members. Our system includes several functions which give a first hand experience to a user of how a real factory operates.

1.2 Problem Statement

Design a database management system for factory management such that when the user accesses the system, they can perform a variety of functions that involve the manipulation of the data which has been added in the system beforehand like deleting, adding, updating, etc. The functions included in the system should correspond with the actual operations which are performed by a manager in a factory.

1.3 Functional requirements of the system

i) Search record

In the functionality, the user has a choice to either search a customer or an employee. If the user wants to search either of the two then the particular ID must be entered (customer ID or Employee ID).

ii) Delete customer record

If the user wants to delete a customer record then the customer ID should be entered.

iii) Check inventory

This displays the current inventory in the form of a table. The table is updated as per certain actions of the user.

iv) Manufacture product

To manufacture a product, the user first enters the material ID and the quantity of the material needed. If the quantity demanded is more than the available quantity then it shows an error message. If it’s lower then the user enters the product ID to manufacture that particular product.

v) Place order for raw material

To place order, the user enters the material ID, supplier ID, and the quantity needed. The total cost (which also includes the delivery charge of each supplier) is deducted from the funds if the cost is lesser than the original funds else an error message is displayed.

vi) Add customer

To add a customer, the user enters the customer ID, name, city, phone number, and the balance.

vii) Sell product

The user will enter the product ID, the customer ID, employee ID and the quantity ordered. First, the quantity will be checked to ensure that it is sufficient. Then, the balance of the customer will be checked to see if he/she can afford it. If both the conditions are fulfilled then the balance of the customer is deducted and the incentive will be given to the employee.

viii) Check orders

The function displays a table which consists of all the orders made. The table gets updated whenever a new order is made.

ix) Find out employee details

It displays the highest and the average salaries as well as a table of employee details.

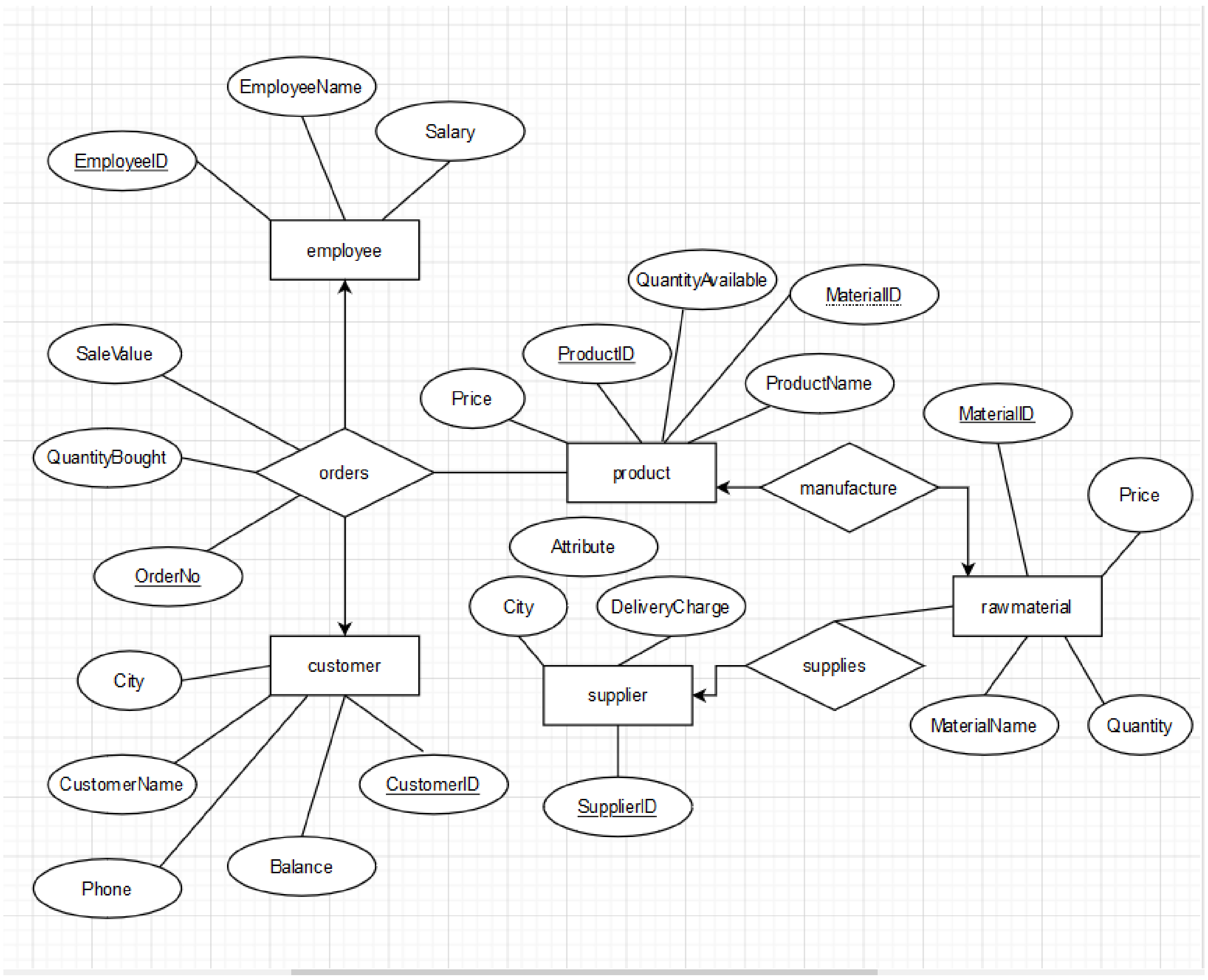
1.4 Users of the System

The user accesses the system as the manager of the factory. The user can perform any of the nine functions-

- Search record
- Delete customer record
- Check Inventory
- Manufacture product
- Place order for raw material
- Add customer
- Sell product
- Check orders
- Find out employee details

Chapter 2: System Design and constraints

2.1 ER Model



(Figure 1)

2.2 Reduction of ER model to relational model

employee (**<u>EmployeeID</u>**, EmployeeName, Salary)

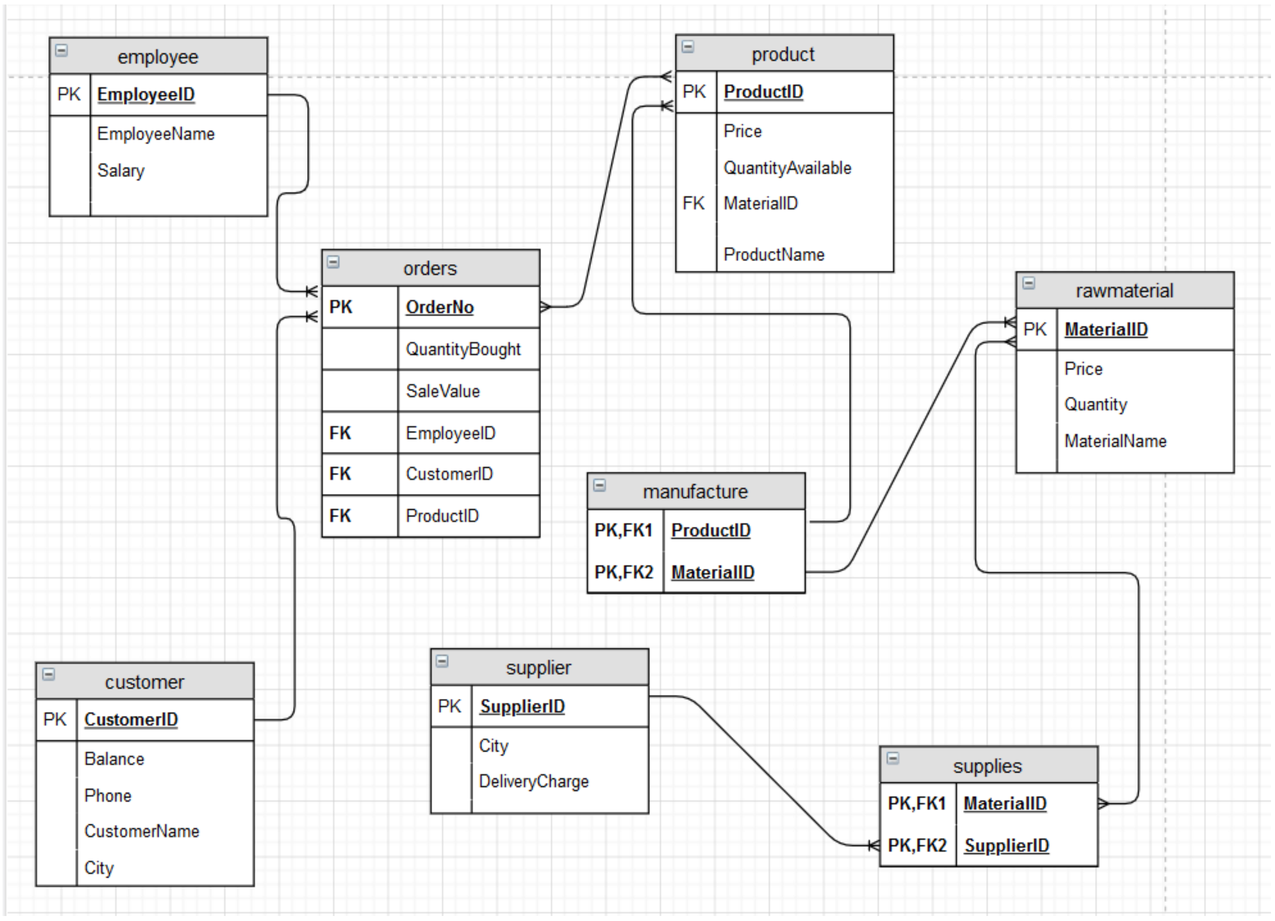
customer (**<u>CustomerID,</u>** Balance, Phone, CustomerName, City)

product (**<u>ProductID,</u>** Price, QuantityAvailable, MaterialID, ProductName)

rawmaterial (**<u>MaterialID,</u>** Price, Quantity, MaterialName)

supplier **<u>(SupplierID,</u>** City, DeliveryCharge) orders (**<u>OrderNo,</u>** EmployeeID, CustomerID, ProductID, QuantityBought)

2.3 Schema Diagram



(Figure 2)

2.4 Constraints

Certain attributes of each table possess some constraint.

Customer table-

CustomerID- primary key, not null

CustomerName- not null

City- not null

Phone- not null, unique

Employee table-

EmployeeID- primary key, not null

EmployeeName- not null

Orders table-

OrderNo- primary key, not null

CustomerID- not null

CustomerName- not null

ProductID- not null

ProductName- not null

EmployeeID- not null

Product table-

ProdcutID- primary key, not null

ProductName- not null

MaterialID- not null

Raw Material Table-

MaterialID- primary key, not null

MaterialName- not null

Supplier Table-

SupplierID- primary key, not null

City- not null

2.5 Normalization techniques applied on relational model

If a database design is not perfect, it may contain anomalies like update, deletion and insert anomalies. This is hazardous for any database management system developer.

Normalization is a method to remove all these anomalies and bring the database to a consistent state.

Normalization consists of a number of normalization forms as follows:

1. First Normal Form (NF1):

First Normal Form is defined in the definition of relations (tables) itself. This rule defines that all the attributes in a relation must have atomic domains. The values in an atomic domain are indivisible units.

Each attribute must contain only a single value from its pre-defined domain.

2. Second Normal Form (NF2):

There are two types of attributes as given below:

- **Prime attribute** − An attribute, which is a part of the candidate-key, is known as a prime attribute.
- **Non-prime attribute** − An attribute, which is not a part of the prime-key, is said to be a non-prime attribute.

According to the NF2, every non-prime attribute should be fully functionally dependent on prime key attribute i.e. **partial dependency** is not allowed in Second Normal Form.

3. Third Normal Form (NF3):

For a relation to be in Third Normal Form, it must be in Second Normal form and the following must satisfy −

- No non-prime attribute is transitively dependent on prime key attribute.
- For any non-trivial functional dependency, X → A, then either −
  - X is a super-key or,
  - A is prime attribute.

Our project satisfied all the above normal forms therefore there is no need of further normalization.

Chapter 3: Implementation

3.1 Hardware and Software details (Front end and Back end details)

Hardware-

Processor- Intel core i3 processor or higher

RAM- 2 GB or more

Operating System- Windows 7 or higher

Software-

Front End- Python was used to develop the front end. Python is an interpreted, high level, general purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Python is dynamically typed and garbage-collected. It 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.

Python was conceived in the late 1980s as a successor to the ABC language. Python 2.0, released in 2000, introduced features like list comprehensions and a garbage collection system capable of collecting reference cycles. Python 3.0, released in 2008, was a major revision of the language that is not completely backward-compatible, and much Python 2 code does not run unmodified on Python 3. The Python 2 language, i.e. Python 2.7.x, was officially discontinued on 1 January 2020 (first planned for 2015) after which security patches and other improvements will not be released for it. With Python 2's end-of-life, only Python 3.5.x and later are supported.

Python interpreters are available for many operating systems. A global community of programmers develops and maintains CPython, an open source reference implementation. A non-profit organization, the Python Software Foundation, manages and directs resources for Python and CPython development.

Back End- SQL was used to develop the back end. SQL is a domain-specific language used in programming and designed for managing data held in a relational database management system (RDBMS), or for stream processing in a relational data stream management system (RDSMS). It is particularly useful in handling structured data, i.e. data incorporating relations among entities and variables.

SQL offers two main advantages over older read–write APIs such as ISAM or VSAM. Firstly, it introduced the concept of accessing many records with one single command. Secondly, it eliminates the need to specify how to reach a record, e.g. with or without an index.

Originally based upon relational algebra and tuple relational calculus, SQL consists of many types of statements, which may be informally classed as sublanguages, commonly: a data query language (DQL), a data definition language (DDL), a data control language (DCL), and a data manipulation language (DML). The scope of SQL includes data query, data manipulation (insert, update and delete), data definition (schema creation and modification), and data access control. Although SQL is essentially a declarative language (4GL), it includes also procedural elements.

SQL was one of the first commercial languages to utilize Edgar F. Codd's relational model. The model was described in his influential 1970 paper, "A Relational Model of Data for Large Shared Data Banks". Despite not entirely adhering to the relational model as described by Codd, it became the most widely used database language.

3.2 Tools or Library used

The tools used are-

PyCharm- It is an integrated development environment (IDE) used in computer programming, specifically for the Python language. It is developed by the Czech company JetBrains. It provides code analysis, a graphical debugger, an integrated unit tester, integration with version control systems (VCSes), and supports web development with Django as well as Data Science with Anaconda. PyCharm is cross-platform, with Windows, macOS and Linux versions. The Community Edition is released under the Apache License, and there is also Professional Edition with extra features – released under a proprietary license.

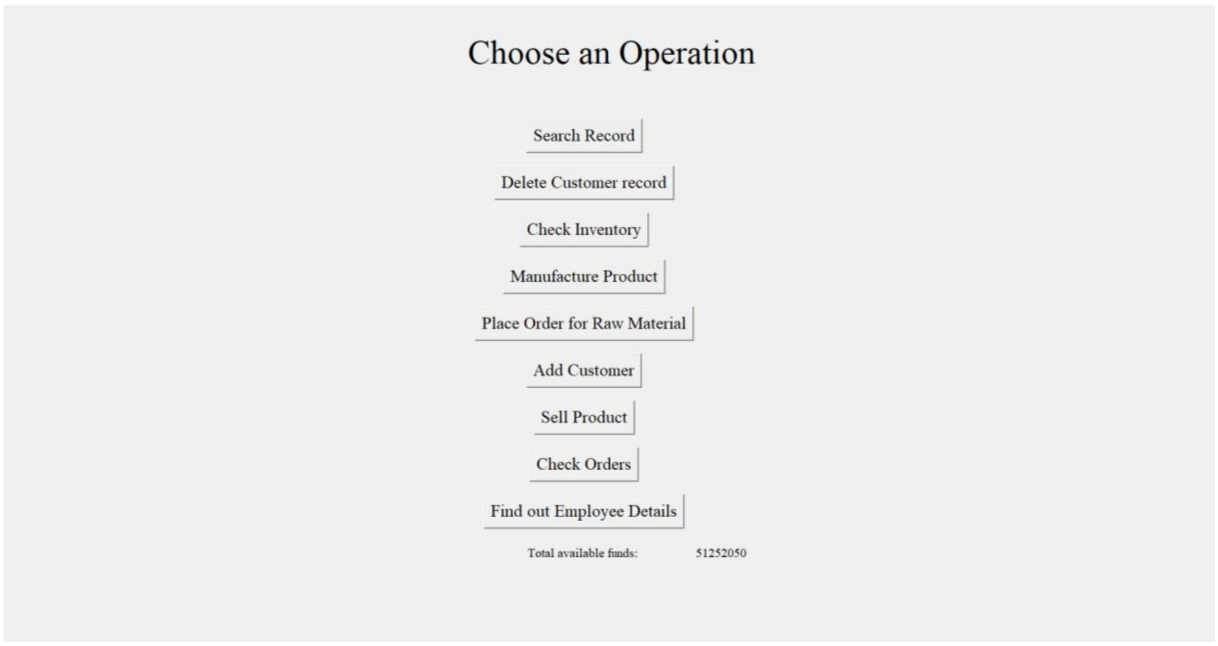
MySQL- It is an open-source relational database management system (RDBMS). MySQL is free and open-source software under the terms of the

GNU General Public License, and is also available under a variety of proprietary licenses. MySQL was owned and sponsored by the Swedish company MySQL AB, which was bought by Sun Microsystems (now Oracle Corporation). In 2010, when Oracle acquired Sun, Widenius forked the open-source MySQL project to create MariaDB. MySQL is a component of the LAMP web application software stack (and others), which is an acronym for Linux, Apache, MySQL, Perl/PHP/Python. MySQL is used by many database-driven web applications, including Drupal, Joomla, phpBB, and WordPress. MySQL is also used by many popular websites, including Facebook, Flickr, MediaWiki, Twitter, and YouTube.

The library used is-

Tkinter- It is a Python binding to the Tk GUI toolkit. It is the standard Python interface to the Tk GUI toolkit, and is Python's de facto standard GUI. Tkinter is included with standard Linux, Microsoft Windows and Mac OS X installs of Python. The name Tkinter comes from Tk interface. Tkinter was written by Fredrik Lundh. Tkinter is free software released under a Python license. As with most other modern Tk bindings, Tkinter is implemented as a Python wrapper around a complete Tcl interpreter embedded in the Python interpreter. Tkinter calls are translated into Tcl commands which are fed to this embedded interpreter, thus making it possible to mix Python and Tcl in a single application. There are several popular GUI library alternatives available, such as wxPython, PyQt (PySide), Pygame, Pyglet, and PyGTK.

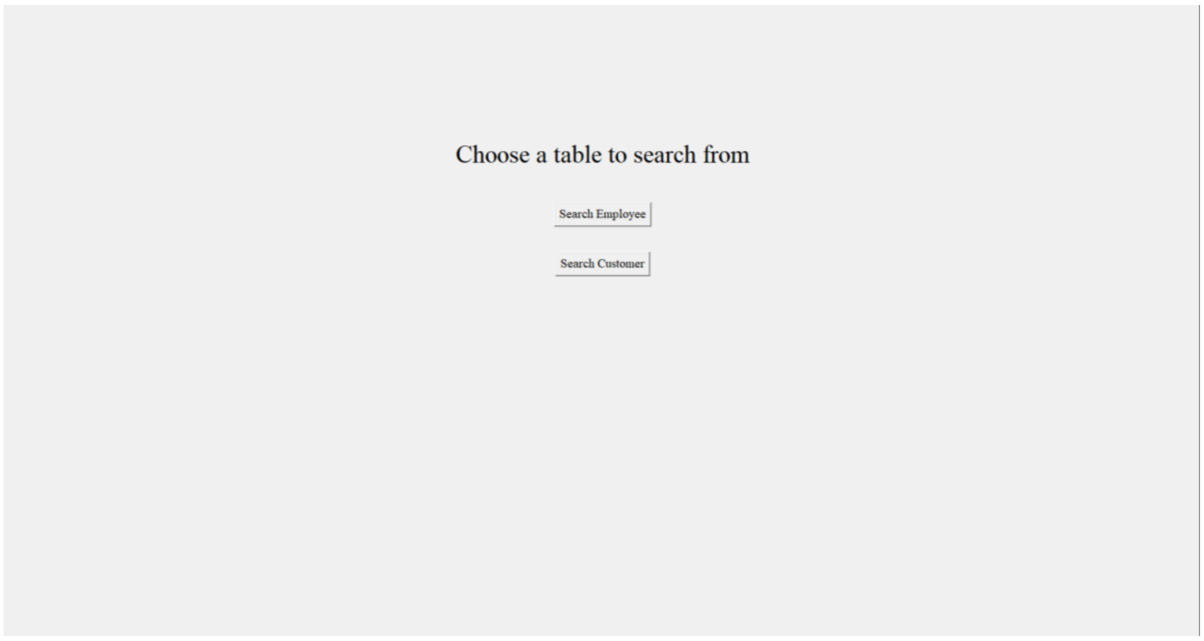
3.3 Screenshots and Description



Total available funds: 51252050

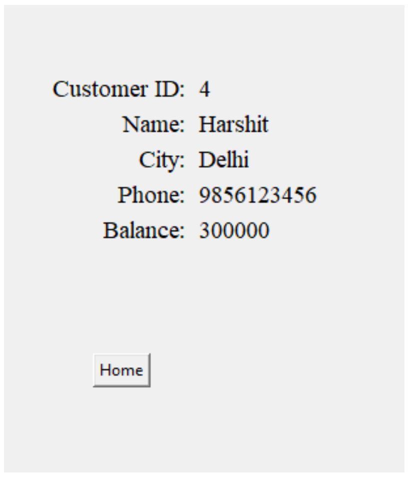
(Figure 3)

Home Screen, funds available displayed at the bottom. Changes after each transaction.



(Figure 4**)**

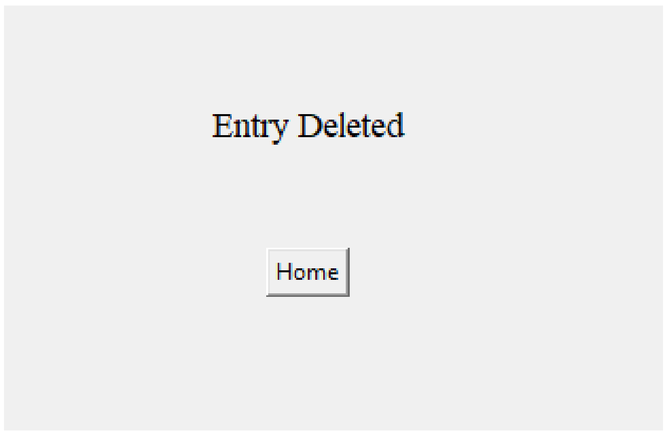
Search Functionality



Customer ID: 4
Name: Harshit
City: Delhi
Phone: 9856123456
Balance: 300000

(Figure 5**)**

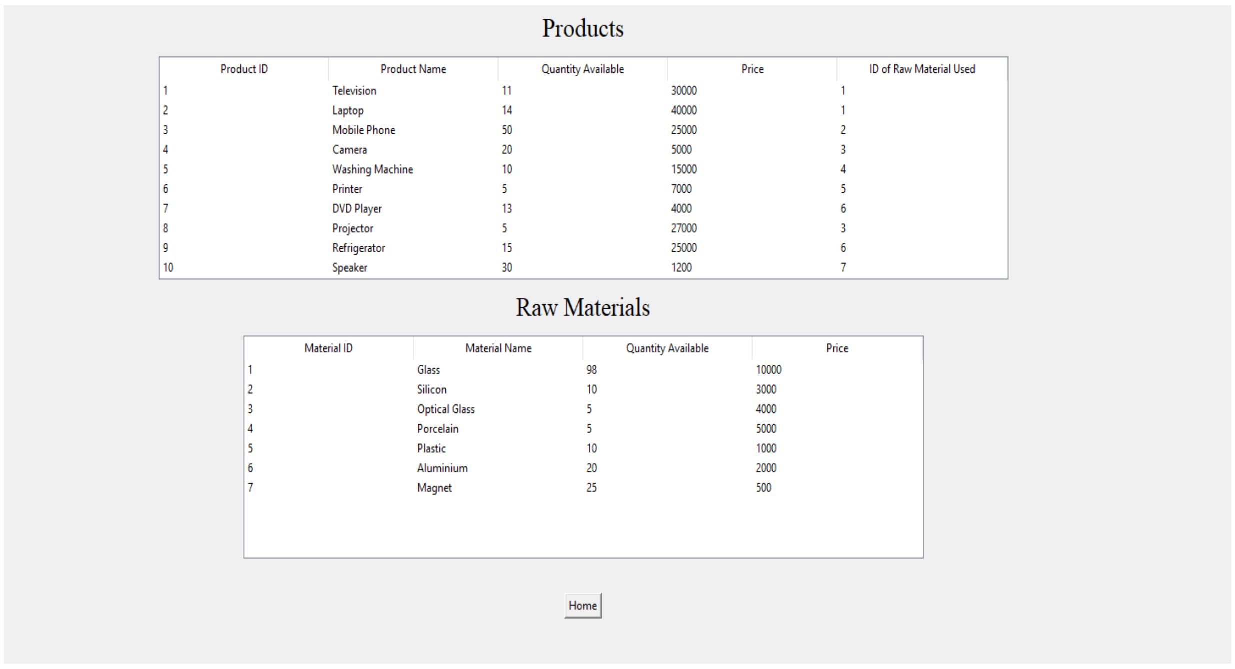
Displaying details of a customer



Entry Deleted

(Figure 6**)**

Message affirming deletion of a customer entry


Products

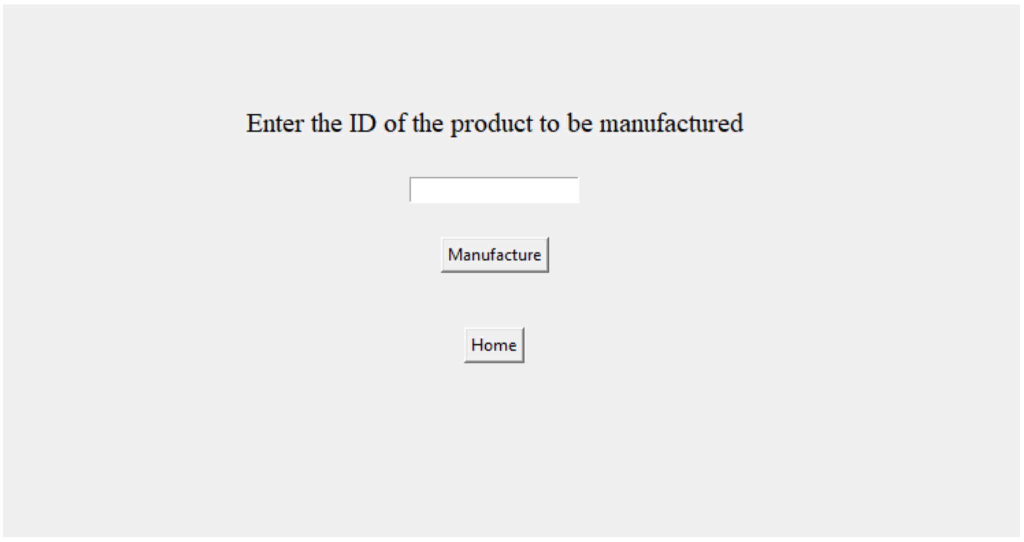
Product ID	Product Name	Quantity Available	Price	ID of Raw Material Used
1	Television	11	30000	1
2	Laptop	14	40000	1
3	Mobile Phone	50	25000	2
4	Camera	20	5000	3
5	Washing Machine	10	15000	4
6	Printer	5	7000	5
7	DVD Player	13	4000	6
8	Projector	5	27000	3
9	Refrigerator	15	25000	6
10	Speaker	30	1200	7

Raw Materials

Material ID	Material Name	Quantity Available	Price
1	Glass	98	10000
2	Silicon	10	3000
3	Optical Glass	5	4000
4	Porcelain	5	5000
5	Plastic	10	1000
6	Aluminium	20	2000
7	Magnet	25	500

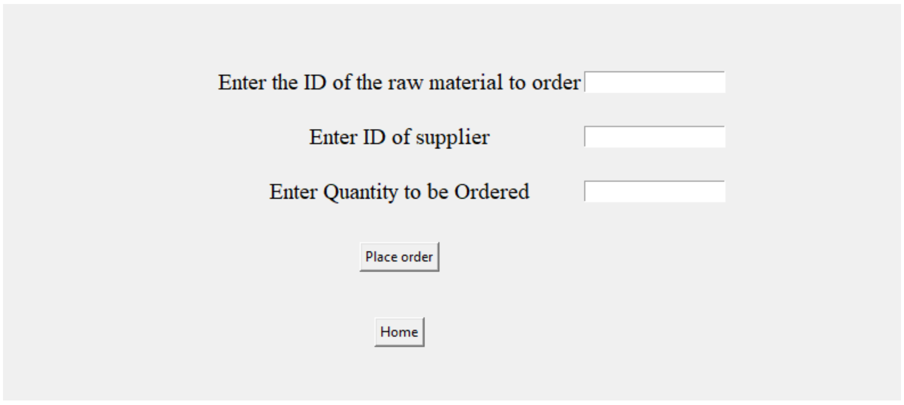
(Figure 7)

Checking inventory


Enter the ID of the product to be manufactured

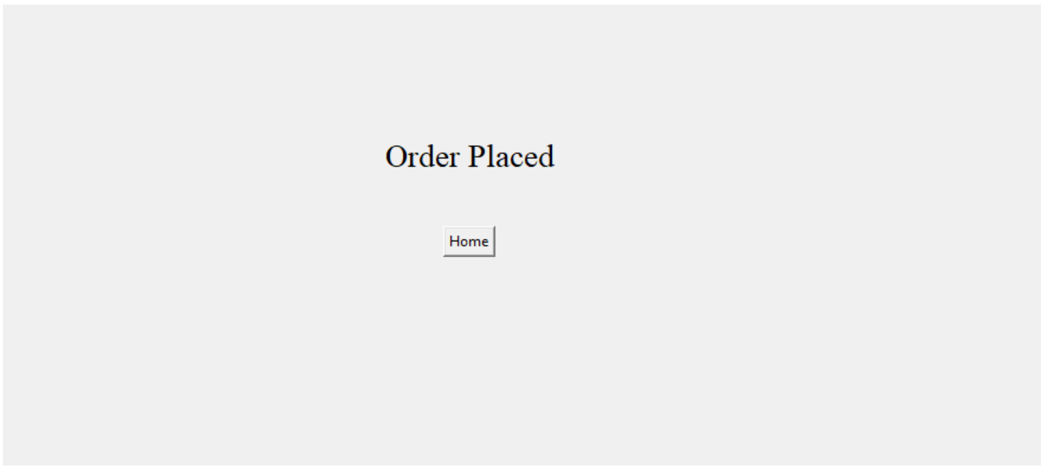
(Figure 8)

Manufacture product screen


Enter the ID of the raw material to order
Enter ID of supplier
Enter Quantity to be Ordered
Place order
Home

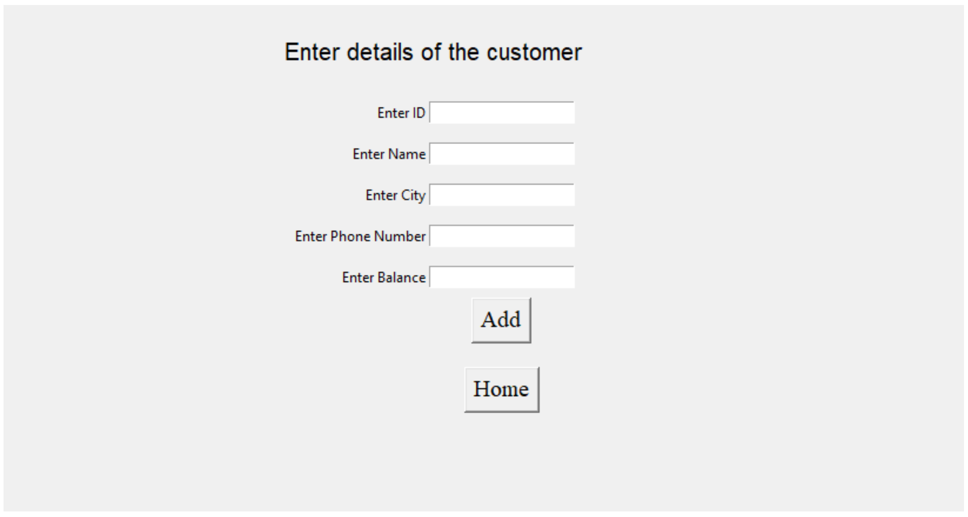
(Figure 9**)**

Page where user can order materials.



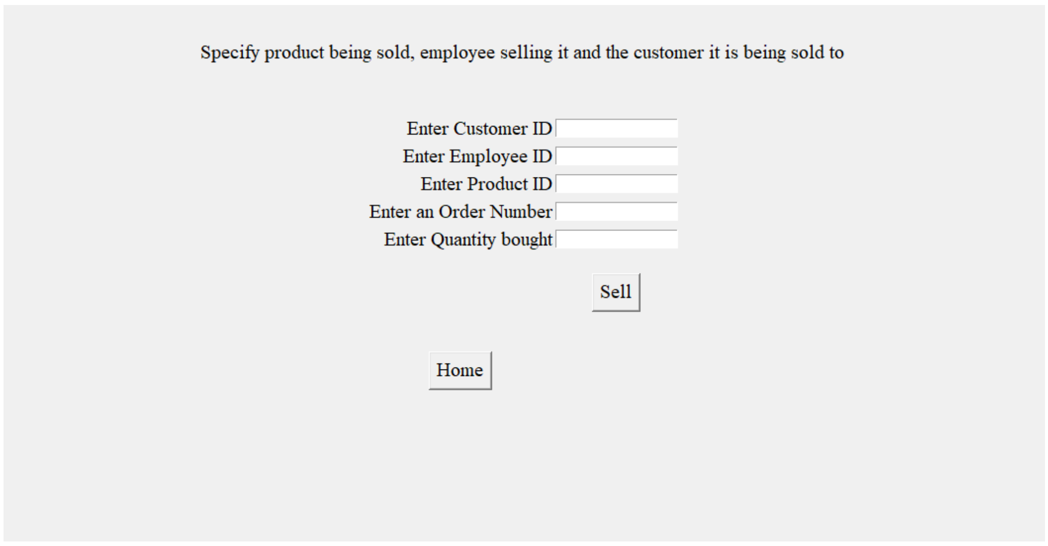
(Figure 10**)**

Message displayed when order is placed successfully



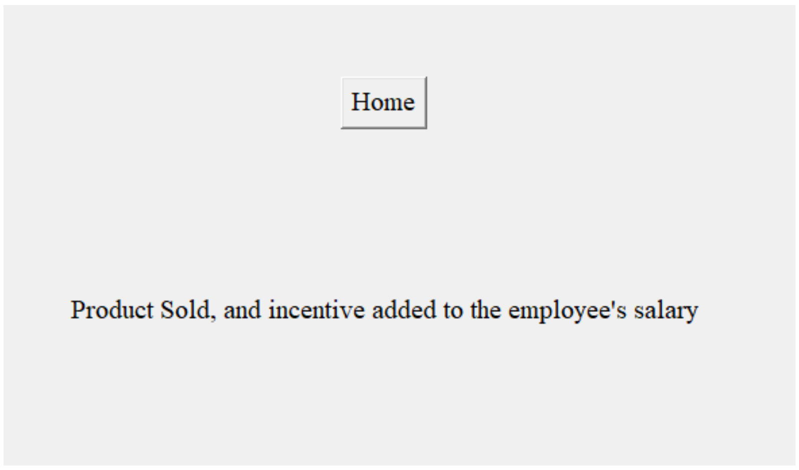
(Figure 11**)**

Screen where customer can be added



(Figure 12**)**

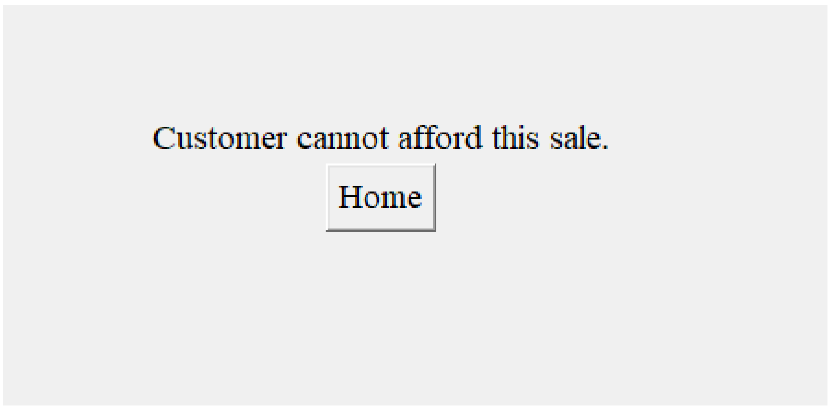
Screen where product can be sold



Product Sold, and incentive added to the employee's salary

(Figure 13**)**

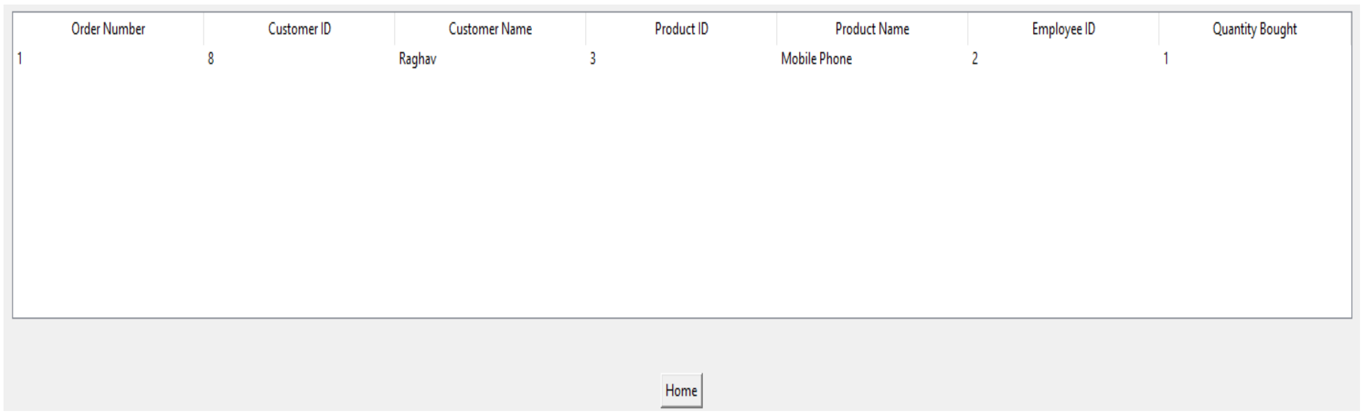
Message displayed after a sale is made



Customer cannot afford this sale.

(Figure 14**)**

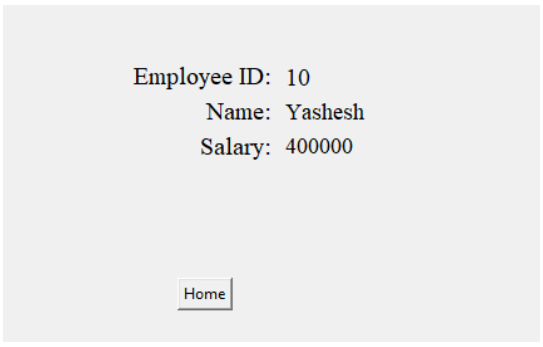
Message displayed when total price of product exceeds the customer's outstanding balance.



Order Number	Customer ID	Customer Name	Product ID	Product Name	Employee ID	Quantity Bought
1	8	Raghav	3	Mobile Phone	2	1

(Figure 15**)**

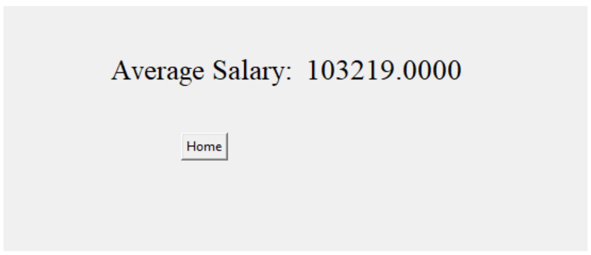
Displaying orders table



Employee ID: 10
Name: Yashesh
Salary: 400000

(Figure 16**)**

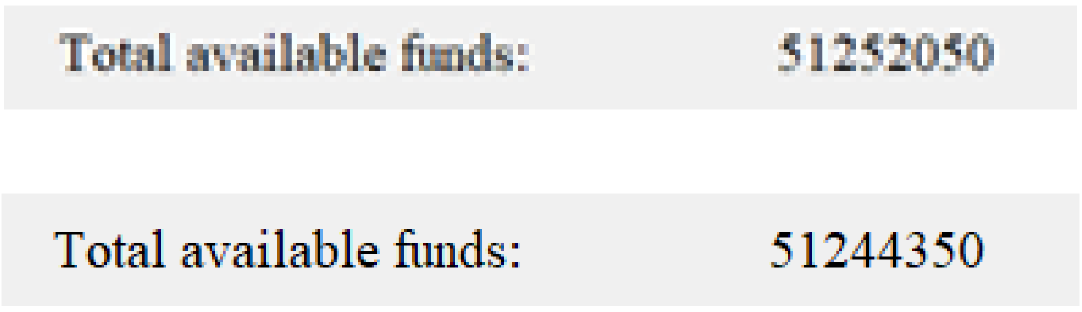
Displaying employee with highest salary



Average Salary: 103219.0000

(Figure 17**)**

Displaying average salary



Total available funds: 51252050

Total available funds: 51244350

(Figure 18**)**

Funds before and after the sale displayed in orders table was made

3.4 Database Structure

i) Customer Table

Field Name	**Data Type**	**Description**	**Remarks**

CustomerID	int	Customer id	primary key, not null
CustomerName	varchar(45)	Customer Name	not null
City	varchar(45)	Customer’s city	not null
Phone	varchar(45)	Phone number	not null, unique
Balance	int	Customer’s budget	

ii) Employee table

Field Name	**Data Type**	**Description**	**Remarks**
EmployeeID	int	Employee id	primary key, not null
EmployeeName	varchar(45)	Employee name	not null
Salary	int	Employee salary	

iii) Orders table

Field Name	**Data Type**	**Description**	**Remarks**
OrderNo	int	Order number	primary key, not null
CustomerID	int	Customer id	not null
CustomerName	varchar(45)	Customer Name	not null
ProductID	int	Product id	not null
EmployeeID	int	Employee id	not null
QuantityBought	int	Quantity of the product purchased	

iv) Product table

Field Name	**Data Type**	**Description**	**Remarks**
ProductID	int	Product id	primary key, not null
ProductName	varchar(45)	Product Name	not null
QuantityAvailable	int	Quantity of product available	
Price	int	Price of product	
MaterialID	int	Material id	not null

v) Rawmaterial table

Field Name	**Data Type**	**Description**	**Remarks**
MaterialID	int	Material ID	primary key, not null
MaterialName	varchar(45)	Material Name	not null
Quantity	int	Quantity of the material	
Price	int	Cost of the material	

vi) Supplier table

Field Name	**Data Type**	**Description**	**Remarks**
SupplierID	int	Supplier id	primary key, not null
City	varchar(45)	City where the supplier is located	not null
DeliveryCharge	int	Amount charged by the supplier for shipping	

Chapter 4: Conclusion and future work

In our project, we developed a factory management system in which we added several functions which mirror the operations that are performed by the manager. We were thorough with the DBMS concepts to put into use but had to look up on the internet for the front-end part so as to code the functionalities efficiently as well as use certain libraries to beautify. We also studied the connectivity concept so as to link our database with our front-end code. Though we performed our project on a small scale, we would have done the same if we were to implement for an actual operation. Not only we learnt a lot of new concepts while doing this project, the previous concepts were also effectively revised as those were hugely implemented by us. The project was an excellent learning experience for us as we also got an insight of how a typical factory is managed. This experience along with the knowledge gained will hugely benefit us for the future tasks when we would be using more advanced concepts as well as performing operations at a big scale.

Chapter 5: Appendix

```
from tkinter import *
from tkinter import ttk
import mysql.connector

#Establishing connection to the database
mydb = mysql.connector.connect(host="localhost", user="root", passwd="password",
database="factory")
mycursor = mydb.cursor()

gui = Tk()  # Creating window
gui.title("Factory")
gui.geometry('1920x1080+0+0')

firstframe = Frame(gui)
startframe = Frame(gui)
empsearchframe = Frame(gui)
customersearchframe = Frame(gui)
deleteframe = Frame(gui)
inventoryframe = Frame(gui)
manufactureframe = Frame(gui)
rawframe = Frame(gui)
addframe = Frame(gui)
sellframe = Frame(gui)
sellheadframe = Frame(gui)
checkorderframe = Frame(gui)
avsalframe = Frame(gui)

firstdestroy = 0
startdestroy = 0
empsearchdestroy = 0
customersearchdestroy = 0
deletedestroy = 0
inventorydestroy = 0
manufacturedestroy = 0
rawdestroy = 0
adddestroy = 0
selldestroy = 0
checkorderdestroy = 0
avsaldestroy = 0


def first():
    global firstframe
    firstframe = Frame(gui)

    global firstdestroy
    firstdestroy = 1

    headingframe = Frame(firstframe)

    heading = Label(headingframe, text="FACTORY SIMULATOR", fg="brown",
font='Helvetica 40 bold italic', bg="lightblue",
                    relief=RAISED)

    loginframe = Frame(firstframe)
    loginbutton = Button(loginframe, text="Begin", font="time 15",
command=startwindow,
                         activebackground="grey")  # Program starts once "Begin"
button is clicked

    creator1 = Label(headingframe,
                     text="Project Created by:\n\nRushang Phira (E011)\nTarran
Pidugu (E012)\nShaurya Rawat (E018)",
                     font="times 20 italic")

    creator1.grid(row=1)
    heading.grid(row=0, column=0, padx=100, pady=80)
    headingframe.grid()
    loginframe.grid(row=2, column=0)
    loginbutton.grid(row=3, padx=20, pady=50)
    firstframe.grid()
    firstframe.place(relx=0.5, rely=0.3, anchor=CENTER)


def startwindow():  # Home button leads to this page. Contains code to destroy all
frames as soon as this page is
    # reached
    # This page then leads to the next page with no input/outputs, making it a
sort of a dummy page.

    global startframe
    global firstdestroy
    global startdestroy
    global empsearchdestroy
    global customersearchdestroy
    global deletedestroy
    global inventorydestroy
    global manufacturedestroy
    global rawdestroy
    global adddestroy
    global selldestroy
    global checkorderdestroy
    global avsaldestroy

    startframe = Frame(gui)
    startdestroy = 1

    # Next few lines check which page we are reaching here from. Then the frames
corresponding to those pages only
    # are destroyed.

    if firstdestroy == 1:
        firstframe.destroy()
        firstdestroy = 0
    elif empsearchdestroy == 1:
        empsearchframe.destroy()
        empsearchdestroy = 0
    elif customersearchdestroy == 1:
        customersearchframe.destroy()
        customersearchdestroy = 0
    elif deletedestroy == 1:
        deleteframe.destroy()
        deletedestroy = 0
    elif inventorydestroy == 1:
        inventoryframe.destroy()
        inventorydestroy = 0
    elif manufacturedestroy == 1:
        manufactureframe.destroy()
        manufacturedestroy = 0
    elif rawdestroy == 1:
        rawframe.destroy()
        rawdestroy = 0
    elif adddestroy == 1:
        addframe.destroy()
        adddestroy = 0
    elif selldestroy == 1:
        sellframe.destroy()
        sellheadframe.destroy()
        selldestroy = 0
    elif checkorderdestroy == 1:
        checkorderframe.destroy()
        checkorderdestroy = 0
    elif avsaldestroy == 1:
        avsalframe.destroy()
        avsaldestroy = 0

    login()  # Leads to the next page, which presents user with all the choices.


def login():  # Function for begin command, executed once "Begin" button is
clicked

    global startdestroy

    if startdestroy == 1:  # Destroying all existing frames before proceeding
        startdestroy = 0
        startframe.destroy()

    def search():  # Function for searching

        def emp():  # Function for Employee search

            global empsearchframe  # Frame within which all widgets are contained
in the employee search page.
            global empsearchdestroy

            empsearchframe = Frame(gui)
            empsearchdestroy = 1  # Setting this to 1 means that when we click the
home button, this page gets

            # destroyed.

            def detail2():  # Function to display employee

                uid = searchentry.get()  # Stores ID of employee

                label1.destroy()
                searchentry.destroy()
                searchbutton.destroy()

                detailframe = Frame(empsearchframe)
                detailframe.grid(row=0, column=0, pady=50)
                displayframe = Frame(empsearchframe)
                displayframe.grid(row=0, column=1, pady=50)

                detaillabel1 = Label(detailframe, text="Employee ID: ",
font="times 14")
                detaillabel1.grid(row=0, sticky=E)

                detaillabel2 = Label(detailframe, text="Name: ", font="times 14")
                detaillabel2.grid(row=1, sticky=E)

                detaillabel3 = Label(detailframe, text="Salary: ", font="times
14")
                detaillabel3.grid(row=2, sticky=E)

                detailID = Label(displayframe, text="", font="times 14 ")
                mycursor.execute(
                    "select EmployeeID from employee where EmployeeID =
('{}')".format(uid))  # Fetching data
                data1 = mycursor.fetchall()
                detailID.config(text=data1)
                detailID.grid(row=0, sticky=W)

                detailname = Label(displayframe, text="", font="times 14")
                mycursor.execute(
                    "select EmployeeName from employee where EmployeeID =
('{}')".format(uid))  # Fetching data
                data2 = (str(mycursor.fetchall()))[3:-4]
                detailname.config(text=data2)
                detailname.grid(row=1, sticky=W)

                detailsal = Label(displayframe, text="", font="times 14")
                mycursor.execute("select Salary from employee where EmployeeID =
('{}')".format(uid))  # Fetching data
                data3 = (str(mycursor.fetchall()))[2:-3]
                detailsal.config(text=data3)
                detailsal.grid(row=2, sticky=W)

                back = Button(empsearchframe, text="Home", command=startwindow)
                back.grid(pady=30, row=2, column=0)

            # Destroying all existing frames before proceeding
            searchframe.destroy()
            search1.destroy()
            search2.destroy()

            empsearchframe.grid(padx=100)
            label1 = Label(empsearchframe, text="Enter unique ID of the Employee",
font="times 15")
            label1.grid(padx=100, pady=20)
            searchentry = Entry(empsearchframe)
            searchentry.grid(row=1, column=0)
            searchbutton = Button(empsearchframe, text="Show Details",
command=detail2, font="times 10")
            searchbutton.grid(row=2, column=0, pady=50)
            empsearchframe.place(relx=0.5, rely=0.3, anchor=CENTER)

        def cmer():  # Function for Customer search

            global empsearchframe
            global empsearchdestroy
            empsearchframe = Frame(gui)

            empsearchdestroy = 1

            def detail1():
                uid = searchentry.get()
                label1.destroy()
                searchentry.destroy()
                searchbutton.destroy()

                detailframe = Frame(empsearchframe)
                detailframe.grid(row=0, column=0, pady=50)

                displayframe = Frame(empsearchframe)
                displayframe.grid(row=0, column=1, pady=50)

                detaillabel1 = Label(detailframe, text="Customer ID: ",
font="times 14")
                detaillabel1.grid(row=0, sticky=E)

                detaillabel2 = Label(detailframe, text="Name: ", font="times 14")
                detaillabel2.grid(row=1, sticky=E)

                detaillabel3 = Label(detailframe, text="City: ", font="times 14")
                detaillabel3.grid(row=2, sticky=E)

                detaillabel4 = Label(detailframe, text="Phone: ", font="times 14")
                detaillabel4.grid(row=3, sticky=E)

                detaillabel5 = Label(detailframe, text="Balance: ", font="times
14")
                detaillabel5.grid(row=4, sticky=E)

                detailID = Label(displayframe, text="", font="times 14 ")
                mycursor.execute(
                    "select CustomerID from customer where CustomerID =
('{}')".format(uid))  # Fetching data
                data1 = mycursor.fetchall()
                detailID.config(text=data1)
                detailID.grid(row=0, sticky=W)

                detailname = Label(displayframe, text="", font="times 14")
                mycursor.execute(
                    "select CustomerName from customer where CustomerID =
('{}')".format(uid))  # Fetching data
                data2 = (str(mycursor.fetchall()))[3:-4]
                detailname.config(text=data2)
                detailname.grid(row=1, sticky=W)

                detailcity = Label(displayframe, text="", font="times 14")
                mycursor.execute("select City from customer where CustomerID =
('{}')".format(uid))  # Fetching data
                data3 = (str(mycursor.fetchall()))[3:-4]
                detailcity.config(text=data3)
                detailcity.grid(row=2, sticky=W)

                detailphone = Label(displayframe, text="", font="times 14")
                mycursor.execute("select Phone from customer where CustomerID =
('{}')".format(uid))  # Fetching data
                data4 = mycursor.fetchall()
                detailphone.config(text=data4)
                detailphone.grid(row=3, sticky=W)

                detailbal = Label(displayframe, text="", font="times 14")
                mycursor.execute("select Balance from customer where CustomerID =
('{}')".format(uid))  # Fetching data
                data5 = mycursor.fetchall()
                detailbal.config(text=data5)
                detailbal.grid(row=4, sticky=W)

                back = Button(empsearchframe, text="Home", command=startwindow)
                back.grid(pady=30, row=2, column=0)

            # Destroying all existing frames before proceeding
            searchframe.destroy()
            search1.destroy()
            search2.destroy()

            label1 = Label(empsearchframe, text="Enter unique ID of the Customer",
font="times 15")
            label1.grid(padx=100, pady=20)
            searchentry = Entry(empsearchframe)
            searchentry.grid(row=1, column=0)
            searchbutton = Button(empsearchframe, text="Show Details",
command=detail1, font="times 14")
            searchbutton.grid(row=2, column=0, pady=50)
            empsearchframe.grid(padx=100)
            empsearchframe.place(relx=0.5, rely=0.3, anchor=CENTER)

        # Destroying all frames before proceeding
        choiceframe.destroy()
        mainframe.destroy()

        searchframe = Frame(gui)
        searchframe.grid(row=0, column=0)
        searchtable = Label(searchframe, text="Choose a table to search from",
font="times 30")
        searchtable.grid(padx=100, pady=50)
        search1 = Button(searchframe, text="Search Employee", command=emp,
font="times 15")
        search2 = Button(searchframe, text="Search Customer", command=cmer,
font="times 15")
        search1.grid(row=1, column=0)
        search2.grid(row=2, column=0, pady=40)
        searchframe.place(relx=0.5, rely=0.3, anchor=CENTER)

    def delete():
        def cmer():  # Function for Customer delete
            def delete1():
                uid = searchentry.get()
                label1.destroy()
                searchentry.destroy()
                searchbutton.destroy()
                mycursor.execute("delete from customer where CustomerID =
('{}')".format(uid))
                mydb.commit()
                dellab = Label(deleteframe, text="Entry Deleted", font="times 14")
                dellab.grid(row=3, pady=20, padx=50)

            label1 = Label(deleteframe, text="Enter unique ID of the Customer",
font="times 15")
            label1.grid(padx=100, pady=20)
            searchentry = Entry(deleteframe)
            searchentry.grid(row=1, column=0)
            searchbutton = Button(deleteframe, text="Delete Details",
command=delete1, font="times 15")
            searchbutton.grid(row=2, column=0, pady=50)

            back = Button(deleteframe, text="Home", command=startwindow)
            back.grid(row=4, column=0, pady=30)

        global deleteframe
        global deletedestroy
        deletedestroy = 1

        # Destroying all frames before proceeding
        choiceframe.destroy()
        mainframe.destroy()

        deleteframe = Frame(gui)
        deleteframe.grid(row=0, column=0)
        deleteframe.place(relx=0.5, rely=0.3, anchor=CENTER)

        cmer()

    def inventory():  # Function to display products as well as raw materials in
stock.

        global inventoryframe
        inventoryframe = Frame(gui)

        global inventorydestroy
        inventorydestroy = 1  # Set to 1 so when the home button is clicked it
goes to startwindow after destroying
        # this frame.

        choiceframe.destroy()
        mainframe.destroy()

        label1 = Label(inventoryframe, text="Products", font="times 20")
        mycursor.execute("select * from product")
        rows = mycursor.fetchall()

        # Creating TreeViews to display data in the form of tables
        tv1 = ttk.Treeview(inventoryframe, columns=(1, 2, 3, 4, 5),
show="headings")
        tv1.heading(1, text="Product ID")
        tv1.heading(2, text="Product Name")
        tv1.heading(3, text="Quantity Available")
        tv1.heading(4, text="Price")
        tv1.heading(5, text="ID of Raw Material Used")
        for i in rows:
            tv1.insert('', 'end', values=i)
        label1.grid(row=0, padx=10, pady=10)
        tv1.grid(row=1)
        label2 = Label(inventoryframe, text="Raw Materials", font="times 20")
        mycursor.execute("select * from rawmaterial")
        rows = mycursor.fetchall()
        tv2 = ttk.Treeview(inventoryframe, columns=(1, 2, 3, 4), show="headings")
        tv2.heading(1, text="Material ID")
        tv2.heading(2, text="Material Name")
        tv2.heading(3, text="Quantity Available")
        tv2.heading(4, text="Price")
        for i in rows:
            tv2.insert('', 'end', values=i)
        label2.grid(row=2, padx=10, pady=10)
        tv2.grid(row=3)

        back = Button(inventoryframe, text="Home", command=startwindow)
        back.grid(row=4, column=0, pady=35)

        inventoryframe.grid()
        inventoryframe.place(relx=0.5, rely=0.4, anchor=CENTER)

    def manufacture():  # Function to manufacture a product

        def manufactured():
            prodid = int(enterprod.get())  # Fetching the product ID entered by
the user

            # Fetching the quantity of the raw material available
            mycursor.execute("select MaterialID from product where ProductID =
('{}')".format(prodid))
            matid = int((str(mycursor.fetchall())[2:-3]))  # Fetching material ID

            mycursor.execute("select Quantity from rawmaterial where MaterialID =
('{}')".format(matid))
            available = int((str(mycursor.fetchall())[2:-3]))  # Fetching
available quantity of raw material

            if available >> 0:  # Checking if quantity available is enough for
manufacturing
                mycursor.execute(
                    "update product set QuantityAvailable = QuantityAvailable+1
where ProductID = ('{}')".format(
                        prodid))
                mycursor.execute("update rawmaterial set Quantity = Quantity-1
where MaterialID= ('{}')".format(matid))
                mydb.commit()
                manlabel = Label(manufactureframe, text="Product Manufactured",
font="times 15")
                manlabel.grid(row=3)
            else:
                label1.destroy()
                enterprod.destroy()
                manbutton.destroy()
                fail = Label(manufactureframe, text="Not enough raw materials,
more stock needed", font="times 14")
                fail.grid(row=0)

        choiceframe.destroy()
        mainframe.destroy()

        global manufacturedestroy
        global manufactureframe
        manufactureframe = Frame(gui)
        manufacturedestroy = 1

        label1 = Label(manufactureframe, text="Enter the ID of the product to be
manufactured", font="times 15")
        enterprod = Entry(manufactureframe)
        manbutton = Button(manufactureframe, command=manufactured,
text="Manufacture")
        back = Button(manufactureframe, text="Home", command=startwindow)

        label1.grid(padx=50, pady=20)
        enterprod.grid(row=1, column=0, padx=50, pady=5)
        manbutton.grid(row=2, column=0, pady=20)
        back.grid(row=5, column=0, pady=20)

        manufactureframe.grid()
        manufactureframe.place(relx=0.5, rely=0.3, anchor=CENTER)

    def raw():  # Function for ordering raw materials

        def order():
            # Next few lines deducts cost of the transaction from factory funds

            supid = int(supplierentry.get())
            mycursor.execute("select DeliveryCharge from supplier where SupplierId
= ('{}')".format(supid))
            deliverycharge = int((str(mycursor.fetchall())[2:-3]))
            rawmat = int(rawentry.get())  # Stores Material ID entered by user
            quantity = int(quantenter.get())  # Stores quantity of raw material
entered by user
            mycursor.execute("select Price from rawmaterial where MaterialID =
('{}')".format(rawmat))
            materialcost = int((str(mycursor.fetchall())[2:-3]))
            totalcost = (materialcost * quantity) + deliverycharge
            mycursor.execute("update funds set Amount = Amount -
('{}')".format(totalcost))

            # Destroying all frames before proceeding.
            rawlabel.destroy()
            rawbutton.destroy()
            rawentry.destroy()
            supplierentry.destroy()
            supplierlabel.destroy()
            quantlabel.destroy()
            quantenter.destroy()

            mycursor.execute(
                "update rawmaterial set Quantity = Quantity + ('{}') where
MaterialID = ('{}')".format(quantity,

rawmat))

            rawdone = Label(rawframe, text="Order Placed", font="times 20")
            rawdone.grid(row=0, pady=20)

            mydb.commit()

        global rawframe
        rawframe = Frame(gui)

        global rawdestroy
        rawdestroy = 1

        choiceframe.destroy()
        mainframe.destroy()

        rawlabel = Label(rawframe, text="Enter the ID of the raw material to
order", font="times 15")
        rawentry = Entry(rawframe)

        supplierlabel = Label(rawframe, text="Enter ID of supplier", font="times
15")
        supplierentry = Entry(rawframe)

        quantlabel = Label(rawframe, text="Enter Quantity to be Ordered",
font="times 15")
        quantenter = Entry(rawframe)
        rawbutton = Button(rawframe, command=order, text="Place order")
        back = Button(rawframe, text="Home", command=startwindow)

        rawlabel.grid(row=0, column=0, pady=10)
        rawentry.grid(row=0, column=1)
        supplierlabel.grid(row=1, column=0, pady=10)
        supplierentry.grid(row=1, column=1, pady=10)
        quantlabel.grid(row=2, column=0, pady=10)
        quantenter.grid(row=2, column=1, pady=10)
        rawbutton.grid(row=3, pady=20)
        back.grid(pady=20, row=4)
        rawframe.grid(padx=100)
        rawframe.place(relx=0.5, rely=0.3, anchor=CENTER)

    def add():  # Function to add a customer to database

        def added():
            mycursor.execute(
                "insert into customer values(('{}'), ('{}'), ('{}'), ('{}'),
('{}'))".format(int(enterid.get()),

entername.get(),

entercity.get(),

enterph.get(),

int(enterbal.get())))
            done = Label(addframe, text="Customer added")
            done.grid(row=8, pady=30)
            mydb.commit()

        choiceframe.destroy()
        mainframe.destroy()

        global addframe
        addframe = Frame(gui)

        global adddestroy
        adddestroy = 1
        addlabelframe = Frame(addframe)
        addlabel = Label(addlabelframe, text="Enter details of the customer",
font="time 15")

        enterframe = Frame(addframe)

        addID = Label(enterframe, text="Enter ID")
        addname = Label(enterframe, text="Enter Name")
        addcity = Label(enterframe, text="Enter City")
        addph = Label(enterframe, text="Enter Phone Number")
        addbal = Label(enterframe, text="Enter Balance")

        enterid = Entry(enterframe)
        entername = Entry(enterframe)
        entercity = Entry(enterframe)
        enterph = Entry(enterframe)
        enterbal = Entry(enterframe)

        addlabel.grid(pady=20)

        addID.grid(row=1, column=0, pady=7, sticky=E)
        addname.grid(row=2, column=0, pady=7, sticky=E)
        addcity.grid(row=3, column=0, pady=7, sticky=E)
        addph.grid(row=4, column=0, pady=7, sticky=E)
        addbal.grid(row=5, column=0, pady=7, sticky=E)

        enterid.grid(row=1, column=1, pady=7)
        entername.grid(row=2, column=1, pady=7)
        entercity.grid(row=3, column=1, pady=7)
        enterph.grid(row=4, column=1, pady=7)
        enterbal.grid(row=5, column=1, pady=7)

        addbutton = Button(enterframe, command=added, text="Add", font="times 15")
        addbutton.grid(column=1)

        back = Button(enterframe, text="Home", font="times 15",
command=startwindow)
        back.grid(pady=20, row=7, column=1)

        addframe.grid(padx=100)
        addlabelframe.grid()
        addframe.place(relx=0.5, rely=0.3, anchor=CENTER)
        enterframe.grid()

    def sell():  # Function to sell a product to a customer based on details
entered by user
        def sold():
            pid = enterprod.get()
            cid = entercust.get()
            orderno = int(enterorder.get())
            quantity = int(enterquant.get())
            eid = int(enteremp.get())

            mycursor.execute("select Price from product where ProductID =
('{}')".format(pid))
            price = int((str(mycursor.fetchall())[2:-3]))
            mycursor.execute("select ProductName from product where ProductID =
('{}')".format(pid))
```

```
            pname = (str(mycursor.fetchall())[3:-4])
            mycursor.execute("select CustomerName from customer where CustomerID =
('{}')".format(cid))
            cname = (str(mycursor.fetchall())[3:-4])
            mycursor.execute("select QuantityAvailable from product where
ProductID = ('{}')".format(pid))
            quantavailable = int((str(mycursor.fetchall())[2:-3]))

            mycursor.execute("select Balance from customer where CustomerID =
('{}')".format(cid))
            custbal = int((str(mycursor.fetchall())[2:-3]))
            print(custbal)
            print("PRICE ", quantity*price)

            if (quantity * price) > custbal:  # Checks if customer can afford
order
                sellheadframe.destroy()
                sellcust.destroy()
                entercust.destroy()
                sellemp.destroy()
                enteremp.destroy()
                sellprod.destroy()
                enterprod.destroy()
                sellbutton.destroy()
                quantlabel.destroy()
                enterquant.destroy()
                enterorder.destroy()
                ordernolabel.destroy()
                back.destroy()
                faillab = Label(sellframe, text="Customer cannot afford this
sale.", font="times 14")
                faillab.grid()
                failbutton = Button(sellframe, text="Home", command=startwindow,
font="times 14")
                failbutton.grid(row=1)
            else:

                if quantity < quantavailable:

                    mycursor.execute("update funds set Amount = Amount +
('{}')".format((quantity * price)))

                    # The next few lines calculate the incentives to the employees
depending on the
                    # amount made during the sale
                    if price * quantity < 20000:
                        incentive = 0.25 / 100 * (price * quantity)
                        mycursor.execute("update employee set Salary = Salary +
('{}') "
                                         "where EmployeeID =
('{}')".format(incentive, eid))
                    elif price * quantity >= 20000 & price * quantity < 35000:
                        incentive = 1.00 / 100 * (price * quantity)
                        mycursor.execute("update employee set Salary = Salary +
('{}') "
                                         "where EmployeeID =
('{}')".format(incentive, eid))
                    elif price * quantity >= 35000 & price * quantity < 60000:
                        incentive = 2.00 / 100 * (price * quantity)
                        mycursor.execute("update employee set Salary = Salary +
('{}') "
                                         "where EmployeeID =
('{}')".format(incentive, eid))
                    elif price * quantity >= 60000 & price * quantity < 90000:
                        incentive = 3.50 / 100 * (price * quantity)
                        mycursor.execute("update employee set Salary = Salary +
('{}') "
                                         "where EmployeeID =
('{}')".format(incentive, eid))
                    elif price * quantity >= 90000 & price * quantity < 120000:
                        incentive = 5.50 / 100 * (price * quantity)
                        mycursor.execute("update employee set Salary = Salary +
('{}') "
                                         "where EmployeeID =
('{}')".format(incentive, eid))

                    sellheadframe.destroy()
                    sellcust.destroy()
                    entercust.destroy()
                    sellemp.destroy()
                    enteremp.destroy()
                    sellprod.destroy()
                    enterprod.destroy()
                    sellbutton.destroy()
                    quantlabel.destroy()
                    enterquant.destroy()
                    enterorder.destroy()
                    ordernolabel.destroy()

                    mycursor.execute(
                        "update product set QuantityAvailable = QuantityAvailable-
('{}') where "
                        "ProductID = ('{}')".format(
                            quantity, pid))
                    mycursor.execute(
                        "update  customer set Balance = Balance - ('{}') where
CustomerID = ('{}')".format(
                            (quantity * price),
                            cid))
                    mycursor.execute(
                        "insert into orders values(('{}'),"
                        " ('{}'), ('{}'), ('{}'), ('{}'),"
                        " ('{}'),('{}'))".format(orderno, int(cid), cname,
int(pid), pname, eid, quantity))

                    mydb.commit()
                    soldlabel = Label(sellframe, text="Product Sold, and incentive
added to the "
                                                      "employee's salary",
font="times 15")
                    soldlabel.grid(padx=100, pady=100)

                else:
                    sellheadframe.destroy()
                    sellcust.destroy()
                    entercust.destroy()
                    sellemp.destroy()
                    enteremp.destroy()
                    sellprod.destroy()
                    enterprod.destroy()
                    sellbutton.destroy()
                    quantlabel.destroy()
                    enterquant.destroy()
                    enterorder.destroy()
                    ordernolabel.destroy()
                    fail = Label(sellframe, text="Product not available, more need
to be manufactured", font="times 14")
                    fail.grid()

        choiceframe.destroy()
        mainframe.destroy()

        global sellframe  # This and the next 4 lines sets this frame to be
destroyed
        # once the user clicks the Home button
        sellframe = Frame(gui)
        global sellheadframe

        global selldestroy
        selldestroy = 1

        sellheadframe = Frame(gui)
        sellhead = Label(sellheadframe, text="Specify product being sold, employee
selling it and the customer it is "
                                             "being sold to",
                         font="times 15")
        sellcust = Label(sellframe, text="Enter Customer ID", font="times 15")
        sellemp = Label(sellframe, text="Enter Employee ID", font="times 15")
        sellprod = Label(sellframe, text="Enter Product ID", font="times 15")
        ordernolabel = Label(sellframe, text="Enter an Order Number", font="times
15")
        quantlabel = Label(sellframe, text="Enter Quantity bought", font="times
15")
        enterorder = Entry(sellframe)
        entercust = Entry(sellframe)
        enteremp = Entry(sellframe)
        enterquant = Entry(sellframe)
        enterprod = Entry(sellframe)
        sellbutton = Button(sellframe, command=sold, text="Sell", font="times 15")

        sellheadframe.grid(pady=20, row=0)
        sellframe.grid(padx=100, row=1, pady=30)
        sellhead.grid()
        ordernolabel.grid(row=4, column=0, sticky=E)
        enterorder.grid(row=4, column=1)
        sellcust.grid(row=1, column=0, sticky=E)
        entercust.grid(row=1, column=1)
        sellemp.grid(row=2, column=0, sticky=E)
        enteremp.grid(row=2, column=1)
        sellprod.grid(row=3, column=0, sticky=E)
        enterprod.grid(row=3, column=1)
        quantlabel.grid(row=5, column=0, sticky=E)
        enterquant.grid(row=5, column=1)
        sellbutton.grid(row=7, column=1, pady=20)
        sellheadframe.place(relx=0.5, rely=0.1, anchor=CENTER)
        sellframe.place(relx=0.5, rely=0.3, anchor=CENTER)
        back = Button(sellframe, text="Home", font="times 15",
command=startwindow)
        back.grid(pady=20, row=9, column=0)

    def checkorder():  # Function to display the orders table

        choiceframe.destroy()
        mainframe.destroy()

        global checkorderdestroy
        checkorderdestroy = 1

        global checkorderframe
        checkorderframe = Frame(gui)

        # Creating a TreeView to display data in the form of a table
        mycursor.execute("select * from orders")
        rows = mycursor.fetchall()
        tv = ttk.Treeview(checkorderframe, columns=(1, 2, 3, 4, 5, 6, 7),
show="headings")
        tv.heading(1, text="Order Number")
        tv.heading(2, text="Customer ID")
        tv.heading(3, text="Customer Name")
        tv.heading(4, text="Product ID")
        tv.heading(5, text="Product Name")
        tv.heading(6, text="Employee ID")
        tv.heading(7, text="Quantity Bought")
        for i in rows:
            tv.insert('', 'end', values=i)
        tv.grid()

        checkorderframe.grid(padx=100, pady=100)
        checkorderframe.place(relx=0.5, rely=0.3, anchor=CENTER)

        back = Button(checkorderframe, text="Home", command=startwindow)
        back.grid(pady=40)

    def avsal():
        def maxsal():
            maxsalbutton.destroy()
            avsalbutton.destroy()

            mycursor.execute("select MAX(Salary) from employee")
            uid = int((str(mycursor.fetchall())[2:-3]))

            detailframe = Frame(avsalframe)
            detailframe.grid(row=0, column=0, pady=50)

            displayframe = Frame(avsalframe)
            displayframe.grid(row=0, column=1, pady=50)

            detaillabel1 = Label(detailframe, text="Employee ID: ", font="times
15")
            detaillabel1.grid(row=0, sticky=E)

            detaillabel2 = Label(detailframe, text="Name: ", font="times 15")
            detaillabel2.grid(row=1, sticky=E)

            detaillabel3 = Label(detailframe, text="Salary: ", font="times 15")
            detaillabel3.grid(row=2, sticky=E)
            detailID = Label(displayframe, text="", font="times 15 ")
            mycursor.execute(
                "select EmployeeID from employee where Salary =
('{}')".format(uid))  # Fetching data
            data1 = mycursor.fetchall()
            detailID.config(text=data1)
            detailID.grid(row=0, sticky=W)

            detailname = Label(displayframe, text="", font="times 14")
            mycursor.execute(
                "select EmployeeName from employee where Salary =
('{}')".format(uid))  # Fetching data
            data2 = (str(mycursor.fetchall()))[3:-4]
            detailname.config(text=data2)
            detailname.grid(row=1, sticky=W)

            detailsal = Label(displayframe, text="", font="times 14")
            mycursor.execute("select Salary from employee where Salary =
('{}')".format(uid))  # Fetching data
            data3 = (str(mycursor.fetchall()))[2:-3]
            detailsal.config(text=data3)
            detailsal.grid(row=2, sticky=W)

        def average():
            maxsalbutton.destroy()
            avsalbutton.destroy()

            mycursor.execute("select AVG(Salary) from employee")
            uid = (str(mycursor.fetchall())[11:-5])

            displaylabel1 = Label(avsalframe, text="Average Salary: ", font="times
20")
            displaylabel1.grid(row=0, column=0)

            displaylabel2 = Label(avsalframe, text="", font="times 20")
            displaylabel2.config(text=uid)
            displaylabel2.grid(row=0, column=1)

        choiceframe.destroy()
        mainframe.destroy()

        global avsaldestroy
        avsaldestroy = 1

        global avsalframe
        avsalframe = Frame(gui)

        maxsalbutton = Button(avsalframe, command=maxsal, text="Employee With
Maximum Salary", font="times 15")
        avsalbutton = Button(avsalframe, command=average, text="Average Employee
Salary", font="times 15")

        back = Button(avsalframe, text="Home", command=startwindow)
        back.grid(row=3, pady=40)
        maxsalbutton.grid(row=0, column=0, pady=15)
        avsalbutton.grid(row=1, column=0, pady=15)
        avsalframe.grid(padx=100, pady=100)
        avsalframe.place(relx=0.5, rely=0.3, anchor=CENTER)
```

```python
    choiceframe = Frame(gui)
    choiceframe.grid(pady=15)
    choiceframe.place(relx=0.5, rely=0.07, anchor=CENTER)

    mainframe = Frame(gui)
    mainframe.grid(padx=100, pady=50)
    mainframe.place(relx=0.5, rely=0.5, anchor=CENTER)

    # Presenting functionalities to the user
    choicetext = Label(choiceframe, text="Choose an Operation", font="times 40")
    choicetext.grid(row=0, column=1, pady=20)

    choice1 = Button(mainframe, command=search, text="Search Record", font="times 20")
    choice1.grid(row=1, column=0, pady=10)

    choice2 = Button(mainframe, command=delete, text="Delete Customer record", font="times 20")
    choice2.grid(row=2, column=0, pady=10)

    choice3 = Button(mainframe, command=inventory, text="Check Inventory", font="times 20")
    choice3.grid(row=3, column=0, pady=10)

    choice4 = Button(mainframe, command=manufacture, text="Manufacture Product", font="times 20")
    choice4.grid(row=4, column=0, pady=10)

    choice5 = Button(mainframe, command=raw, text="Place Order for Raw Material", font="times 20")
    choice5.grid(row=5, column=0, pady=10)

    choice6 = Button(mainframe, command=add, text="Add Customer", font="times 20")
    choice6.grid(row=6, column=0, pady=10)

    choice7 = Button(mainframe, command=sell, text="Sell Product", font="times 20")
    choice7.grid(row=7, column=0, pady=10)

    choice8 = Button(mainframe, command=checkorder, text="Check Orders", font="times 20")
    choice8.grid(row=8, column=0, pady=10)

    choice9 = Button(mainframe, command=avsal, text="Find out Employee Details", font="times 20")
    choice9.grid(row=9, column=0, pady=10)

    # Funds is a table in the database that stores the total funds available to the factory
    fundlabel = Label(mainframe, text="Total available funds: ", font="times 15")
    fundlabel.grid(row=10, column=0, pady=15)
    fundlabel2 = Label(mainframe, text="", font="times 15")
    mycursor.execute("select * from funds")
    funds = int((str(mycursor.fetchall()))[2:-3])
    fundlabel2.config(text=funds)
    fundlabel2.grid(row=10, column=1, pady=15)
```

```
first()
# Execute GUI
gui.mainloop()
```

Connection to the database is established by the line *mydb = mysql.connector.connect(host="localhost", user="root", passwd="password", database="factory")*. Variable 'mycursor' is the cursor variable which is used to executes commands on our database.

Whenever we fetch data from the database, we receive it in the for of a tuple. In order to use that data or to display it, we need to convert it into a string, slice it to remove commas or parentheses and then convert it into an integer (if necessary). An example of this is: *funds = int((str(mycursor.fetchall()))[2:-3])* This command fetches the value stored in funds and converts it into an integer to display it on the home screen.

Important methods used in the code are:

1. first(): Function for the start screen, displayed only once during execution
2. startwindow(): Home button leads to this page. Contains code to destroy all frames as soon as this page is reached. This page then leads to the next page with no input/outputs, making it a sort of a dummy page.

3. login(): Function for begin command, executed once "Begin" button is clicked

4. search(): Function to search for a record

5. emp(): Function for employee search

6. cmer(): Function for customer search

7. detail2(): Function to display employee details

8. detail1(): Function to display customer details

9. delete(): Function to delete record of a customer

10. inventory(): Function to display products as well as raw materials in stock.

11. manufacture(): Function to manufacture a product

12. raw(): Function for ordering raw materials

13. add(): Function to add a customer to database

14. sell(): Function to sell a product to a customer based on details entered by user

15. checkorder(): Function to display the orders table

16. def avsal(): Function to display average or maximum salaries

17. maxsal(): Function within avsal() to display details of employee with maximum salary.

18. average(): Function within avsal() to display average employee salary