**Part B Micro-Project Report**

**Employee Record Management System**

1. **Rationale:**

The Employee Record Management System, built using Python, offers a pragmatic approach for businesses aiming to streamline the management of their employee data. Our application, powered by Python, facilitates effortless addition, viewing, updating and deletion of employee details, empowering organizations to uphold precise records and facilitate informed HR decisions. With ongoing enhancements to the system, we strive to augment its capabilities continuously, thereby fostering enhanced practices in workforce management.

1. **Course Outcomes Achieved:**
2. Develop Python program to demonstrate use of Operators
3. Perform operations on data structures in Python.
4. Design classes for given problems
5. **Actual Methodology Followed:**

1. Define Requirements: Understand the needs of the system. Identify what features are essential, such as adding, viewing, updating and deleting employee records. Determine if any additional features or functionalities are required.

2. Design Database Schema: Design the structure of the database to store employee records. Decide what information needs to be stored for each employee, such as name, ID, department, etc. Choose an appropriate database system like SQLite, MySQL, or PostgreSQL.

3. Set Up the Python Environment: Install Python on your system if not already installed. Decide if you want to use any frameworks or libraries such as Flask or Django for web development or Tkinter for desktop GUI applications.

4. Create the Employee Class: Define a Python class to represent an employee. Include attributes and methods to manipulate employee data, such as adding, updating, or deleting employee records.

5. Implement CRUD Operations: Implement functions or methods to perform CRUD (Create, Read, Update, Delete) operations on employee records. These functions will interact with the database to store or retrieve employee information.

6. Develop User Interface (UI): Depending on your requirements, create a user interface for interacting with the system. This could be a command-line interface (CLI), a desktop GUI using Tkinter, or a web interface using Flask or Django.

7. Implement Authentication and Authorization: If necessary, implement authentication and authorization mechanisms to control access to the system. This ensures that only authorized users can perform certain actions, such as adding or editing employee records.

8. Testing: Test the system thoroughly to ensure that it functions as expected. Test each feature and functionality to identify and fix any bugs or issues.

9. Documentation: Document the system, including its architecture, functionalities, usage instructions, and any other relevant information.

10. Deployment: Deploy the system in the desired environment, whether it's on a local machine, a server, or a cloud platform. Ensure that the system is accessible to users and meets performance requirements.

11. Maintenance and Updates: Regularly maintain and update the system to address any bugs, security vulnerabilities, or new requirements that arise over time. Consider user feedback and make improvements accordingly.

1. **Algorithm**

**Step 1:** Import necessary libraries: tkinter, messagebox, and pymysql.

**Step 2:** Establish a connection to the MySQL database.

**Step 3:** Define functions for various operations such as adding, viewing, deleting, updating, and exiting employee details.

**Step 4:** For adding employee details:

a. Define a function `emp\_add()` which creates a new Tkinter window for adding employee details.

b. Within the `emp\_add()` function, define nested functions for handling back, submit, and clearing entries.

c. Perform data validation for each input field.

d. Check if the employee ID already exists in the database.

e. If not, insert the employee details into the database.

**Step 5:** For viewing employee details:

a. Define a function `emp\_view()` which creates a new Tkinter window for viewing employee details.

b. Retrieve all employee records from the database and display them in a tabular format.

**Step 6:** For deleting employee details:

a. Define a function `emp\_delete()` which creates a new Tkinter window for deleting employee details.

b. Define nested functions for handling back, clearing entries, and submitting the delete operation.

c. Check if the employee ID exists in the database, if yes, delete the corresponding record.

**Step 7:** For updating employee details:

a. Define a function `emp\_update()` which creates a new Tkinter window for updating employee details.

b. Define nested functions for handling back, clearing entries, and submitting the update operation.

c. Perform data validation for the new value to be updated.

d. Check if the employee ID exists in the database, if yes, update the corresponding field with the new value.

**Step 8:** Define a function `exit\_btn()` to exit the application.

**Step 9:** Create the main Tkinter window with buttons for adding, viewing, deleting, updating, and exiting employee details.

**Step 10:** Run the main event loop using `root.mainloop()` to display the GUI and handle user interactions.

1. **Flowchart**
2. **Source Code**

from tkinter import \*

from tkinter import messagebox

import pymysql as mq

import re

mysql = mq.connect(host="localhost", user="root", password="", database="emp")

mycursor = mysql.cursor()

def emp\_add():

def back():

add.destroy()

root.deiconify()

def submit():

# Retrieve data from entry fields

emp\_id = id\_entry.get()

emp\_name = name\_entry.get()

emp\_age = age\_entry.get()

emp\_phone = phone\_entry.get()

emp\_email = email\_entry.get()

emp\_gender = gender\_var.get()

emp\_designation = designation\_entry.get()

emp\_salary = salary\_entry.get()

# Data Validation

errors = []

if not emp\_id.isdigit():

errors.append("Invalid ID")

if not emp\_name.replace(" ", "").isalpha():

errors.append("Invalid Name")

if not emp\_age.isdigit() or int(emp\_age) <= 18:

errors.append("Age should be a number greater than 18")

if not re.match(r"[^@]+@[^@]+\.[^@]+", emp\_email):

errors.append("Invalid Email")

if not emp\_phone.isdigit() or len(emp\_phone) != 10:

errors.append("Invalid Phone Number (should be 10 digits)")

if not emp\_designation.replace(" ", "").isalpha():

errors.append("Invalid Designation")

if not emp\_salary.isdigit():

errors.append("Invalid Salary")

if errors:

messagebox.showerror("Error", "\n".join(errors))

else:

# Check if employee ID already exists in the database

mycursor.execute(

"SELECT emp\_id FROM emp\_details WHERE emp\_id = %s", (emp\_id,))

existing\_emp = mycursor.fetchone()

if existing\_emp:

messagebox.showerror("Error", "Employee ID already exists")

else:

# Insert data into the database

sql = "INSERT INTO emp\_details (emp\_id, name, phone, email, age, gender, desi, salary) VALUES (%s, %s, %s, %s, %s, %s, %s, %s)"

val = (emp\_id, emp\_name, emp\_phone, emp\_email, emp\_age,

emp\_gender, emp\_designation, emp\_salary)

mycursor.execute(sql, val)

mysql.commit()

# Clear entry fields after submission

clear\_entries()

messagebox.showinfo(

"Success", "Employee Record Added Successfully")

add.destroy()

root.deiconify()

def clear\_entries():

id\_entry.delete(0, END)

name\_entry.delete(0, END)

age\_entry.delete(0, END)

phone\_entry.delete(0, END)

email\_entry.delete(0, END)

designation\_entry.delete(0, END)

salary\_entry.delete(0, END)

root.withdraw()

root.withdraw()

add = Tk()

add.title("Employee Management System - Add Employee Details")

add.geometry("800x800") # Set window size here

add.minsize(600, 400)

add.maxsize(800, 800)

heading\_label = Label(

add, text="Enter Employee Details", font=("Times New Roman", 30, "bold"), padx=30, pady=30)

heading\_label.grid(row=0, column=2, columnspan=2)

id\_label = Label(add, text="ID", font=("Times New Roman", 20))

id\_label.grid(row=1, column=1, padx=10, pady=10)

id\_entry = Entry(add, font=("Times New Roman", 20))

id\_entry.grid(row=1, column=2, padx=10, pady=10)

name\_label = Label(add, text="Name", font=("Times New Roman", 20))

name\_label.grid(row=2, column=1, padx=10, pady=10)

name\_entry = Entry(add, font=("Times New Roman", 20))

name\_entry.grid(row=2, column=2, padx=10, pady=10)

age\_label = Label(add, text="Age", font=("Times New Roman", 20))

age\_label.grid(row=3, column=1, padx=10, pady=10)

age\_entry = Entry(add, font=("Times New Roman", 20))

age\_entry.grid(row=3, column=2, padx=10, pady=10)

phone\_label = Label(add, text="Phone", font=("Times New Roman", 20))

phone\_label.grid(row=4, column=1, padx=10, pady=10)

phone\_entry = Entry(add, font=("Times New Roman", 20))

phone\_entry.grid(row=4, column=2, padx=10, pady=10)

email\_label = Label(add, text="Email", font=("Times New Roman", 20))

email\_label.grid(row=5, column=1, padx=10, pady=10)

email\_entry = Entry(add, font=("Times New Roman", 20))

email\_entry.grid(row=5, column=2, padx=10, pady=10)

gender\_label = Label(add, text="Gender", font=("Times New Roman", 20))

gender\_label.grid(row=6, column=1, padx=10, pady=10)

gender\_var = StringVar()

gender\_var.set("Male")

male\_radio = Radiobutton(

add, text="Male", variable=gender\_var, value="Male", font=("Times New Roman", 20))

female\_radio = Radiobutton(

add, text="Female", variable=gender\_var, value="Female", font=("Times New Roman", 20))

male\_radio.grid(row=6, column=2, sticky=W)

female\_radio.grid(row=6, column=3, sticky=W)

designation\_label = Label(

add, text="Designation", font=("Times New Roman", 20))

designation\_label.grid(row=7, column=1, padx=10, pady=10)

designation\_entry = Entry(add, font=("Times New Roman", 20))

designation\_entry.grid(row=7, column=2, padx=10, pady=10)

salary\_label = Label(add, text="Salary", font=("Times New Roman", 20))

salary\_label.grid(row=8, column=1, padx=10, pady=10)

salary\_entry = Entry(add, font=("Times New Roman", 20))

salary\_entry.grid(row=8, column=2, padx=10, pady=10)

back\_btn = Button(add, text="Back", font=("Times New Roman", 20),

padx=5, pady=5, command=back)

back\_btn.grid(row=9, column=1, padx=10, pady=10)

submit\_btn = Button(add, text="Submit", font=("Times New Roman", 20),

padx=10, pady=10, command=submit)

submit\_btn.grid(row=9, column=2, padx=10, pady=10)

add.mainloop()

def emp\_view():

def back():

view.destroy()

root.deiconify()

root.withdraw()

view = Tk()

view.title("Employee Management System - View Employee Details")

view.geometry("800x800")

# Retrieve data from the database

mycursor.execute("SELECT \* FROM emp\_details")

rows = mycursor.fetchall()

# Create headings

headings = ["ID", "Name", "Phone", "Email",

"Age", "Gender", "Designation", "Salary"]

for col, heading in enumerate(headings):

label = Label(view, text=heading, font=("Times New Roman", 20, "bold"))

label.grid(row=0, column=col, padx=5, pady=5)

# Insert data into table

for row\_idx, row\_data in enumerate(rows, start=1):

for col\_idx, cell\_data in enumerate(row\_data):

label = Label(view, text=cell\_data, font=("Times New Roman", 15))

label.grid(row=row\_idx, column=col\_idx, padx=5, pady=5)

back\_btn = Button(view, text="Back", font=("Times New Roman", 15),

padx=5, pady=5, command=back)

back\_btn.grid(row=9, column=1, padx=10, pady=10)

view.mainloop()

def emp\_delete():

def back():

delete.destroy()

root.deiconify()

def clear\_entries():

id\_entry.delete(0, END)

def submit():

emp\_id = id\_entry.get()

mycursor.execute(

"SELECT emp\_id FROM emp\_details WHERE emp\_id = %s", (emp\_id,))

existing\_emp = mycursor.fetchone()

if existing\_emp:

sql = "DELETE FROM emp\_details WHERE emp\_id=%s"

val = (emp\_id,)

mycursor.execute(sql, val)

mysql.commit()

clear\_entries()

messagebox.showinfo(

"Success", "Employee Record Deleted Successfully")

delete.destroy()

root.deiconify()

else:

messagebox.showinfo(

"Error", "Employee Record does not exists")

clear\_entries()

root.withdraw()

delete = Tk()

delete.title("Employee Management System - Delete Employee Details")

delete.geometry("800x800") # Set window size here

delete.minsize(600, 400)

delete.maxsize(800, 800)

heading\_label = Label(

delete, text="Delete Employee Details", font=("Times New Roman", 30, "bold"), padx=30, pady=30)

heading\_label.grid(row=0, column=1, columnspan=2)

id\_label = Label(delete, text="Enter Employee ID to delete:",

font=("Times New Roman", 20))

id\_label.grid(row=4, column=1, padx=10, pady=10)

id\_entry = Entry(delete, font=("Times New Roman", 20))

id\_entry.grid(row=4, column=2, padx=10, pady=10)

back\_btn = Button(delete, text="Back", font=("Times New Roman", 20),

padx=5, pady=5, command=back)

back\_btn.grid(row=6, column=1, padx=10, pady=10)

submit\_btn = Button(delete, text="Submit", font=("Times New Roman", 20),

padx=10, pady=10, command=submit)

submit\_btn.grid(row=6, column=2, padx=10, pady=10)

delete.mainloop()

def emp\_update():

def back():

update.destroy()

root.deiconify()

def clear\_entries():

id\_entry.delete(0, END)

new\_value\_entry.delete(0, END)

def submit():

emp\_id = id\_entry.get()

mycursor.execute(

"SELECT emp\_id FROM emp\_details WHERE emp\_id = %s", (emp\_id,))

existing\_emp = mycursor.fetchone()

field\_to\_update = update\_var.get()

new\_value = new\_value\_entry.get()

if existing\_emp:

if field\_to\_update == "Select Field":

messagebox.showerror(

"Error", "Please select a field to update")

if field\_to\_update == "Name":

if not new\_value.isalpha():

messagebox.showerror("Name Error", "Invalid Name")

else:

sql = "UPDATE emp\_details SET name = %s WHERE emp\_id = %s"

elif field\_to\_update == "Phone":

if not new\_value.isdigit() or len(new\_value) != 10:

messagebox.showerror(

"Phone Error", "Invalid Phone Number (should be 10 digits)")

else:

print("Updating phone number...")

sql = "UPDATE emp\_details SET phone = %s WHERE emp\_id = %s"

elif field\_to\_update == "Email":

if not re.match(r"[^@]+@[^@]+\.[^@]+", new\_value):

messagebox.showerror("Email Error", "Invalid Email")

else:

sql = "UPDATE emp\_details SET email = %s WHERE emp\_id = %s"

elif field\_to\_update == "Age":

if not new\_value.isdigit() or int(new\_value) <= 18:

messagebox.showerror("Age Error", "Invalid Age")

else:

sql = "UPDATE emp\_details SET age = %s WHERE emp\_id = %s"

elif field\_to\_update == "Designation":

if not new\_value.isalpha():

messagebox.showerror(

"Designation Error", "Invalid Designation")

else:

sql = "UPDATE emp\_details SET desi = %s WHERE emp\_id = %s"

elif field\_to\_update == "Salary":

if not new\_value.isdigit():

messagebox.showerror("Salary Error", "Invalid Salary")

else:

sql = "UPDATE emp\_details SET salary = %s WHERE emp\_id = %s"

try:

mycursor.execute(sql, (new\_value, emp\_id))

mysql.commit()

messagebox.showinfo(

"Success", "Employee Record Updated Successfully")

update.destroy()

root.deiconify()

except mq.Error as e:

messagebox.showerror("Error", f"Error updating record: {e}")

else:

messagebox.showinfo(

"Error", "Employee Record does not exist")

clear\_entries()

root.withdraw()

update = Tk()

update.title("Employee Management System - Update Employee Details")

update.geometry("800x800")

update.minsize(600, 400)

update.maxsize(800, 800)

heading\_label = Label(update, text="Update Employee Details", font=(

"Times New Roman", 30, "bold"), padx=30, pady=30)

heading\_label.grid(row=0, column=1, columnspan=2)

id\_label = Label(update, text="Enter Employee ID:",

font=("Times New Roman", 20))

id\_label.grid(row=1, column=1, padx=10, pady=10)

id\_entry = Entry(update, font=("Times New Roman", 20))

id\_entry.grid(row=1, column=2, padx=10, pady=10)

update\_options = ["Select Field", "Name", "Phone",

"Email", "Age", "Designation", "Salary"]

update\_var = StringVar()

update\_var.set(update\_options[0]) # Default value

update\_label = Label(

update, text="Select Field to Update:", font=("Times New Roman", 20))

update\_label.grid(row=2, column=1, padx=10, pady=10)

update\_dropdown = OptionMenu(update, update\_var, \*update\_options)

update\_dropdown.config(font=("Times New Roman", 20))

update\_dropdown.grid(row=2, column=2, padx=10, pady=10)

new\_value\_label = Label(update, text="New Value:",

font=("Times New Roman", 20))

new\_value\_label.grid(row=3, column=1, padx=10, pady=10)

new\_value\_entry = Entry(update, font=("Times New Roman", 20))

new\_value\_entry.grid(row=3, column=2, padx=10, pady=10)

submit\_btn = Button(update, text="Submit", font=(

"Times New Roman", 20), padx=10, pady=10, command=submit)

submit\_btn.grid(row=4, column=2, padx=10, pady=10)

back\_btn = Button(update, text="Back", font=(

"Times New Roman", 20), padx=5, pady=5, command=back)

back\_btn.grid(row=4, column=1, padx=10, pady=10)

update.mainloop()

def exit\_btn():

root.destroy()

root = Tk()

root.geometry("800x800")

root.minsize(600, 400)

root.maxsize(800, 800)

root.title("Employee Management System")

heading\_label = Label(

text="Welcome to Employee Management System", font=("Times New Roman", 30, "bold"), padx=30, pady=30)

heading\_label.pack(fill=X)

button\_padding\_y = 10

add\_emp = Button(text="Add Employee Details", font=(

"Times New Roman", 22), padx=57, pady=20, command=emp\_add)

view\_emp = Button(text="View Employee Details",

font=("Times New Roman", 22), padx=53, pady=20, command=emp\_view)

delete\_emp = Button(text="Delete Employee Details",

font=("Times New Roman", 22), padx=45, pady=20, command=emp\_delete)

update\_emp = Button(text="Update Employee Details",

font=("Times New Roman", 22), padx=42, pady=20, command=emp\_update)

exit\_btn = Button(text="Exit",

font=("Times New Roman", 22), padx=160, pady=20, command=exit\_btn)

add\_emp.pack(pady=10)

view\_emp.pack(pady=10)

delete\_emp.pack(pady=10)

update\_emp.pack(pady=10)

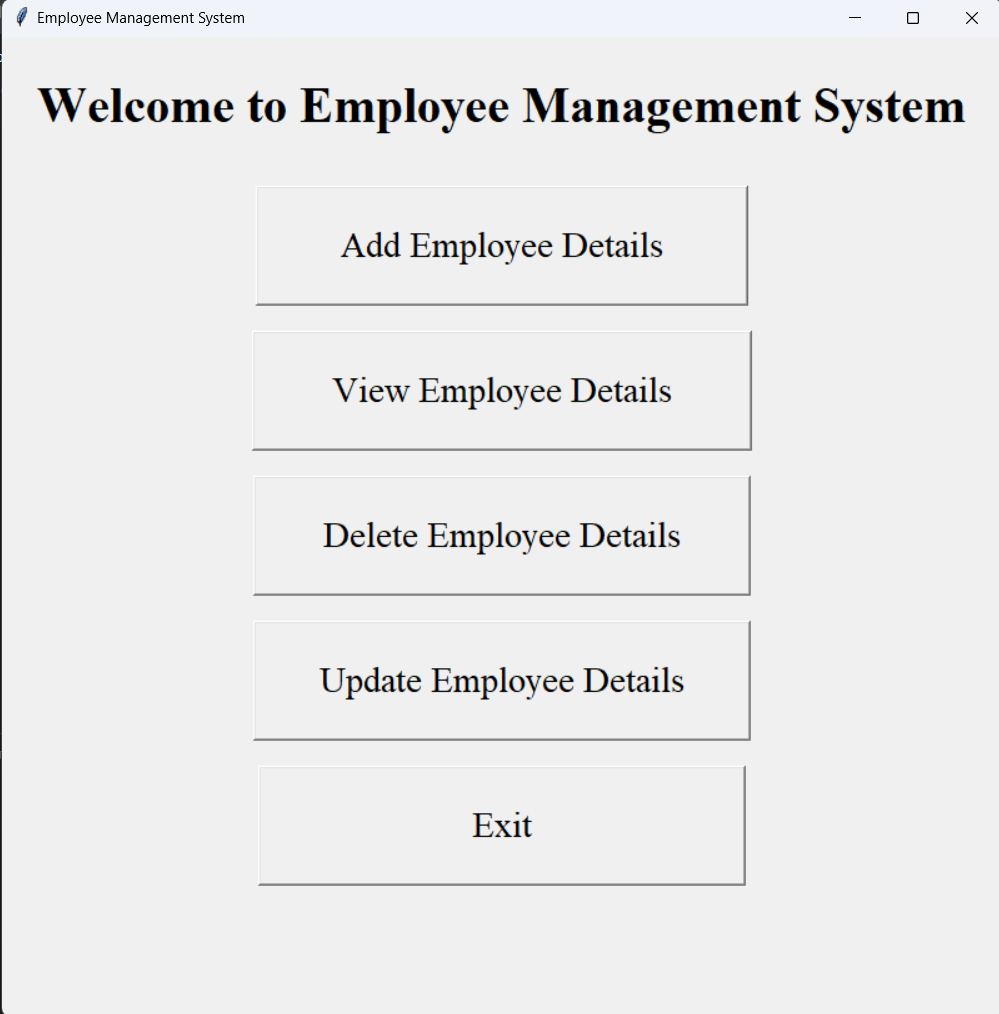
exit\_btn.pack(pady=10)

root.mainloop()

1. **Actual Resources Required:**

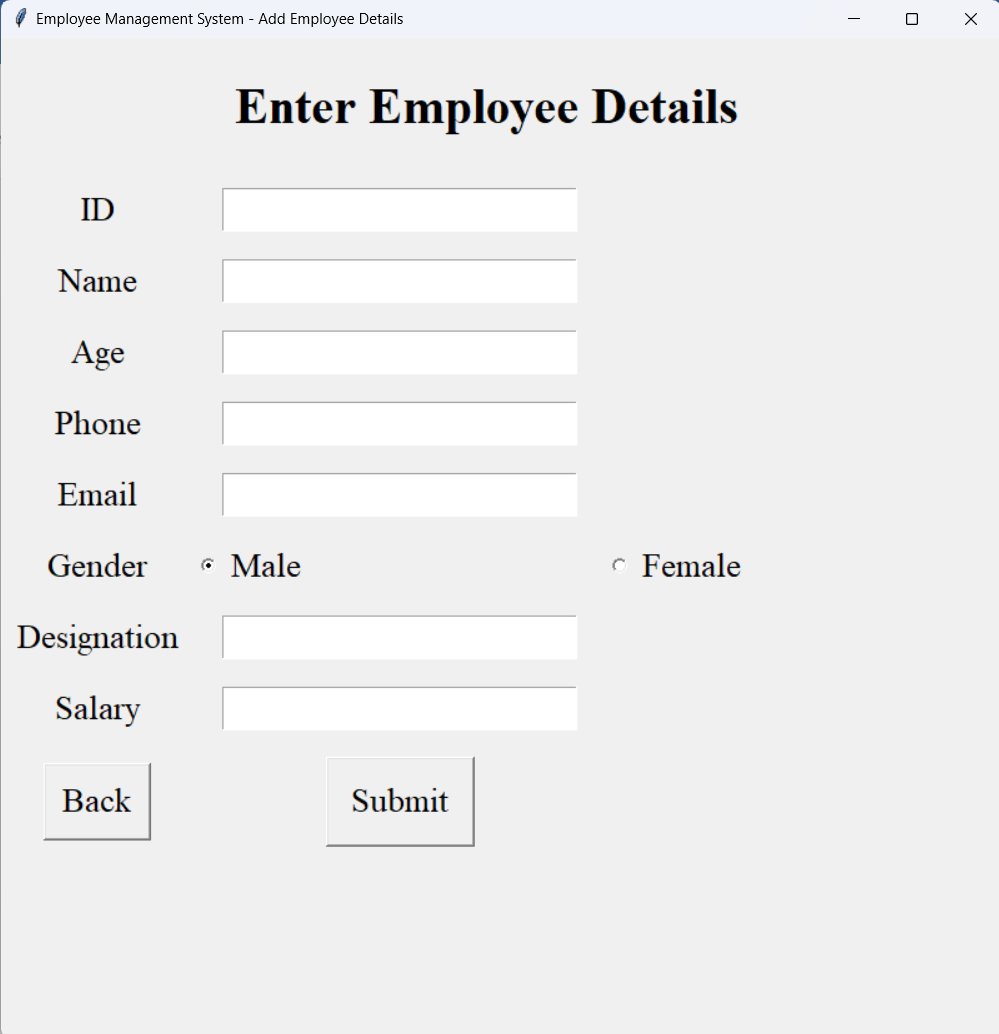
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. no** | **Name of resources** | **Specification** | **Quantity** | **Remark** |
| 1 | Computer system | **Processor:** Intel(R) Pentium(R) Dual CPU [E2140@1.60GHz](mailto:E2140@1.60GHz)1.60GHz  **RAM:** 512 MB | 1 | - |
| 2 | Operating System | **OS:** Windows 7(32bit) | 1 | - |

1. **Output of Micro-Project:**

****

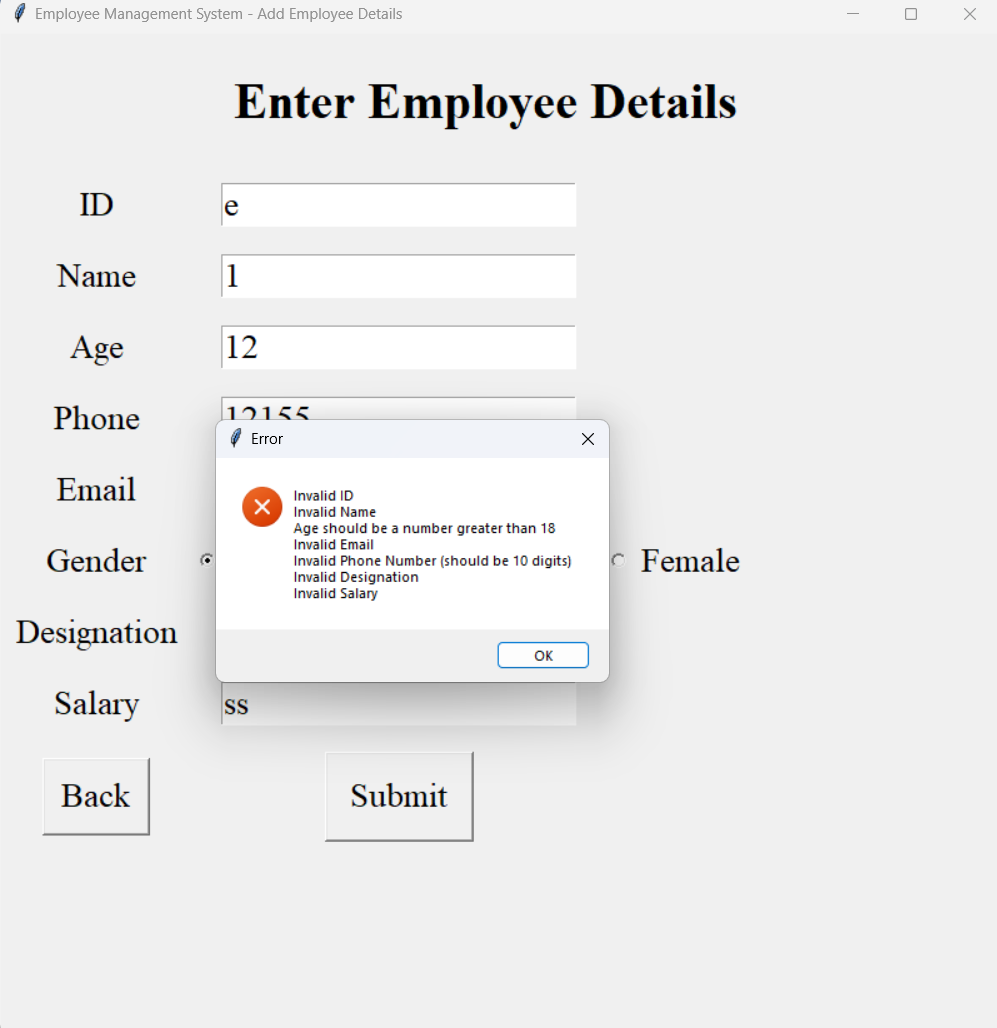
**Fig(5.1): Home Page**

**Fig(5.1):** Contains Welcome Message and Add employee detail, View employee detail, Delete employee detail, Update employee detail, Buttons to navigate to their specific function

****

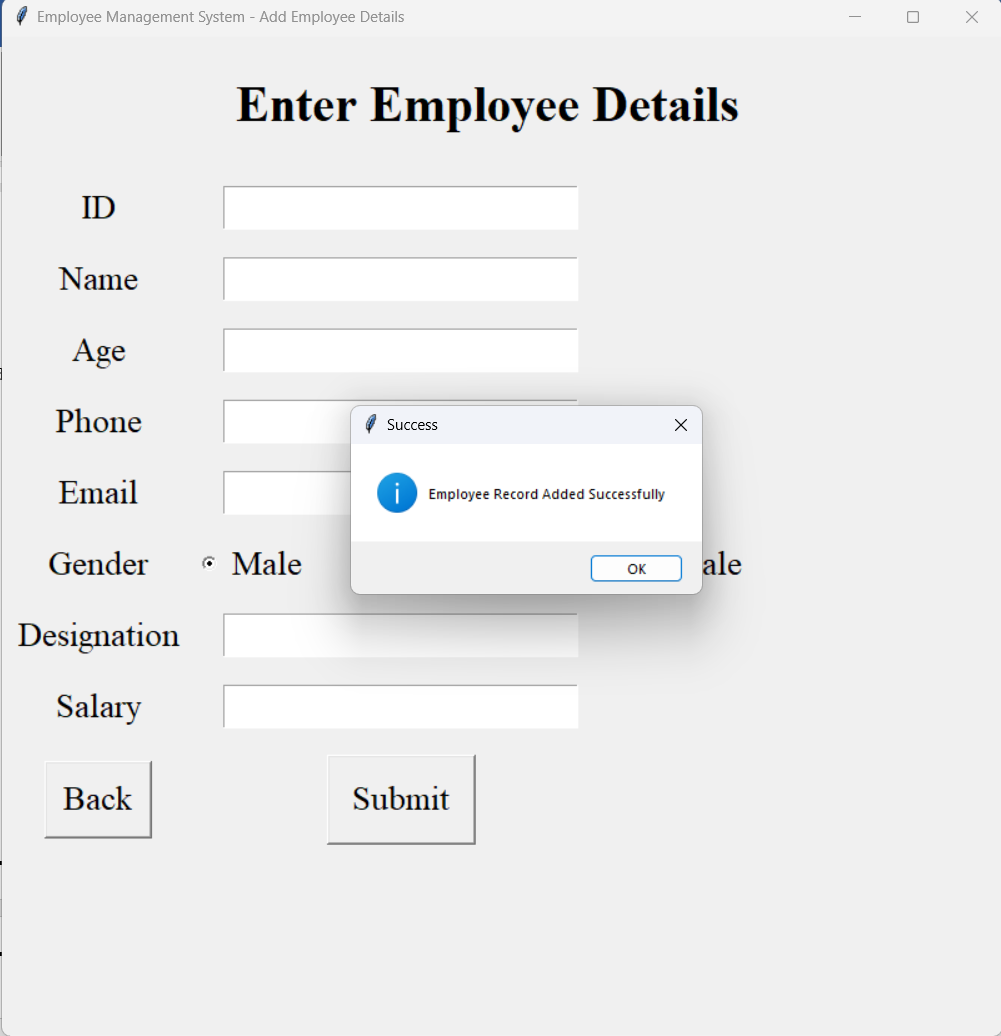
**Fig(5.2):Add Employee Details**

**Fig(5.2):-**Contains label and Entry forAdding new Employee details in the database

****

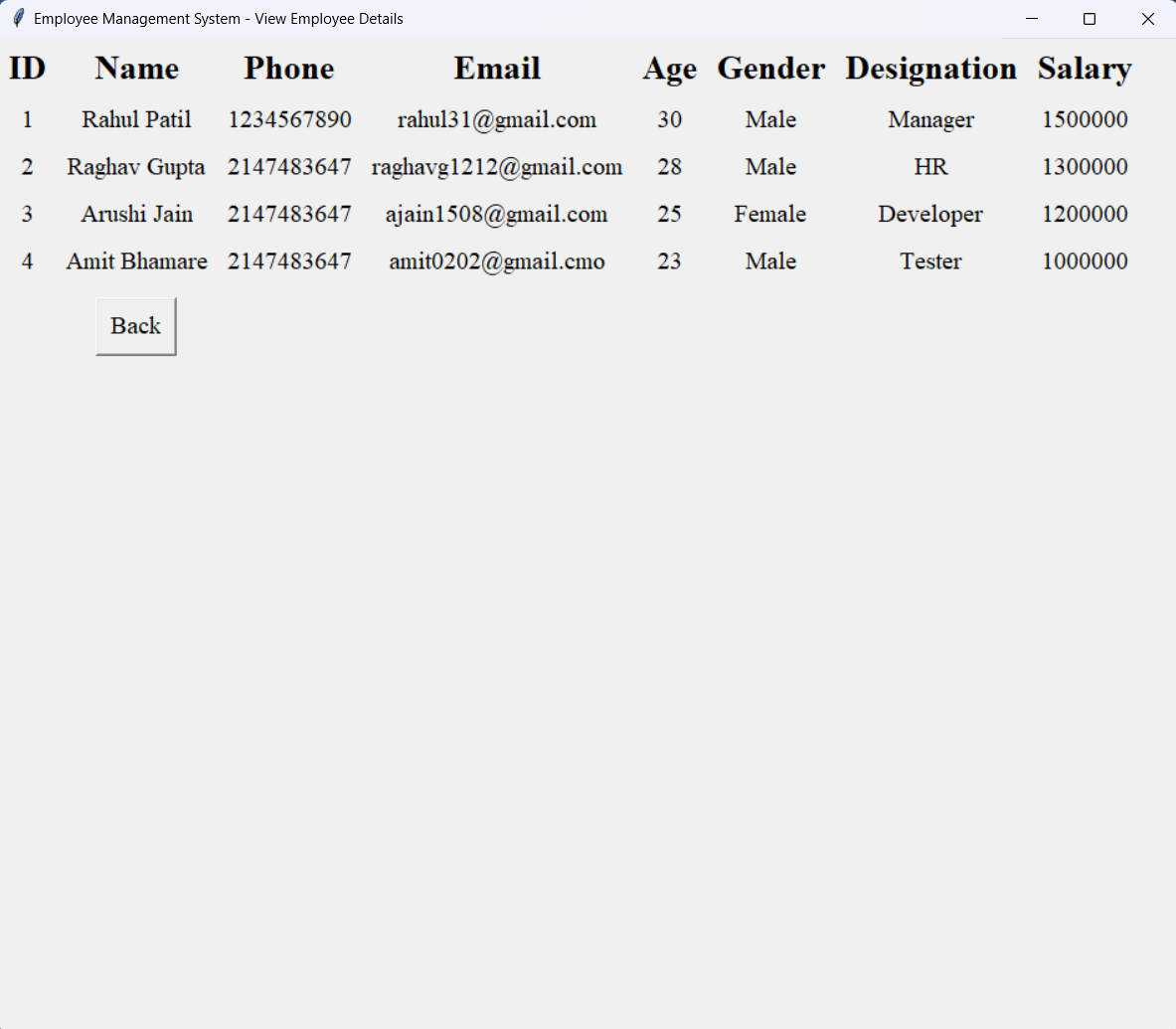
**Fig(5.3):Add Employee Details – Validation**

**Fig(5.3):**Contains Validation of all the entry fields before inserting data into the database, if data is incorrect display error message for specific field

****

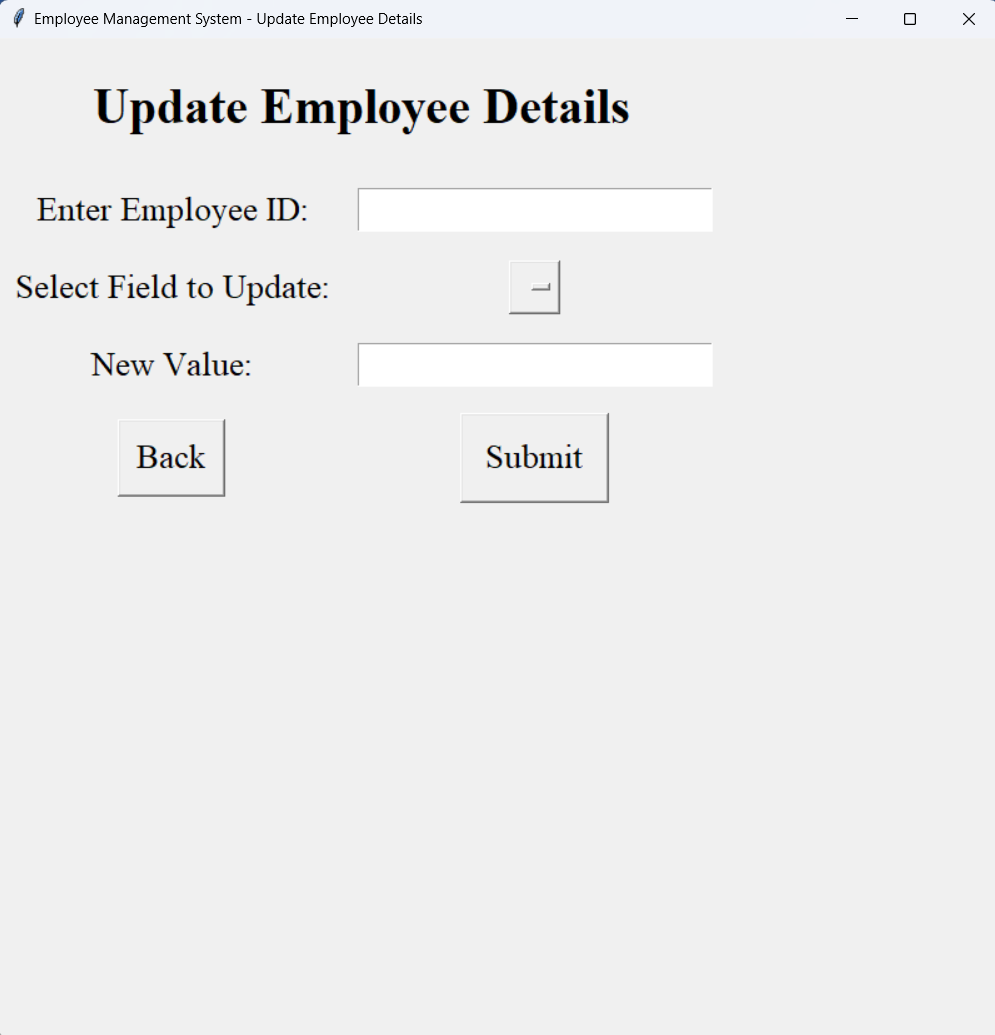
**Fig(5.4):Add Employee Details(Success Message)**

**Fig(5.4):**Contains After validating and verifying that all data is correct insert it into the database. If data is successfully inserted into the database then show success message

****

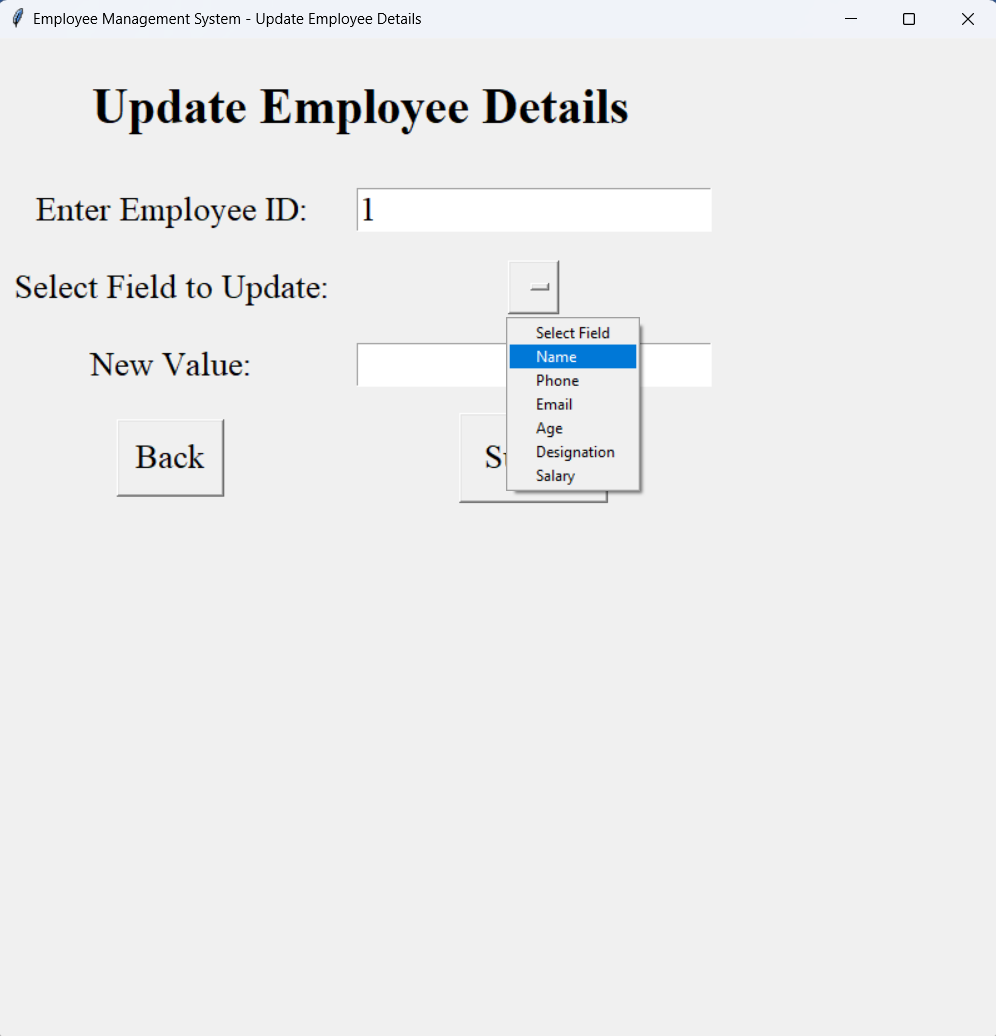
**Fig(5.5):View Employee Details**

**Fig(5.5):** Displaying all the Employers record from the database in table format

****

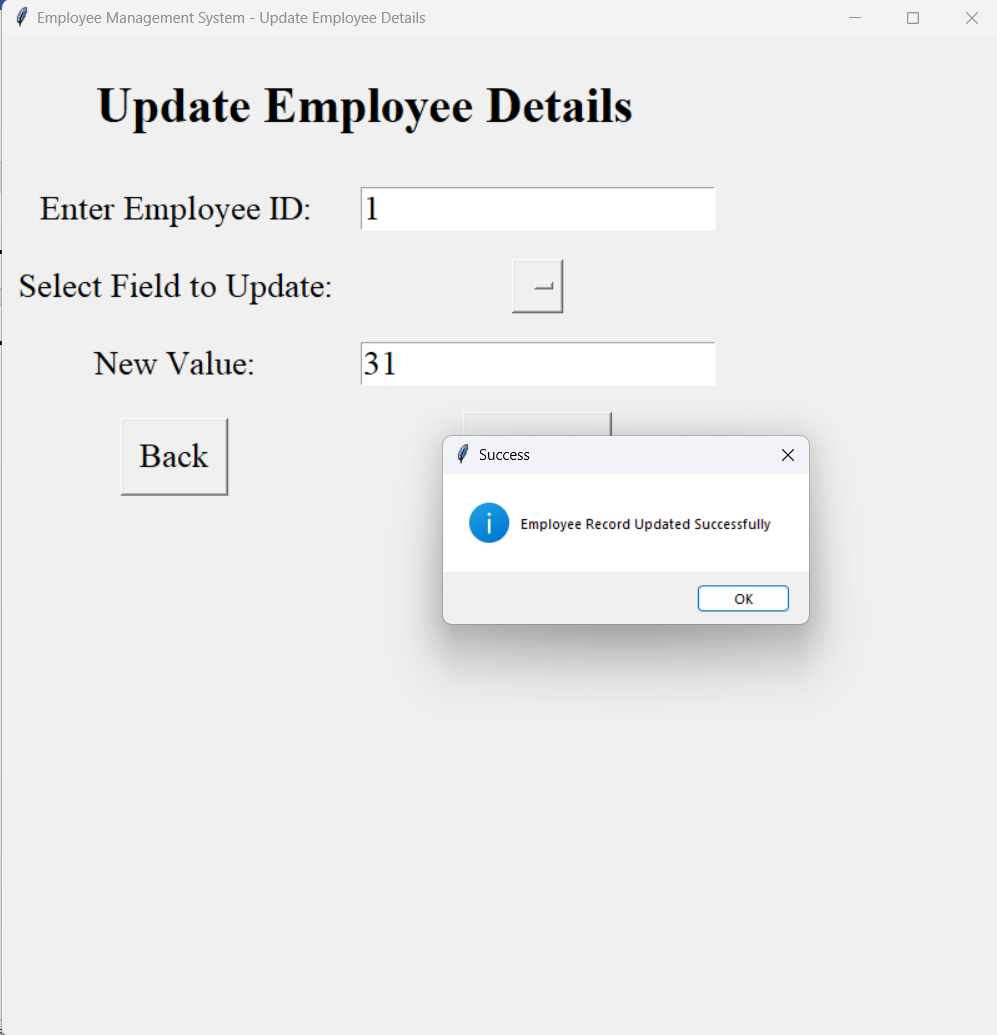
**Fig(5.6):Update Employee Details**

Fig(5.6): Contains Updating details of Employee if employee with entered employee id exists in database

****

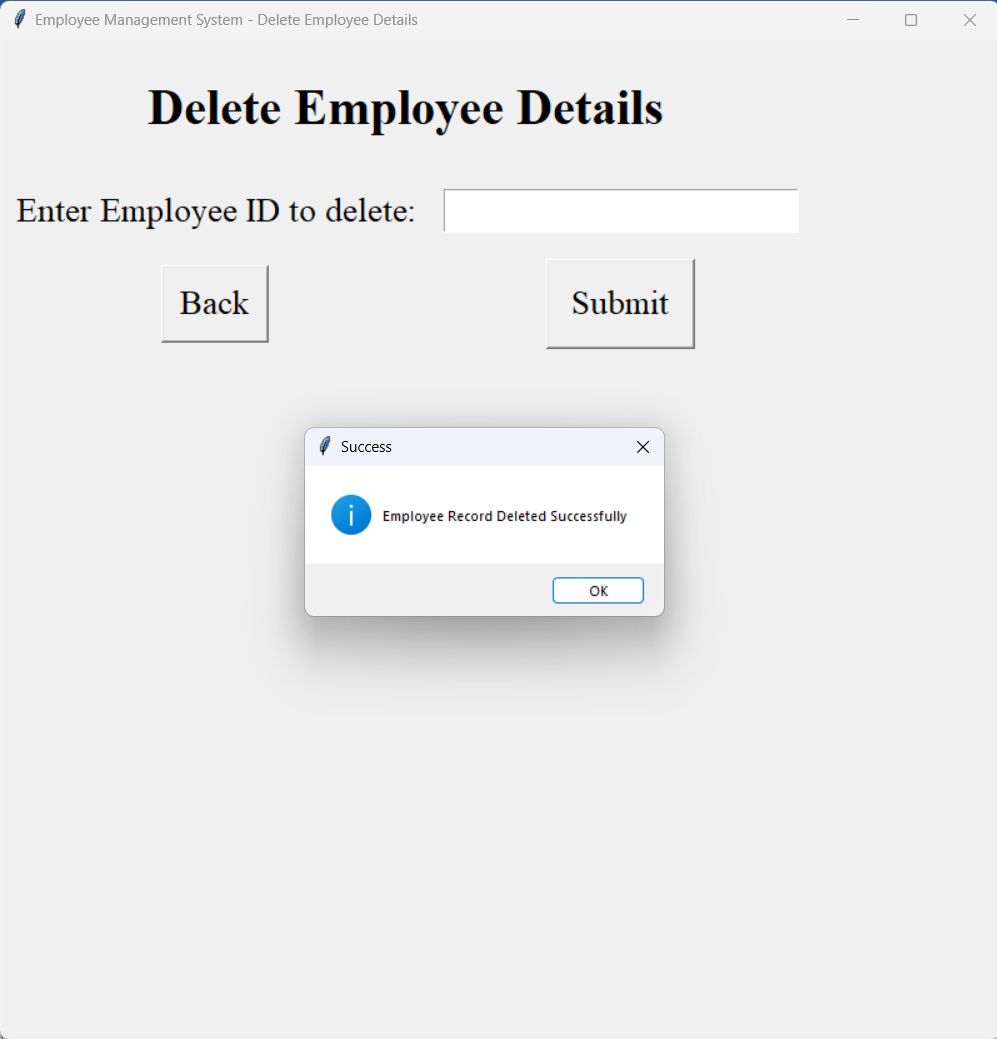
**Fig(5.7):Update Employee Details (Select Field)**

**Fig(5.7):**Contains Selecting which field to update of employee with entered employee id

****

**Fig(5.8):Update Employee Details – Record Updated Successfully**

**Fig(5.8):**Contains Validating the entered field, if correct the update the record of employee with that employee id, else show error message

****

**Fig(5.9):Delete Employee Details**

**Fig(5.9):**Contains Delete record of employee if employee with entered employee id exists in database, else show error message

1. **Skill Developed:**
2. We learned how to define a class and how to create object of class.
3. We learned how to implement the array of object.
4. We learned the concept of menu driven program.
5. We learned how to import package in the program.
6. **Applications of this Micro-Project:**
7. The employee record management system is use in the offices for managing the information of employee.