Ex. No: 13 08.05.2025	Attendance MANAGEMENT SYSTEM

AIM:

To create a Attendance management system dashboard and perform CRUD operation using Tkinter and Python.

Attendance MANAGEMENT DASHBOARD:

- It is web-based application built using Tkinter, SQL-Work Bench that provides user's info in the management.
- The dashboard displays varieties of Attendance management with customers Attendance .
- This project aims to demonstrate the use of work bench that visualizes the info given by customers.

FRONT-END CODING AND BACKEND CODING:

App.py

```
import tkinter as tk
from tkinter import ttk, messagebox
import mysql.connector
from datetime import datetime
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

class AttendanceSystem:
    def _init_(self, root):
        self.root = root
        self.root.title("Attendance Management System")
        self.root.geometry("1200x700")
        self.root.resizable(True, True)

        # Configure style
        self.style = ttk.Style()
        self.style.theme_use('clam')
        self.style.configure('TFrame', background='#f0f0f0')
        self.style.configure('TLabel', background='#f0f0f0', font=('Helvetica', 10))
        self.style.configure('TButton', font=('Helvetica', 10), padding=5)
        self.style.configure('Header.TLabel', font=('Helvetica', 16, 'bold'))
        self.style.configure('Treeview', font=('Helvetica', 10), rowheight=25)
        self.style.configure('Treeview.Heading', font=('Helvetica', 10, 'bold'))
```

```
        # Connect to MySQL Workbench
        self.db_connection = self.connect_to_database()
        self.create_tables()

        # Create GUI
        self.create_main_frame()

    def connect_to_database(self):
        try:
            connection = mysql.connector.connect(
                host='localhost',
                user='root',  # Replace with your MySQL username
                password='1234',  # Replace with your MySQL password
                database='attendance_system'
            )
            return connection
        except mysql.connector.Error as err:
            messagebox.showerror("Database Error", f"Error connecting to MySQL: {err}")
            self.root.destroy()
            exit()

    def create_tables(self):
        cursor = self.db_connection.cursor()

        # Create students table if not exists
        cursor.execute("""
        CREATE TABLE IF NOT EXISTS students (
            student_id INT AUTO_INCREMENT PRIMARY KEY,
            name VARCHAR(100) NOT NULL,
            roll_number VARCHAR(20) UNIQUE NOT NULL,
            class VARCHAR(50) NOT NULL,
            email VARCHAR(100),
            phone VARCHAR(20),
            created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
        )
        """)

        # Create attendance table if not exists
        cursor.execute("""
        CREATE TABLE IF NOT EXISTS attendance (
            attendance_id INT AUTO_INCREMENT PRIMARY KEY,
            student_id INT NOT NULL,
            date DATE NOT NULL,
            status ENUM('Present', 'Absent') NOT NULL,
            FOREIGN KEY (student_id) REFERENCES students(student_id),
            UNIQUE KEY (student_id, date)
        )
        """)
```

```
        self.db_connection.commit()
        cursor.close()

    def create_main_frame(self):
        # Main container
        self.main_frame = ttk.Frame(self.root)
        self.main_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

        # Header
        header_frame = ttk.Frame(self.main_frame)
        header_frame.pack(fill=tk.X, pady=(0, 10))

        ttk.Label(header_frame, text="Attendance Management System",
style='Header.TLabel').pack(side=tk.LEFT)

        # Navigation buttons
        nav_frame = ttk.Frame(self.main_frame)
        nav_frame.pack(fill=tk.X, pady=(0, 10))

        self.student_btn = ttk.Button(nav_frame, text="Manage Students",
command=self.show_student_management)
        self.student_btn.pack(side=tk.LEFT, padx=5)

        self.attendance_btn = ttk.Button(nav_frame, text="Mark Attendance",
command=self.show_attendance)
        self.attendance_btn.pack(side=tk.LEFT, padx=5)

        self.reports_btn = ttk.Button(nav_frame, text="View Reports", command=self.show_reports)
        self.reports_btn.pack(side=tk.LEFT, padx=5)

        # Content frame
        self.content_frame = ttk.Frame(self.main_frame)
        self.content_frame.pack(fill=tk.BOTH, expand=True)

        # Show student management by default
        self.show_student_management()

    def show_student_management(self):
        self.clear_content_frame()

        # Left frame - Add student form
        left_frame = ttk.Frame(self.content_frame)
        left_frame.pack(side=tk.LEFT, fill=tk.Y, padx=10, pady=10)

        ttk.Label(left_frame, text="Add New Student", style='Header.TLabel').pack(pady=(0, 10))

        # Form fields
        ttk.Label(left_frame, text="Full Name:").pack(anchor=tk.W)
        self.name_entry = ttk.Entry(left_frame, width=30)
```

```
        self.name_entry.pack(fill=tk.X, pady=(0, 10))

        ttk.Label(left_frame, text="Roll Number:").pack(anchor=tk.W)
        self.roll_entry = ttk.Entry(left_frame, width=30)
        self.roll_entry.pack(fill=tk.X, pady=(0, 10))

        ttk.Label(left_frame, text="Class:").pack(anchor=tk.W)
        self.class_entry = ttk.Entry(left_frame, width=30)
        self.class_entry.pack(fill=tk.X, pady=(0, 10))

        ttk.Label(left_frame, text="Email:").pack(anchor=tk.W)
        self.email_entry = ttk.Entry(left_frame, width=30)
        self.email_entry.pack(fill=tk.X, pady=(0, 10))

        ttk.Label(left_frame, text="Phone:").pack(anchor=tk.W)
        self.phone_entry = ttk.Entry(left_frame, width=30)
        self.phone_entry.pack(fill=tk.X, pady=(0, 10))

        add_btn = ttk.Button(left_frame, text="Add Student", command=self.add_student)
        add_btn.pack(pady=10)

        # Right frame - Student list
        right_frame = ttk.Frame(self.content_frame)
        right_frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True, padx=10, pady=10)

        ttk.Label(right_frame, text="Student List", style='Header.TLabel').pack(pady=(0, 10))

        # Search frame
        search_frame = ttk.Frame(right_frame)
        search_frame.pack(fill=tk.X, pady=(0, 10))

        ttk.Label(search_frame, text="Search:").pack(side=tk.LEFT)
        self.search_entry = ttk.Entry(search_frame, width=30)
        self.search_entry.pack(side=tk.LEFT, padx=5)
        search_btn = ttk.Button(search_frame, text="Search", command=self.search_students)
        search_btn.pack(side=tk.LEFT, padx=5)
        refresh_btn = ttk.Button(search_frame, text="Refresh", command=self.load_students)
        refresh_btn.pack(side=tk.LEFT)

        # Treeview for student list
        self.student_tree = ttk.Treeview(right_frame, columns=('id', 'name', 'roll', 'class', 'email', 'phone'),
show='headings')
        self.student_tree.heading('id', text='ID')
        self.student_tree.heading('name', text='Name')
        self.student_tree.heading('roll', text='Roll No.')
        self.student_tree.heading('class', text='Class')
        self.student_tree.heading('email', text='Email')
        self.student_tree.heading('phone', text='Phone')
```

```
        self.student_tree.column('id', width=50, anchor=tk.CENTER)
        self.student_tree.column('name', width=150)
        self.student_tree.column('roll', width=100, anchor=tk.CENTER)
        self.student_tree.column('class', width=100, anchor=tk.CENTER)
        self.student_tree.column('email', width=150)
        self.student_tree.column('phone', width=100, anchor=tk.CENTER)

        scrollbar = ttk.Scrollbar(right_frame, orient=tk.VERTICAL, command=self.student_tree.yview)
        self.student_tree.configure(yscroll=scrollbar.set)
        scrollbar.pack(side=tk.RIGHT, fill=tk.Y)
        self.student_tree.pack(fill=tk.BOTH, expand=True)

        # Action buttons
        action_frame = ttk.Frame(right_frame)
        action_frame.pack(fill=tk.X, pady=(10, 0))

        edit_btn = ttk.Button(action_frame, text="Edit", command=self.edit_student)
        edit_btn.pack(side=tk.LEFT, padx=5)

        delete_btn = ttk.Button(action_frame, text="Delete", command=self.delete_student)
        delete_btn.pack(side=tk.LEFT, padx=5)

        # Load student data
        self.load_students()

    def add_student(self):
        name = self.name_entry.get()
        roll = self.roll_entry.get()
        class_ = self.class_entry.get()
        email = self.email_entry.get()
        phone = self.phone_entry.get()

        if not name or not roll or not class_:
            messagebox.showerror("Error", "Name, Roll Number and Class are required!")
            return

        try:
            cursor = self.db_connection.cursor()
            cursor.execute(
                "INSERT INTO students (name, roll_number, class, email, phone) VALUES (%s, %s, %s, %s, %s)",
                (name, roll, class_, email, phone)
            )
            self.db_connection.commit()
            messagebox.showinfo("Success", "Student added successfully!")
            self.clear_student_form()
            self.load_students()
        except mysql.connector.Error as err:
            if err.errno == 1062:  # Duplicate entry
                messagebox.showerror("Error", "Roll number already exists!")
```

```
            else:
                messagebox.showerror("Database Error", f"Error adding student: {err}")
        finally:
            cursor.close()

    def clear_student_form(self):
        self.name_entry.delete(0, tk.END)
        self.roll_entry.delete(0, tk.END)
        self.class_entry.delete(0, tk.END)
        self.email_entry.delete(0, tk.END)
        self.phone_entry.delete(0, tk.END)

    def load_students(self):
        try:
            cursor = self.db_connection.cursor()
            cursor.execute("SELECT student_id, name, roll_number, class, email, phone FROM students")
            rows = cursor.fetchall()

            self.student_tree.delete(*self.student_tree.get_children())
            for row in rows:
                self.student_tree.insert('', tk.END, values=row)
        except mysql.connector.Error as err:
            messagebox.showerror("Database Error", f"Error loading students: {err}")
        finally:
            cursor.close()

    def search_students(self):
        search_term = self.search_entry.get()
        if not search_term:
            self.load_students()
            return

        try:
            cursor = self.db_connection.cursor()
            query = """
            SELECT student_id, name, roll_number, class, email, phone FROM students
            WHERE name LIKE %s OR roll_number LIKE %s OR class LIKE %s
            """
            cursor.execute(query, (f"%{search_term}%", f"%{search_term}%", f"%{search_term}%"))
            rows = cursor.fetchall()

            self.student_tree.delete(*self.student_tree.get_children())
            for row in rows:
                self.student_tree.insert('', tk.END, values=row)
        except mysql.connector.Error as err:
            messagebox.showerror("Database Error", f"Error searching students: {err}")
        finally:
            cursor.close()
```

```
    def edit_student(self):
        selected_item = self.student_tree.selection()
        if not selected_item:
            messagebox.showerror("Error", "Please select a student to edit!")
            return

        item = self.student_tree.item(selected_item[0])
        student_id = item['values'][0]

        # Open edit window
        edit_window = tk.Toplevel(self.root)
        edit_window.title("Edit Student")
        edit_window.geometry("400x300")

        # Get student details
        try:
            cursor = self.db_connection.cursor()
            cursor.execute("SELECT name, roll_number, class, email, phone FROM students WHERE student_id
= %s", (student_id,))
            student = cursor.fetchone()
        except mysql.connector.Error as err:
            messagebox.showerror("Database Error", f"Error fetching student: {err}")
            edit_window.destroy()
            return
        finally:
            cursor.close()

        # Form fields
        ttk.Label(edit_window, text="Edit Student", style='Header.TLabel').pack(pady=(10, 20))

        ttk.Label(edit_window, text="Full Name:").pack(anchor=tk.W)
        name_entry = ttk.Entry(edit_window, width=30)
        name_entry.insert(0, student[0])
        name_entry.pack(fill=tk.X, pady=(0, 10))

        ttk.Label(edit_window, text="Roll Number:").pack(anchor=tk.W)
        roll_entry = ttk.Entry(edit_window, width=30)
        roll_entry.insert(0, student[1])
        roll_entry.pack(fill=tk.X, pady=(0, 10))

        ttk.Label(edit_window, text="Class:").pack(anchor=tk.W)
        class_entry = ttk.Entry(edit_window, width=30)
        class_entry.insert(0, student[2])
        class_entry.pack(fill=tk.X, pady=(0, 10))

        ttk.Label(edit_window, text="Email:").pack(anchor=tk.W)
        email_entry = ttk.Entry(edit_window, width=30)
        email_entry.insert(0, student[3])
        email_entry.pack(fill=tk.X, pady=(0, 10))
```

```
        ttk.Label(edit_window, text="Phone:").pack(anchor=tk.W)
        phone_entry = ttk.Entry(edit_window, width=30)
        phone_entry.insert(0, student[4])
        phone_entry.pack(fill=tk.X, pady=(0, 10))

        def update_student():
            name = name_entry.get()
            roll = roll_entry.get()
            class_ = class_entry.get()
            email = email_entry.get()
            phone = phone_entry.get()

            if not name or not roll or not class_:
                messagebox.showerror("Error", "Name, Roll Number and Class are required!")
                return

            try:
                cursor = self.db_connection.cursor()
                cursor.execute(
                    "UPDATE students SET name = %s, roll_number = %s, class = %s, email = %s, phone = %s WHERE
student_id = %s",
                    (name, roll, class_, email, phone, student_id)
                )
                self.db_connection.commit()
                messagebox.showinfo("Success", "Student updated successfully!")
                edit_window.destroy()
                self.load_students()
            except mysql.connector.Error as err:
                if err.errno == 1062:  # Duplicate entry
                    messagebox.showerror("Error", "Roll number already exists!")
                else:
                    messagebox.showerror("Database Error", f"Error updating student: {err}")
            finally:
                cursor.close()

        ttk.Button(edit_window, text="Update", command=update_student).pack(pady=10)

    def delete_student(self):
        selected_item = self.student_tree.selection()
        if not selected_item:
            messagebox.showerror("Error", "Please select a student to delete!")
            return

        if not messagebox.askyesno("Confirm", "Are you sure you want to delete this student?"):
            return

        item = self.student_tree.item(selected_item[0])
        student_id = item['values'][0]
```

```
        try:
            cursor = self.db_connection.cursor()

            # First delete attendance records for this student
            cursor.execute("DELETE FROM attendance WHERE student_id = %s", (student_id,))

            # Then delete the student
            cursor.execute("DELETE FROM students WHERE student_id = %s", (student_id,))

            self.db_connection.commit()
            messagebox.showinfo("Success", "Student deleted successfully!")
            self.load_students()
        except mysql.connector.Error as err:
            messagebox.showerror("Database Error", f"Error deleting student: {err}")
            self.db_connection.rollback()
        finally:
            cursor.close()

    def show_attendance(self):
        self.clear_content_frame()

        # Top frame - Date selection
        top_frame = ttk.Frame(self.content_frame)
        top_frame.pack(fill=tk.X, pady=(0, 10))

        ttk.Label(top_frame, text="Select Date:").pack(side=tk.LEFT)
        self.attendance_date = ttk.Entry(top_frame, width=15)
        self.attendance_date.insert(0, datetime.now().strftime('%Y-%m-%d'))
        self.attendance_date.pack(side=tk.LEFT, padx=5)

        load_btn = ttk.Button(top_frame, text="Load", command=self.load_attendance)
        load_btn.pack(side=tk.LEFT, padx=5)

        # Middle frame - Attendance treeview
        middle_frame = ttk.Frame(self.content_frame)
        middle_frame.pack(fill=tk.BOTH, expand=True)

        self.attendance_tree = ttk.Treeview(middle_frame, columns=('id', 'name', 'roll', 'class', 'status'),
show='headings')
        self.attendance_tree.heading('id', text='ID')
        self.attendance_tree.heading('name', text='Name')
        self.attendance_tree.heading('roll', text='Roll No.')
        self.attendance_tree.heading('class', text='Class')
        self.attendance_tree.heading('status', text='Status')

        self.attendance_tree.column('id', width=50, anchor=tk.CENTER)
        self.attendance_tree.column('name', width=150)
        self.attendance_tree.column('roll', width=100, anchor=tk.CENTER)
```

```
        self.attendance_tree.column('class', width=100, anchor=tk.CENTER)
        self.attendance_tree.column('status', width=100, anchor=tk.CENTER)

        scrollbar = ttk.Scrollbar(middle_frame, orient=tk.VERTICAL, command=self.attendance_tree.yview)
        self.attendance_tree.configure(yscroll=scrollbar.set)
        scrollbar.pack(side=tk.RIGHT, fill=tk.Y)
        self.attendance_tree.pack(fill=tk.BOTH, expand=True)

        # Bind double click to toggle status
        self.attendance_tree.bind('<Double-1>', self.toggle_attendance_status)

        # Bottom frame - Save button
        bottom_frame = ttk.Frame(self.content_frame)
        bottom_frame.pack(fill=tk.X, pady=(10, 0))

        save_btn = ttk.Button(bottom_frame, text="Save Attendance", command=self.save_attendance)
        save_btn.pack(pady=5)

        # Load attendance for today by default
        self.load_attendance()

    def load_attendance(self):
        date = self.attendance_date.get()

        try:
            datetime.strptime(date, '%Y-%m-%d')  # Validate date format
        except ValueError:
            messagebox.showerror("Error", "Invalid date format! Please use YYYY-MM-DD")
            return

        try:
            cursor = self.db_connection.cursor()

            # Get all students
            cursor.execute("SELECT student_id, name, roll_number, class FROM students ORDER BY
roll_number")
            students = cursor.fetchall()

            # Get attendance for selected date
            cursor.execute("""
            SELECT a.student_id, a.status
            FROM attendance a
            WHERE a.date = %s
            """, (date,))
            attendance_records = {row[0]: row[1] for row in cursor.fetchall()}

            self.attendance_tree.delete(*self.attendance_tree.get_children())
            for student in students:
                student_id, name, roll, class_ = student
```

```
            status = attendance_records.get(student_id, 'Absent')
            self.attendance_tree.insert('', tk.END, values=(student_id, name, roll, class_, status))
        except mysql.connector.Error as err:
            messagebox.showerror("Database Error", f"Error loading attendance: {err}")
        finally:
            cursor.close()

    def toggle_attendance_status(self, event):
        item = self.attendance_tree.selection()[0]
        values = self.attendance_tree.item(item, 'values')

        if values[4] == 'Present':
            new_status = 'Absent'
        else:
            new_status = 'Present'

        self.attendance_tree.item(item, values=(values[0], values[1], values[2], values[3], new_status))

    def save_attendance(self):
        date = self.attendance_date.get()

        try:
            datetime.strptime(date, '%Y-%m-%d')  # Validate date format
        except ValueError:
            messagebox.showerror("Error", "Invalid date format! Please use YYYY-MM-DD")
            return

        try:
            cursor = self.db_connection.cursor()

            # Delete existing attendance for this date
            cursor.execute("DELETE FROM attendance WHERE date = %s", (date,))

            # Insert new attendance records
            for item in self.attendance_tree.get_children():
                values = self.attendance_tree.item(item, 'values')
                student_id = values[0]
                status = values[4]

                if status == 'Present':
                    cursor.execute(
                        "INSERT INTO attendance (student_id, date, status) VALUES (%s, %s, %s)",
                        (student_id, date, status)
                    )

            self.db_connection.commit()
            messagebox.showinfo("Success", "Attendance saved successfully!")
        except mysql.connector.Error as err:
            messagebox.showerror("Database Error", f"Error saving attendance: {err}")
```

```
            self.db_connection.rollback()
        finally:
            cursor.close()

    def show_reports(self):
        self.clear_content_frame()

        # Create notebook for multiple report tabs
        notebook = ttk.Notebook(self.content_frame)
        notebook.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

        # Daily Attendance Report
        daily_frame = ttk.Frame(notebook)
        notebook.add(daily_frame, text="Daily Attendance")

        # Date selection
        date_frame = ttk.Frame(daily_frame)
        date_frame.pack(fill=tk.X, pady=(0, 10))

        ttk.Label(date_frame, text="Select Date:").pack(side=tk.LEFT)
        self.report_date = ttk.Entry(date_frame, width=15)
        self.report_date.insert(0, datetime.now().strftime('%Y-%m-%d'))
        self.report_date.pack(side=tk.LEFT, padx=5)

        load_daily_btn = ttk.Button(date_frame, text="Load Report", command=self.load_daily_report)
        load_daily_btn.pack(side=tk.LEFT, padx=5)

        # Daily report treeview
        self.daily_report_tree = ttk.Treeview(daily_frame, columns=('id', 'name', 'roll', 'class', 'status'),
show='headings')
        self.daily_report_tree.heading('id', text='ID')
        self.daily_report_tree.heading('name', text='Name')
        self.daily_report_tree.heading('roll', text='Roll No.')
        self.daily_report_tree.heading('class', text='Class')
        self.daily_report_tree.heading('status', text='Status')

        self.daily_report_tree.column('id', width=50, anchor=tk.CENTER)
        self.daily_report_tree.column('name', width=150)
        self.daily_report_tree.column('roll', width=100, anchor=tk.CENTER)
        self.daily_report_tree.column('class', width=100, anchor=tk.CENTER)
        self.daily_report_tree.column('status', width=100, anchor=tk.CENTER)

        scrollbar = ttk.Scrollbar(daily_frame, orient=tk.VERTICAL, command=self.daily_report_tree.yview)
        self.daily_report_tree.configure(yscroll=scrollbar.set)
        scrollbar.pack(side=tk.RIGHT, fill=tk.Y)
        self.daily_report_tree.pack(fill=tk.BOTH, expand=True)

        # Summary frame
        summary_frame = ttk.Frame(daily_frame)
```

```
        summary_frame.pack(fill=tk.X, pady=(10, 0))

        self.present_count = ttk.Label(summary_frame, text="Present: 0")
        self.present_count.pack(side=tk.LEFT, padx=10)

        self.absent_count = ttk.Label(summary_frame, text="Absent: 0")
        self.absent_count.pack(side=tk.LEFT, padx=10)

        # Monthly Report
        monthly_frame = ttk.Frame(notebook)
        notebook.add(monthly_frame, text="Monthly Report")

        # Month selection
        month_frame = ttk.Frame(monthly_frame)
        month_frame.pack(fill=tk.X, pady=(0, 10))

        ttk.Label(month_frame, text="Select Month:").pack(side=tk.LEFT)
        self.report_month = ttk.Combobox(month_frame, width=10, values=[
            '01', '02', '03', '04', '05', '06', '07', '08', '09', '10', '11', '12'
        ])
        self.report_month.set(datetime.now().strftime('%m'))
        self.report_month.pack(side=tk.LEFT, padx=5)

        ttk.Label(month_frame, text="Year:").pack(side=tk.LEFT)
        self.report_year = ttk.Entry(month_frame, width=8)
        self.report_year.insert(0, datetime.now().strftime('%Y'))
        self.report_year.pack(side=tk.LEFT, padx=5)

        load_monthly_btn = ttk.Button(month_frame, text="Load Report",
command=self.load_monthly_report)
        load_monthly_btn.pack(side=tk.LEFT, padx=5)

        # Monthly report treeview
        self.monthly_report_tree = ttk.Treeview(monthly_frame, columns=('id', 'name', 'roll', 'class', 'present',
'absent'), show='headings')
        self.monthly_report_tree.heading('id', text='ID')
        self.monthly_report_tree.heading('name', text='Name')
        self.monthly_report_tree.heading('roll', text='Roll No.')
        self.monthly_report_tree.heading('class', text='Class')
        self.monthly_report_tree.heading('present', text='Present Days')
        self.monthly_report_tree.heading('absent', text='Absent Days')

        self.monthly_report_tree.column('id', width=50, anchor=tk.CENTER)
        self.monthly_report_tree.column('name', width=150)
        self.monthly_report_tree.column('roll', width=100, anchor=tk.CENTER)
        self.monthly_report_tree.column('class', width=100, anchor=tk.CENTER)
        self.monthly_report_tree.column('present', width=100, anchor=tk.CENTER)
        self.monthly_report_tree.column('absent', width=100, anchor=tk.CENTER)
```

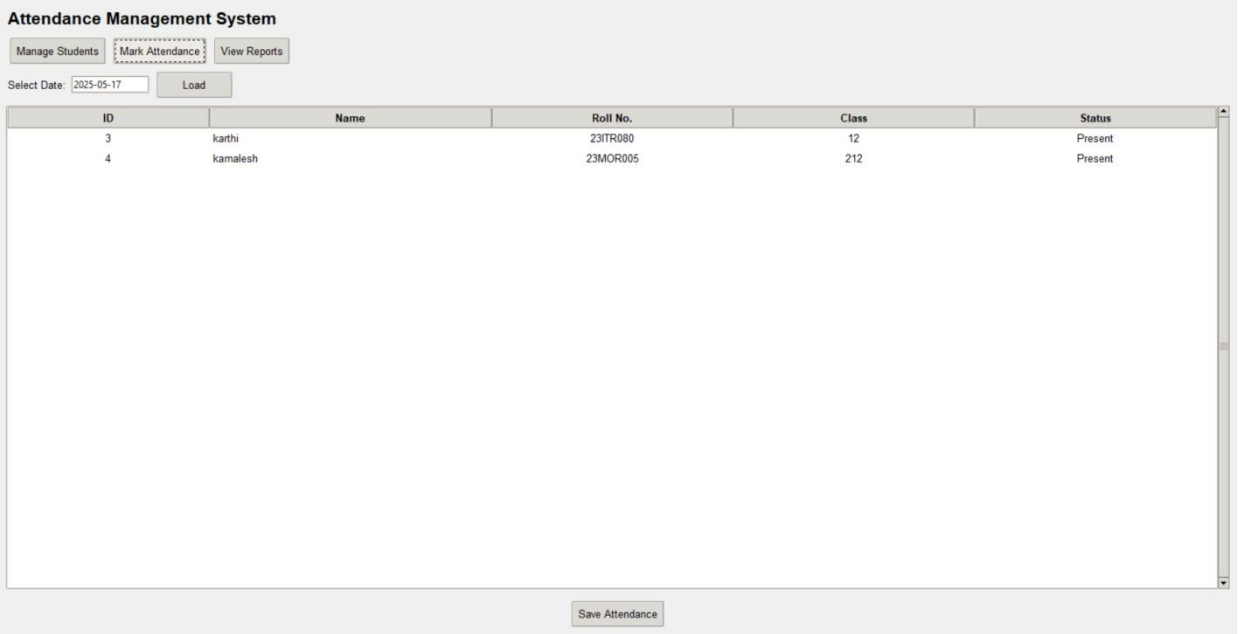
```
        scrollbar = ttk.Scrollbar(monthly_frame, orient=tk.VERTICAL,
command=self.monthly_report_tree.yview)
        self.monthly_report_tree.configure(yscroll=scrollbar.set)
        scrollbar.pack(side=tk.RIGHT, fill=tk.Y)
        self.monthly_report_tree.pack(fill=tk.BOTH, expand=True)

        # Chart frame
        chart_frame = ttk.Frame(monthly_frame)
        chart_frame.pack(fill=tk.BOTH, expand=True, pady=(10, 0))

        self.fig, self.ax = plt.subplots(figsize=(6, 3), dpi=100)
        self.canvas = FigureCanvasTkAgg(self.fig, master=chart_frame)
        self.canvas.get_tk_widget().pack(fill=tk.BOTH, expand=True)

        # Load today's report by default
        self.load_daily_report()

    def load_daily_report(self):
        date = self.report_date.get()

        try:
            datetime.strptime(date, '%Y-%m-%d')  # Validate date format
        except ValueError:
            messagebox.showerror("Error", "Invalid date format! Please use YYYY-MM-DD")
            return

        try:
            cursor = self.db_connection.cursor()

            # Get attendance for selected date
            cursor.execute("""
            SELECT s.student_id, s.name, s.roll_number, s.class,
                COALESCE(a.status, 'Absent') as status
            FROM students s
            LEFT JOIN attendance a ON s.student_id = a.student_id AND a.date = %s
            ORDER BY s.roll_number
            """, (date,))
            rows = cursor.fetchall()

            self.daily_report_tree.delete(*self.daily_report_tree.get_children())

            present_count = 0
            absent_count = 0

            for row in rows:
                student_id, name, roll, class_, status = row
                self.daily_report_tree.insert('', tk.END, values=(student_id, name, roll, class_, status))

                if status == 'Present':
```

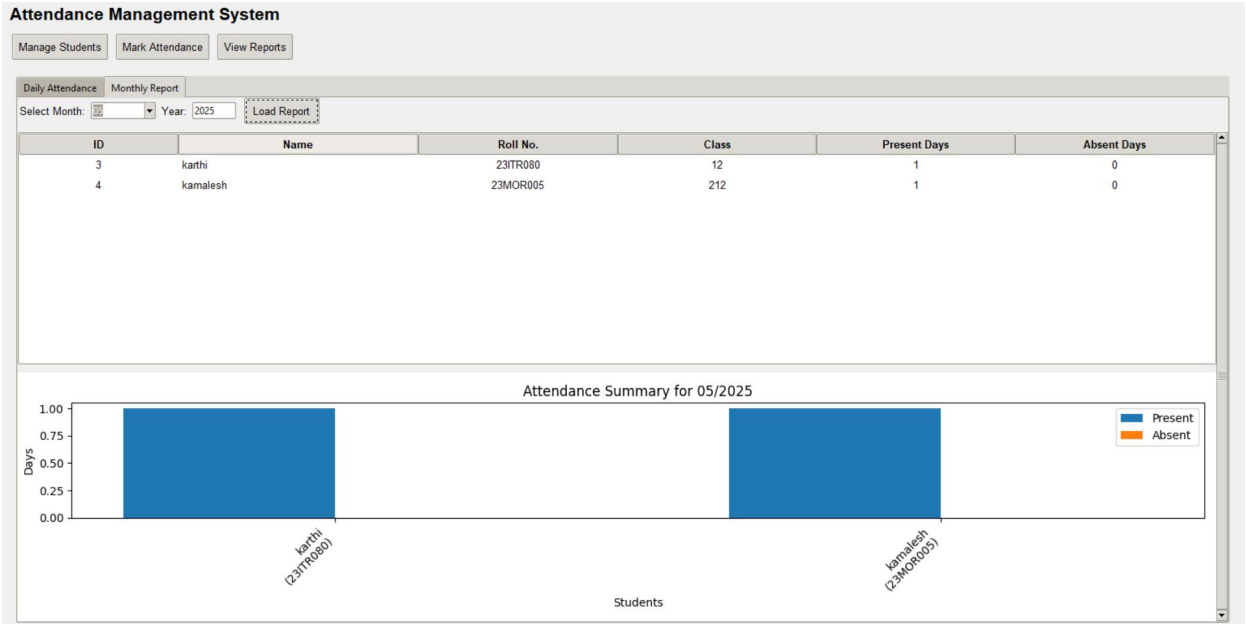
```
                present_count += 1
            else:
                absent_count += 1

        # Update summary
        self.present_count.config(text=f"Present: {present_count}")
        self.absent_count.config(text=f"Absent: {absent_count}")

    except mysql.connector.Error as err:
        messagebox.showerror("Database Error", f"Error loading daily report: {err}")
    finally:
        cursor.close()

def load_monthly_report(self):
    month = self.report_month.get()
    year = self.report_year.get()

    try:
        datetime.strptime(f"{year}-{month}-01", '%Y-%m-%d')  # Validate date
    except ValueError:
        messagebox.showerror("Error", "Invalid month/year!")
        return

    try:
        cursor = self.db_connection.cursor()

        # Get attendance summary for the month
        cursor.execute("""
        SELECT s.student_id, s.name, s.roll_number, s.class,
               SUM(CASE WHEN a.status = 'Present' THEN 1 ELSE 0 END) as present_days,
               SUM(CASE WHEN a.status = 'Absent' THEN 1 ELSE 0 END) as absent_days
        FROM students s
        LEFT JOIN attendance a ON s.student_id = a.student_id
            AND YEAR(a.date) = %s AND MONTH(a.date) = %s
        GROUP BY s.student_id
        ORDER BY s.roll_number
        """, (year, month))
        rows = cursor.fetchall()

        self.monthly_report_tree.delete(*self.monthly_report_tree.get_children())

        present_counts = []
        absent_counts = []
        student_names = []

        for row in rows:
            student_id, name, roll, class_, present, absent = row
            self.monthly_report_tree.insert('', tk.END, values=(student_id, name, roll, class_, present, absent))
```

```
                present_counts.append(present)
                absent_counts.append(absent)
                student_names.append(f"{name}\n({roll})")

            # Update chart
            self.ax.clear()

            if rows:
                x = range(len(student_names))
                width = 0.35

                self.ax.bar(x, present_counts, width, label='Present')
                self.ax.bar([p + width for p in x], absent_counts, width, label='Absent')

                self.ax.set_xlabel('Students')
                self.ax.set_ylabel('Days')
                self.ax.set_title(f'Attendance Summary for {month}/{year}')
                self.ax.set_xticks([p + width/2 for p in x])
                self.ax.set_xticklabels(student_names, rotation=45, ha='right')
                self.ax.legend()

                self.fig.tight_layout()
                self.canvas.draw()

        except mysql.connector.Error as err:
            messagebox.showerror("Database Error", f"Error loading monthly report: {err}")
        finally:
            cursor.close()

    def clear_content_frame(self):
        for widget in self.content_frame.winfo_children():
            widget.destroy()

if _name_ == "_main_":
    root = tk.Tk()
    app = AttendanceSystem(root)
    root.mainloop()
```

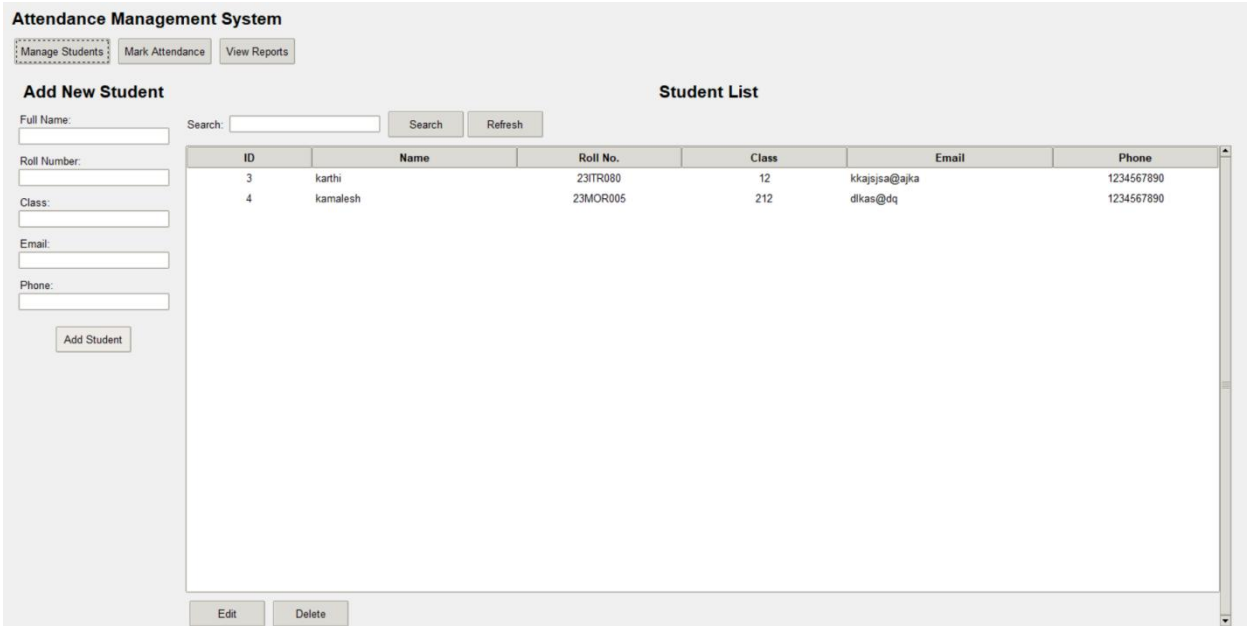
Output:

Home page:





Attendance Management page:



SQL QUERY:

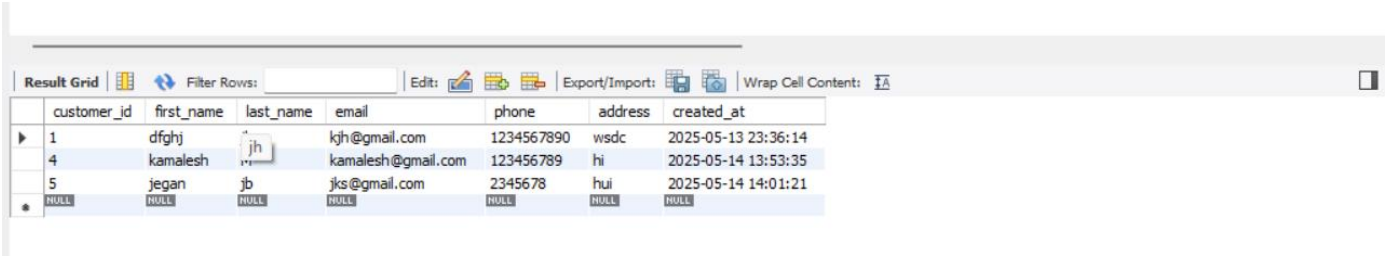
CREATING TABLE:

```
CREATE TABLE IF NOT EXISTS student (
   student_id INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(100) NOT NULL,
   roll_number VARCHAR(20) UNIQUE NOT NULL,
   class VARCHAR(50) NOT NULL,
   email VARCHAR(100),
   phone VARCHAR(20),
   created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

DISPLAY:

```
SELECT * FROM students;
```

OUTPUT:



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

customer_id	first_name	last_name	email	phone	address	created_at
1	dfghj	jh	kjh@gmail.com	1234567890	wsdc	2025-05-13 23:36:14
4	kamalesh	[illegible]	kamalesh@gmail.com	123456789	hi	2025-05-14 13:53:35
5	jegan	jb	jks@gmail.com	2345678	hui	2025-05-14 14:01:21
NULL	NULL	NULL	NULL	NULL	NULL	NULL

CONTENTS	MARKS ALLOTED	MARKS OBTAINED
Aim, Algorithm, SQL, PL/SQL	30	
Execution and Result	20	
Viva	10	
Total	60	

RESULT:

Thus a Attendance management system dashboard was created and performed CRUD operation using Tkinter and Python.