EMPLOYEE MANAGEMENT SYSTEM

MINI PROJECT REPORT

Submitted by

VINOTH J 220701322

THEJOEAM S 220701304

In partial fulfilment for the award of degree
of

BACHELOR OF ENGINEERING
IN
COMPUTER SCIENCE AND ENGINEERING

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)
THANDALAM CHENNAI-602105

JUNE-2024

BONAFIDE CERTIFICATE

Certified that this project report "EMPLOYEE MANAGEMENT SYSTEM" is the bonafide work of "**VINOTH J(220701322),THEJORAM S(220701304)**"

who carried out the project work under my supervision.

SIGNATURE

Dr.R.SABITHA

Professor and II Year Academic Head

Computer Science and Engineering,

Rajalakshmi Engineering College

(Autonomous)

Thandalam, Chennai-602 105

SIGNATURE

Ms.V.JANANEE

Assistant Professor (SG),

Computer Science and Engineering,

Rajalakshmi Engineering College

(Autonomous)

Thandalam, Chennai-602 105

Submitted for the Practical Examination held on _____

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ABSTRACT

The Employee Management System (EMS) is a robust and comprehensive application designed to streamline and optimize the management of employee records and processes within an organization. This system facilitates efficient handling of employee information, including personal details, job roles, and salary data. EMS enables administrators to manage employee records, track performance, and monitor attendance seamlessly.

Key features include the capability to add new employee records, update existing information, and delete records as necessary. By automating routine HR tasks, the EMS reduces manual workload, minimizes errors, and enhances the overall productivity of the HR department.

Built using a combination of SQL for database management and Python for application logic, the system ensures data integrity and provides a user-friendly interface for ease of operation. The integration of these technologies allows for real-time data processing and retrieval, ensuring that information is always current and accessible.

In summary, the Employee Management System offers a comprehensive solution for managing employee-related activities, fostering a more organized and efficient work environment.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

1.1 INTRODUCTION

The Employee Management System (EMS) is designed to streamline the management of employee records and related processes. It handles personal details, job roles, and salary data, allowing for efficient management of employee records, performance tracking, and attendance monitoring. The EMS provides functionalities for adding, updating, and deleting employee records and generating detailed reports on performance, attendance, and departmental statistics.

OBJECTIVES

1. **User Authentication**:
    - Verify user credentials to allow access to the system.
    - Ensure secure login and restrict unauthorized access.

2. **Employee Management**:
    - Enable adding new employee details.
    - Provide functionality to update existing employee records.
    - Allow deletion of employee records from the system.
    - Facilitate viewing of individual and all employee records.

3. **Data Integrity and Consistency**:
    - Maintain accurate and up-to-date records in the database.
    - Ensure data consistency across various related tables.

4. **User Interface**:
    - Design a user-friendly interface for easy navigation and interaction.

- Provide clear feedback messages to users during operations.

MODULES

1. **Login Module**:
    - **Objective**: To authenticate users and provide secure access to the system.
        - **Functions**:
            - check(): Verifies user credentials using the db.verify_credentials method and provides feedback on login success or failure.

2. **Dashboard Module**:
    - **Objective**: To provide a centralized interface for managing employee records after successful login.
    - **Functions**:
        - open_dashboard(): Opens the dashboard window with options to add, update, delete, and view employee records.
        - **UI Elements**:

          Labels and Entry fields for employee details.

          Buttons for performing add, update, delete, and view operations.

          Dropdown menu for selecting fields to update.

3. **Add Employee Module**:
    - **Objective**: To add new employee records to the system.
    - **Functions**:
        - add(): Collects employee details from the UI and inserts them into the database using the db.insert_row method.

4. **Update Employee Module**:
    - **Objective**: To update existing employee records in the system.
    - **Functions**:

        - update(): Updates a specific field of an employee's record in the database using the db.update_row method.

5. **Delete Employee Module**:
    - **Objective**: To delete employee records from the system.
    - **Functions**:
        - delete(): Deletes an employee's record from the database using the db.delete_row method.

6. **View Employee Module**:
    - **Objective**: To view individual or all employee records.
    - **Functions**:
        - view(): Fetches and displays a specific employee's record using the db.read_row method.
        - view_all(): Fetches and displays all employee records using the db.get_all_rows method.

Database Interaction Module

This module handles all database-related operations, ensuring data integrity and consistency.

1. **Database Connection**:
    - **Objective**: To establish a connection with the MySQL database.
    - **Functions**:
        - con: Connection object for the database.
        - cur: Cursor object for executing queries.
2. **CRUD Operations**:
    - **Objective**: To perform Create, Read, Update, and Delete operations on the database.
    - **Functions**:
        - verify_credentials(username, password): Verifies user credentials for login.
        - insert_row(empid, name, department, position, location, hire_date, salary): Inserts a new employee record into the database.

- read_row(emp_id): Reads an employee's record from the database.
- update_row(empid, column, value): Updates a specific field of an employee's record.
- delete_row(emp_id): Deletes an employee's record from the database.
- get_all_rows(): Retrieves all employee records from the database.

User Interface Elements

1. **Labels and Entries**:
    - For input fields such as EMPID, NAME, DEPARTMENT, etc.
2. **Buttons**:
    - For actions such as ADD NEW EMPLOYEE, UPDATE FIELD, DELETE EMPLOYEE, VIEW EMPLOYEE, VIEW ALL EMPLOYEES, LOGIN, and NEXT.
3. **Message Labels**:
    - To provide feedback to the user on actions performed.
4. **Frames**:
    - For organizing UI elements, especially in the view section to display records.

By organizing the code into these modules and detailing their objectives and functions, we can ensure a structured approach to developing and maintaining the Employee Management System.

CHAPTER 2: SURVEY OF TECHNOLOGIES

2.1 SOFTWARE DESCRIPTION

The Employee Management System is built using Python for application logic and MySQL Database for data storage and retrieval.

2.2 Languages

2.2.1 Python

Description: Python is a high-level, interpreted programming language known for its simplicity and readability. It has a large standard library and supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is widely used in web development, data analysis, artificial intelligence, scientific computing, and more.

Features and Benefits:

- **Ease of Use and Readability:** Python's syntax is clear and concise, making it easy to learn and use. This readability reduces the cost of program maintenance and enhances collaboration among developers.
- **Extensive Libraries and Frameworks:** Python has a rich set of libraries and frameworks (like Tkinter for GUI applications, Pandas for data manipulation, and Flask/Django for web development) that can be leveraged to build robust applications quickly.
- **Cross-Platform Compatibility:** Python is available on multiple operating systems, ensuring that the code can run on different platforms without modification.
- **Community Support:** Python has a large and active community, providing extensive documentation, tutorials, and support forums that are invaluable for development and troubleshooting.

Why Python was Chosen: Python was chosen for this project due to its simplicity, which accelerates development time and reduces the likelihood of

bugs. Its extensive libraries provide ready-to-use tools for various functionalities, minimizing the need to write code from scratch. Python's strong support for GUI development through Tkinter made it an ideal choice for creating the user interface of the Employee Management System.

2.2.2 MySQL Database

Description: MySQL is an open-source relational database management system (RDBMS). It uses Structured Query Language (SQL) for accessing and managing the data stored in databases. MySQL is known for its reliability, performance, and ease of use.

Features and Benefits:

- **Performance and Scalability:** MySQL is optimized for high performance and can handle large databases and high volumes of transactions. It is scalable to accommodate growing data and user load.
- **Security:** MySQL provides robust security features, including user authentication, access privileges, and SSL support, ensuring data protection.
- **Flexibility and Extensibility:** MySQL supports a wide range of storage engines, which can be chosen based on specific application requirements. It also integrates easily with other technologies and platforms.
- **Community and Enterprise Editions:** MySQL offers both community and enterprise editions, catering to different needs and budgets. The community edition is free and open-source, making it accessible for development and deployment.
- **Wide Adoption and Support:** MySQL is widely adopted across various industries, ensuring extensive community and professional support. This widespread use guarantees a wealth of resources, documentation, and expertise.

Why MySQL was Chosen: MySQL was chosen for this project due to its robust performance and reliability, which are crucial for managing employee data. Its strong security features help protect sensitive employee information. The relational model of MySQL is well-suited for structuring the data in a way that reflects the real-world relationships between employees, departments, positions, and salaries. Additionally, MySQL's widespread adoption means there is ample support and resources available for development and troubleshooting.

CHAPTER 3: REQUIREMENTS AND ANALYSIS

3.1 Requirement Specification

This section outlines the functional and non-functional requirements of the Employee Management System, detailing the necessary features and performance criteria.

3.2 Hardware and Software Requirements

Hardware:

- A modern computer with at least 4GB of RAM and sufficient disk space for storing the database.

Software:

- **Operating System:** Windows 11
- **Programming Language:** Python (version 3.12)
- **Database Management System:** MySQL (version 8.0)
- **Python Libraries:** Tkinter for GUI, mysql-connector-python for database connectivity

3.3 Architecture

The Employee Management System is designed with a multi-tier architecture, leveraging a combination of the client-server model and a database backend. The architecture can be divided into three main layers: Presentation Layer, Business Logic Layer, and Data Layer. Below is a detailed description of each layer and its components.

1. Presentation Layer

Components:

- **Tkinter-based GUI**: Provides the graphical user interface for user interactions.
    - **Login Page**: The initial interface where users enter their credentials.
    - **Dashboard**: The main interface for managing employee records, accessible after successful login.
    - **Labels, Entries, Buttons**: Various UI elements for input and interaction.

Responsibilities:

- Captures user input for login, adding, updating, deleting, and viewing employee records.
- Displays feedback messages to users.
- Navigates between different views (e.g., from login to dashboard).

2. Business Logic Layer

Components:

- **Application Logic**: Contains the core logic for handling user requests and interfacing with the database.
    - **Login Validation**: Verifies user credentials.
    - **CRUD Operations**: Handles creation, reading, updating, and deletion of employee records.

Responsibilities:

- Validates user credentials by communicating with the Data Layer.
- Processes user actions from the GUI (add, update, delete, view).
- Ensures data integrity and consistency during CRUD operations.
- Provides appropriate feedback to the Presentation Layer based on operation outcomes.

3. Data Layer

Components:

- **MySQL Database**: Stores employee data and credentials.
    - **Tables**:
        - empdetails: Stores detailed information about employees.
        - empsalaryperm: Stores salary and department information.
        - empcredentials: Stores user credentials.
        - empleave: Stores leave details.

Responsibilities:

- Executes SQL queries to perform CRUD operations.
- Maintains persistent storage of employee data.
- Ensures secure storage and retrieval of user credentials.
- Provides data access methods for the Business Logic Layer.

Data Flow

1. **User Login**:
    - The user enters credentials on the Login Page.
    - The check() function validates the credentials via db.verify_credentials(username, password).
    - If valid, the user is granted access to the Dashboard; otherwise, an error message is displayed.
2. **Employee Management**:
    - **Add Employee**:
        - User inputs employee details on the Dashboard.
        - The add() function collects details and calls db.insert_row() to add the record to the database.
    - **Update Employee**:
        - User specifies EMPID, field to update, and new value.
        - The update() function calls db.update_row() to update the specified field.
    - **Delete Employee**:
        - User specifies EMPID to delete.
        - The delete() function calls db.delete_row() to remove the record from the database.
    - **View Employee**:
        - User specifies EMPID to view.
        - The view() function calls db.read_row() to fetch and display the record.
    - **View All Employees**:
        - The view_all() function calls db.get_all_rows() to fetch and display all records.

Technology Stack

- **Python**: The programming language used to implement the application logic and GUI.
- **Tkinter**: The Python library used for creating the graphical user interface.

- **MySQL**: The relational database management system used for storing and managing employee data.
- **PIL (Python Imaging Library)**: Used for handling images (if required in the GUI).

Security Considerations

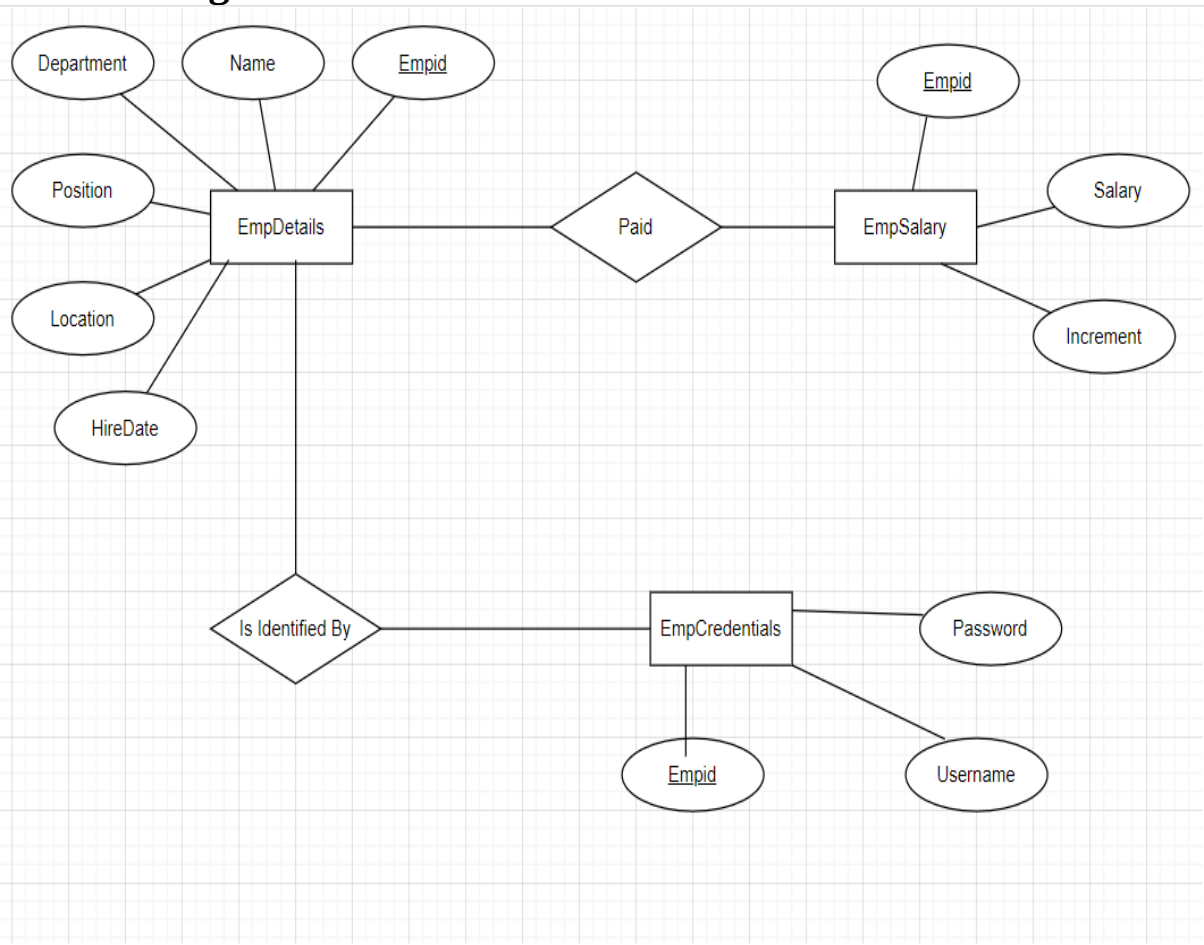
- **User Authentication**: Ensures that only authorized users can access the system.
- **Data Validation**: Ensures that input data is validated to prevent SQL injection and other forms of data corruption.
- **Error Handling**: Provides mechanisms to handle and log database errors, ensuring system stability.

Future Enhancements

- **Role-Based Access Control (RBAC)**: Implementing different access levels for users based on roles (e.g., admin, HR, employee).
- **Enhanced UI**: Improving the user interface for better user experience.
- **Additional Features**: Adding functionalities like employee performance tracking, detailed leave management, and reporting tools.

By dividing the system into these layers and components, the architecture ensures modularity, scalability, and maintainability, making it easier to develop, test, and enhance the Employee Management System.

3.5 E R Diagram



3.5 Data Modeling in SQL

Data modeling in MySQL involves designing a database schema that defines how data is stored, organized, and accessed. This process includes identifying entities, their attributes, and relationships, and then implementing this structure using MySQL. Here's a step-by-step guide to data modeling in MySQL:

1. Identify Entities and Attributes

Entities represent real-world objects or concepts. Each entity has attributes that describe its properties.

- **Employee**: Represents a company employee.

- Attributes: Employee ID, Name, Department, Position, Location, Hire Date, Salary

2. Define Relationships

Identify how entities are related to each other.

- **One-to-One (1:1)**: Rare in databases but occurs when each record in one table is linked to one record in another table.

- **One-to-Many (1)**: Common relationship where a record in one table is related to multiple records in another table.

- **Many-to-Many (M)**: Requires a junction table to manage the relationship between two tables.

3. Normalize the Data

Normalization ensures data is logically stored, reducing redundancy and improving integrity.

- **First Normal Form (1NF)**: Ensure each column contains atomic values, and each column value is unique.
- **Second Normal Form (2NF)**: Ensure the table is in 1NF and all non-key attributes are fully functionally dependent on the primary key.
- **Third Normal Form (3NF)**: Ensure the table is in 2NF and all attributes are only dependent on the primary key.

4. Create Tables

Use the CREATE TABLE statement to define tables, specifying columns, data types, and constraints.

CHAPTER 4: PROGRAM CODE

This chapter includes the complete source code for the Employee Management System.

1.dbmsConnector.py

```
import mysql.connector

# Establish database connection
con = mysql.connector.connect(user="root", password="root", database="vinoth")
cur = con.cursor()

def verify_credentials(username, password):
    try:
        query = "SELECT empid FROM empcredentials WHERE username = %s AND
password = %s"
        cur.execute(query, (username, password))
        result = cur.fetchone()
        return result is not None
    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return False

def insert_row(empid, name, department, position, location, hire_date,
salary):
    try:
        empdetails_sql = """
            INSERT INTO empdetails (empid, name, department, position,
location, hire_date)
            VALUES (%s, %s, %s, %s, %s, %s)
        """
        cur.execute(empdetails_sql, (int(empid), name, department, position,
location, hire_date))

        empsalaryperm_sql = """
            INSERT INTO empsalaryperm (empid, department, salary, increment)
            VALUES (%s, %s, %s, %s)
        """
        cur.execute(empsalaryperm_sql, (int(empid), department, float(salary),
0.05))

        empcredentials_sql = """
            INSERT INTO empcredentials (empid, username, password)
```

```
            VALUES (%s, %s, CONCAT(SUBSTRING_INDEX(%s, ' ', 1), %s))
        """
        cur.execute(empcredentials_sql, (int(empid), name, name, str(empid)))

        empleave_sql = """
            INSERT INTO empleave (empid, pto)
            VALUES (%s, %s)
        """
        cur.execute(empleave_sql, (int(empid), 15))

        con.commit()

    except mysql.connector.Error as err:
        print(f"Error: {err}")
        con.rollback()

def read_row(emp_id):
    try:
        query = """
            SELECT ed.empid, ed.name, ed.department, ed.position, ed.location,
ed.hire_date,
                   es.salary, el.pto, ec.username
            FROM empdetails ed
            JOIN empsalaryperm es ON ed.empid = es.empid
            JOIN empleave el ON ed.empid = el.empid
            JOIN empcredentials ec ON ed.empid = ec.empid
            WHERE ed.empid = %s
        """
        cur.execute(query, (int(emp_id),))
        result = cur.fetchone()
        return result

    except mysql.connector.Error as err:
        print(f"Error: {err}")
        con.rollback()
        return None
def update_row(empid, column, value):
    try:
        if column in ['name', 'department', 'position', 'location',
'hire_date']:
            empdetails_sql = f"UPDATE empdetails SET {column} = %s WHERE empid
= %s"
            cur.execute(empdetails_sql, (value, int(empid)))
        elif column in ['salary', 'increment']:
            empsalary_sql = f"UPDATE empsalaryperm SET {column} = %s WHERE
empid = %s"
            cur.execute(empsalary_sql, (float(value), int(empid)))
```

```
        con.commit()

    except mysql.connector.Error as err:
        print(f"Error: {err}")
        con.rollback()


def delete_row(emp_id):
    try:
        delete_queries = [
            "DELETE FROM empsalaryperm WHERE empid = %s",
            "DELETE FROM empcredentials WHERE empid = %s",
            "DELETE FROM empleave WHERE empid = %s",  # Added comma here
            "DELETE FROM empdetails WHERE empid = %s"  # Moved the corrected
line up
        ]

        for query in delete_queries:
            cur.execute(query, (int(emp_id),))

        con.commit()

    except mysql.connector.Error as err:
        print(f"Error: {err}")
        con.rollback()


def get_all_rows():
    try:
        query = """
            SELECT ed.empid, ed.name, ed.department, ed.position, ed.location,
ed.hire_date,
                   es.salary, el.pto, ec.username
            FROM empdetails ed
            JOIN empsalaryperm es ON ed.empid = es.empid
            JOIN empleave el ON ed.empid = el.empid
            JOIN empcredentials ec ON ed.empid = ec.empid
        """
        cur.execute(query)
        results = cur.fetchall()
        return results

    except mysql.connector.Error as err:
        print(f"Error: {err}")
        con.rollback()
        return None
```

This script establishes a connection to a MySQL database and provides functions for managing employee records, including verifying credentials, inserting, reading, updating, deleting, and retrieving records.

Components and Functionality

1. **Database Connection**
    - Establishes a connection to a MySQL database using mysql.connector.
2. **Verify Credentials**
    - verify_credentials(username, password): Checks if the provided username and password match a record in the empcredentials table. Returns True if a match is found, False otherwise.
3. **Insert Row**
    - insert_row(empid, name, department, position, location, hire_date, salary): Inserts a new employee's details into multiple tables (empdetails, empsalaryperm, empcredentials, empleave). It handles the employee's personal details, salary, credentials, and leave information, and commits the transaction.
4. **Read Row**
    - read_row(emp_id): Fetches and returns details of an employee based on the provided employee ID by joining multiple tables (empdetails, empsalaryperm, empleave, empcredentials).
5. **Update Row**
    - update_row(empid, column, value): Updates a specific field of an employee's record. Depending on the column, it updates either the empdetails or empsalaryperm table and commits the changes.
6. **Delete Row**
    - delete_row(emp_id): Deletes an employee's record from all related tables (empsalaryperm, empcredentials, empleave, empdetails). It executes multiple delete queries and commits the transaction.
7. **Get All Rows**
    - get_all_rows(): Retrieves and returns details of all employees by joining multiple tables (empdetails, empsalaryperm, empleave, empcredentials).

Error Handling

- Each function includes error handling using try-except blocks to catch and print database errors. If an error occurs, the transaction is rolled back to maintain database integrity.

Usage

- These functions can be integrated into a larger application, such as a GUI for managing employee records, providing backend support for database operations.

2.FrontEnd.py

```
import dbmsConnector as db
from tkinter import *
from tkinter import messagebox
from PIL import Image, ImageTk

flag = 0
entries = {}

root = Tk()
root.title("Login Page")
root.geometry("400x300")
root.configure(bg="#66C6BA")

def check():
    global flag
    username = username_Entry.get()
    password = password_Entry.get()
    if db.verify_credentials(username, password):
        message_Label.config(text="LOGIN SUCCESSFUL", font=("Arial", 15),
fg="#3957EB", bg="#66C6BA")
        flag = 1
    else:
        message_Label.config(text="LOGIN FAILED", font=("Arial", 15),
fg="red", bg="#66C6BA")

def create_label_and_entry(parent, text, row, column):
    label = Label(parent, text=text, fg="#043087", bg="#66C6BA")
    label.grid(row=row, column=column, padx=10, pady=10)
    entry = Entry(parent, borderwidth=10, bg="#66C6BA", fg="#043087")
    entry.grid(row=row, column=column + 1)
    entries[text] = entry

def add():
    emp_details = {key: entry.get() for key, entry in entries.items()}
    db.insert_row(emp_details['EMPID'], emp_details['NAME'],
emp_details['DEPARTMENT'], emp_details['POSITION'], emp_details['LOCATION'],
emp_details['HIRE DATE'], emp_details['SALARY'])
```

```
def update():
    empid = updateempid_entry.get()
    field = option.get()
    value = update_entry.get()
    db.update_row(empid, field, value)

def delete():
    empid = deleteempid_entry.get()
    db.delete_row(empid)

def view():
    empid = viewemp_entry.get()
    if not empid:
        messagebox.showerror("Error", "Please enter a valid EMPID")
        return
    for widget in view_frame.winfo_children():
        widget.destroy()
    record = db.read_row(empid)
    if record:
        record_str = f"EMPID: {record[0]}, NAME: {record[1]}, DEPARTMENT:
{record[2]}, POSITION: {record[3]}, LOCATION: {record[4]}, HIRE DATE:
{record[5]}, SALARY: {record[6]}"
        print_record = Label(view_frame, text=record_str, bg="#66C6BA",
fg="#043087")
        print_record.pack(anchor="w", padx=10, pady=5)
    else:
        print_record = Label(view_frame, text="No record found", bg="#66C6BA",
fg="#043087")
        print_record.pack(anchor="w", padx=10, pady=5)

def view_all():
    for widget in view_frame.winfo_children():
        widget.destroy()
    records = db.get_all_rows()
    print_record = ""
    for record in records:
        print_record += f"EMPID: {record[0]}, NAME: {record[1]}, DEPARTMENT:
{record[2]}, POSITION: {record[3]}, LOCATION: {record[4]}, HIRE DATE:
{record[5]}, SALARY: {record[6]}\n"
    print_label = Label(view_frame, text=print_record, bg="#66C6BA",
fg="#043087", justify=LEFT)
    print_label.pack(anchor="w", padx=10, pady=5)

def open_dashboard():
    if flag == 1:
        root.destroy()
        global dashboard
        dashboard = Tk()
```

```
        dashboard.title("Dashboard")
        dashboard.geometry("800x600")
        dashboard.configure(bg="#66C6BA")

        # Add section
        create_label_and_entry(dashboard, "EMPID", 1, 0)
        create_label_and_entry(dashboard, "NAME", 1, 2)
        create_label_and_entry(dashboard, "DEPARTMENT", 2, 0)
        create_label_and_entry(dashboard, "POSITION", 2, 2)
        create_label_and_entry(dashboard, "LOCATION", 3, 0)
        create_label_and_entry(dashboard, "HIRE DATE", 3, 2)
        create_label_and_entry(dashboard, "SALARY", 4, 0)

        add_Button = Button(dashboard, text="ADD NEW EMPLOYEE", borderwidth=5,
bg="#66C6BA", fg="#043087", command=add)
        add_Button.grid(row=5, column=0)

        # Update section
        updateempid_label = Label(dashboard, text="ENTER EMPID",
borderwidth=10, bg="#66C6BA", fg="#043087")
        updateempid_label.grid(row=6, column=0)
        global updateempid_entry
        updateempid_entry = Entry(dashboard, borderwidth=10, bg="#66C6BA",
fg="#043087")
        updateempid_entry.grid(row=6, column=1)
        updatecolumn_label = Label(dashboard, text="UPDATE FIELD",
bg="#66C6BA", fg="#043087")
        updatecolumn_label.grid(row=6, column=2)
        global update_entry
        update_entry = Entry(dashboard, borderwidth=10, bg="#66C6BA",
fg="#043087")
        update_entry.grid(row=6, column=4)
        global option
        option = StringVar(dashboard)
        option.set("SELECT COLUMN")
        columnlist = OptionMenu(dashboard, option, "name", "department",
"position", "location", "hire_date", "salary", "increment")
        columnlist.grid(row=6, column=3)
        update_button = Button(dashboard, text="UPDATE FIELD", borderwidth=5,
bg="#66C6BA", fg="#043087", command=update)
        update_button.grid(row=7, column=0, padx=10, pady=10)

        # Delete section
        deleteempid_label = Label(dashboard, text="ENTER EMPID",
borderwidth=10, bg="#66C6BA", fg="#043087")
        deleteempid_label.grid(row=8, column=0)
        global deleteempid_entry
```

```
        deleteempid_entry = Entry(dashboard, borderwidth=10, bg="#66C6BA",
fg="#043087")
        deleteempid_entry.grid(row=8, column=1)
        delete_button = Button(dashboard, text="DELETE EMPLOYEE",
borderwidth=5, bg="#66C6BA", fg="#043087", command=delete)
        delete_button.grid(row=9, column=0, padx=10, pady=10)

        # View section
        viewemp_label = Label(dashboard, text="ENTER EMPID", bg="#66C6BA",
fg="#043087")
        viewemp_label.grid(row=10, column=0)
        global viewemp_entry
        viewemp_entry = Entry(dashboard, borderwidth=10, bg="#66C6BA",
fg="#043087")
        viewemp_entry.grid(row=10, column=1)
        view_button = Button(dashboard, text="VIEW EMPLOYEE", borderwidth=5,
bg="#66C6BA", fg="#043087", command=view)
        view_button.grid(row=11, column=0, padx=10, pady=10)
        viewall_button = Button(dashboard, text="VIEW ALL EMPLOYEES",
borderwidth=5, bg="#66C6BA", fg="#043087", command=view_all)
        viewall_button.grid(row=13, column=0, padx=10, pady=10)

        global view_frame
        view_frame = Frame(dashboard, bg="#66C6BA")
        view_frame.grid(row=15, column=0, columnspan=4, sticky='w')

login_label = Label(root, text="LOGIN PAGE", font=("Arial", 25), bg="#66C6BA",
fg="#043087")
login_label.grid(row=0, column=2)

username_Label = Label(root, text="USERNAME", borderwidth=2, bg="#66C6BA",
fg="#043087")
username_Label.grid(row=1, column=1, pady=10)

password_Label = Label(root, text="PASSWORD", borderwidth=2, bg="#66C6BA",
fg="#043087")
password_Label.grid(row=2, column=1)

username_Entry = Entry(root, borderwidth=10, bg="#66C6BA", fg="#043087")
username_Entry.grid(row=1, column=2)

password_Entry = Entry(root, borderwidth=10, show="*", bg="#66C6BA",
fg="#043087")
password_Entry.grid(row=2, column=2)

login_Button = Button(root, text="LOGIN", borderwidth=5, bg="#66C6BA",
fg="#043087", command=check)
login_Button.grid(row=3, column=2, pady=10)
```

```
next_Button = Button(root, text="NEXT", borderwidth=5, command=open_dashboard,
bg="#66C6BA", fg="#043087")
next_Button.grid(row=3, column=3, pady=10)

message_Label = Label(root, text="", borderwidth=5, bg="#66C6BA",
fg="#043087")
message_Label.grid(row=4, column=2)

root.mainloop()
```

This script is a Tkinter-based GUI application for managing employee records, with a login interface and a dashboard for CRUD (Create, Read, Update, Delete) operations. Here's a breakdown of its components and functionality:

Login Page

- **Login Interface**: The main window prompts the user to enter a username and password.
- **Login Verification**: The check() function verifies the credentials using a hypothetical db.verify_credentials() function. If successful, a success message is displayed and a flag is set.

Dashboard

- **Opening the Dashboard**: If the login is successful (flag == 1), the main window is destroyed, and a new window (dashboard) is created.
- **Add Section**: Contains labels and entry fields to input new employee details and a button to add the employee to the database using db.insert_row().
- **Update Section**: Allows updating an existing employee's details. It includes entry fields to specify the employee ID, the field to update, and the new value, and a button to execute the update using db.update_row().
- **Delete Section**: Provides an entry field to input the employee ID and a button to delete the employee from the database using db.delete_row().
- **View Section**:
    - **View Specific Employee**: Entry field for employee ID and a button to view the details of that employee using db.read_row().
    - **View All Employees**: A button to display all employee records from the database using db.get_all_rows(). The records are displayed in a frame within the dashboard.

Helper Functions

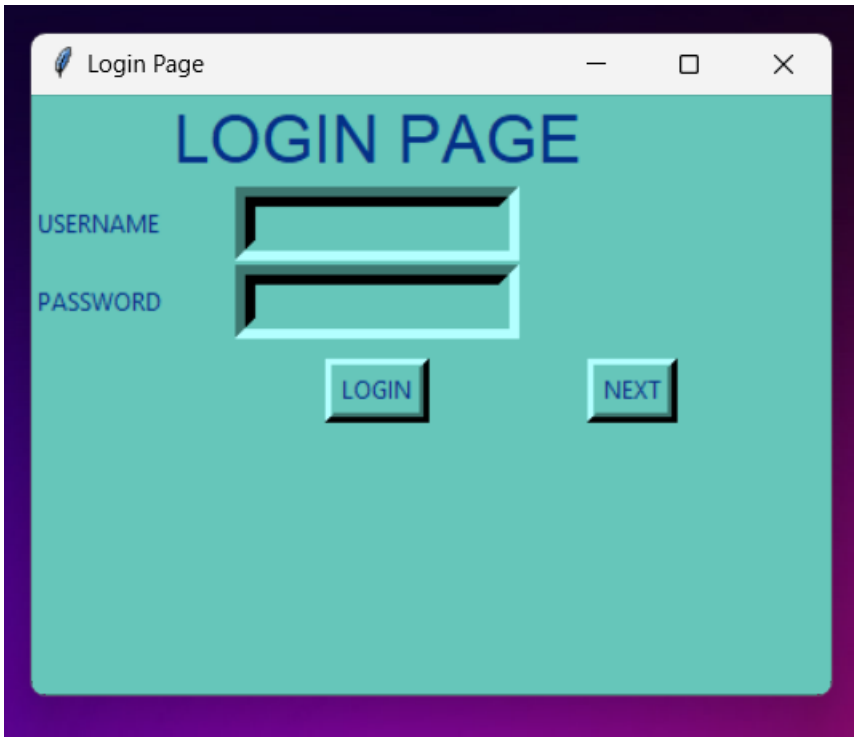
- **create_label_and_entry()**: Creates labels and entry fields dynamically for the dashboard sections.
- **Database Operations**: Functions add(), update(), delete(), view(), and view_all() interact with the database through the dbmsConnector module.

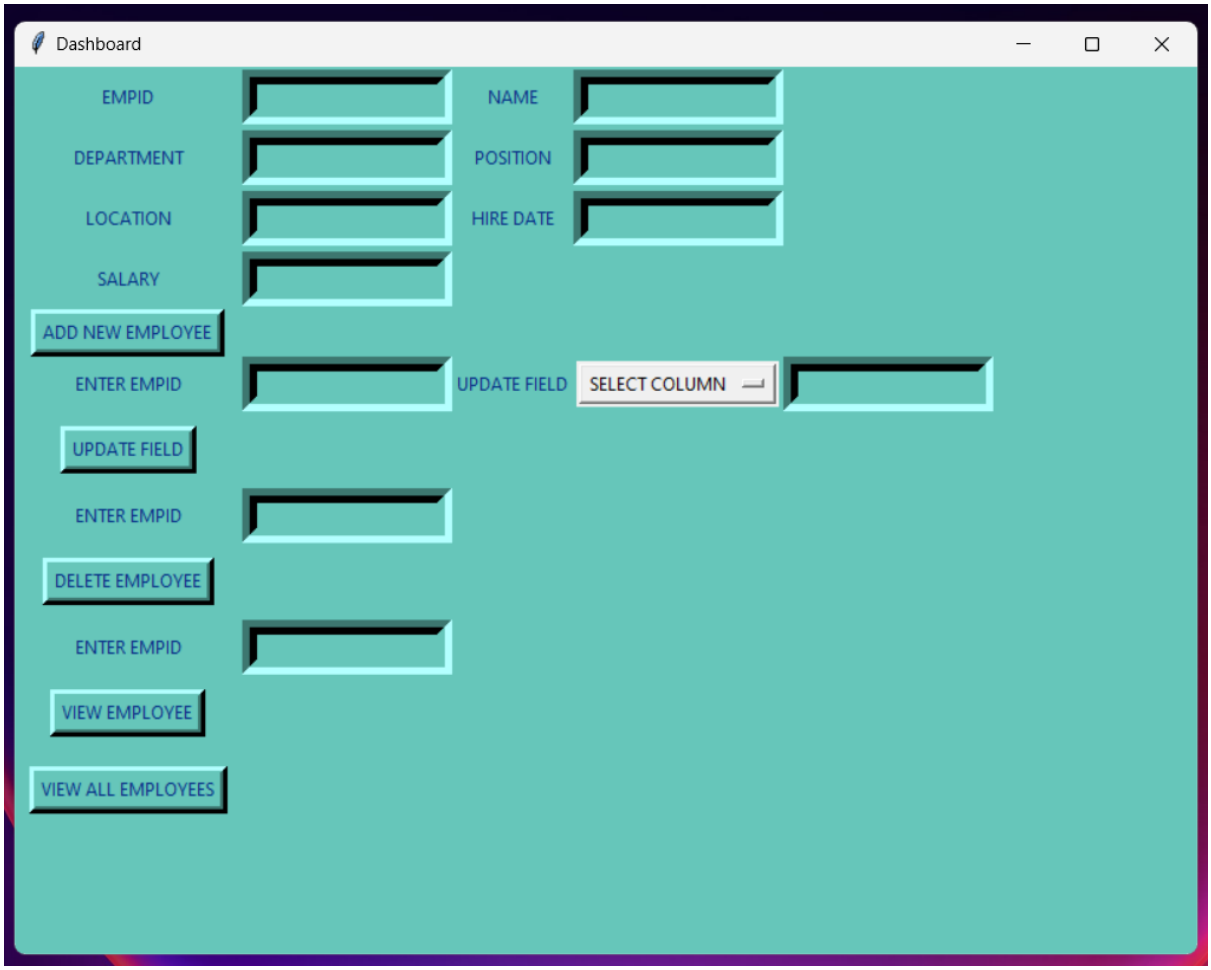
GUI Styling

- Consistent color scheme (#66C6BA for background and #043087 for text).
- Various widgets including Label, Entry, Button, OptionMenu, and Frame for layout and interaction.

This application facilitates easy management of employee records through a user-friendly graphical interface, leveraging a modular approach to handle various CRUD operations.

Output




Dashboard
EMPID
NAME
DEPARTMENT
POSITION
LOCATION
HIRE DATE
SALARY
ADD NEW EMPLOYEE
ENTER EMPID
UPDATE FIELD
SELECT COLUMN
UPDATE FIELD
ENTER EMPID
DELETE EMPLOYEE
ENTER EMPID
VIEW EMPLOYEE
VIEW ALL EMPLOYEES

CHAPTER 5: RESULTS AND DISCUSSION

Results

The Employee Management System developed using Python and MySQL successfully demonstrates various functionalities for managing employee records. The key results of the project are:

- **User Authentication:** Implemented a secure login system, verifying credentials using the MySQL database to ensure only authorized users can access the system.
- **Employee Records Management:** Enabled adding, updating, deleting, and viewing employee records. New employee records include personal information, department, position, location, hire date, and salary. Specific fields of an employee's record can be updated, and employee records can be deleted along with all related records in different tables. The system also allows viewing individual employee details or a list of all employees.
- **Database Operations:** Ensured robust database operations with transaction management to maintain data integrity and consistency, leveraging MySQL for efficient data handling.
- **User Interface:** Developed a user-friendly interface using Tkinter for the login page and dashboard, with intuitive input forms for managing employee records.

CHAPTER 6: CONCLUSION

Conclusion

The Employee Management System project effectively demonstrates the use of Python and MySQL to create a functional and efficient system for managing employee data. The project highlights several key aspects:

- **Functionality:** Successfully implemented core functionalities for managing employee records.
- **Database Management:** Utilized MySQL for complex queries and data relationships, ensuring data integrity and consistency.
- **User Interface:** Created a user-friendly interface for simplified management of employee records.
- **Security:** Implemented a secure login system to protect sensitive employee data.

Overall, the project achieves its objectives and provides a solid foundation for further enhancements, such as adding advanced features, generating reports and analytics, enhancing the user interface, and integrating with other enterprise applications. The Employee Management System is a valuable tool for organizations to efficiently manage employee data, streamline HR processes, and maintain accurate records.

CHAPTER 7: REFERENCES

1. https://www.w3schools.com/python/python_mysql_getstarted.asp

2. https://www.geeksforgeeks.org/crud-operation-in-python-using-mysql/w

3. https://dev.mysql.com/doc/mysql-getting-started/en/

4.https://www.youtube.com/watch?v=yQSEXcf6s2I&list=PLCC34OHNcOtoC6GglhF3ncJ5rLwQrLGnV