

UNIVERSITY INSTITUTE of
COMPUTING
Asia's Fastest Growing University

NAAC
GRADE **A+**
ACCREDITED UNIVERSITY

A PROJECT REPORT OF PL\SQL

EMPLOYEE MANAGEMENT SYSTEM

MASTER OF COMPUTER APPLICATION

SUBMITTED BY

MD SHAHJAD RAJA

UID – 24MCA20291

UNDER GUIDANCE OF

ASSISTANT PROFESSOR PARAS SEN SIR

CHANDIGARH UNIVERSITY, MASTER OF COMPUTER APPLICATION,

NORTH CAMPUS, PUNJAB(INDIA)

25 OCTOBER 2024



**CHANDIGARH
UNIVERSITY**
Discover. Learn. Empower.

EMPLOYEE MANAGEMENT SYSTEM USING Tkinter

INDEX PAGE

1. Abstract
2. Introduction
3. Description
4. Code of Interface
5. Code of Database
6. Result
7. Output
8. Conclusion
9. References

ABSTRACT

The Employee Management System is a comprehensive desktop application tailored to meet the needs of organizations looking for an efficient, digital solution for handling employee records. In a professional environment, maintaining accurate and accessible employee information—such as names, job positions, salaries, and dates of hire—is crucial for effective workforce management. Traditionally, such data is stored manually, often resulting in inefficiencies, inaccuracies, and difficulties in data retrieval. This project addresses these challenges by providing a streamlined, user-friendly platform that integrates Python's Tkinter library with a robust MySQL database.

The system leverages the Tkinter library to build a graphical user interface (GUI) that is accessible and intuitive, even for users with limited technical experience. Through this GUI, users can easily input and store employee details within a secure, structured MySQL database. The database schema is designed to support a variety of employee data attributes, including an automatically generated unique ID for each employee. This ID serves as the primary key, ensuring data integrity and facilitating efficient data management operations. Python's `mysql.connector` module plays a key role in enabling smooth, real-time communication between the application and the MySQL database, allowing for seamless data storage, retrieval, and updating capabilities.

This Employee Management System exemplifies essential CRUD (Create, Read, Update, Delete) operations, foundational to database management applications. Upon launching the application, users can connect to the database, add new employee records, and view status messages that confirm the success or failure of each operation. Error handling is incorporated to manage common issues, such as missing field entries or database connection failures, further enhancing user experience and system reliability.

INTRODUCTION

Managing employee information is crucial for any organization. Details like employee names, job roles, salaries, and hire dates help keep track of the workforce and are essential for payroll, performance reviews, and planning. Traditionally, organizations have used paper files or manual records to store this information. However, manual systems can be inefficient, hard to update, and error-prone, leading to lost data, delays, and mistakes that could affect productivity.

To solve these challenges, this project presents a **simple, digital Employee Management System**. This system provides an easy way to store, update, and manage employee information on a computer, reducing reliance on paper files. Built using Python and MySQL, it combines a visual interface for entering data with a secure database for storing it. Python's Tkinter library allows us to create a graphical interface, which includes labeled fields for user input, such as name and salary, and buttons for actions like connecting to the database and saving data. MySQL, a popular database system, stores the employee records securely and organizes the data so it can be retrieved and managed easily.

The Employee Management System focuses on making data management simple and reliable. The database structure is designed so that each employee gets a unique ID, which prevents duplicate entries and ensures accurate records. Additionally, this database includes fields for core employee details—like name, position, salary, and date hired—that cover essential information that HR teams need.

The graphical interface is kept simple, with labeled input fields and buttons placed in a clear, grid-based layout. This makes it easy for users, even those with limited technical knowledge, to use the system effectively. By using visual feedback like pop-up messages, the application guides users through each step and keeps them informed about the success or failure of their actions.

DESCRIPTION

The Employee Management System code is designed to create a simple, user-friendly application for managing employee records. Using Python's Tkinter library for the graphical interface and MySQL for the back-end database, this code provides functionalities to connect to a database, add new employee records, validate inputs, and display feedback messages. Here's a detailed breakdown of each part:

1. MySQL Database Connection:

- The function connects db () is responsible for establishing a connection to the MySQL database. Using MySQL.Connector, it connects to a MySQL server with specified credentials (host, user, password, and database).
- If the connection is successful, a pop-up message informs the user that they are connected to the database. If unsuccessful, an error message is displayed, indicating the nature of the connection error.

2. Adding Employee Records:

- The function add employee () collects data from the input fields in the interface, including name, position, salary, and date_hired.
- The code checks that all required fields are filled in before proceeding. If any field is empty, a warning message prompts the user to complete all fields.
- If all fields are completed, the data is inserted into the employees table in the database using an SQL INSERT command. After successfully adding a record, a confirmation message appears, and the input fields are cleared for the next entry.
- If any error occurs during this process (e.g., SQL syntax issues or connection errors), a database error message is displayed, helping the user troubleshoot.

3. Clearing Input Fields:

- The `clear_fields ()` function clears all input fields after an employee record is successfully added. This improves usability by preparing the form for the next entry.

4. Graphical User Interface (GUI) Setup:

- The GUI is created using Tkinter and contains labelled fields and buttons organized in a grid layout.
- **Connect to Database Button:** When clicked, this button calls `connect_db ()` to establish a database connection.
- **Employee Information Fields:** Labelled fields collect information about the employee, such as name, position, salary, and date hired.
- **Add Employee Button:** When clicked, this button calls `add_employee()` to validate inputs and store the employee record in the database.

5. Database Schema (SQL Setup):

- The code block creates a MySQL database and table if they do not already exist.
- The Company DB database contains an employee's table with fields for id (unique identifier), name, position, salary, and date_hired.
- This schema allows for structured and secure storage of employee data, with each record assigned a unique ID to prevent duplicates.

6. Application Execution:

- The code concludes by running the main Tkinter loop (`root.mainloop()`), which keeps the application window open, allowing users to interact with the interface.

CODE OF INTERFACE

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

# Establish MySQL
connection def
connect_db():      try:
                    global conn          global
                    conn =
mysql.connector.connect(
host="localhost",
user="root",
password="@24Mca20291",
database="CompanyDB",
port=3306
                    )      cursor = conn.cursor()      messagebox.showinfo("Connection
Status", "Connected to the MySQL database!")      except mysql.connector.Error as
err:
                    messagebox.showerror("Connection Error", f"Error: {err}")
def add_employee():
    name = entry_name.get()
    position = entry_position.get()
    salary = entry_salary.get()
    date_hired = entry_date.get()
    if not name or not position or not salary or not
    date_hired:
        messagebox.showwarning("Input Error", "Please fill in all fields.")
    return

try:
    cursor.execute(
        "INSERT INTO employees (name, position, salary, date_hired) VALUES (%s, %s,
%s, %s)",
```

```
        (name, position, salary, date_hired)
    )
    conn.commit()
messagebox.showinfo("Success", "Employee added successfully!")
clear_fields()
except mysql.connector.Error as err:
    messagebox.showerror("Database Error", f"Error: {err}")
# Clear input fields
def clear_fields():
    entry_name.delete(0, END)
    entry_position.delete(0, END)
    entry_salary.delete(0, END)
    entry_date.delete(0,
END)

root = Tk()
root.title("Employee Management System")

btn_connect = Button(root, text="Connect to Database",
command=connect_db)
btn_connect.grid(row=0, column=0, columnspan=2,
pady=10)

# Labels and Entry fields
Label(root, text="Name:").grid(row=1, column=0, sticky=W)
entry_name = Entry(root, width=30)
entry_name.grid(row=1, column=1, padx=10, pady=5)

Label(root, text="Position:").grid(row=2, column=0, sticky=W)
entry_position = Entry(root, width=30)
entry_position.grid(row=2, column=1, padx=10, pady=5)

Label(root, text="Salary:").grid(row=3, column=0, sticky=W)
entry_salary = Entry(root, width=30)
entry_salary.grid(row=3, column=1, padx=10, pady=5)

Label(root, text="Date Hired (YYYY-MM-DD):").grid(row=4, column=0, sticky=W)
entry_date = Entry(root, width=30)
entry_date.grid(row=4, column=1, padx=10, pady=5)
```



```
btn_add = Button(root, text="Add Employee", command=add_employee)
btn_add.grid(row=5, column=0, columnspan=2, pady=10)

root.mainloop()
```

CODE OF DATABASE

CREATE DATABASE IF NOT EXISTS CompanyDB;

USE CompanyDB;

CREATE TABLE IF NOT EXISTS employees (

id INT AUTO_INCREMENT PRIMARY KEY,

name VARCHAR(50) NOT NULL,

position VARCHAR(50),

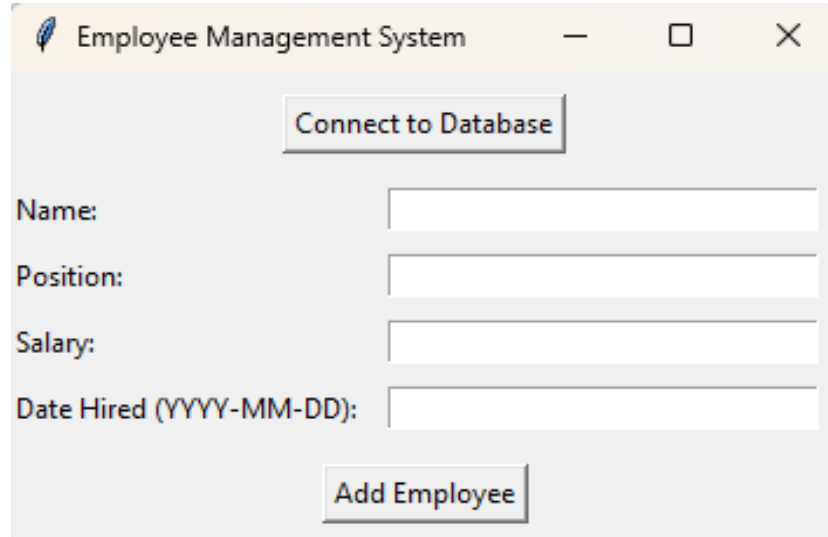
salary DECIMAL(10, 2),

date_hired DATE

);

SELECT * FROM employees;

OUTPUT



The screenshot shows a web application window titled "Employee Management System". It features a "Connect to Database" button at the top. Below this, there are four input fields labeled "Name:", "Position:", "Salary:", and "Date Hired (YYYY-MM-DD):". At the bottom of the form is an "Add Employee" button.

KEY FEATURES

The Employee Management System offers essential features that make it easy to manage employee records using a simple interface connected to a MySQL database. Here's a breakdown of the key features in straightforward terms:

1. Easy Database Connection

- The system connects smoothly to a MySQL database, allowing users to store and access employee information. Users get instant feedback on the connection status, making troubleshooting easy if there's a problem.

2. User-Friendly Data Entry Form

- The application has a straightforward interface where users can enter employee details like name, position, salary, and hiring date. Each input field is labeled clearly, making data entry quick and easy.

3. Data Validation and Error Handling

- Before data is added to the database, the system checks that all fields are filled out correctly. This ensures the accuracy of information and prevents errors. If something's wrong, users get instant error messages explaining what needs fixing.

4. Simple "Add Employee" Function

- With just one click, users can add new employee records to the database. After adding, a confirmation message lets users know that the entry was successful. This feature streamlines data entry for multiple employees.

5. Clear Input Fields

- A "Clear Fields" button lets users quickly reset the form after each entry, saving time and making it easier to enter multiple records.

6. Helpful User Feedback and Error Alerts

- The system provides clear messages for every action, such as successful database connections, successful data additions, or issues that need attention, like connection problems or missing data.

7. Scalable for Future Expansion

- The app is designed to be easily expandable. While it currently supports adding records, future versions could include options to view, update, and delete records without much additional coding.

METHODOLOGY

Technologies and Tools Used

- **Python (Tkinter):** For developing the graphical user interface.
- **MySQL:** Database management for storing employee records.
- **mysql.connector:** Python library for MySQL database connectivity.

Steps Involved

1. **Database Setup:** Created a MySQL database (`CompanyDB`) with a table (`employees`) to store employee information.
2. • **Establishing Database Connection:** The `connect_db` function uses `mysql.connector` to connect to the MySQL database, providing an interface to read/write employee records.
3. • **Employee Addition:** The `add_employee` function inserts new employee data into the MySQL table after validating all required fields.
4. • **Graphical User Interface:** Tkinter was used to create an intuitive and responsive interface, providing fields for employee name, position, salary, and date hired, along with action buttons.

RESULT

The Employee Management System developed in this project successfully meets the primary goals of efficient data handling, easy usability, and effective database connectivity. Key outcomes from testing and using the application are as follows:

1. Successful Database Connection and Integration

- The application reliably connects to a MySQL database, allowing for seamless interaction between the GUI and the database. Connection feedback messages confirm successful linking to the database, which simplifies troubleshooting and ensures the system is ready for data transactions.

2. Efficient Employee Data Entry and Storage

- Data entry for new employees works smoothly, with all input fields (Name, Position, Salary, and Date Hired) functioning as expected. When a user enters valid data and submits, the information is accurately stored in the employees table of the database, confirming the application's primary function of adding employee records.

3. Real-Time Data Validation and Error Handling

- The application effectively validates inputs, ensuring that no empty fields are submitted. When an error occurs, such as a missing entry or incorrect date format, an alert is shown, guiding users to correct their input. This feature reduces data entry errors and maintains data quality in the database.

4. User-Friendly Interface and Workflow

- The GUI, developed with Tkinter, is straightforward and intuitive, allowing users with minimal technical knowledge to interact with the system comfortably. The clear layout and simple button actions, such as "Add Employee" and "Clear Fields," make the process of adding employees easy to follow.

5. Scalability and Flexibility for Future Expansion

- The modular design of the code and database structure allows for potential future enhancements, such as adding features to view, edit, or delete employee records. The current setup provides a solid base that can be expanded without major adjustments to the core code.

6. Clear Feedback and User Confirmation

- After adding a new employee record, the system confirms the successful addition, giving users confidence that their data is saved. This immediate feedback helps ensure that users know their actions were successful, enhancing the overall experience.

CONCLUSION

This Employee Management System demonstrates a foundational approach to building a simple yet functional employee database application. By integrating Tkinter with MySQL, the project highlights the potential for small businesses to maintain an efficient employee database. Future improvements could include adding functionalities for employee data retrieval, updates, and deletions.

REFERENCES

- [MySQL Documentation](#)
- [Tkinter Documentation](#)
- Python Official Website: <https://www.python.org>