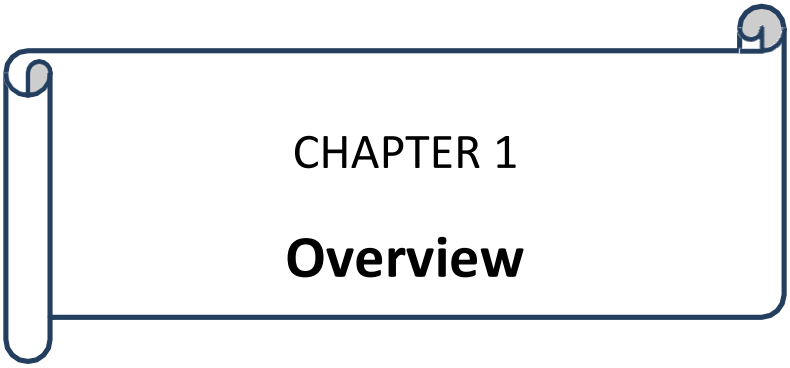
A

Project Report

On

Employee Management System

Contents



CHAPTER 1

Overview

Overview

What is management system?

A management system is the way in which an organization manages the interrelated parts of its business in order to achieve its objectives. These objectives can relate to a number of different topics, including product or service quality, operational efficiency, environmental performance, health and safety in the workplace and many more. Management system can be used to manage different aspect like Environmental, Organisational, and Government.

If we consider organisational aspect we can use management system to manage employee data, customer data, product data etc. For example in employee management system we can add, modify, and delete data related to employees in the organisation.

In this project we are going to build a web application accessed over LAN (Local Area Network) that can handle all information related employees, like employee ID, employee name, salary code, date of joining, birth date, Work location, reporting manager name, PF number etc. Project will use MySQL as DBMS and Python as programming language.

Why Employee Management System?

A management system is a key tool in helping to streamline your business processes and build-in efficiency. Implementation of the appropriate management system and certifying to the appropriate standard to your business improves business performance and embeds safe and sustainable practices into your operations.

It is very important to be able to manage all the data in order to work efficiently. Management system helps us to work in time not only in professional but also in personal life. Also allows us to learn, apply and showcase different technologies like programming languages, data management applications, web development skills etc.

How Employee Management System Composed?

This project is divided in two parts:

A. Backend
B. Frontend

Backend: Will have all stuff related to python programming, features, MySQL database.

Frontend: As end user might be anyone who doesn't know about technical things, frontend will deal with the user friendly web interface.

CHAPTER 2

Database Structure

Database Structure

EMS

To build employee management system, we need to build a database first. We are using MySQL DBMS to create a database called “ems”.

To create database we have used below query:

```
create database ems;
```

We have to use database in order to create table and make changes.

```
use ems;
```

The database “ems” contains six tables as below:

SQL query for creating respective tables is as below.

```
create table Employee_Basic_Details
(employee_id varchar(255),
name_prefix varchar(255),
first_name varchar(255),
last_name varchar(255),
date_of_birth varchar(255),
mobile_number varchar(255),
landline_number varchar(255),
city varchar(255),
qualification varchar(255),
year_of_experience INT,
date_of_joining varchar(255),
date_of_leaving varchar(255),
employee_type varchar(255),
gender varchar(255),
marital_status varchar(255),
primary key(employee_id));
```

Table 1: Parent Table

Employee Basic details
employee_id

name_prefix
first_name
last_name
date_of_birth(YYYY-MM-DD)
mobile_number
landline_number
city
qualification
years_of_experience
date_of_joining
date_of_leaving
type_of_employee
gender
marital_status

```
create table Working_History
(
employee_id varchar(255),
previous_company_name varchar(255),
previous_date_of_joining varchar(255),
previous_date_of_leaving varchar(255),
FOREIGN KEY (employee_id) REFERENCES
Employee_Basic_Details(employee_id));
```

Table 2: Child Table 1

Working History
employee_id
previous_employer_name
previous_company_name
previous_date_of_joining
previous_date_of_leaving

```
create table Time_Information_Per_Week
(
employee_id varchar(255),
Worked_hours INT,
Off_hours INT,
days_off INT,
```

```
overtime INT,
FOREIGN KEY (employee_id) REFERENCES
Employee_Basic_Details(employee_id));
```

Table 3: Child Table 2

Time InformationTime_Information_Per_Week
employee_id
Worked_hours
Off_hours
days_off
overtime

```
create table Salary_Information
(
employee_id varchar(255),
annual_salary varchar(255),
annual_tax varchar(255),
monthly_PF_deductions varchar(255),
medical_policy_premium_deduction varchar(255),
FOREIGN KEY (employee_id) REFERENCES
Employee_Basic_Details(employee_id));
```

Table 4: Child Table 3

Salary Information
employee_id
annual_salary
annual_tax
monthly_PF_deductions
medical_policy_premium_deduction

```
create table Contact_Person_Information
(
employee_id varchar(255),
c_first_name varchar(255),
c_last_name varchar(255),
relationship varchar(255),
```

```
c_mobile_number varchar(255),
c_landline_number varchar(255),
c_city varchar(255),
FOREIGN KEY (employee_id) REFERENCES
Employee_Basic_Details(employee_id));
```

Table 5: Child Table 4

Contact Person Information
employee_id
c_first_name
c_last_name
relationship
c_mobile_number
c_landline_number
c_city

```
create table Holiday_Information
(
employee_id varchar(255),
previlage_leave_balance INT,
sick_leave_balance INT,
emergency_leave_balance INT,
FOREIGN KEY (employee_id) REFERENCES
Employee_Basic_Details(employee_id));
```

Table 6: Child Table 5

Holiday Information
employee_id
previlage_leave_balance
sick_leave_balance
emergency_leave_balance

There will be one main table (parent table = employee_basic_details) and five child tables, related to each other. Each employee of the staff is intended to have several records, responding to his Working History, Contact Person Information, Salary Information, Time Information and Holiday Information, and only one record containing his basic information within the company – his personal details as: date of birth, gender, marital status, address and phone details, and his current working record.

The relationships between the data tables are shown in Figure 1

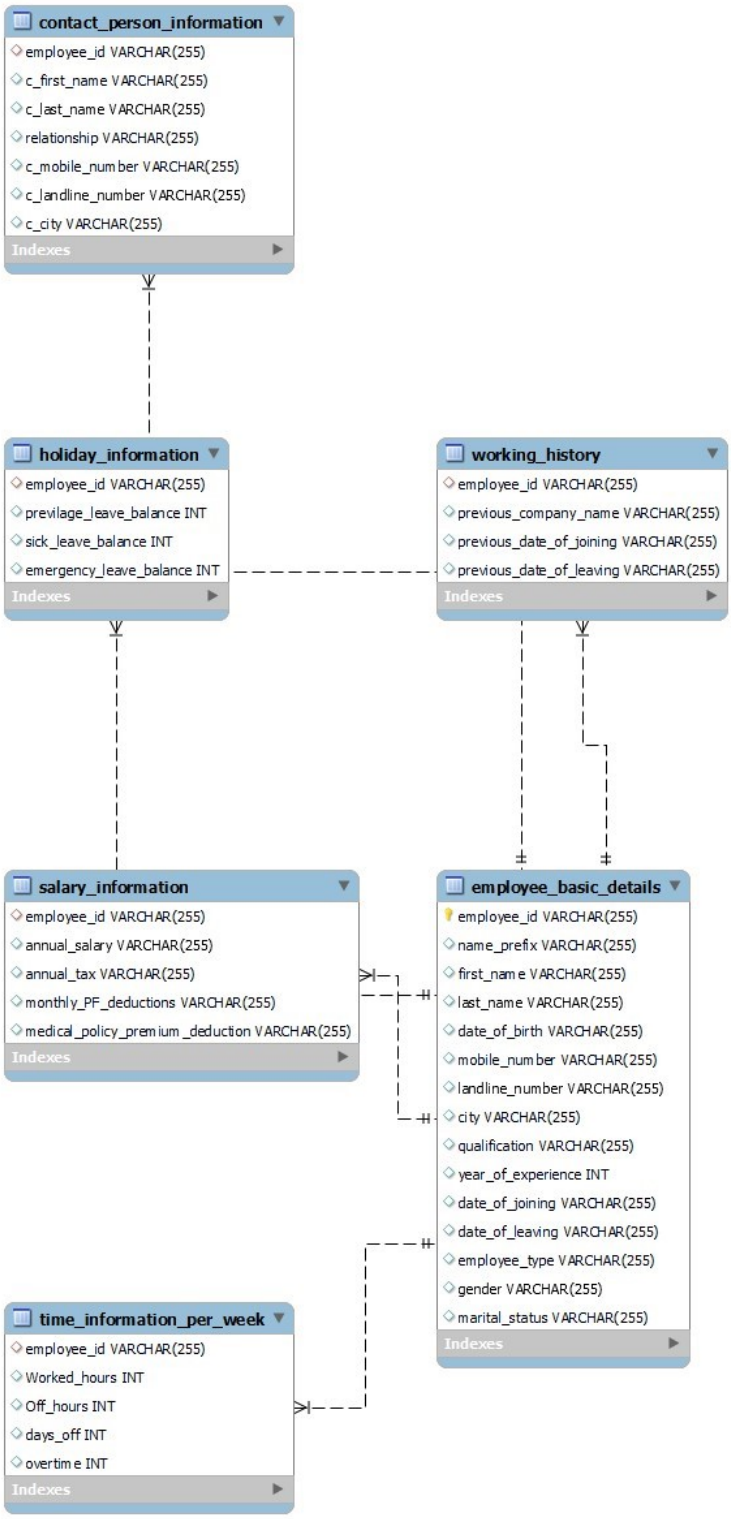


Fig. 1 EMS EER Diagram

Retire

We have created one more database named ‘retire’ which is a replica of ‘ems’ database, which will store information of retired employee or the employees who has left the company. Below is relation diagram of ‘retire’ database.

Purpose of this database is to keep track of employee IDs and make sure that no ID is repeated. So when an employee data gets deleted it will get copied to ‘retire’ database and then gets deleted from main ‘ems’ database. So when we add new employee it will take next id.

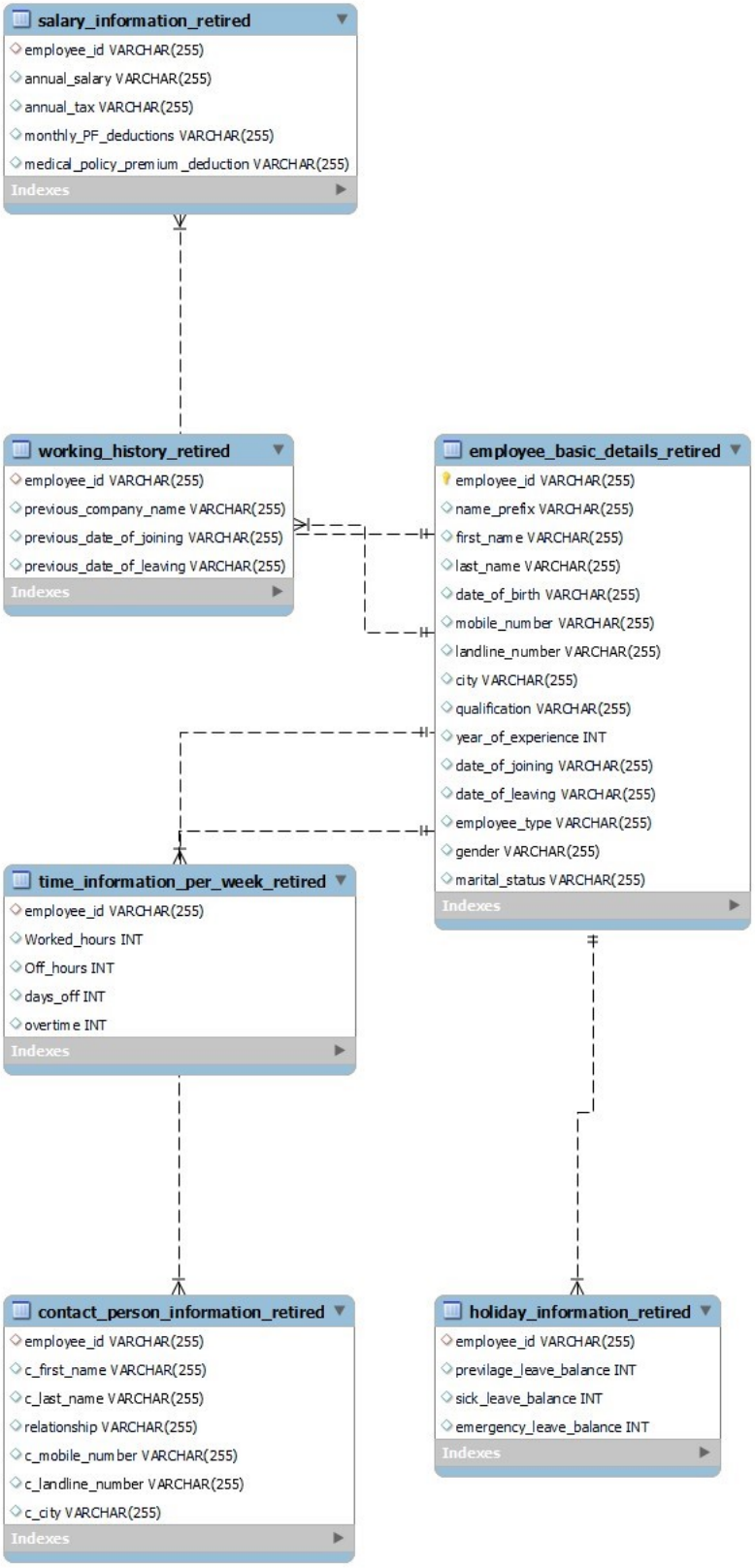


Fig. 2 Retire EER Diagram

CHAPTER 3

Implementation

Implementation

Once database is created we have build main program structure so that company admin can interact will employee management system. There are multiple features to the system which are explained as below.

Packages

Before starting to write our main program we have import all necessary packages those are required to build our web application.

```
import streamlit as st
from PIL import Image
import pickle
from pathlib import Path
import streamlit_authenticator as stauth
import mysql.connector
import re
import datetime
import time
from streamlit_option_menu import option_menu
import pandas as pd
```

Login

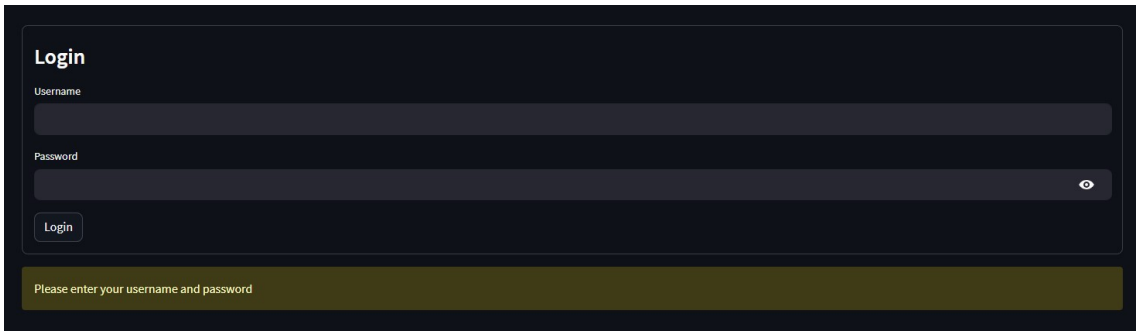
For login purpose we have used streamlit authentication feature in which we have written a code to set passwords for all allowed persons and generated a pickle file (.pkl) that contains all hashed password in binary form. That file is then passed when our main program checks for authorized login.

```python
import pickle
from pathlib import Path
import streamlit_authenticator as stauth

names = ["Sharayu Salunke", "Onlei Technologies"]

usernames = ["ssalunke", "otech"]

passwords = ["XXX", "XXX"]

hashed_passwords = stauth.Hasher(passwords).generate()

file_path = Path(__file__).parent / "hashed_pwd.pkl"
with file_path.open("wb") as file:
    pickle.dump(hashed_passwords, file)
```

In above snippet we can see only users with id “ssalunke” and “otech” are authorized to log into the system. Once the .pkl file is generated with required password it is good practice to change passwords list in the code with random passwords as above.

Now we have used .pkl file in main program to validate login as below.

```python
# Employess of an organisation can only
# able to login and connect to EMS database

names = ["Sharayu Salunke", "Onlei Technologies"]

usernames = ["ssalunke", "otech"]


with open("C:\Projects\EMS\Scripts\hashed_pwd.pkl", "rb") as file:
    hashed_passwords = pickle.load(file)


authenticator = stauth.Authenticate(names, usernames, hashed_passwords,
                                    "ems", "abcdef", cookie_expiry_days = 1)

name, authentication_status, username = authenticator.login("Login", "main")
st.sidebar.title(f"Welcome {name}")
if authentication_status == False:
    st.error("Username/Password is incorrect")

if authentication_status == None:
    st.warning("Please enter your username and password")

if authentication_status:

    #Only when employee logged into the system will connect to database
    #before that no one can access database.
```



Database and Web Application

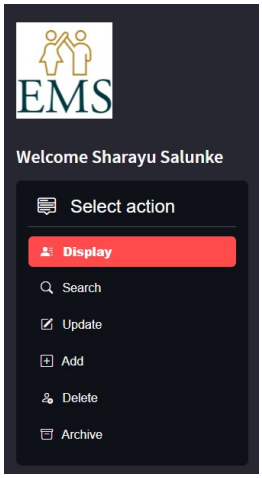
Now once we are successfully logged in to the system we will be able to connect to our database automatically and start using them.

```
if authentication_status:

    #Only when employee logged into the system will connect to database
    #before that no one can access database.
    mydb = mysql.connector.connect(host = "localhost", user = "root", password ="XXXX", database = "ems")

    mydbarchive = mysql.connector.connect(host = "localhost", user = "root", password ="XXXX", database = "retire")
    #connecting to mysql and retire Db
```

Menu bar

Now to be able to select different actions we have implemented a menu bar using streamlit which contains main features of our project.

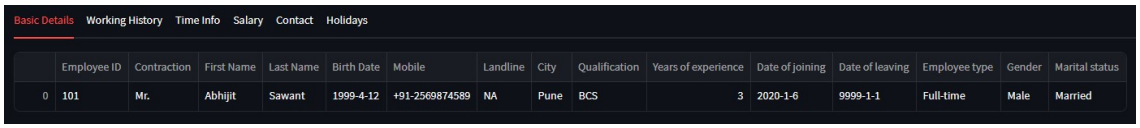
```
    with st.sidebar:
        action = option_menu(
            menu_title = "Select action",
            options = ["Display","Search", "Update", "Add", "Delete", "Archive"],
            icons = ["person-lines-fill","search", "pencil-square", "plus-square", "person-x", "archive"]

            )
```



With each menu user can able to do respective action as described in further section.

Display

When user clicks on display menu, all employees in the company will get displayed in different tabbed format as shown in below figure. We have used pandas dataframe library to display data in tabular form.
User can navigate over the tabs to view different information.

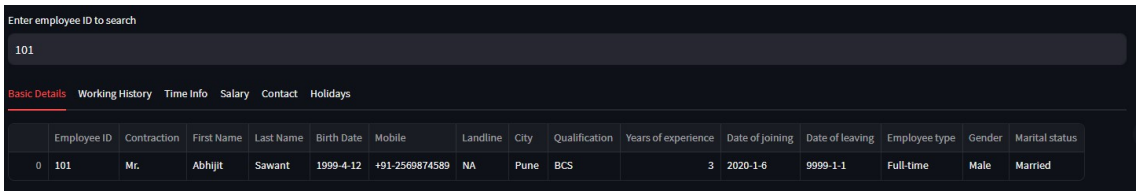


	Employee ID	Contraction	First Name	Last Name	Birth Date	Mobile	Landline	City	Qualification	Years of experience	Date of joining	Date of leaving	Employee type	Gender	Marital status
0	101	Mr.	Abhijit	Sawant	1999-4-12	+91-2569874589	NA	Pune	BCS	3	2020-1-6	9999-1-1	Full-time	Male	Married

```
if(action == 'Display'):
    tab_titles = ["Basic Details", "Working History", "Time Info", "Salary", "Contact", "Holidays"]
    tabs = st.tabs(tab_titles)
    with tabs[0]:
        L1 = []
        c1 = mydb.cursor()
        c1.execute("select * from Employee_Basic_Details")
        for row in c1:

            L1.append(row)
        df1 = pd.DataFrame(data = L1, columns = ['Employee ID','Contraction','First Name','Last Name', 'Birth Date','Mobile', 'Landline',
                                                 'City', 'Qualification', 'Years of experience', 'Date of joining',
                                                 'Date of leaving', 'Employee type', 'Gender', 'Marital status'])
        st.dataframe(df1)

    with tabs[1]:
        L2 = []
        c2 = mydb.cursor()
        c2.execute("select * from Working_History")
        for row in c2:
            L2.append(row)
        df2 = pd.DataFrame(data = L2, columns = [ 'Employee ID', 'Previous Company Name', 'Previous Date of Joining', 'Previous Date of Leaving'])
        st.dataframe(df2)
    with tabs[2]:
        L3 = []
        c3 = mydb.cursor()
        c3.execute("select * from Time_Information_Per_Week")
        for row in c3:

            L3.append(row)
        df3 = pd.DataFrame(data = L3, columns = [ 'Employee ID' ,'Worked Hours','Off Hours', 'Days Off', 'Overtime'])
        st.dataframe(df3)

    with tabs[3]:
        L4 = []
        c4 = mydb.cursor()
        c4.execute("select * from Salary_Information")
        for row in c4:
            L4.append(row)
        df4 = pd.DataFrame(data = L4, columns = ['Employee ID' ,'Annual Salary', 'Annual Tax', 'Monthly PF Deductions','Medical Policy Premium Deduction'])

        st.dataframe(df4)
    with tabs[4]:
        L5 = []
        c5 = mydb.cursor()
        c5.execute("select * from  Contact_Person_Information")
        for row in c5:
             L5.append(row)
        df5 = pd.DataFrame(data = L5, columns = ['Employee ID', 'First Name', 'Last Name', 'Relationship', 'Mobile Number', 'Landline Number','City'])
        st.dataframe(df5)

    with tabs[5]:
        L6 = []
        c6 = mydb.cursor()
        c6.execute("select * from Holiday_Information")
        for row in c6:

            L6.append(row)
        df6 = pd.DataFrame(data = L6, columns = ['Employee ID','Previlage Leave Balance', 'Sick Leave Balance', 'Emergency Leave Balance'])
        st.dataframe(df6)
```

Search

Now if user wants to search for one employee out of thousands can click on search menu. It will pop up for employee ID from user whose data needs to search. When user will enter employee ID then all data will get displayed for that particular employee.



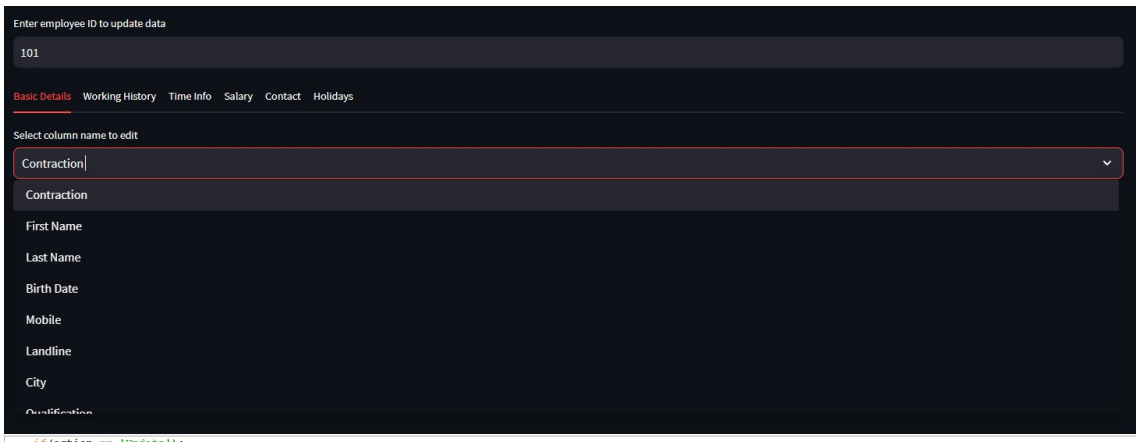

```
if(action == 'Search'):
        eid = st.text_input(label = "Enter employee ID to search")
        if(not eid):
            st.warning('Enter employee ID', icon="⚠")
        else:
            eid_list = [] #creating list to store employee id
            c = mydb.cursor()
            c.execute("select employee_id from Employee_Basic_Details")
            for row in c:
                eid_list.append(row[0])
            if eid not in eid_list:
                st.error('Employee not found', icon="🚨")
            if eid in eid_list:
                tab_titles = ["Basic Details", "Working History", "Time Info", "Salary", "Contact", "Holidays"]
                tabs = st.tabs(tab_titles)
                with tabs[0]:

                    L1 = []
                    c1 = mydb.cursor()
                    c1.execute("select * from Employee_Basic_Details")
                    for row in c1:
                        if(row[0] == eid ):
                            L1.append(row)
                    df1 = pd.DataFrame(data = L1, columns = ['Employee ID','Contraction','First Name','Last Name', 'Birth Date','Mobile', 'Landline',
                                                             'City', 'Qualification', 'Years of experience', 'Date of joining',
                                                             'Date of leaving', 'Employee type', 'Gender', 'Marital status'])

                    st.dataframe(df1)

                with tabs[1]:

                    L2 = []
                    c2 = mydb.cursor()
                    c2.execute("select * from Working_History")
                    for row in c2:
                        if(row[0] == eid ):
                            L2.append(row)
                    df2 = pd.DataFrame(data = L2, columns = [ 'Employee ID','Previous Company Name', 'Previous Date of Joining', 'Previous Date of Leaving'])
                    st.dataframe(df2)
```

```
                with tabs[2]:
                    L3 = []
                    c3 = mydb.cursor()
                    c3.execute("select * from Time_Information_Per_Week")
                    for row in c3:
                        if(row[0] == eid ):
                            L3.append(row)
                    df3 = pd.DataFrame(data = L3, columns = [ 'Employee ID' ,'Worked Hours','Off Hours', 'Days Off', 'Overtime'])
                    st.dataframe(df3)

                with tabs[3]:

                    L4 = []
                    c4 = mydb.cursor()
                    c4.execute("select * from Salary_Information")
                    for row in c4:
                        if(row[0] == eid ):
                            L4.append(row)
                    df4 = pd.DataFrame(data = L4, columns = [ 'Employee ID' ,'Annual Salary', 'Annual Tax', 'Monthly PF Deductions',
                                                              'Medical Policy Premium Deduction'])
                    st.dataframe(df4)

                with tabs[4]:

                    L5 = []
                    c5 = mydb.cursor()
                    c5.execute("select * from  Contact_Person_Information")
                    for row in c5:
                        if(row[0] == eid ):
                            L5.append(row)
                    df5 = pd.DataFrame(data = L5, columns = ['Employee ID', 'First Name', 'Last Name', 'Relationship', 'Mobile Number', 'Landline Number','City'])
                    st.dataframe(df5)

                with tabs[5]:
                    L6 = []
                    c6 = mydb.cursor()
                    c6.execute("select * from Holiday_Information")
                    for row in c6:
                        if(row[0] == eid ):
                            L6.append(row)
                    df6 = pd.DataFrame(data = L6, columns = ['Employee ID','Previlage Leave Balance', 'Sick Leave Balance', 'Emergency Leave Balance'])
                    st.dataframe(df6)
```

Update

Now if user wants to update for one employee out of thousands can click on update menu. It will pop up for employee ID from user whose data needs to update. When user will enter

employee ID then tabs will get displayed and in each tab user can update respective data by selecting from dropdown.




```
if(action == 'Update'):
    obj_ems = ems()
    eid = st.text_input(label = "Enter employee ID to update data")
    if(not eid):
        st.warning('Enter employee ID', icon="⚠")
    else:
        eid_list = [] #creating list to store employee id
        c = mydb.cursor()
        c.execute("select employee_id from Employee_Basic_Details")
        for row in c:
            eid_list.append(row[0])
        if eid not in eid_list:
            st.error('Employee not found', icon="🚨")
        if eid in eid_list:
            tab_titles = ["Basic Details", "Working History", "Time Info", "Salary", "Contact", "Holidays"]
            tabs = st.tabs(tab_titles)
            with tabs[0]:
                table_name = "Employee_Basic_Details"
                editcolumn = st.selectbox("Select column name to edit",('Contraction','First Name','Last Name', 'Birth Date','Mobile', 'Landline',
                                                                        'City', 'Qualification', 'Years of experience', 'Date of joining',
                                                                        'Date of leaving',  'Gender', 'Marital status') )
                st.write('You selected:', editcolumn)
                dictionary = {'Contraction':'name_prefix','First Name' : 'first_name', 'Last Name' : 'last_name', 'Birth Date' :  'date_of_birth',
                              'Mobile' : 'mobile_number',
                              'Landline' :  'landline_number','City' : 'city', 'Qualification' :  'qualification' ,
                              'Years of experience' : 'year_of_experience', 'Date of joining' : 'date_of_joining',
                              'Date of leaving' : 'date_of_leaving',  'Gender' : 'gender' , 'Marital status' :  'marital_status'}
                if(editcolumn == 'Contraction'or'First Name' or 'Last Name' or 'Birth Date' or 'Mobile'or 'Landline'or 'City'
                   or 'Qualification'or 'Years of experience' or 'Date of joining' or'Date of leaving' or  'Gender' or 'Marital status' ):
                    for k,v in dictionary.items():
                        if k == editcolumn:
                            column_name = v
                    new_value =  st.text_input(label = f'Enter new value for column {column_name} ', max_chars = 14,
                                               help = 'Date: YYYY-MM-DD, Mobile: +91-XXXXXXXXXX, Landline: 020-XXXXXXXX')
                    if(new_value):
                        command = "update {0} set {1} ='{2}' where employee_id = '{3}'".format(table_name, column_name, new_value, eid )
                        obj_ems.update(command)
                        st.success('Data updated successfully', icon="☑")
            with tabs[1]:
                table_name = "Working_History"
                editcolumn = st.selectbox("Select column name to edit",('Previous Company Name', 'Previous Date of Joining', 'Previous Date of Leaving') )
                st.write('You selected:', editcolumn)
                dictionary = {'Previous Company Name' : 'previous_company_name', 'Previous Date of Joining': 'previous_date_of_joining',
                              'Previous Date of Leaving':' previous_date_of_leaving'}
                if(editcolumn == 'Previous Company Name' or 'Previous Date of Joining' or 'Previous Date of Leaving'):
                    for k,v in dictionary.items():
                        if k == editcolumn:
                            column_name = v
                    new_value =  st.text_input(label = f'Enter new value for column {column_name} ', max_chars = 14,
                                               help = 'Date: YYYY-MM-DD, Mobile: +91-XXXXXXXXXX, Landline: 020-XXXXXXXX')
                    if(new_value):
                        command = "update {0} set {1} ='{2}' where employee_id = '{3}'".format(table_name, column_name, new_value, eid )
                        obj_ems.update(command)
                        st.success('Data updated successfully', icon="☑")
            with tabs[1]:
                table_name = "Working_History"
                editcolumn = st.selectbox("Select column name to edit",('Previous Company Name', 'Previous Date of Joining', 'Previous Date of Leaving') )
                st.write('You selected:', editcolumn)
                dictionary = {'Previous Company Name' : 'previous_company_name', 'Previous Date of Joining': 'previous_date_of_joining',
                              'Previous Date of Leaving':' previous_date_of_leaving'}
                if(editcolumn == 'Previous Company Name' or 'Previous Date of Joining' or 'Previous Date of Leaving'):
                    for k,v in dictionary.items():
                        if k == editcolumn:
                            column_name = v
                    new_value =  st.text_input(label = f'Enter new value for column {column_name}', max_chars = 14,
                                               help = 'Date: YYYY-MM-DD, Mobile: +91-XXXXXXXXXX, Landline: 020-XXXXXXXX')
                    if(new_value):
                        command = "update {0} set {1} ='{2}' where employee_id = '{3}'".format(table_name, column_name, new_value, eid )
                        obj_ems.update(command)
                        st.success('Data updated successfully', icon="☑")
            with tabs[2]:
                table_name = "Time_Information_Per_Week"
                editcolumn = st.selectbox("Select column name to edit",('Worked hours','Off Hours', 'Days Off', 'Overtime') )
                st.write('You selected:', editcolumn)
                dictionary = {'Worked hours' : 'Worked_hours', 'Off Hours' : 'Off_hours', 'Days Off' : 'days_off','Overtime' : 'overtime'}
                if(editcolumn == 'Worked hours'or'Off Hours'or 'Days Off'or 'Overtime'):
                    for k,v in dictionary.items():
                        if k == editcolumn:
                            column_name = v
                    new_value =  st.text_input(label = f'Enter new value for column {column_name}', max_chars = 14,
                                               help = 'Date: YYYY-MM-DD, Mobile: +91-XXXXXXXXXX, Landline: 020-XXXXXXXX')
                    if(new_value):
                        command = "update {0} set {1} ='{2}' where employee_id = '{3}'".format(table_name, column_name, new_value, eid )
                        obj_ems.update(command)
                        st.success('Data updated successfully', icon="☑")
            with tabs[3]:
                table_name = "Salary_Information"
                editcolumn = st.selectbox("Select column name to edit",('Annual Salary',None) )
                st.write('You selected:', editcolumn)
                dictionary = {'Annual Salary':'annual_salary'}
                if(editcolumn == 'Annual Salary'):
```

```
            if(editcolumn == 'Annual Salary'):
                for k,v in dictionary.items():
                    if k == editcolumn:
                        column_name = v
                new_value =  st.number_input(label = f'Enter new value for column {column_name}',
                                             help = 'Date: YYYY-MM-DD, Mobile: +91-XXXXXXXXXX, Landline: 020-XXXXXXXX')
                new_annual_tax = 0
                #int_salary = int(new_value)
                #Calculating annual tax as per annual salary
                if(250000 < int(new_value) <= 500000 ):
                    new_annual_tax = str(int(new_value) * 0.05)
                if(500000 < int(new_value) <= 1000000 ):
                    new_annual_tax = str(int(new_value) * 0.2)
                if(1000000 < int(new_value)):
                    new_annual_tax = str(int(new_value) * 0.3)
                command = "update {0} set {1} ='{2}' where employee_id = '{3}'".format(table_name, 'annual_tax', new_annual_tax, eid )
                obj_ems.update(command)
                #Calculating monthly PF deductions as per rules
                new_monthly_PF_deductions = str((int(new_value)/12)*0.12 + (int(new_value)/12)*0.0367)
                command = "update {0} set {1} ='{2}' where employee_id = '{3}'".format(table_name, 'monthly_PF_deductions', new_monthly_PF_deductions, eid )
                obj_ems.update(command)
                #Calculating monthly Medical policy premium
                new_premium = 0
                if(250000 < int(new_value) <= 500000 ):
                    new_premium = str(int(new_value) * 0.0025)
                if(500000 < int(new_value) <= 1000000 ):
                    new_premium = str(int(new_value) * 0.01)
                if(1000000 < int(new_value)):
                    new_premium = str(int(new_value) * 0.015)
                command = "update {0} set {1} ='{2}' where employee_id = '{3}'".format(table_name, 'medical_policy_premium_deduction', new_premium, eid )
                obj_ems.update(command)

                if(new_value):
                    command = "update {0} set {1} ='{2}' where employee_id = '{3}'".format(table_name, column_name, new_value, eid )
                    obj_ems.update(command)
                    st.success('Data updated successfully', icon="☑")
        with tabs[4]:
            table_name = "Contact_Person_Information"
            editcolumn = st.selectbox("Select column name to edit",('First Name', 'Last Name', 'Relationship', 'Mobile Number', 'Landline Number','City') )
            st.write('You selected:', editcolumn)
            dictionary = {'First Name':'c_first_name','Last Name': 'c_last_name', 'Relationship' : 'relationship',
                          'Mobile Number' : 'c_mobile_number', 'Landline Number':'c_landline_number','City' : 'c_city'}
            if(editcolumn == 'First Name'or 'Last Name'or 'Relationship'or 'Mobile Number'or 'Landline Number'or'City'):
                for k,v in dictionary.items():
                    if k == editcolumn:
                        column_name = v
```



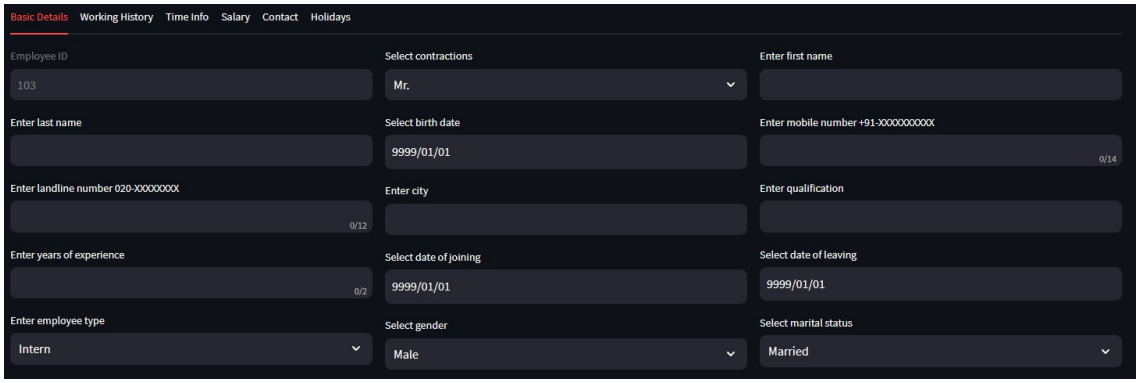
```
                        column_name = v
                new_value =  st.text_input(label = f'Enter new value for column {column_name}', max_chars = 14,
                                           help = 'Date: YYYY-MM-DD, Mobile: +91-XXXXXXXXXX, Landline: 020-XXXXXXXX')
                if(new_value):
                    command = "update {0} set {1} ='{2}' where employee_id = '{3}'".format(table_name, column_name, new_value, eid )
                    obj_ems.update(command)
                    st.success('Data updated successfully', icon="☑")
        with tabs[3]:
            table_name = "Salary_Information"
            editcolumn = st.selectbox("Select column name to edit",('Annual Salary',None) )
            st.write('You selected:', editcolumn)
            dictionary = {'Annual Salary':'annual_salary'}
            if(editcolumn == 'Annual Salary'):
                for k,v in dictionary.items():
                    if k == editcolumn:
                        column_name = v
                new_value =  st.number_input(label = f'Enter new value for column {column_name}',
                                             help = 'Date: YYYY-MM-DD, Mobile: +91-XXXXXXXXXX, Landline: 020-XXXXXXXX')
                new_annual_tax = 0
                #int_salary = int(new_value)
                #Calculating annual tax as per annual salary
                if(250000 < int(new_value) <= 500000 ):
                    new_annual_tax = str(int(new_value) * 0.05)
                if(500000 < int(new_value) <= 1000000 ):
                    new_annual_tax = str(int(new_value) * 0.2)
                if(1000000 < int(new_value)):
                    new_annual_tax = str(int(new_value) * 0.3)
                command = "update {0} set {1} ='{2}' where employee_id = '{3}'".format(table_name, 'annual_tax', new_annual_tax, eid )
                obj_ems.update(command)
                #Calculating monthly PF deductions as per rules
                new_monthly_PF_deductions = str((int(new_value)/12)*0.12 + (int(new_value)/12)*0.0367)
                command = "update {0} set {1} ='{2}' where employee_id = '{3}'".format(table_name, 'monthly_PF_deductions', new_monthly_PF_deductions, eid )
                obj_ems.update(command)
                #Calculating monthly Medical policy premium
                new_premium = 0
                if(250000 < int(new_value) <= 500000 ):
                    new_premium = str(int(new_value) * 0.0025)
                if(500000 < int(new_value) <= 1000000 ):
                    new_premium = str(int(new_value) * 0.01)
                if(1000000 < int(new_value)):
                    new_premium = str(int(new_value) * 0.015)
                command = "update {0} set {1} ='{2}' where employee_id = '{3}'".format(table_name, 'medical_policy_premium_deduction', new_premium, eid )
                obj_ems.update(command)

                if(new_value):
                    command = "update {0} set {1} ='{2}' where employee id = '{3}'".format(table name, column name, new value, eid )
```

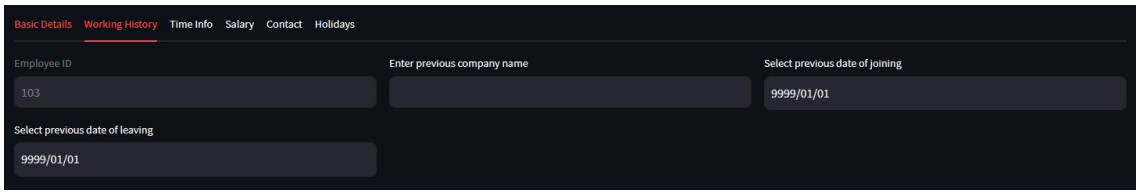


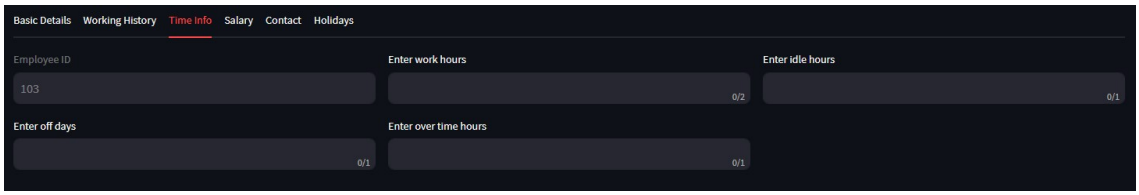
```
                    command = "update {0} set {1} ='{2}' where employee_id = '{3}'".format(table_name, column_name, new_value, eid )
                    obj_ems.update(command)
                    st.success('Data updated successfully', icon="☑")
        with tabs[4]:
            table_name = "Contact_Person_Information"
            editcolumn = st.selectbox("Select column name to edit",('First Name', 'Last Name', 'Relationship', 'Mobile Number', 'Landline Number','City') )
            st.write('You selected:', editcolumn)
            dictionary = {'First Name':'c_first_name','Last Name': 'c_last_name', 'Relationship' : 'relationship',
                          'Mobile Number' : 'c_mobile_number', 'Landline Number':'c_landline_number','City' : 'c_city'}
            if(editcolumn == 'First Name'or 'Last Name'or 'Relationship'or 'Mobile Number'or 'Landline Number'or'City'):
                for k,v in dictionary.items():
                    if k == editcolumn:
                        column_name = v
                new_value =  st.text_input(label = f'Enter new value for column {column_name}', max_chars = 14,
                                           help = 'Date: YYYY-MM-DD, Mobile: +91-XXXXXXXXXX, Landline: 020-XXXXXXXX')
                if(new_value):
                    command = "update {0} set {1} ='{2}' where employee_id = '{3}'".format(table_name, column_name, new_value, eid )
                    obj_ems.update(command)
                    st.success('Data updated successfully', icon="☑")
        with tabs[5]:
            table_name = "Holiday_Information"
            editcolumn = st.selectbox("Select column name to edit",('Previlage Leave Balance', 'Sick Leave Balance', 'Emergency Leave Balance') )
            st.write('You selected:', editcolumn)
            dictionary = {'Previlage Leave Balance' : 'previlage_leave_balance', 'Sick Leave Balance' : 'sick_leave_balance',
                          'Emergency Leave Balance':'emergency_leave_balance'}
            if(editcolumn == 'Previlage Leave Balance'or 'Sick Leave Balance'or 'Emergency Leave Balance'):
                for k,v in dictionary.items():
                    if k == editcolumn:
                        column_name = v
                new_value =  st.text_input(label = f'Enter new value for column {column_name}', max_chars = 2,
                                           help = 'Date: YYYY-MM-DD, Mobile: +91-XXXXXXXXXX, Landline: 020-XXXXXXXX')
                if(new_value):
                    command = "update {0} set {1} ='{2}' where employee_id = '{3}'".format(table_name, column_name, new_value, eid )
                    obj_ems.update(command)
                    st.success('Data updated successfully', icon="☑")
```

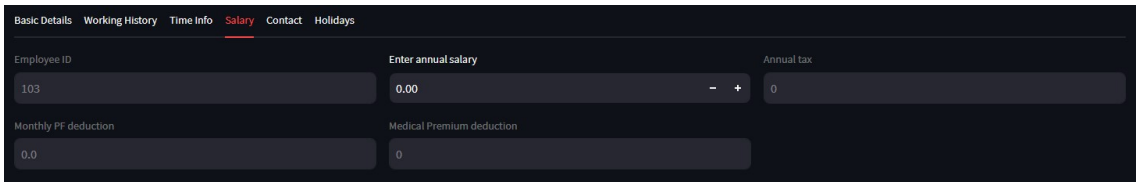
Add

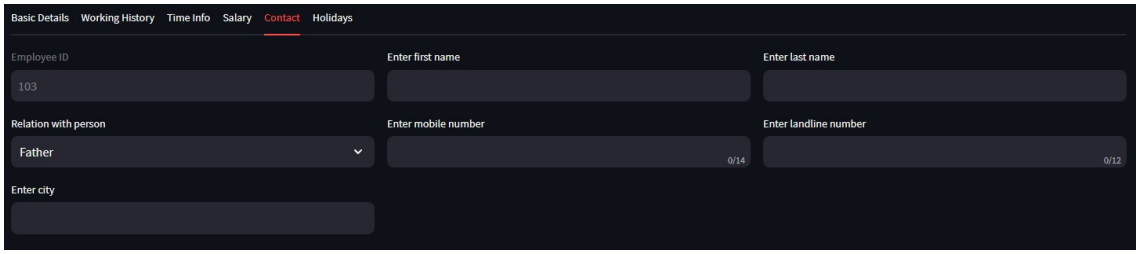
Now if user wants to add new employee by selecting add menu user can enter new data. It will display a form for data in all tabs as shown in below snippets; unless user enters all data forms cannot be submitted. When new employee is going to add employee ID will be automatically taken from system as coded in program. When all data is filled submit button will appear and user can add new employee to database.

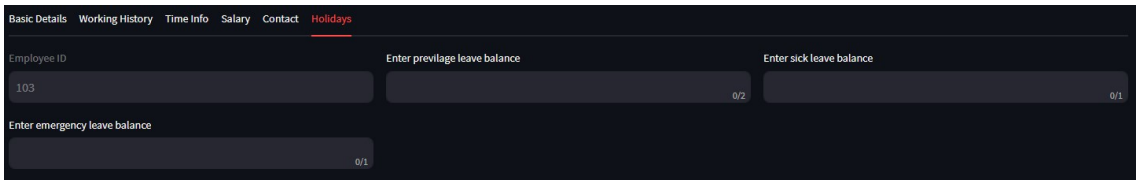













```
    if(action == 'Add'):
        tab_titles = ["Basic Details", "Working History", "Time Info", "Salary", "Contact", "Holidays"]
        tabs = st.tabs(tab_titles)
        def next_eid():#function to Generating next employee ID
            last_employee_id = ""
            last_employee_id_r = ""
            c = mydb.cursor()
            c.execute("select employee_id from Employee_Basic_Details")
            for row in c:
                last_employee_id = row[0]
            c = mydbarchive.cursor()
            c.execute("select employee_id from Employee_Basic_Details_retired")
            for row in c:
                last_employee_id_r = row[0]
            if(last_employee_id > last_employee_id_r ):
                return int(last_employee_id) + 1
            elif(last_employee_id < last_employee_id_r ):
                return int(last_employee_id_r) + 1
            else:
                return 101
        with st.form("form0",clear_on_submit = False):
            with tabs[0]:
                c1, c2, c3 = st.columns((1, 1, 1))

                with c1:
                    employeeid = st.text_input(label = "Employee ID",value= next_eid(),disabled=True )

                with c2:
                    prefix = st.selectbox("Select contractions",('Mr.','Mrs.','Miss','Ms.'))
                with c3:
                    firstname = st.text_input(label = "Enter first name")
                with c1:
                    lastname = st.text_input(label = "Enter last name")
                with c2:
                    birthdate = st.date_input(label = "Select birth date", value = datetime.datetime(9999, 1, 1),
                      min_value=datetime.date(year=1800, month=12, day=31),
                      max_value=datetime.date(year=9999, month=12, day=31))
                with c3:
                    mobile = st.text_input(label = "Enter mobile number +91-XXXXXXXXXX",  max_chars = 14)

                with c1:
                    landline = st.text_input(label = "Enter landline number 020-XXXXXXXX", max_chars = 12)
                with c2:
                    city = st.text_input(label = "Enter city")
                with c3:
                    qualification = st.text input(label = "Enter qualification")
```

```
        with c3:
            qualification = st.text_input(label = "Enter qualification")
        with c1:
            yearsofexperience = st.text_input(label = "Enter years of experience", max_chars = 2)
        with c2:
            joindate = st.date_input(label = "Select date of joining",value = datetime.datetime(9999, 1, 1),
             min_value=datetime.date(year=1900, month=12, day=31),
             max_value=datetime.date(year=9999, month=12, day=31))
        with c3:
            leavedate = st.date_input(label = "Select date of leaving", value = datetime.datetime(9999, 1, 1),
              min_value=datetime.date(year=1900, month=12, day=31),
              max_value=datetime.date(year=9999, month=12, day=31))
        with c1:
            employeetype = st.selectbox("Enter employee type", ('Intern','Part-time','Full-time','Contractual'))
        with c2:

            gender = st.selectbox("Select gender",('Male','Female'))
        with c3:
            maritalstatus = st.selectbox("Select marital status",('Married','Single','Divorced', 'Do not want to disclose'))


    with tabs[1]:
        c1, c2, c3 = st.columns((1, 1, 1))
        #with st.form("form1",clear_on_submit = False):
        with c1:
            employeeid = st.text_input(label = "Employee ID ",value= next_eid(),disabled=True )

        with c2:
            precompany = st.text_input(label = "Enter previous company name")
        with c3:
            prevjoindate = st.date_input(label = "Select previous date of joining",value = datetime.datetime(9999, 1, 1),
             min_value=datetime.date(year=1900, month=12, day=31),
             max_value=datetime.date(year=9999, month=12, day=31))
        with c1:
            prevleavedate = st.date_input(label = "Select previous date of leaving", value = datetime.datetime(9999, 1, 1),
              min_value=datetime.date(year=1900, month=12, day=31),
              max_value=datetime.date(year=9999, month=12, day=31))


    with tabs[2]:
        c1, c2, c3 = st.columns((1, 1, 1))
        #with st.form("form2",clear_on_submit = False):
        with c1:
            employeeid = st.text input(label = " Employee ID ",value= next eid(),disabled=True )
```

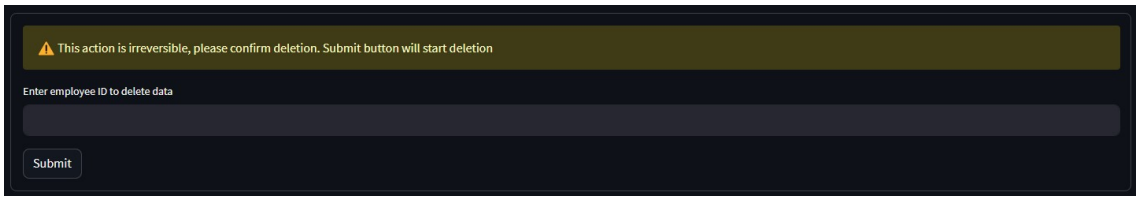
```
            employeeid = st.text_input(label = " Employee ID ",value= next_eid(),disabled=True )

        with c2:
            workedhours = st.text_input(label = "Enter work hours", max_chars = 2)
        with c3:
            offhours = st.text_input(label = "Enter idle hours" , max_chars = 1 )
        with c1:
            daysoff = st.text_input(label = "Enter off days", max_chars = 1)
        with c2:
            ot = st.text_input(label = "Enter over time hours", max_chars = 1)

    with tabs[3]:
        c1, c2, c3 = st.columns((1, 1, 1))
        #with st.form("form3",clear_on_submit = False):
        with c1:
            employeeid = st.text_input(label = " Employee ID  ",value= next_eid(),disabled=True )

        with c2:
            salary  = st.number_input(label = "Enter annual salary")
        with c3:
            def tax():
                annual_tax = 0
                #int_salary = int(new_value)
                #Calculating annual tax as per annual salary
                if(250000 < int(salary) <= 500000 ):
                    annual_tax = str(int(salary) * 0.05)

                if(500000 < int(salary) <= 1000000 ):
                    top = int(salary) - 500000
                    annual_tax = str((top * 0.2) + 12500)

                if(1000000 < int(salary)):
                    top1 = int(salary) - 1000000
                    top2 = 500000
                    annual_tax = str((top1 * 0.3) + (top2 * 0.2) +  12500)
                return annual_tax
            tax = st.text_input(label = "Annual tax" ,value= tax(),disabled=True )
        with c1:
            PF_deductions = str((int(salary)/12)*0.12 + (int(salary)/12)*0.0367)
            pf = st.text_input(label = "Monthly PF deduction",value= PF_deductions,disabled=True)
        with c2:
            new_premium = 0
            if(250000 < int(salary) <= 500000 ):
                    new_premium = str(int(salary) * 0.0025)
            if(500000 < int(salary) <= 1000000 ):
                    new premium = str(int(salary) * 0.01)
```

```
                    new_premium = str(int(salary) * 0.01)
                if(1000000 < int(salary)):
                    new_premium = str(int(salary) * 0.015)
                premium = st.text_input(label = "Medical Premium deduction", value= new_premium,disabled=True)

        with tabs[4]:
            c1, c2, c3 = st.columns((1, 1, 1))
            #with st.form("form4",clear_on_submit = False):
            with c1:
                employeeid = st.text_input(label = " Employee  ID ",value= next_eid(),disabled=True )

            with c2:
                fname = st.text_input(label = "Enter first name ")
            with c3:
                lname = st.text_input(label = "Enter last name "  )
            with c1:
                relation = st.selectbox("Relation with person", ('Father','Mother','Sister','Brother','Other'))

            with c2:
                mobno = st.text_input(label = "Enter mobile number ", max_chars = 14)
            with c3:
                lanno = st.text_input(label = "Enter landline number ", max_chars = 12)
            with c1:
                cty = st.text_input(label = "Enter city ")
        with tabs[5]:
            c1, c2, c3 = st.columns((1, 1, 1))
            #with st.form("form5",clear_on_submit = False):
            with c1:
                employeeid = st.text_input(label = "  Employee  ID ",value= next_eid(),disabled=True )

            with c2:
                pl = st.text_input(label = "Enter previlage leave balance ", max_chars = 2)
            with c3:
                sl = st.text_input(label = "Enter sick leave balance ", max_chars = 1  )
            with c1:
                el = st.text_input(label = "Enter emergency leave balance  ", max_chars = 1)


        if(employeeid and prefix and firstname and lastname and birthdate and mobile and landline and city and qualification
           and yearsofexperience and joindate and leavedate and employeetype and gender and  precompany and prevjoindate and prevleavedate
            and workedhours and offhours and daysoff and ot and  salary and tax and pf and premium and  fname and lname and
            relation and mobno and lanno and cty and  pl and sl and el):
            submit = st.form_submit_button("Submit" )
            if(submit):
                c = mydb.cursor(buffered=True)
                newempdata = (employeeid, prefix, firstname, lastname, birthdate, mobile, landline, city, qualification,
```

```
                newempdata = (employeeid, prefix, firstname, lastname, birthdate, mobile, landline, city, qualification,
                    yearsofexperience, joindate, leavedate, employeetype, gender, maritalstatus )
                add = """insert into Employee_Basic_Details(`employee_id`,`name_prefix`, `first_name`,`last_name`,
                                                `date_of_birth`,`mobile_number`,`landline_number`,
                                                `city`,`qualification`,`year_of_experience`,`date_of_joining`,
                                                `date_of_leaving`,`employee_type`,`gender` , `marital_status`)
                                                values(%s,%s,%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)"""
                c.execute(add,newempdata)
                mydb.commit()

                c = mydb.cursor(buffered=True)
                newempdata = (employeeid, precompany, prevjoindate, prevleavedate )
                add = """insert into Working_History(`employee_id`,`previous_company_name`,`previous_date_of_joining`,`previous_date_of_leaving`)
                                values(%s, %s,%s, %s)"""
                c.execute(add,newempdata)
                mydb.commit()

                c = mydb.cursor(buffered=True)
                newempdata = (employeeid, workedhours, offhours, daysoff, ot )
                add = """insert into Time_Information_Per_Week(`employee_id`,`Worked_hours`,`Off_hours`,`days_off`, `overtime`)
                                                values(%s,%s,%s, %s, %s)"""
                c.execute(add,newempdata)
                mydb.commit()

                c = mydb.cursor(buffered=True)
                newempdata = ( employeeid,salary , tax, pf, premium )
                add = """insert into Salary_Information(`employee_id`,`annual_salary`,`annual_tax`,`monthly_PF_deductions`, `medical_policy_premium_deduction`)
                                values(%s,%s,%s, %s, %s)"""
                c.execute(add,newempdata)
                mydb.commit()

                c = mydb.cursor(buffered=True)
                newempdata = ( employeeid,fname, lname, relation,  mobno,lanno, cty )
                add = """insert into Contact_Person_Information(`employee_id`,`c_first_name`,`c_last_name`,`relationship`,
                                `c_mobile_number`, `c_landline_number`, `c_city`)
                                values(%s,%s,%s, %s, %s, %s, %s)"""
                c.execute(add,newempdata)
                mydb.commit()

                c = mydb.cursor(buffered=True)
                newempdata = (employeeid, pl,sl,el )
                add = """insert into Holiday_Information(`employee_id`,`previlage_leave_balance`,`sick_leave_balance`,`emergency_leave_balance`)
                                values(%s,%s,%s, %s)"""
                c.execute(add,newempdata)
                mydb.commit()
                st.success('Employee added successfully', icon="☑")
```


Delete

When user selects delete menu it will ask for employee ID to delete, when user click on submit button data related with that employee will move to retire database and gets deleted from ems database.



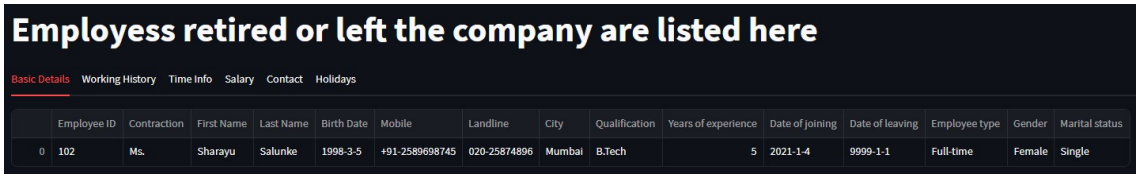

```
if(action == 'Delete'):
    with st.form("deleteform"):
        st.warning('This action is irreversible, please confirm deletion. Submit button will start deletion', icon="⚠")
        eid = st.text_input(label = "Enter employee ID to delete data")
        submit = st.form_submit_button("Submit")
        if(submit):
            if(not eid):
                st.warning('Enter employee ID', icon="⚠")
            else:
                eid_list = [] #creating list to store employee id
                c = mydb.cursor()
                c.execute("select employee_id from Employee_Basic_Details")
                for row in c:
                    eid_list.append(row[0])
                if eid not in eid_list:
                    st.error('Employee not found', icon="🚨")
                if eid in eid_list:
                    c = mydb.cursor()
                    c.execute("select employee_id from Employee_Basic_Details")
                    for row in c: # iterating throgh multiple rows of tuple containg employee id
                        if eid in row: # Searching in tuple
                            delid = row
                    c = mydb.cursor(buffered=True)
                    c.execute("insert into retire.Employee_Basic_Details_retired select * from ems.Employee_Basic_Details where employee_id = %s",delid )
                    c = mydb.cursor(buffered=True)
                    c.execute("insert into retire.Working_History_retired select * from ems.Working_History where employee_id = %s",delid)
                    c = mydb.cursor(buffered=True)
                    c.execute("insert into retire.Time_Information_Per_Week_retired select * from ems.Time_Information_Per_Week where employee_id = %s",delid)
                    c = mydb.cursor(buffered=True)
                    c.execute("insert into retire.Salary_Information_retired select * from ems.Salary_Information where employee_id = %s",delid)
                    c = mydb.cursor(buffered=True)
                    c.execute("insert into retire.Contact_Person_Information_retired select * from ems.Contact_Person_Information where employee_id = %s",delid)
                    c = mydb.cursor(buffered=True)
                    c.execute("insert into retire.Holiday_Information_retired select * from ems.Holiday_Information where employee_id = %s",delid)

                    st.success('Employee archived', icon="☑")
                    progress_text = "Deletion in progress. Please wait."
                    my_bar = st.progress(0, text=progress_text)
```



```
                for percent_complete in range(100):
                    time.sleep(0.01)
                    my_bar.progress(percent_complete + 1, text=progress_text)
                #We are deleteing information from child tables
                #first and lastly deleting dat of parent table,
                # As deleteing parent table raises error
                c = mydb.cursor(buffered=True)
                c.execute("delete from Working_History where employee_id = %s ",delid)
                mydb.commit()
                c = mydb.cursor(buffered=True)
                c.execute("delete from Time_Information_Per_Week where employee_id = %s ",delid)
                mydb.commit()
                c = mydb.cursor(buffered=True)
                c.execute("delete from Salary_Information where employee_id = %s ",delid)
                mydb.commit()
                c = mydb.cursor(buffered=True)
                c.execute("delete from Contact_Person_Information where employee_id= %s ",delid)
                mydb.commit()
                c = mydb.cursor(buffered=True)
                c.execute("delete from Holiday_Information where employee_id = %s",delid)
                mydb.commit()
                c = mydb.cursor(buffered=True)
                c.execute("delete from Employee_Basic_Details where employee_id = %s",delid)
                mydb.commit()
                st.success('Data deleted successfully', icon="☑")
```

Archive

When user selects archive menu, it will display all employee data who has either retired or have left the company.



	Employee ID	Contraction	First Name	Last Name	Birth Date	Mobile	Landline	City	Qualification	Years of experience	Date of joining	Date of leaving	Employee type	Gender	Marital status
0	102	Ms.	Sharayu	Salunke	1998-3-5	+91-2589698745	020-25874896	Mumbai	B.Tech	5	2021-1-4	9999-1-1	Full-time	Female	Single

Logout

When user click on logout button, system will log out successfully and will display login page again.


```
authenticator.logout("Logout", "sidebar")
```