Problem Statement #4: Console-Based/Backend & Business Logic Employee Management Application

Overview :

To create a Console-Based Employee Management Application with various CRUD operations such as :

- Creating the Employee Records
- Updating the Employee Records
- Showing all the Employee Data
- Showing all the Employees in the Company
- Search for a Particular Employee Records based on various criteria
- Deleting the Employee's Record
- Department-wise Average salary
- Overall Average salary of Employees in the Company

Installation:

These are some of the step-by-step instructions to install and set up the application, including various dependencies and configurations.

1. Download or clone the source code from the GitHub repository https://github.com/NafiyaThehreem/Presidio_Task.git
2. Open the file in PyCharm Python IDE or cd into the respective directory using the terminal (I have used hyper terminal).
3. Install the dependencies like PrettyTable by running a command on the PyCharm Terminal : pip install PrettyTable
4. All set we are ready to run the code.

Database Setup:

To set up a Database named *Employeedatabase.db* and create a table named *Employees* we have the below code:

```
import sqlite3
mydb = sqlite3.connect('Employeedatabase.db')
mycursor = mydb.cursor()
print('Database Connected Successfully')
mycursor.execute('''CREATE TABLE IF NOT EXISTS
employees(emp_id INTEGER PRIMARY KEY,emp_name
VARCHAR,age INTEGER,
    date_of_birth DATE, salary REAL, emp_dept
VARCHAR)''')
mydb.commit()
```

Code Structure:

The code consists of 8 Functions, each handling different functionalities of the Employee Management System. The functions are as mentioned below:

- `insertIntoTable(mycursor, mydb)`
- `updateEmployeeDetails(mycursor, mydb)`
- `showAllEmployees(mycursor)`
- `employeesInCompany(mycursor)`
- `searchEmployee(mycursor)`
- `deleteEmployeeRecord(mycursor, mydb)`
- `averageSalaryInDept(mycursor)`
- `averageSalaryOfEmployees(mycursor)`

The function **insertIntoTable(mycursor,mydb)** performs the operations of creating the employee records in the database.

```
def insertIntoTable(mycursor, mydb):

    print("Enter Employee id")
    emp_id = int(input())
    print("Enter Employee Name")
    emp_name = input()
    print("Enter the age of Employee")
    age = int(input())
    print("Enter DOB of the Employee")
    dob = input()
    print("Enter the salary of Employee")
    salary = float(input())
    print("Enter the department of Employee")
    emp_dept = input()
    employee_data = (emp_id, emp_name, age, dob,salary,
emp_dept)
    mycursor.execute('INSERT INTO employees(emp_id,
emp_name, age, date_of_birth, salary, emp_dept)
VALUES(?,?,?,?,?,?)',employee_data)
    mydb.commit()
```

The function **updateEmployeeDetails(mycursor,mydb)** performs the operations of updating the employee records in the database such as their Name, age, dob, salary or the department of the respective employee with their specified Employee ID as it is the primary key of *Employees* Table.

```
def updateEmployeeDetails(mycursor, mydb):

    print("What do you want to update in the Employee
record:")
    print("1.Employee Name \n2.Employee Age
```

```
\n3.Employee DOB \n4.Employee Salary \n5.Employee
Department")
    ch = int(input())

    if (ch == 1):
        print("Enter the Employee Id whose name has to
be Updated")
        emp_id = int(input())
        print("Enter Employee's Updated Name")
        emp_name = input()
        mycursor.execute("UPDATE employees SET emp_name
= ? WHERE emp_id = ?", (emp_name, emp_id))
        mydb.commit()

    elif (ch == 2):
        print("Enter the Employee Id whose Age has to
be Updated")
        emp_id = int(input())
        print("Enter Employee's Updated Age")
        age = int(input())
        mycursor.execute("UPDATE employees SET age = ?
WHERE emp_id = ?", (age, emp_id))
        mydb.commit()

    elif (ch == 3):
        print("Enter the Employee Id whose DOB has to
be Updated")
        emp_id = int(input())
        print("Enter Employee's Updated DOB")
        dob = input()
        mycursor.execute("UPDATE employees SET
date_of_birth = ? WHERE emp_id = ?", (dob, emp_id))
        mydb.commit()

    elif (ch == 4):
        print("Enter the Employee Id whose salary has
to be Updated")
        emp_id = int(input())
        print("Enter Employee's Updated salary")
        salary = float(input())
```

```
        mycursor.execute("UPDATE employees SET salary =
? WHERE emp_id = ?", (salary, emp_id))
        mydb.commit()

    elif (ch == 5):
        print("Enter the Employee Id whose Department
has to be Updated")
        emp_id = int(input())
        print("Enter Employee's Updated Department")
        dept = input()
        mycursor.execute("UPDATE employees SET emp_dept
= ? WHERE emp_id = ?", (dept, emp_id))
        mydb.commit()

    else:
        print("Invalid Choice")

    print("Employee Records Updated Successfully")
```

The function **showAllEmployees(mycursor)** performs the operations of Printing or displaying all the employee records in the database.To make it more attractive I have installed PrettyTable package for displaying the records in a table format. To check the working of PrettyTable, I used try and catch blocks to avoid any sort of exception.

```
try:
    from prettytable import PrettyTable

    print("PrettyTable is installed.")
except ImportError:
    print("PrettyTable is not installed.")
```

```
def printEmployeeDetails(t):
    table = PrettyTable(["emp_id", "emp_name", "age",
"dob", "salary", "emp_dept"])
    for i in t:
```

```
        table.add_row(i)
    print(table)
```

```
def showAllEmployees(mycursor):
    mycursor.execute('SELECT * FROM employees')
    t = mycursor.fetchall()
    printEmployeeDetails(t)
```

The function **employeesInCompany(mycursor)** performs the operations of Printing or displaying all the ID's and names of the Employees working in the company.

```
def printEmployees(t, emp_id, emp_name):
    table = PrettyTable([emp_id, emp_name])
    for i in t:
        table.add_row(i)
    print(table)
```

```
def employeesInCompany(mycursor):
    mycursor.execute("SELECT emp_id, emp_name FROM
employees")
    t = mycursor.fetchall()
    printEmployees(t, "ID", "Employee Name")
```

The function **searchEmployee (mycursor)** performs the operations of searching or filtering the Employees based on some criteria such as its ID, name or the Department the Employee works in.

```
def searchEmployee(mycursor):
    print("How do you want to search an Employee
record:")
    print("1.Using Employee Id \n2.Using Employee Name
\n3.Using Department")
```

```
    ch = int(input())

    if (ch == 1):
        print("Enter Employee Id")
        emp_id = int(input())
        mycursor.execute('SELECT * FROM employees WHERE
emp_id = ?', (emp_id,))
        t = mycursor.fetchall()
        printEmployeeDetails(t)

    elif (ch == 2):
        print("Enter Employee Name")
        emp_name = input()
        mycursor.execute('SELECT * FROM employees WHERE
emp_name = ?', (emp_name,))
        t = mycursor.fetchall()
        printEmployeeDetails(t)

    elif (ch == 3):
        print("Enter Department")
        emp_dept = input()
        mycursor.execute('SELECT * FROM employees WHERE
emp_dept = ?', (emp_dept,))
        t = mycursor.fetchall()
        printEmployeeDetails(t)

    else:
        print("Invalid Choice")
```

The function **deleteEmployeeRecord(mycursor, mydb)** performs the operations of deleting the entire record of the Employee from the database with their specified Employee ID.

```
def deleteEmployeeRecord(mycursor, mydb):
    print("Enter the Employee Id to delete the
records")
    reg = int(input())
    mycursor.execute("DELETE FROM employees WHERE
```

```
emp_id = ?", (reg,))
    mydb.commit()
    print("The Record with Employee id = ", reg, " is
Deleted successfully")
```

The method **averageSalaryInDept(mycursor)** performs the functionality of Calculating and displaying the Department wise average salary of the company.

```
def printDeptAvg(t, dept, avg):
    table = PrettyTable([dept, avg])
    for i in t:
        table.add_row(i)
    print(table)
```

```
def averageSalaryInDept(mycursor):
    mycursor.execute("SELECT emp_dept AS Department,
AVG(salary) AS Avg_salary FROM employees GROUP BY
emp_dept")
    t = mycursor.fetchall()
    printDeptAvg(t, "Department", "Avg_salary")
```

The method **averageSalaryOfEmployees(mycursor)** performs the functionality of Calculating and displaying the average salary of the Employees in the company.

```
def printAverage(t, avg):
    table = PrettyTable([avg])
    for i in t:
        table.add_row(i)
    print(table)
```

```
def averageSalaryOfEmployees(mycursor):
    mycursor.execute("SELECT AVG(salary) AS avg_salary
FROM employees")
    t = mycursor.fetchall()
    printAverage(t, "Average Salary")
```

Dependencies:

There are two external libraries or modules used in this project :

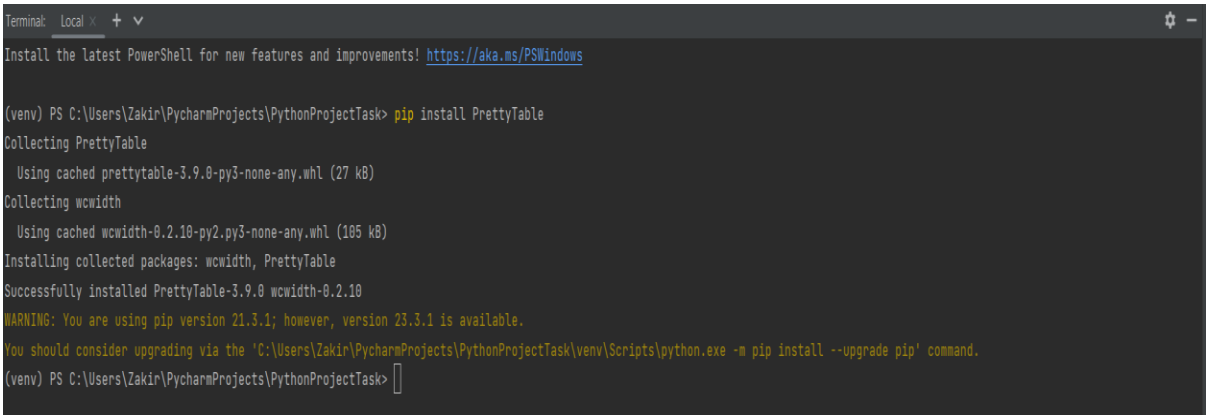
- SQLite
- PrettyTable

To import these two we have to include :

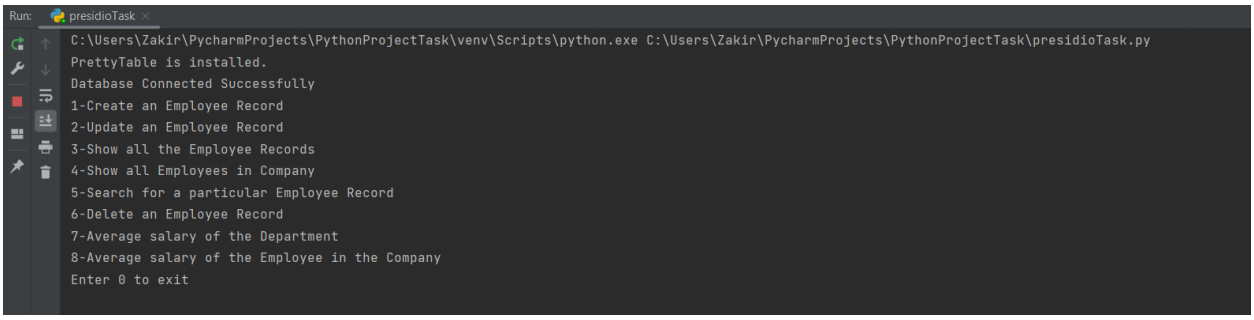
- import sqlite3
- import prettytable from PrettyTable

Sample Input/Output:

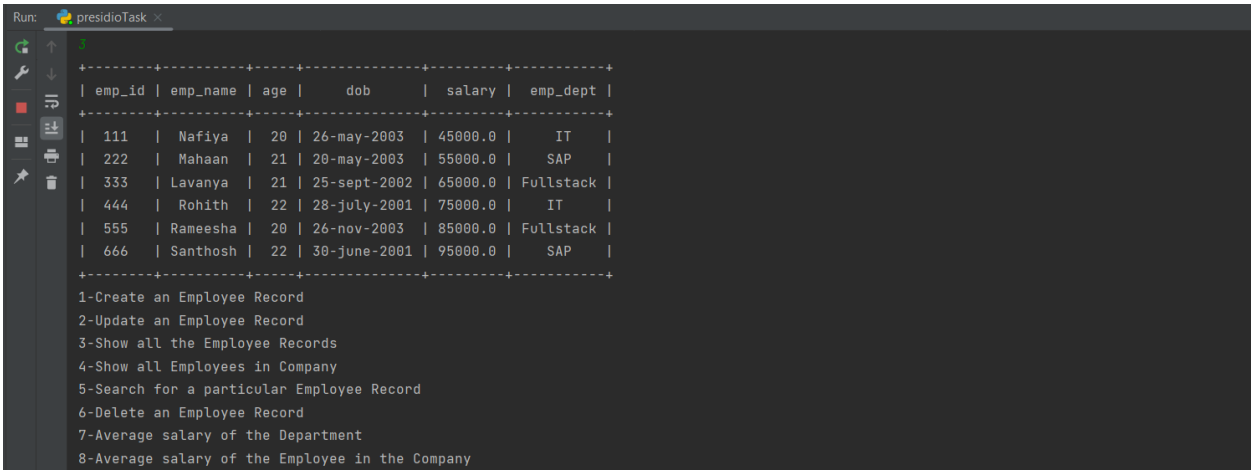
To check the proper functionalities of the code we have some sample input and output. First, install the PrettyTable using the pip install PrettyTable command in PyCharm terminal.



The landing page with all the displayed options for the user to select.

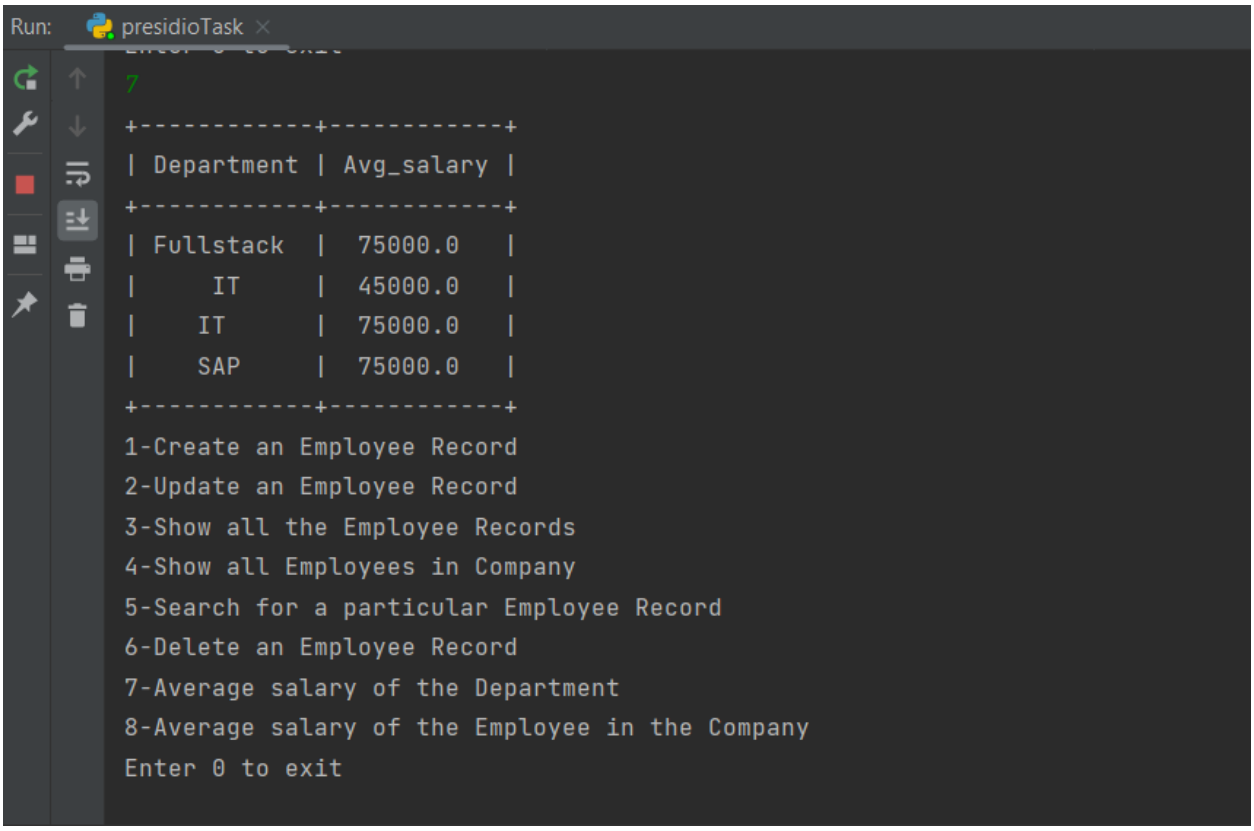


Created some employee records and had them displayed in a pretty formatted table using the PrettyTable library.

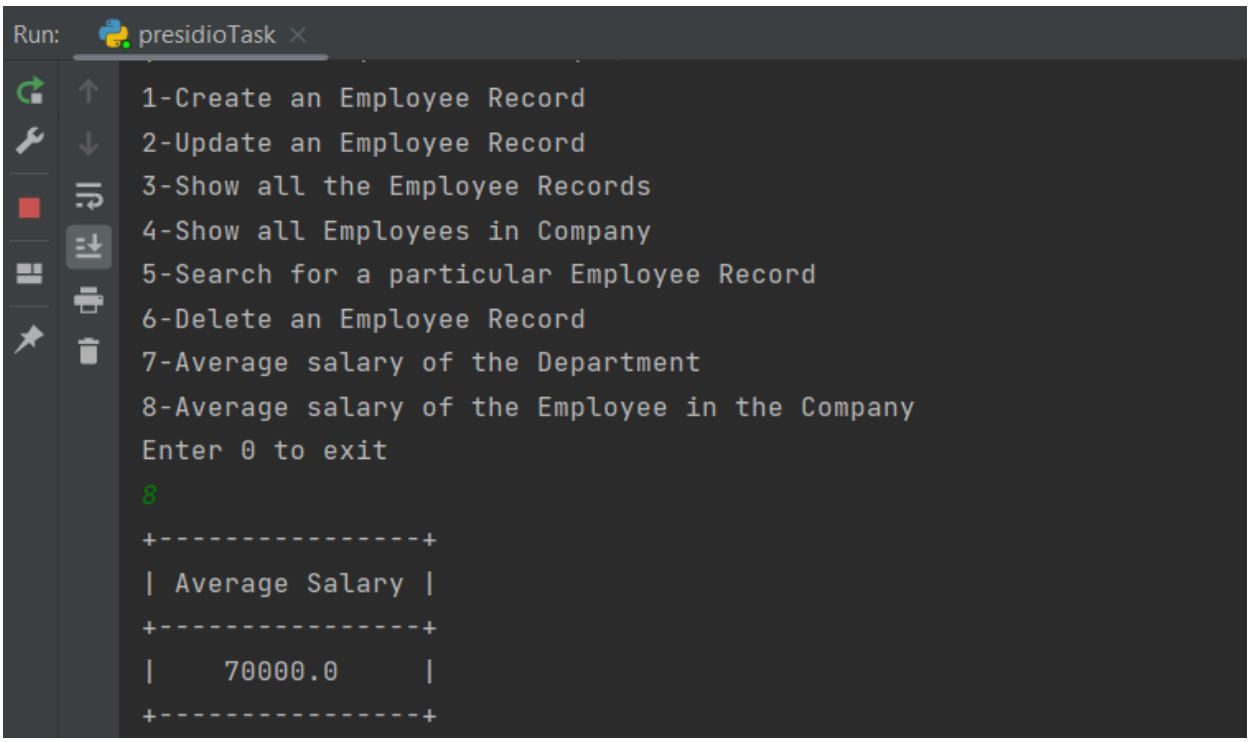


Showing all the employees that work in the company with their respective employeeID’s.

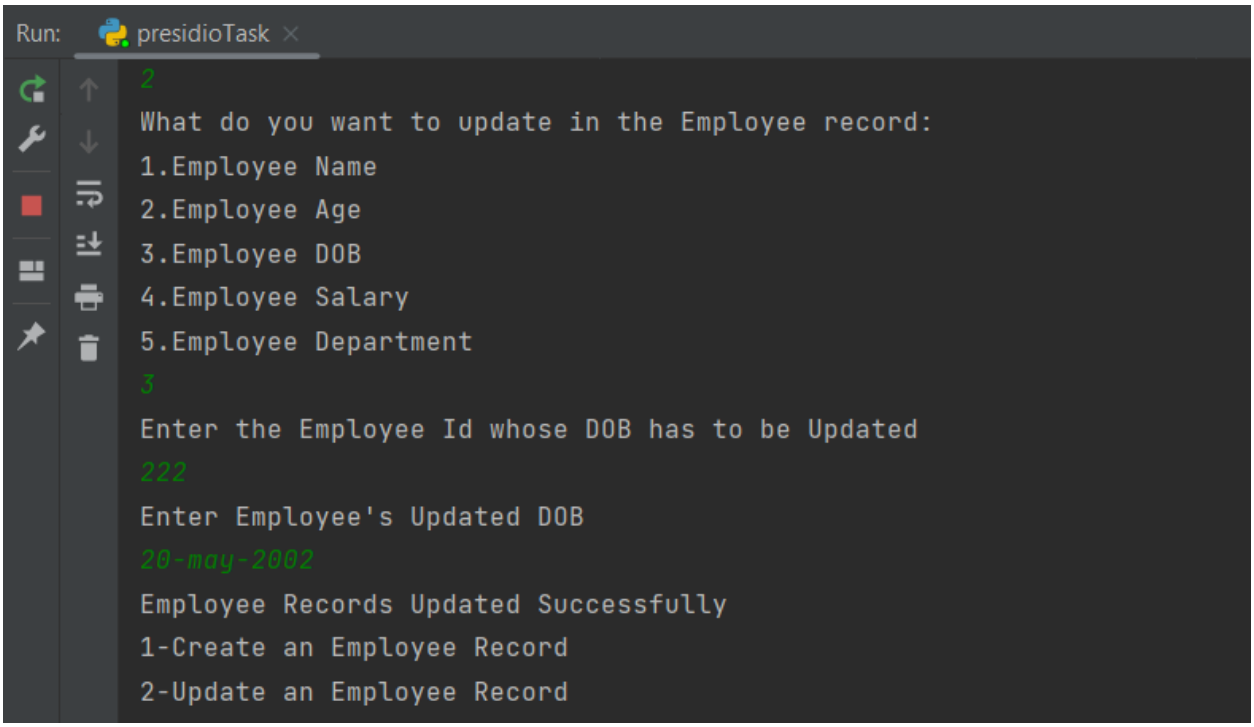
Displaying the Department-wise salary average.



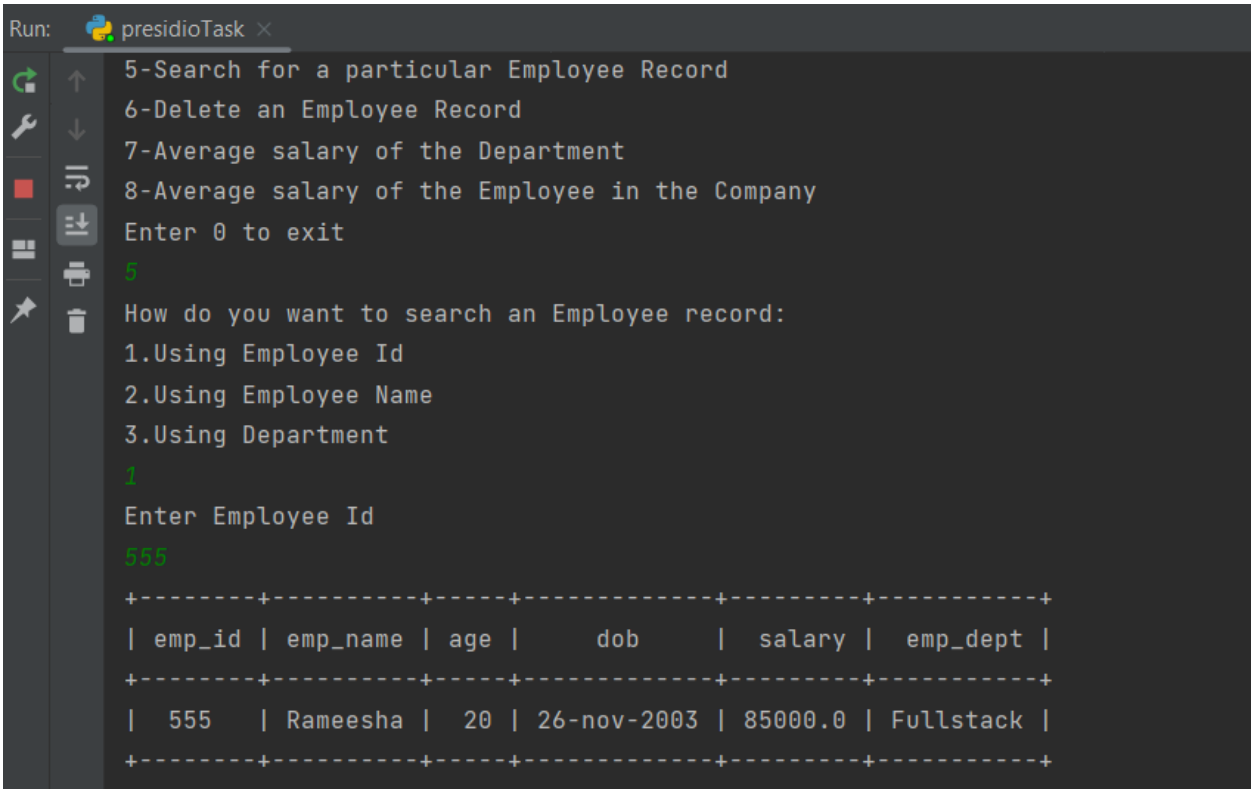
Displaying the overall average salary of the employees in the Company.



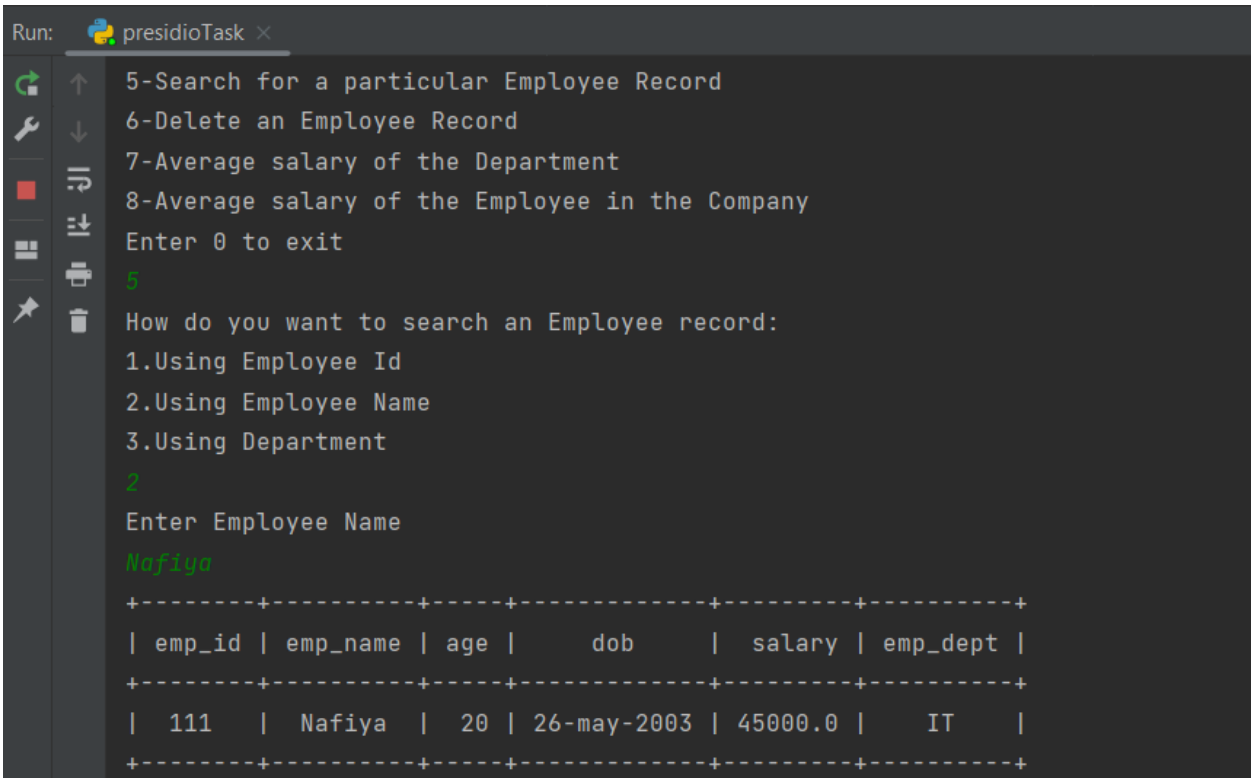
Updating the Employee's DOB using Employee ID as that is the primary key of the table.



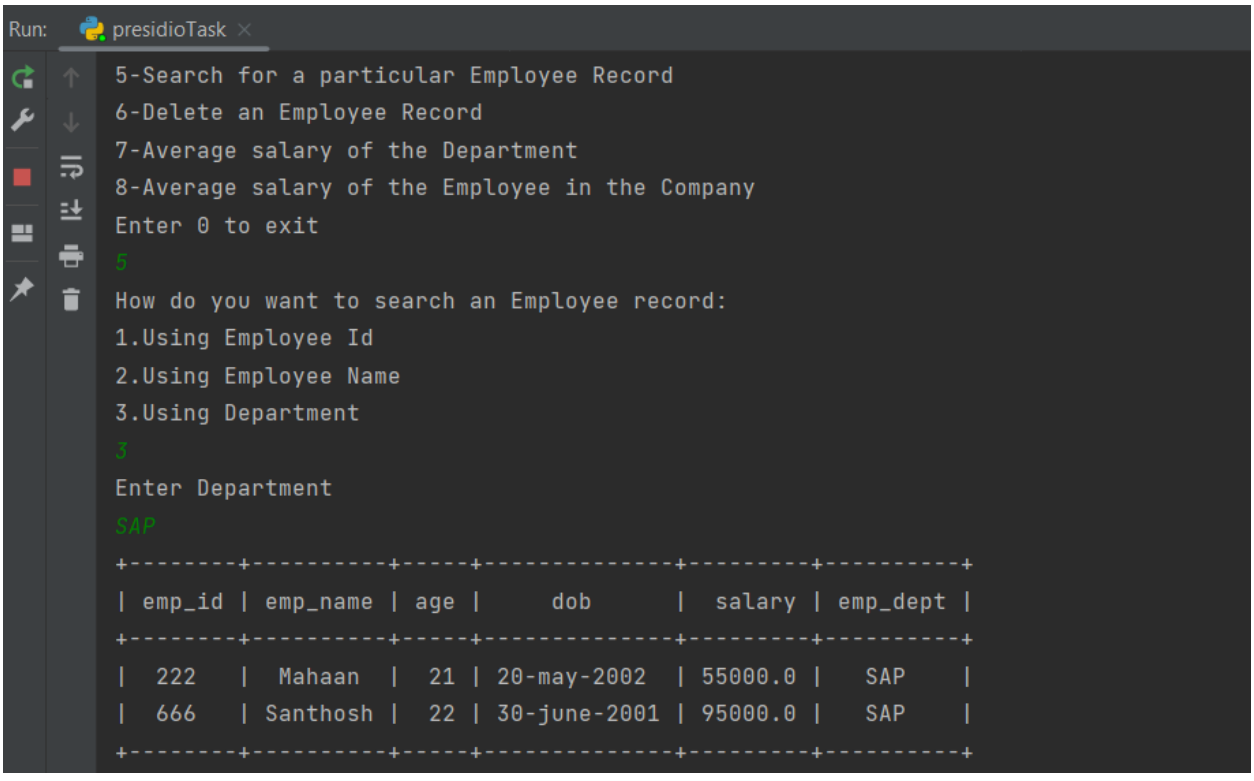
Search for a particular Employee record based on Employee ID:



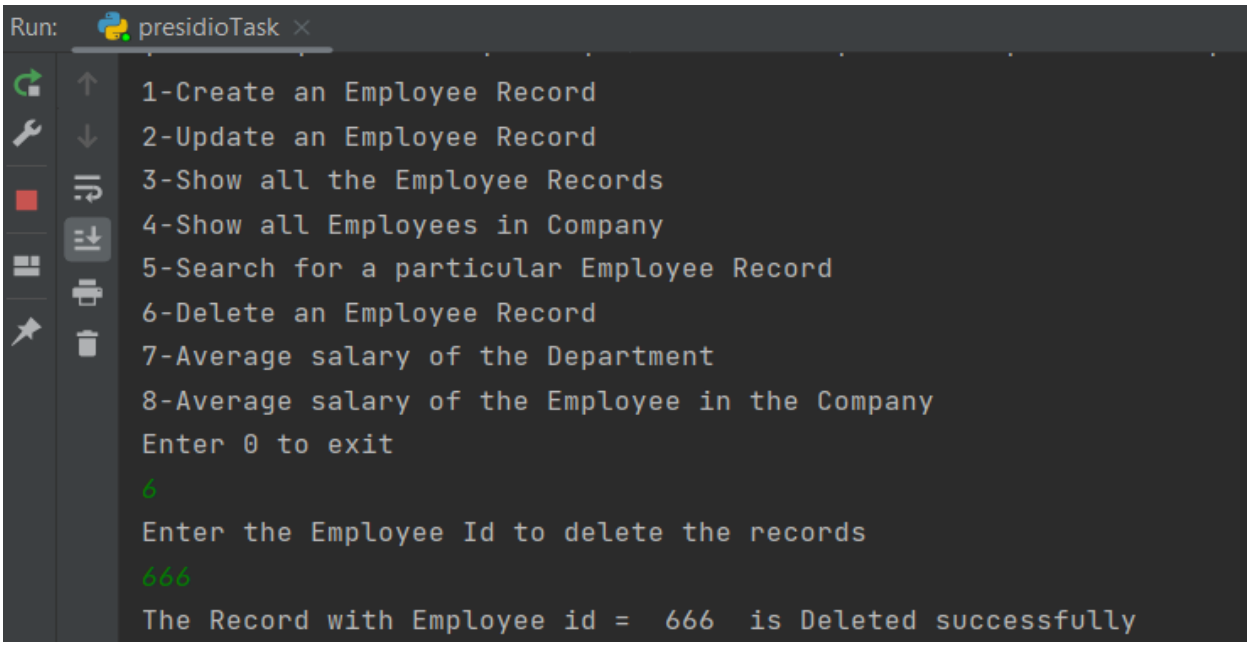
Search for a particular Employee record based on Employee name.



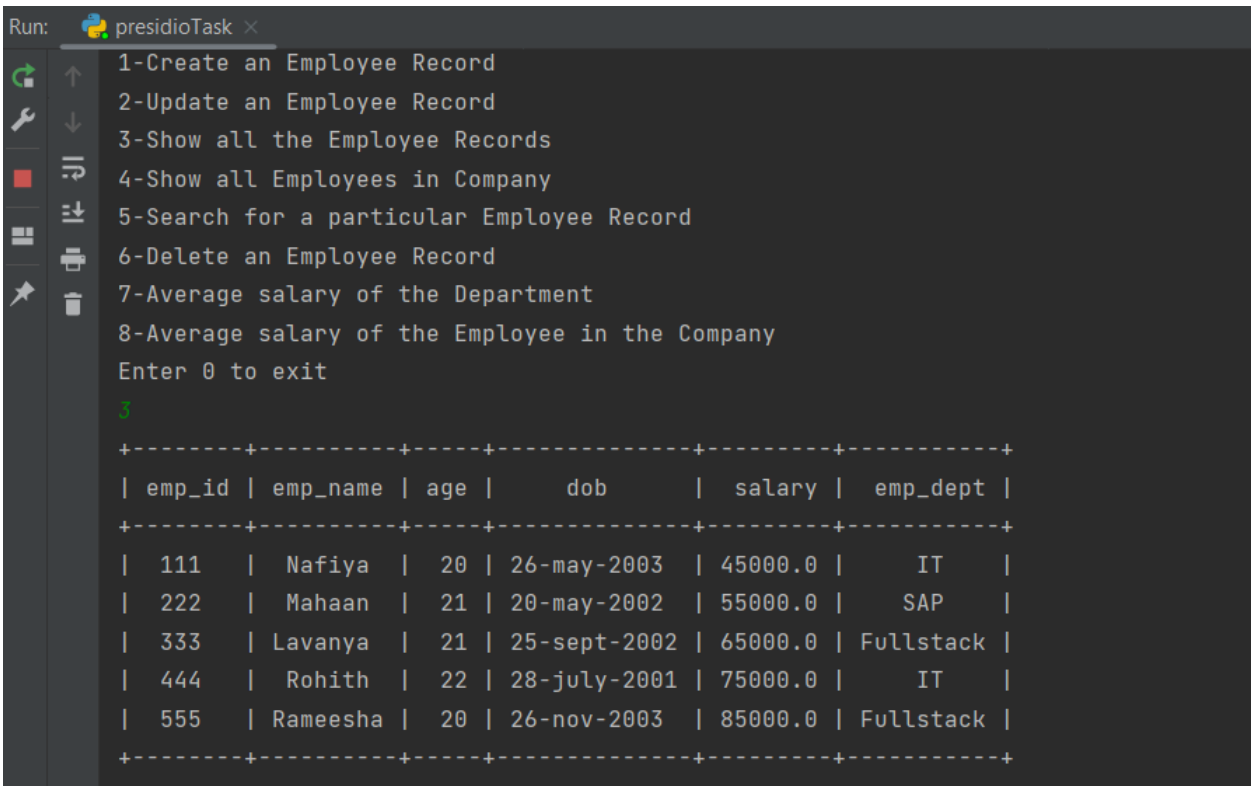
Search for a particular Employee record based on Department.



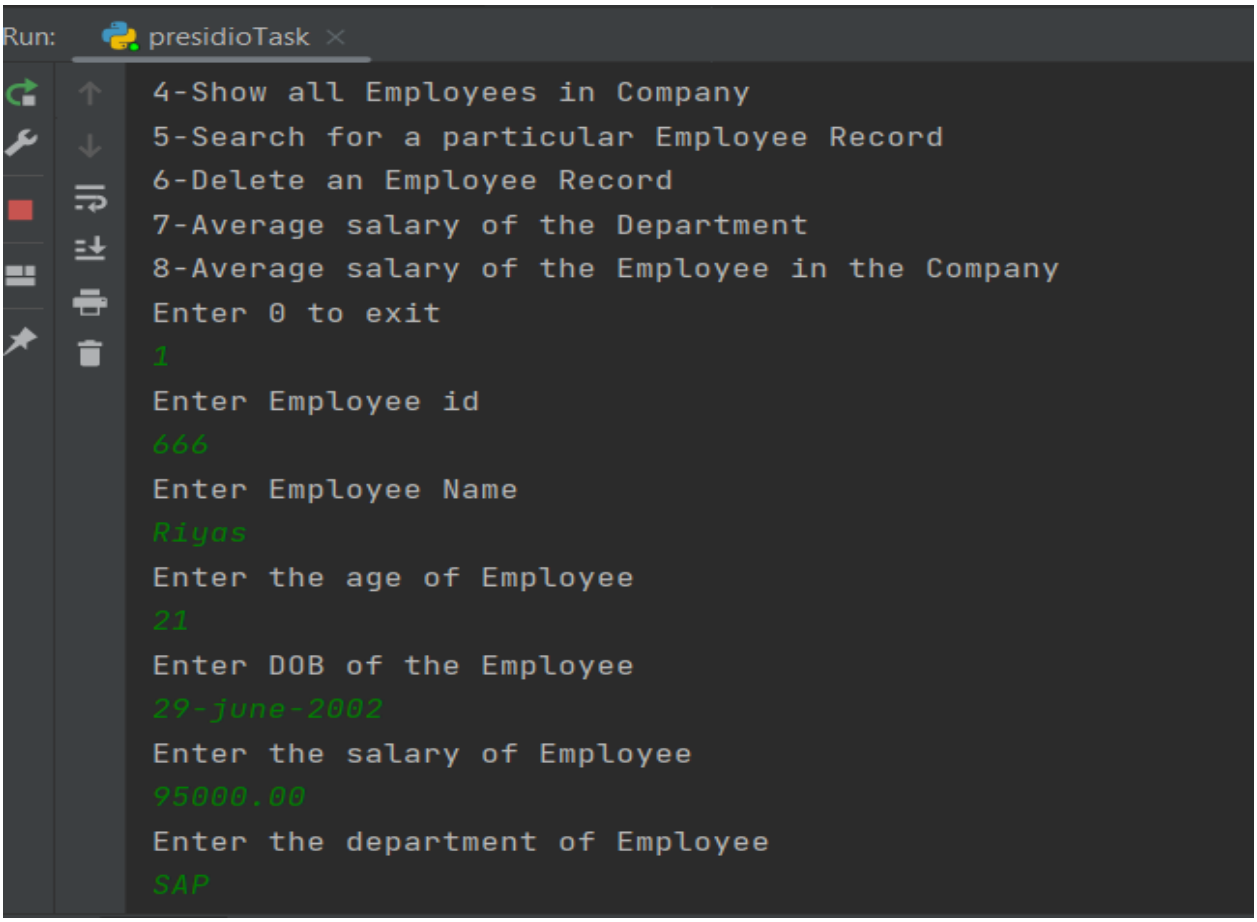
Deleting an Employee Record using their Employee ID.



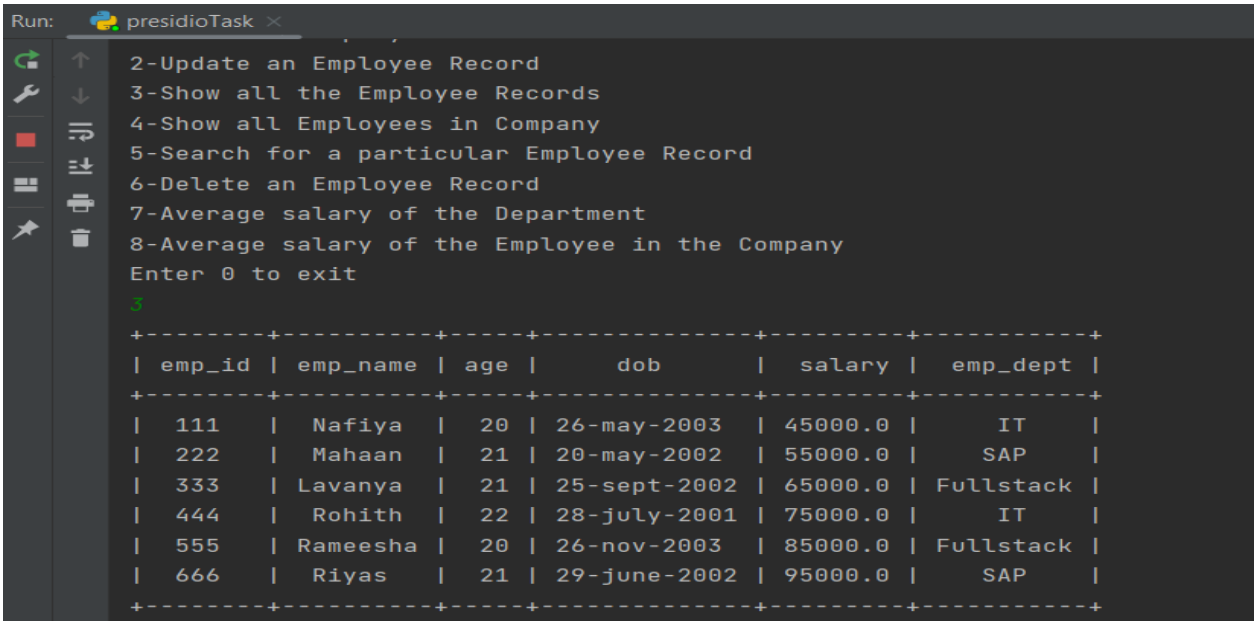
Display all records to check Deletion operation.



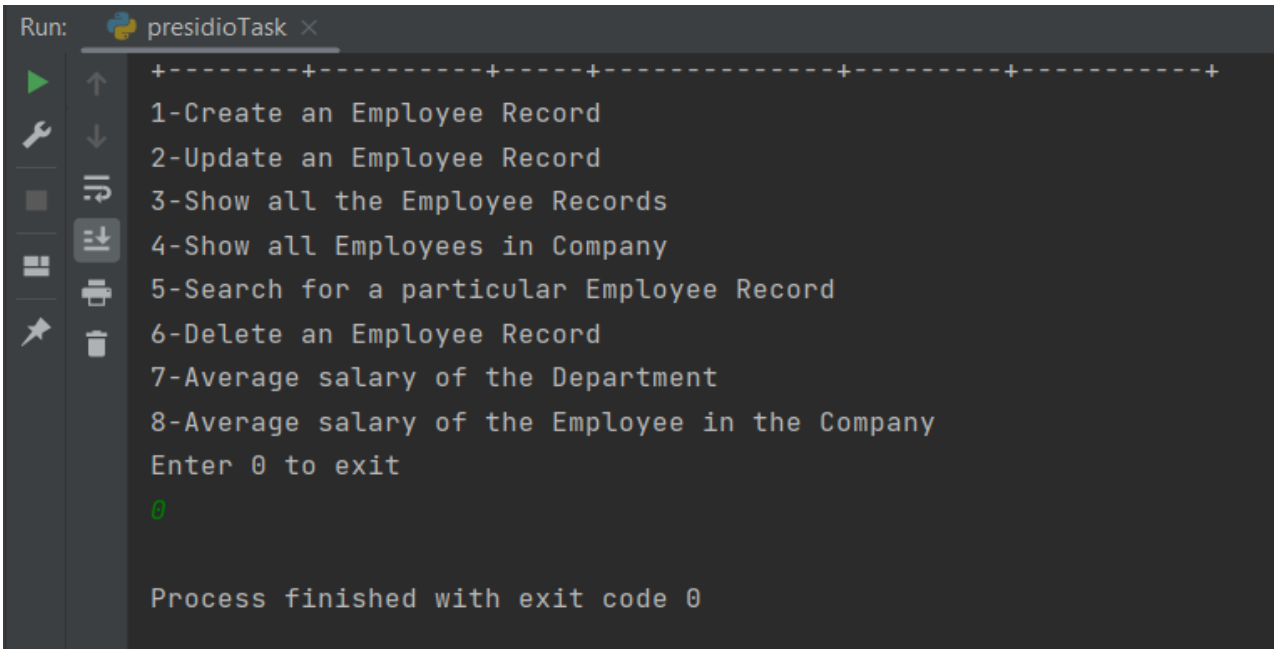
Add another new record to check the insertion operation.



Display the Final Records stored in the Employees Table.



Exit the console.



Conclusion:

Therefore the Console-based Project (Backend Application) named **Employee Record Management System** is accomplished successfully and all the operations and functionalities are verified using sample input and output.