

# **Task Report**

# INDEX

<b>1.Task 1.....</b>	<b>3</b>
<b>1.1 Objective.....</b>	<b>4</b>
<b>1.2 What the Algorithm Must Achieve.....</b>	<b>4</b>
<b>1.3 Input Data Details.....</b>	<b>4</b>
<b>1.4 Algorithm Design and Explanation.....</b>	<b>6</b>
<b>1.5 Flow chart.....</b>	<b>8</b>
<b>1.6 Sample Run with Test Data.....</b>	<b>9</b>
<b>2.Task 2.....</b>	<b>10</b>
<b>2 Objective.....</b>	<b>11</b>
<b>2.1 Features .....</b>	<b>11</b>
<b>2.2 Running from Source Code .....</b>	<b>11</b>
<b>2.3 Application Menu.....</b>	<b>12</b>
<b>2.4 Final Output.....</b>	<b>13</b>
<b>3. Task 3.....</b>	<b>21</b>
<b>3.1 Objective.....</b>	<b>22</b>
<b>3.2 Create Database.....</b>	<b>22</b>
<b>3.3 Tables.....</b>	<b>23</b>
<b>3.4 Data Insertion.....</b>	<b>25</b>
<b>3.5 Queries.....</b>	<b>26</b>
<b>3.6 Stored Procedures.....</b>	<b>28</b>
<b>3.7 Views.....</b>	<b>30</b>

## **Task 1**

# **Algorithm Development Task: Inventory Reordering System**

## 1.1 Objective

The goal of this task is to design an algorithm that ensures uninterrupted stock availability in a warehouse by determining an optimal reordering schedule. The algorithm aims to prevent stockouts by analyzing current inventory levels, forecasted demand, and reordering costs. By doing so, it ensures that inventory levels are sufficient to meet demand while minimizing the total cost associated with reordering, including batch size considerations. This solution is crucial for maintaining operational efficiency and cost-effectiveness in inventory management systems.

## 1.2 What the Algorithm Must Achieve

The algorithm should:

1. **Maintain Stock Levels:**

Ensure that the current stock, supplemented by reordered units, is always sufficient to meet forecasted demand. This prevents stockouts that can disrupt operations.

2. **Optimize Costs:**

Minimize total reordering costs by considering the cost per unit and batch size constraints.

3. **Balance Efficiency and Accuracy:**

Ensure that stock levels are neither too high (overstocking) nor too low (understocking), reducing storage costs and waste.

## 1.3 Input Data Details

The algorithm requires the following data for each item in inventory:

1. **Item Attributes:**

- **item\_id:** A unique identifier for the item.
- **current\_stock:** The quantity of the item currently in the warehouse.

- **forecasted\_demand:** The expected demand for the item in the upcoming period.
- **reorder\_cost\_per\_unit:** The cost to reorder one unit of the item (used to calculate the total cost of ordering).
- **batch\_size:** The minimum quantity that can be ordered for the item, ensuring practical ordering.

### Example Input Data

```
[
  { "item_id": 101, "current_stock": 50, "forecasted_demand": 100, "reorder_cost_per_unit":
2, "batch_size": 20 },
  { "item_id": 102, "current_stock": 30, "forecasted_demand": 80, "reorder_cost_per_unit":
1.5, "batch_size": 10 }
]
```

### Output Requirements

The algorithm produces a **reordering plan** that includes the following for each item:

#### 1. Item Identification:

- **item\_id:** The unique identifier of the item.

#### 2. Reorder Quantity:

- **units\_to\_order:** The number of units to reorder for each item, ensuring that stock meets forecasted demand.

### Example Output

For the input data provided above, the output would look like this:

```
[  
  { "item_id": 101, "units_to_order": 60 },  
  { "item_id": 102, "units_to_order": 50 }  
]
```

## 1.4. Algorithm Design and Explanation

Here's how to structure it:

### Steps of the Algorithm

#### 1. Input Validation

- Check if the input data is complete and valid (e.g., no missing or invalid values for `current_stock`, `forecasted_demand`, etc.).

#### 2. Reorder Quantity Calculation

- For each item in the inventory:
  1. Calculate the required units to meet forecasted demand:  
$$\text{Required Units} = \text{Forecasted Demand} - \text{Current Stock}$$
  2. Determine the number of units to reorder, considering the batch size:

- If Required Units is greater than zero, round up to the nearest multiple of the batch size:

$$\text{Units to Order} = \lceil \text{Required Units} / \text{Batch Size} \rceil * \text{Batch Size}$$

- Otherwise, set Units to Order to zero.

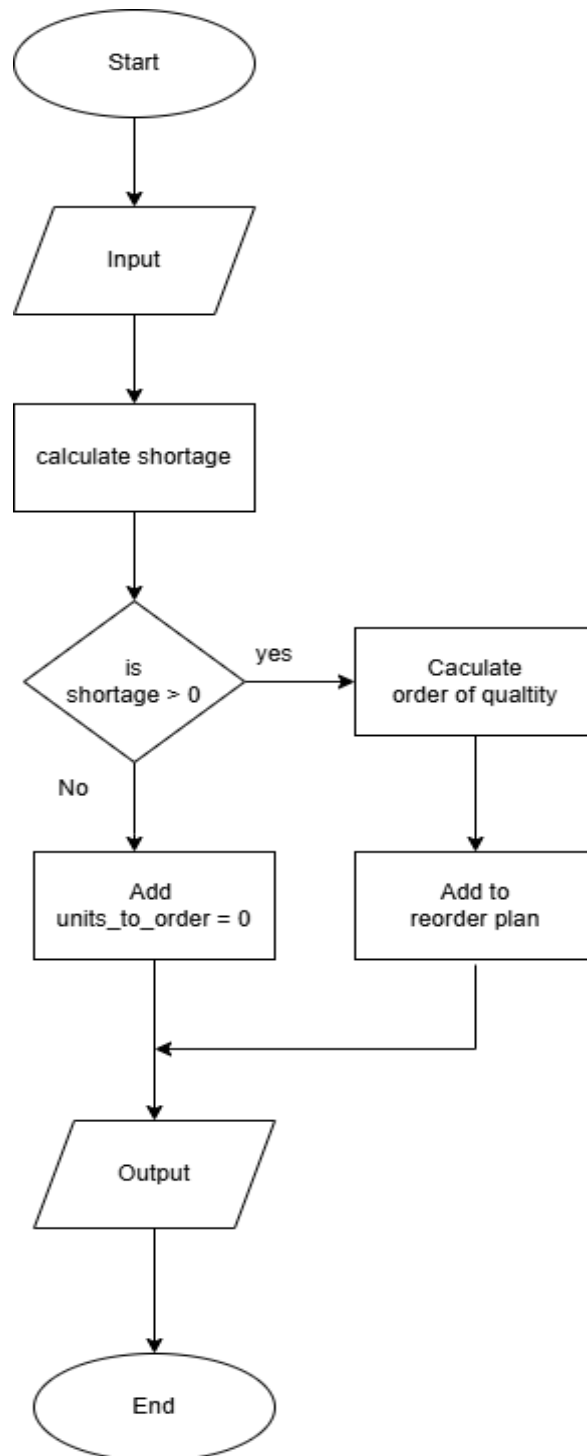
### 3. **Generate Output**

- Compile the reordering plan as a list of objects containing item\_id and units\_to\_order.

### 4. **Output Validation**

- Verify that the calculated reorder quantities meet all constraints:
  - Forecasted demand is covered.
  - No unnecessary overstocking occurs.

## 1.5 Flow chart





## **1.6.Sample Run with Test Data**

### **1. Input Data:**

Item ID: 101

Current Stock: 50

Forecasted Demand: 100

Reorder Cost Per Unit: 2

Batch Size: 20

Item ID: 102

Current Stock: 30

Forecasted Demand: 80

Reorder Cost Per Unit: 1.5

Batch Size: 10

### **Output**

Item ID: 101, Units to Order: 60

Item ID: 102, Units to Order: 50

## **Task 2**

# **Employee Payroll System**

## 2. Objective

The Employee Payroll System is a console-based application built using C# to manage employee records, calculate salaries, and store data using Object-Oriented Programming (OOP) principles. This application supports role-specific operations for Managers, Developers, and Interns.

### 2.1 Features

1. Add and manage employee details, including Name, ID, Role, Basic Pay, and Allowances.
2. Calculate and display employee salaries using the formula:

$$\text{Salary} = \text{Basic Pay} + \text{Allowances} - \text{Deductions}$$

3. Display all employee details.
4. Save and retrieve employee data using file storage.
5. Calculate and display total payroll for all employees.

### 2.2 Running from Source Code (Command Line Instructions)

#### 1. Open a Command Prompt or Terminal:

Navigate to the folder where the project is located using the cd command.

For example: **cd \path-to-project-folder**

#### 2. Build the Project:

Use the dotnet build command to compile the project:

**dotnet build**

### **3. Run the Application:**

Run the application using:

**dotnet run**

## **2.3 Application Menu**

Upon the application displays a menu-driven interface with the following options:

### **1. Add Employee:**

- Allows adding a new employee by entering details like Name, ID, Role, Basic Pay, and Allowances.

### **2. Display All Employees:**

- Displays a list of all employees along with their details.

### **3. Calculate Salary:**

- Calculates and displays the salary for a specific employee by their ID.

### **4. Calculate Total Payroll:**

- (Optional Feature) Calculates the total salary paid to all employees.

### **5. Save/Load Employees:**

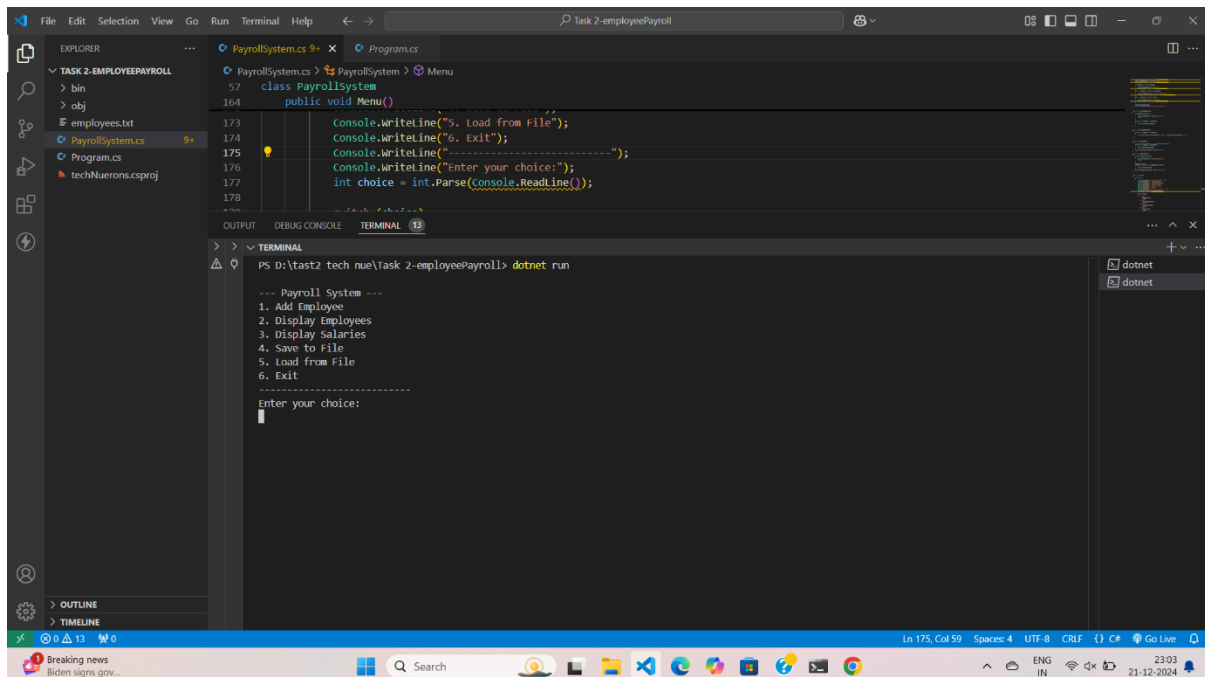
- (Optional Feature) Save employee data to a file or load data from a file.

### **6. Exit:**

- Exits the application.

## 2.4 Final Output

- **Run the Program:** When you run the program, the first thing you'll see is the main menu



The screenshot shows the Visual Studio IDE with the 'TASK 2-EMPLOYEEPAYROLL' project open. The 'PayrollSystem.cs' file is selected in the Explorer, showing the 'Menu' method. The terminal window displays the output of the program, which is the main menu of the Payroll System.

```
class PayrollSystem
{
    public void Menu()
    {
        Console.WriteLine("5. Load from File");
        Console.WriteLine("6. Exit");
        Console.WriteLine("-----");
        Console.WriteLine("Enter your choice:");
        int choice = int.Parse(Console.ReadLine());
    }
}
```

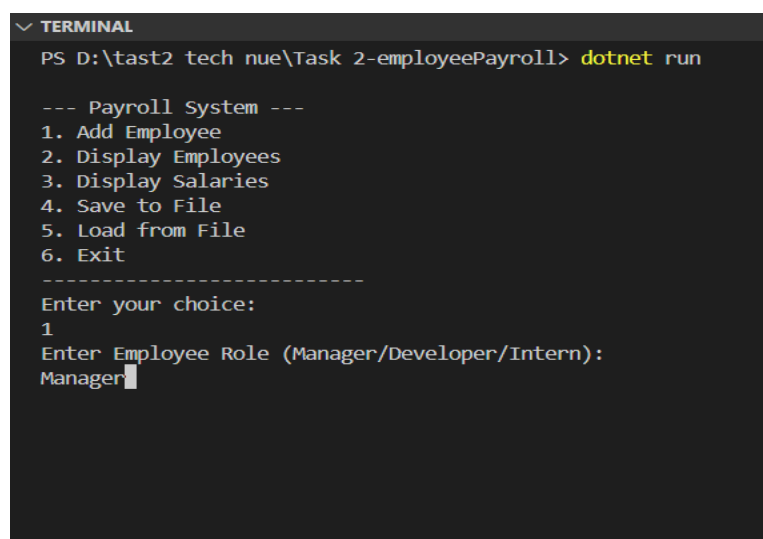
```
PS D:\task2 tech nue\Task 2-employeePayroll> dotnet run

--- Payroll System ---
1. Add Employee
2. Display Employees
3. Display Salaries
4. Save to File
5. Load from File
6. Exit
-----
Enter your choice:

```

- **Add an Employee**

If you select 1 (Add Employee), you'll be prompted to enter details for the new employee (ID, Name, Basic Pay, Allowances, etc.). Depending on the role you select (Manager, Developer, Intern)



The screenshot shows the terminal output after selecting '1. Add Employee'. The program prompts the user to enter the employee role (Manager/Developer/Intern).

```
PS D:\task2 tech nue\Task 2-employeePayroll> dotnet run

--- Payroll System ---
1. Add Employee
2. Display Employees
3. Display Salaries
4. Save to File
5. Load from File
6. Exit
-----
Enter your choice:
1
Enter Employee Role (Manager/Developer/Intern):
Manager

```

- **Enter Employee ID:** After selecting the role, the next prompt will ask for the employee's ID



```
▼ TERMINAL
PS D:\tast2 tech nue\Task 2-employeePayroll> dotnet run

--- Payroll System ---
1. Add Employee
2. Display Employees
3. Display Salaries
4. Save to File
5. Load from File
6. Exit
-----
Enter your choice:
1
Enter Employee Role (Manager/Developer/Intern):
Manager
Enter Employee ID:
501
```

- **Enter Employee Name:** Next, the program will ask for the employee's Name

```
▼ TERMINAL
PS D:\tast2 tech nue\Task 2-employeePayroll> dotnet run

--- Payroll System ---
1. Add Employee
2. Display Employees
3. Display Salaries
4. Save to File
5. Load from File
6. Exit
-----
Enter your choice:
1
Enter Employee Role (Manager/Developer/Intern):
Manager
Enter Employee ID:
501
Enter Employee Name:
Aswanth P
```

- **Enter Basic Pay:** The program will then prompt you to enter the employee's **Basic Pay**

```
✓ TERMINAL
PS D:\tast2 tech nue\Task 2-employeePayroll> dotnet run

--- Payroll System ---
1. Add Employee
2. Display Employees
3. Display Salaries
4. Save to File
5. Load from File
6. Exit
-----
Enter your choice:
1
Enter Employee Role (Manager/Developer/Intern):
Manager
Enter Employee ID:
501
Enter Employee Name:
Aswanth P
Enter Basic Pay:
45000
Enter Allowances:
█
```

- **Enter Allowances:** After entering the basic pay, the program will ask for the **Allowances**

```
✓ TERMINAL
PS D:\tast2 tech nue\Task 2-employeePayroll> dotnet run

--- Payroll System ---
1. Add Employee
2. Display Employees
3. Display Salaries
4. Save to File
5. Load from File
6. Exit
-----
Enter your choice:
1
Enter Employee Role (Manager/Developer/Intern):
Manager
Enter Employee ID:
501
Enter Employee Name:
Aswanth P
Enter Basic Pay:
45000
Enter Allowances:
4500█
```

- **Enter Deductions:** The next prompt will be for **Deductions**

```
▼ TERMINAL
PS D:\tast2 tech nue\Task 2-employeePayroll> dotnet run

--- Payroll System ---
1. Add Employee
2. Display Employees
3. Display Salaries
4. Save to File
5. Load from File
6. Exit
-----
Enter your choice:
1
Enter Employee Role (Manager/Developer/Intern):
Manager
Enter Employee ID:
501
Enter Employee Name:
Aswanth P
Enter Basic Pay:
45000
Enter Allowances:
4500
Enter Deductions:
3000
```

- **Additional Role-Specific Information:** Depending on the role of the employee (Manager, Developer, or Intern), you will be asked for additional information

- For **Managers**, you'll need to enter a **Bonus**

**Enter Bonus:**

- For **Developers**, you'll need to enter a **Project Allowance**

**Enter Project Allowance:**

- For **Interns**, you'll need to enter a **Stipend**

**Enter Stipend:**



```
--- Payroll System ---
1. Add Employee
2. Display Employees
3. Display Salaries
4. Save to File
5. Load from File
6. Exit
-----
Enter your choice:
1
Enter Employee Role (Manager/Developer/Intern):
Manager
Enter Employee ID:
501
Enter Employee Name:
Aswanth P
Enter Basic Pay:
45000
Enter Allowances:
4500
Enter Deductions:
3000
Enter Bonus:
2000
```

**Employee Added:** After all the details have been entered, the program will confirm that the employee has been successfully added

**Employee added successfully.**

```
6. Exit
-----
Enter your choice:
1
Enter Employee Role (Manager/Developer/Intern):
Manager
Enter Employee ID:
501
Enter Employee Name:
Aswanth P
Enter Basic Pay:
45000
Enter Allowances:
4500
Enter Deductions:
3000
Enter Bonus:
2000
Employee added successfully.
```

➤ **Display Employees**

Choose the "Display Employees" Option 2 and press Enter to choose the Display Employees option. The program will list all the employees added to the system along with their details.

```
Employee added successfully.

--- Payroll System ---
1. Add Employee
2. Display Employees
3. Display Salaries
4. Save to File
5. Load from File
6. Exit
-----
Enter your choice:
2
ID: 501, Name: Aswanth P, Role: , Basic Pay: 45000, Allowances: 4500, Deductions: 3000
```

➤ **Display Salaries**

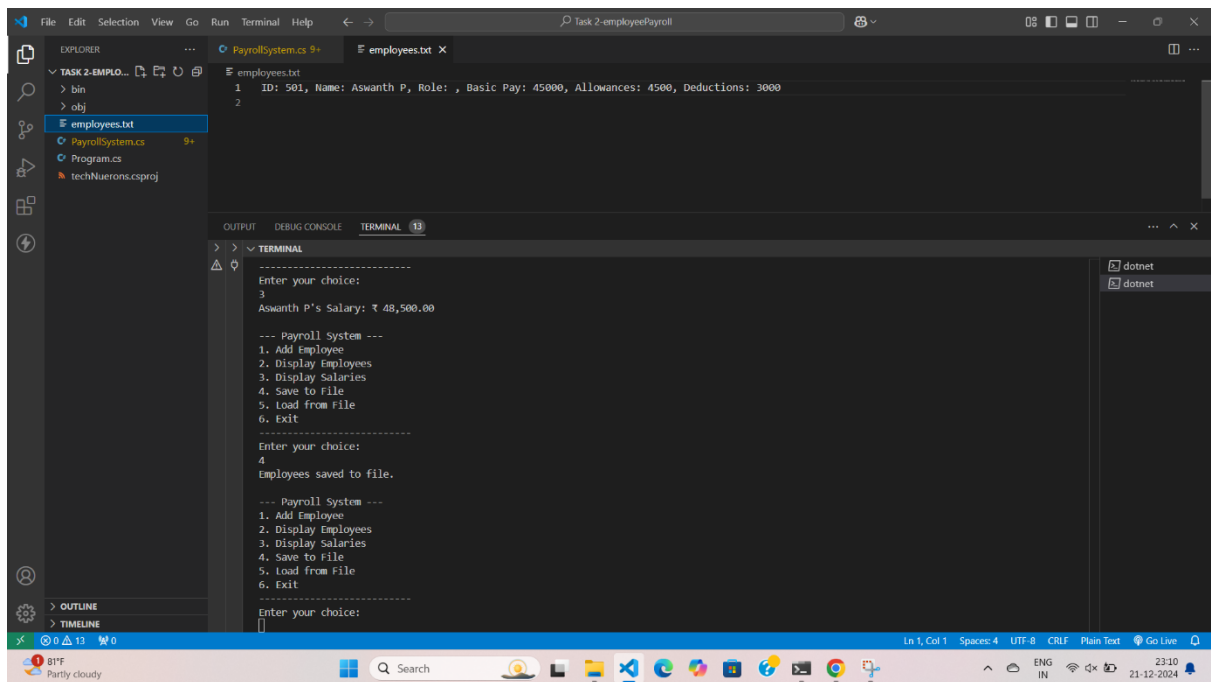
Choose the "Display Salaries" Option 3 and press Enter to choose the Display Salaries option. The program will iterate over all employees in the system and calculate their salary using the CalculateSalary() method, then display the salary for each employee in the following format:

```
--- Payroll System ---
1. Add Employee
2. Display Employees
3. Display Salaries
4. Save to File
5. Load from File
6. Exit
-----
Enter your choice:
3
Aswanth P's Salary: ₹ 48,500.00
```

### ➤ Save to File

Choose the **"Save to File" Option 4** and press **Enter** to choose the **Save to File** option. The program will save the details of all employees (ID, Name, Role, Basic Pay, Allowances, Deductions, Bonus, Project Allowance, Stipend, etc.) into a text file. The file will be named `employees.txt` and will be stored in the application's directory.

**Confirmation Message:** After saving the data, the program will confirm the action by printing a message:



```
1 ID: 501, Name: Aswanth P, Role: , Basic Pay: 45000, Allowances: 4500, Deductions: 3000
2
```

```
Enter your choice:
3
Aswanth P's Salary: ₹ 48,500.00

--- Payroll System ---
1. Add Employee
2. Display Employees
3. Display Salaries
4. Save to File
5. Load from File
6. Exit

Enter your choice:
4
Employees saved to file.

--- Payroll System ---
1. Add Employee
2. Display Employees
3. Display Salaries
4. Save to File
5. Load from File
6. Exit

Enter your choice:
```

### ➤ Load from File

Choose the **"Load from File" Option 5** and press **Enter** to choose the **Load from File** option. **Load Employee Data:** The program will attempt to read the `employees.txt` file and load the employee details into the employees list. It will display each employee's details as it reads them. **Confirmation Message:** After loading the data from the file, the program will print a message indicating that the employees were loaded:

```
File Edit Selection View Go Run Terminal Help
Task 2-employeePayroll
employees.txt
1 ID: 501, Name: Aswanth P, Role: , Basic Pay: 45000, Allowances: 4500, Deductions: 3000
2
PayrollSystem.cs 9+
Program.cs
techNuerons.csproj
employees.txt
--- Payroll System ---
1. Add Employee
2. Display Employees
3. Display Salaries
4. Save to File
5. Load from File
6. Exit
-----
Enter your choice:
4
Employees saved to file.
--- Payroll System ---
1. Add Employee
2. Display Employees
3. Display Salaries
4. Save to File
5. Load from File
6. Exit
-----
Enter your choice:
5
ID: 501, Name: Aswanth P, Role: , Basic Pay: 45000, Allowances: 4500, Deductions: 3000
Employees loaded from file.
Ln 1, Col 1 Spaces: 4 UTF-8 CRLF Plain Text Go Live
79°F Mostly clear
```

## ➤ Exit

Select Exit Option 6 and press Enter to choose the Exit option

```
File Edit Selection View Go Run Terminal Help
Task 2-employeePayroll
employees.txt
1 ID: 501, Name: Aswanth P, Role: , Basic Pay: 45000, Allowances: 4500, Deductions: 3000
2
PayrollSystem.cs 9+
Program.cs
techNuerons.csproj
employees.txt
--- Payroll System ---
1. Add Employee
2. Display Employees
3. Display Salaries
4. Save to File
5. Load from File
6. Exit
-----
Enter your choice:
4
Employees saved to file.
--- Payroll System ---
1. Add Employee
2. Display Employees
3. Display Salaries
4. Save to File
5. Load from File
6. Exit
-----
Enter your choice:
5
ID: 501, Name: Aswanth P, Role: , Basic Pay: 45000, Allowances: 4500, Deductions: 3000
Employees loaded from file.
--- Payroll System ---
1. Add Employee
2. Display Employees
3. Display Salaries
4. Save to File
5. Load from File
6. Exit
-----
Enter your choice:
6
PS D:\task2 tech nue\task 2-employeePayroll>
Ln 1, Col 1 Spaces: 4 UTF-8 CRLF Plain Text Go Live
81°F Partly cloudy
```

# **Task 3**

## **Employee Management System**

## 3.1 Objective

The Employee Management System is a relational database designed to efficiently manage employee records and salaries within an organization. This system aims to streamline various aspects of employee management, such as storing detailed employee information, calculating salaries, and providing insights into departmental performance. By utilizing well-structured tables, stored procedures, and views, the system ensures data integrity, simplifies complex operations like payroll calculation, and enhances reporting capabilities. Its robust design enables organizations to maintain accurate records, automate salary-related processes, and gain valuable insights for better decision-making.

The Employee Management System was developed and tested using **SQLyog**, a graphical interface for MySQL databases. SQLyog was utilized for creating the database schema, writing and executing SQL queries, stored procedures, and views, as well as managing data. Its user-friendly interface and advanced features, such as query building and performance analysis tools, made the development process efficient and streamlined.

## 3.2 Create Database

- **Creation**

Creates a new database named EmployeeManagement where all tables and data will be stored.

**CREATE DATABASE EmployeeManagement;**

- **Use Database**

Switches the active database to EmployeeManagement for executing further commands.

**USE EmployeeManagement;**

### 3.3 Tables

- **Departments Table**

Creates the Departments table to store information about departments within the organization.

```
CREATE TABLE Departments (  
    DepartmentID INT PRIMARY KEY,  
    DepartmentName VARCHAR(100) NOT NULL  
);
```

<input type="checkbox"/>	DepartmentID	DepartmentName
*	(NULL)	(NULL)

- **Employees Table**

Creates the Employees table to store employee details, including a foreign key linking to the Departments table.

```
CREATE TABLE Employees (  
    EmployeeID INT PRIMARY KEY,  
    Name VARCHAR(100) NOT NULL,  
    DepartmentID INT NOT NULL,  
    HireDate DATE NOT NULL,  
    FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID) );
```


<input type="checkbox"/>	EmployeeID	Name	DepartmentID	HireDate
*	(NULL)	(NULL)	(NULL)	(NULL)

- **Salaries Table**

Creates the Salaries table to store salary details for each employee, with a foreign key referencing the Employees table.

```
CREATE TABLE Salaries (  
    EmployeeID INT PRIMARY KEY,  
    BaseSalary DECIMAL(10, 2) NOT NULL,  
    Bonus DECIMAL(10, 2),  
    Deductions DECIMAL(10, 2),  
    FOREIGN KEY (EmployeeID) REFERENCES Employees(EmployeeID) )
```



<input type="checkbox"/>	EmployeeID 	BaseSalary	Bonus	Deductions
*	(NULL)	(NULL)	(NULL)	(NULL)

### 3.4 Data Insertion

- **Insert into Departments Table**

Adds example department records to the Departments table.

**INSERT INTO Departments (DepartmentID, DepartmentName)**

**VALUES (1, 'HR'), (2, 'Engineering'), (3, 'Finance');**

<input type="checkbox"/>	DepartmentID	DepartmentName
<input type="checkbox"/>	1	HR
<input type="checkbox"/>	2	Engineering
<input type="checkbox"/>	3	Finance
*	(NULL)	(NULL)

- **Insert into Employees Table**

Adds sample employee records to the Employees table.

**INSERT INTO Employees (EmployeeID, Name, DepartmentID, HireDate) VALUES (101, 'Aswanth', 2, '2020-01-15'), (102, 'Anu', 3, '2021-03-10'), (103, 'Athul', 1, '2019-08-25');**

<input type="checkbox"/>	EmployeeID	Name	DepartmentID	HireDate
<input type="checkbox"/>	101	Aswanth	2	2020-01-15
<input type="checkbox"/>	102	Anu	3	2021-03-10
<input type="checkbox"/>	103	Athul	1	2019-08-25
*	(NULL)	(NULL)	(NULL)	(NULL)

- **Insert into Salaries Table**

Adds salary details for employees into the Salaries table.

**INSERT INTO Salaries (EmployeeID, BaseSalary, Bonus, Deductions)**

**VALUES (101, 50000, 5000, 2000),**

**(102, 60000, 4000, 3000),**

**(103, 45000, 3000, 1000);**

<input type="checkbox"/>	EmployeeID	BaseSalary	Bonus	Deductions
<input type="checkbox"/>	101	50000.00	5000.00	2000.00
<input type="checkbox"/>	102	60000.00	4000.00	3000.00
<input type="checkbox"/>	103	45000.00	3000.00	1000.00

## 3.5 Queries

- **List Employees with Department Names**

Fetches employee details along with their department names by joining the Employees and Departments tables.

**SELECT e.EmployeeID, e.Name, d.DepartmentName FROM Employees e JOIN Departments d ON e.DepartmentID = d.DepartmentID;**

<input type="checkbox"/>	EmployeeID	Name	DepartmentName
<input type="checkbox"/>	103	Athul	HR
<input type="checkbox"/>	101	Aswanth	Engineering
<input type="checkbox"/>	102	Anu	Finance

- **Calculate Net Salary**

Calculates the net salary for each employee using the formula:

$$\text{Net Salary} = \text{BaseSalary} + \text{Bonus} - \text{Deductions}$$

**SELECT e.Name, (s.BaseSalary + s.Bonus - s.Deductions) AS NetSalary FROM Employees e JOIN Salaries s ON e.EmployeeID = s.EmployeeID;**

<input type="checkbox"/>	Name	NetSalary
<input type="checkbox"/>	Aswanth	53000.00
<input type="checkbox"/>	Anu	61000.00
<input type="checkbox"/>	Athul	47000.00

- **Find Department with Highest Average Salary**

Finds the department with the highest average salary by aggregating data from the Departments, Employees, and Salaries tables.

```
SELECT d.DepartmentName, AVG(s.BaseSalary + s.Bonus - s.Deductions) AS AvgSalary
FROM Departments d JOIN Employees e ON d.DepartmentID = e.DepartmentID JOIN
Salaries s ON e.EmployeeID = s.EmployeeID GROUP BY d.DepartmentName ORDER
BY AvgSalary DESC LIMIT 1;
```

<input type="checkbox"/> DepartmentName	AvgSalary
<input type="checkbox"/> Finance	61000.000000

### 3.6 Stored Procedures

- **Add Employee**

Creates a stored procedure to insert new employee details into the Employees table.

**DELIMITER //**

```
CREATE PROCEDURE AddEmployee ( IN empID INT, IN empName VARCHAR(100),
IN deptID INT, IN hireDate DATE ) BEGIN INSERT INTO Employees (EmployeeID,
Name, DepartmentID, HireDate) VALUES (104, fawas, 3, 2020-01-19); END//
```

**DELIMITER ;**

- **Update Salary**

Updates salary details for a given employee.

**DELIMITER //**

**CREATE PROCEDURE UpdateSalary (**

**IN empID INT,**

**IN newBaseSalary DECIMAL(10, 2),**

**IN newBonus DECIMAL(10, 2),**

**IN newDeductions DECIMAL(10, 2)**

**)**

**BEGIN**

**UPDATE Salaries**

**SET BaseSalary = newBaseSalary,**

**Bonus = newBonus,**

**Deductions = newDeductions**

**WHERE EmployeeID = empID;**

**END // DELIMITER ;**

- **Calculate Payroll**

Calculates the total payroll cost for all employees

**DELIMITER //**

**CREATE PROCEDURE CalculatePayroll ()**

**BEGIN**

**SELECT SUM(BaseSalary + Bonus - Deductions) AS TotalPayroll**

**FROM Salaries;**

**END //**

**DELIMITER ;**

## 3.7 Views

- **EmployeeSalaryView**

Provides a detailed view of employee salaries with department names and net salaries.

```
CREATE VIEW EmployeeSalaryView AS SELECT e.Name, d.DepartmentName,  
(s.BaseSalary + s.Bonus - s.Deductions) AS NetSalary FROM Employees e JOIN Salaries  
s ON e.EmployeeID = s.EmployeeID JOIN Departments d ON e.DepartmentID =  
d.DepartmentID;
```

<input type="checkbox"/>	Name	DepartmentName	NetSalary
<input type="checkbox"/>	Aswanth	Engineering	53000.00
<input type="checkbox"/>	Anu	Finance	61000.00
<input type="checkbox"/>	Athul	HR	47000.00
*	(NULL)	(NULL)	(NULL)

- **HighEarnerView**

Lists employees earning above a specified threshold.

```
CREATE VIEW HighEarnerView AS SELECT e.Name, (s.BaseSalary + s.Bonus -  
s.Deductions) AS NetSalary FROM Employees e JOIN Salaries s ON e.EmployeeID =  
s.EmployeeID WHERE (s.BaseSalary + s.Bonus - s.Deductions) > 50000;
```

<input type="checkbox"/>	Name	NetSalary
<input type="checkbox"/>	Aswanth	53000.00
<input type="checkbox"/>	Anu	61000.00
*	(NULL)	(NULL)