***EMPLOYEE MANAGEMENT SYSTEM***

***ASMA SAJJAD***

***2023-ag-9514***

***BSCS (MORNING)***

***4th SEMESTER (M3)***

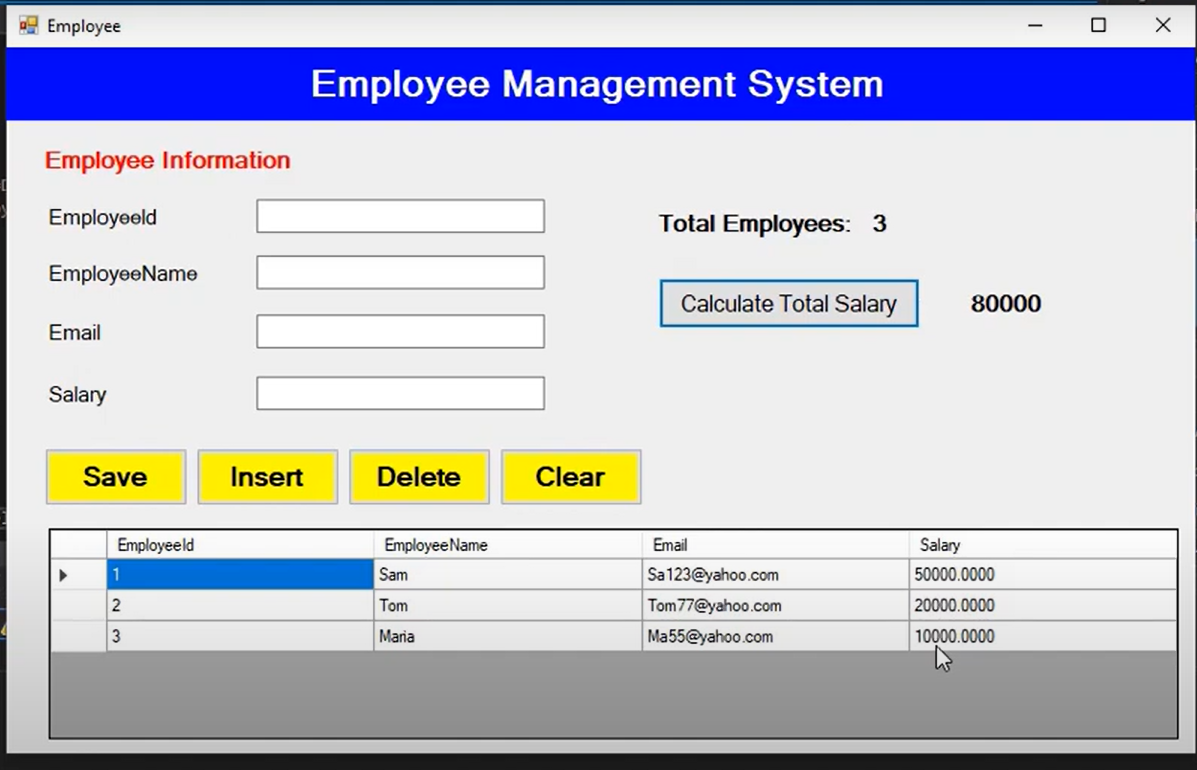
***VISUAL PROGRAMMING***

***SUBMITTED TO : PROF SAEED RASHEED***

***SUBMISSION DATE : 4 JUNE 2025***

******

***Screen shot of the interface***

******

This is a simple **Employee Management System** graphical user interface (GUI), likely built using a visual programming environment like Visual Studio (Windows Forms). Here's a short explanation of the components:

**Main Features:**

**Title: "Employee Management System"**

**Input Fields**:

EmployeeId: For entering the ID of the employee.

EmployeeName: For entering the name.

Email: For the employee's email.

Salary: For the salary amount.

**Right Side Panel**:

Total Employees: Displays the current count .

Calculate Total Salary: Button to compute the sum of all entered salaries.

**Buttons:**

Save: Likely saves current input data.

Insert: Adds a new record.

Delete: Removes a selected record.

Clear: Clears input fields.

**Bottom Section:**

Appears to be a data display area (probably a **DataGridView**) for listing employee records.

***Code***

using System;

using System.Data;

using System.Data.SqlClient;

using System.Windows.Forms;

public partial class EmployeeManagementSystem : Form

{

public EmployeeManagementSystem()

{

InitializeComponent();

}

private void SaveButton\_Click(object sender, EventArgs e)

{

try

{

using (SqlConnection con = new SqlConnection(@"Data Source=DESKTOP-HEEKBHE\SQLEXPRESS;Initial Catalog=EmployeeDB;Integrated Security=True"))

{

con.Open();

using (SqlCommand cnn = new SqlCommand("insert into Employee values(@employeeid,@employeename,@email,@salary)", con))

{

cnn.Parameters.AddWithValue("@EmployeeId", int.Parse(textBox1.Text));

cnn.Parameters.AddWithValue("@EmployeeName", textBox2.Text);

cnn.Parameters.AddWithValue("@Email", textBox3.Text);

cnn.Parameters.AddWithValue("@Salary", Convert.ToDecimal(textBox4.Text));

cnn.ExecuteNonQuery();

}

}

MessageBox.Show("Record Saved Successfully");

DisplayRecords();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

private void DisplayRecords()

{

try

{

using (SqlConnection con = new SqlConnection(@"Data Source=DESKTOP-HEEKBHE\SQLEXPRESS;Initial Catalog=EmployeeDB;Integrated Security=True"))

{

using (SqlCommand cnn = new SqlCommand("select \* from Employee", con))

{

SqlDataAdapter da = new SqlDataAdapter(cnn);

DataTable dt = new DataTable();

da.Fill(dt);

dataGridView1.DataSource = dt;

}

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

private void DeleteButton\_Click(object sender, EventArgs e)

{

try

{

using (SqlConnection con = new SqlConnection(@"Data Source=DESKTOP-HEEKBHE\SQLEXPRESS;Initial Catalog=EmployeeDB;Integrated Security=True"))

{

con.Open();

using (SqlCommand cnn = new SqlCommand("delete Employee where employeeid=@employeeid", con))

{

cnn.Parameters.AddWithValue("@EmployeeId", int.Parse(textBox1.Text));

cnn.ExecuteNonQuery();

}

}

MessageBox.Show("Record Deleted Successfully");

DisplayRecords();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

private void EmployeeManagementSystem\_Load(object sender, EventArgs e)

{

DisplayRecords();

try

{

using (SqlConnection con = new SqlConnection(@"Data Source=DESKTOP-HEEKBHE\SQLEXPRESS;Initial Catalog=EmployeeDB;Integrated Security=True"))

{

using (SqlCommand cnn = new SqlCommand("select \* from Employee", con))

{

SqlDataAdapter da = new SqlDataAdapter(cnn);

DataTable dt = new DataTable();

da.Fill(dt);

dataGridView1.DataSource = dt;

label8.Text = dt.Rows.Count.ToString();

}

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

}

***Explanation of Code***

This is a console-based Employee Management System using ADO.NET to interact with a SQL Server database. Below is a comprehensive documentation of the code, detailing its structure, functionality, and usage.

**Overview**

This application allows users to perform CRUD (Create, Read, Update, Delete) operations on an Employees table in a SQL Server database. It provides functionalities to add, view, delete employees, clear the entire table, and compute total employees and their combined salaries.

**Prerequisites**

**.NET Framework or .NET Core SDK:** Ensure that the appropriate .NET SDK is installed on your system.

**SQL Server**: A running instance of SQL Server where the Employees table will be created and managed.

**Connection String Configuration**: Update the connectionString variable with your SQL Server details:

static string connectionString = "Server=YourServerName;Database=YourDatabaseName;User Id=YourUsername;Password=YourPassword;";

**Code Structure and Functionality**

**1. Main Method**

The entry point of the application. It performs the following:

Calls CreateTableIfNotExists() to ensure the Employees table exists.

Displays a menu with options for various operations.

Processes user input to execute the corresponding functionality.

**2. CreateTableIfNotExists**()

Checks if the Employees table exists in the database. If not, it creates the table with the following schema:

**EmployeeID:** INT, Primary Key, Auto-incremented.

**EmployeeName**: NVARCHAR(100).

**Email:** NVARCHAR(100).

**Salary:** DECIMAL(10,2).

This is achieved using the SqlCommand.ExecuteNonQuery() method, which executes the SQL statement without returning any rows.

**3. SaveEmployee()**

Prompts the user to enter employee details: name, email, and salary. It then inserts this data into the Employees table using a parameterized SQL INSERT statement to prevent SQL injection attacks.

**4. DeleteEmployee**()

Prompts the user to enter the EmployeeID of the employee to be deleted. It then removes the corresponding record from the Employees table using a parameterized SQL DELETE statement.

**5. ClearTable()**

Deletes all records from the Employees table using the TRUNCATE TABLE SQL statement, effectively resetting the table.

**6. ViewEmployees()**

Retrieves and displays all records from the Employees table. It uses the SqlDataReader class to read data in a forward-only, read-only manner, which is efficient for large datasets.

**7. TotalEmployees()**

Calculates and displays the total number of employees by executing a SELECT COUNT(\*) SQL query. The result is obtained using the SqlCommand.ExecuteScalar() method, which retrieves a single value from the database.

**8. TotalSalary**()

Calculates and displays the sum of all employee salaries by executing a SELECT SUM(Salary) SQL query. It also uses the SqlCommand.ExecuteScalar() method to retrieve the result.

**Future Improvements**

Employee Management System could focus on enhancing user experience, security, and functionality. Transitioning from a console to a graphical user interface would improve accessibility, while implementing data validation and error handling would ensure data integrity and reliability. Adding features such as employee search, update capabilities, and data export options would further enhance usability. Additionally, incorporating security measures like data encryption and consistently using parameterized queries would protect sensitive information and prevent SQL injection attacks, making the system more robust and user-friendly**.**

**Conclusion**

This Employee Management System serves as a foundational application that demonstrates essential CRUD operations using ADO.NET with SQL Server. It effectively manages employee records and computes total employees and salaries. With the outlined future improvements, the system can evolve into a more robust, user-friendly, and feature-rich application tailored to meet more complex business needs. By continuously refining the user experience and enhancing functionality, this application can become an integral tool for efficient employee management in diverse environments.

By following the above documentation and best practices, you can effectively utilize and extend the Employee Management System to suit your specific requirements.