# Task 03

### Pseudo Code for Salary Component

1. Start salary calculation process

2. Get input values

- recordId = Input from text\_recordid field

- settingId = Input from txt\_settingid field

- salary = Input from text\_monthlysalaery field (monthly salary)

- OTRate = Input from text\_overtimerate field (overtime rate)

- allowances = Input from text\_allownce field

- taxRate = Input from text\_govermenttaxrate field (tax rate in percentage)

- salCyDateRange = Input from text\_salaryCycleDataRange field (salary cycle date range)

- absentDays = Input from text\_absendday field

- OTHours = Input from text\_overtimehours field (overtime hours)

3. Calculate salary components

- Base Pay Calculation:

- basePay = salary + allowances + (OTRate \* OTHours)

- Overtime Pay Calculation:

- OTPay = OTRate \* OTHours

- Tax Calculation:

- calculatedTax = (basePay \* taxRate) / 100

- No Pay Deduction (for absence):

- noPay = ((salary + allowances) / salCyDateRange) \* absentDays

- Gross Pay Calculation:

- grossPay = basePay - (noPay + calculatedTax)

4. If 'add' operation

- Insert the calculated values into the 'salary' table in the database

- Use SQL INSERT command with the values of recordId, settingId, basePay, OTPay, calculatedTax, noPay, grossPay

5. End of salary calculation process

### Salary Component

using GrifindoToysManagementSystem.System\_Par;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Data.SqlClient;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace GrifindoToysManagementSystem

{

public partial class Salary\_Compenent : Form

{

public Salary\_Compenent()

{

InitializeComponent();

DataGridViewSalary();

}

private void pictureBox3\_Click(object sender, EventArgs e)

{

Form1 f1 = new Form1();

f1.Show();

this.Hide();

}

Setting\_compenent sc1 = new Setting\_compenent();

private bool userInputFormatCheck()

{

bool invalidInput = false;

if (!sc1.numbersOnly(txt\_settingid.Text.Trim()))

{

errorProvider1.SetError(txt\_settingid, "Setting ID can only contain number");

invalidInput = true;

}

return invalidInput;

}

private void button\_add\_Click(object sender, EventArgs e)

{

errorProvider1.Dispose();

if (userInputFormatCheck())

{

MessageBox.Show("Invalid Input Formats", "Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return;

}

else

{

try

{

int recordId = int.Parse(text\_recordid.Text);

int settingId = int.Parse(txt\_settingid.Text);

decimal salary = decimal.Parse(text\_monthlysalaery.Text);

decimal OTRate = decimal.Parse(text\_overtimerate.Text);

decimal allowances = decimal.Parse(text\_allownce.Text);

decimal taxRate = decimal.Parse(text\_govermenttaxrate.Text);

int salCyDateRange = int.Parse(text\_salaryCycleDataRange.Text);

int absentDays = int.Parse(text\_absendday.Text);

decimal OTHours = decimal.Parse(text\_overtimehours.Text);

decimal basePay = salary + allowances + (OTRate \* OTHours);

decimal OTPay = OTRate \* OTHours;

decimal calculatedTax = (basePay \* taxRate) / 100;

decimal noPay = ((salary + allowances) / salCyDateRange) \* absentDays;

decimal grossPay = (basePay - (noPay + calculatedTax));

using (SqlConnection con = Database\_Connection.GetConnection())

{

con.Open();

SqlCommand cmd = new SqlCommand("INSERT INTO salary (recordId, settingId, basePay, OTPay, tax, noPay, grossPay) " +

"VALUES (@recordId, @settingId, @basePay, @OTPay, @tax, @noPay, @grossPay)", con);

cmd.Parameters.AddWithValue("@recordId", recordId);

cmd.Parameters.AddWithValue("@settingId", settingId);

cmd.Parameters.AddWithValue("@basePay", basePay);

cmd.Parameters.AddWithValue("@OTPay", OTPay);

cmd.Parameters.AddWithValue("@tax", calculatedTax);

cmd.Parameters.AddWithValue("@noPay", noPay);

cmd.Parameters.AddWithValue("@grossPay", grossPay);

cmd.ExecuteNonQuery();

DataGridViewSalary();

MessageBox.Show("Record saved successfully.", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

catch (Exception ex)

{

MessageBox.Show("An error occurred: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

private void DataGridViewSalary()

{

try

{

using (SqlConnection con = Database\_Connection.GetConnection())

{

con.Open();

SqlDataAdapter adapter = new SqlDataAdapter("SELECT \* FROM salary", con);

DataTable dt = new DataTable();

adapter.Fill(dt);

dataGridView\_salary.DataSource = dt;

}

}

catch (Exception ex)

{

MessageBox.Show("An error occurred while refreshing the DataGridView: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void button\_update\_Click(object sender, EventArgs e)

{

errorProvider1.Dispose();

if (userInputFormatCheck())

{

MessageBox.Show("Invalid Input Formats", "Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return;

}

else

{

try

{

int recordId = int.Parse(text\_recordid.Text);

int settingId = int.Parse(txt\_settingid.Text);

decimal salary = decimal.Parse(text\_monthlysalaery.Text);

decimal OTRate = decimal.Parse(text\_overtimerate.Text);

decimal allowances = decimal.Parse(text\_allownce.Text);

decimal taxRate = decimal.Parse(text\_govermenttaxrate.Text);

int salCyDateRange = int.Parse(text\_salaryCycleDataRange.Text);

int absentDays = int.Parse(text\_absendday.Text);

decimal OTHours = decimal.Parse(text\_overtimehours.Text);

decimal basePay = salary + allowances + (OTRate \* OTHours);

decimal OTPay = OTRate \* OTHours;

decimal calculatedTax = (basePay \* taxRate) / 100;

decimal noPay = ((salary + allowances) / salCyDateRange) \* absentDays;

decimal grossPay = (basePay - (noPay + calculatedTax));

using (SqlConnection con = Database\_Connection.GetConnection())

{

con.Open();

SqlCommand cmd = new SqlCommand("UPDATE salary SET settingId = @settingId, basePay = @basePay, OTPay = @OTPay, " +

"tax = @tax, noPay = @noPay, grossPay = @grossPay WHERE recordId = @recordId", con);

cmd.Parameters.AddWithValue("@recordId", recordId);

cmd.Parameters.AddWithValue("@settingId", settingId);

cmd.Parameters.AddWithValue("@basePay", basePay);

cmd.Parameters.AddWithValue("@OTPay", OTPay);

cmd.Parameters.AddWithValue("@tax", calculatedTax);

cmd.Parameters.AddWithValue("@noPay", noPay);

cmd.Parameters.AddWithValue("@grossPay", grossPay);

cmd.ExecuteNonQuery();

DataGridViewSalary();

MessageBox.Show("Record updated successfully.", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

catch (Exception ex)

{

MessageBox.Show("An error occurred: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

private void button\_clear\_Click(object sender, EventArgs e)

{

text\_recordid.Clear();

txt\_settingid.Clear();

text\_employeeid.Clear();

text\_salaryCycleDataRange.Clear();

text\_monthlysalaery.Clear();

text\_overtimerate.Clear();

text\_allownce.Clear();

text\_absendday.Clear();

text\_leavestaken.Clear();

text\_overtimehours.Clear();

text\_govermenttaxrate.Clear();

}

private void button\_delete\_Click(object sender, EventArgs e)

{

try

{

int recordId = int.Parse(text\_recordid.Text);

using (SqlConnection con = Database\_Connection.GetConnection())

{

con.Open();

SqlCommand cmd = new SqlCommand("DELETE FROM salary WHERE recordId = @recordId", con);

cmd.Parameters.AddWithValue("@recordId", recordId);

cmd.ExecuteNonQuery();

DataGridViewSalary();

MessageBox.Show("Record deleted successfully.", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

catch (Exception ex)

{

MessageBox.Show("An error occurred: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void picture\_search\_Click(object sender, EventArgs e)

{

try

{

if (!int.TryParse(txt\_search.Text, out int recIdToSearch))

{

MessageBox.Show("Please enter a valid Record ID for searching.", "Invalid Record ID", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return;

}

using (SqlConnection con = Database\_Connection.GetConnection())

{

con.Open();

SqlCommand searchCmd = new SqlCommand("SELECT \* FROM salary WHERE recordId = @recordId", con);

searchCmd.Parameters.AddWithValue("@recordId", recIdToSearch);

SqlDataAdapter adapter = new SqlDataAdapter(searchCmd);

DataTable dt = new DataTable();

adapter.Fill(dt);

if (dt.Rows.Count > 0)

{

dataGridView\_salary.DataSource = dt;

}

else

{

MessageBox.Show("No record found with the provided Record ID.", "Record Not Found", MessageBoxButtons.OK, MessageBoxIcon.Warning);

dataGridView\_salary.DataSource = null;

}

}

}

catch (Exception ex)

{

MessageBox.Show("An error occurred: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void btn\_new\_Click(object sender, EventArgs e)

{

try

{

using (SqlConnection con = Database\_Connection.GetConnection())

{

con.Open();

SqlCommand latestRecordIdCmd = new SqlCommand("SELECT MAX(recordId) FROM salary", con);

object result = latestRecordIdCmd.ExecuteScalar();

int latestRecordId = result != null && result != DBNull.Value ? (int)result : 0;

int newRecordId = latestRecordId + 1;

text\_recordid.Text = newRecordId.ToString();

txt\_settingid.Clear();

text\_employeeid.Clear();

text\_absendday.Clear();

text\_leavestaken.Clear();

text\_overtimehours.Clear();

text\_govermenttaxrate.Clear();

text\_allownce.Clear();

text\_salaryCycleDataRange.Clear();

text\_overtimerate.Clear();

text\_monthlysalaery.Clear();

}

}

catch (Exception ex)

{

MessageBox.Show("An error occurred: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void button1\_search\_Click(object sender, EventArgs e)

{

try

{

if (!int.TryParse(txt\_settingid.Text, out int settingIdToSearch))

{

MessageBox.Show("Please enter a valid Setting ID for searching.", "Invalid Setting ID", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return;

}

using (SqlConnection con = Database\_Connection.GetConnection())

{

con.Open();

SqlCommand searchCmd = new SqlCommand("SELECT \* FROM setting WHERE settingId = @settingId", con);

searchCmd.Parameters.AddWithValue("@settingId", settingIdToSearch);

SqlDataReader reader = searchCmd.ExecuteReader();

if (reader.Read())

{

text\_absendday.Text = reader["absentDays"].ToString();

text\_leavestaken.Text = reader["leavesTaken"].ToString();

text\_overtimehours.Text = reader["OTHours"].ToString();

text\_govermenttaxrate.Text = reader["taxRate"].ToString();

text\_salaryCycleDataRange.Text = reader["sal\_cy\_date\_range"].ToString();

text\_employeeid.Text = reader["empId"].ToString();

}

else

{

MessageBox.Show("No record found with the provided Setting ID.", "Record Not Found", MessageBoxButtons.OK, MessageBoxIcon.Warning);

text\_absendday.Clear();

text\_leavestaken.Clear();

text\_overtimehours.Clear();

text\_govermenttaxrate.Clear();

text\_salaryCycleDataRange.Clear();

text\_employeeid.Clear();

}

reader.Close();

}

}

catch (Exception ex)

{

MessageBox.Show("An error occurred: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void button\_2search\_Click(object sender, EventArgs e)

{

try

{

int empIdToSearch = 0;

if (!int.TryParse(text\_employeeid.Text, out empIdToSearch))

{

MessageBox.Show("Please enter a valid Employee ID for searching.", "Invalid Employee ID", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return;

}

using (SqlConnection con = Database\_Connection.GetConnection())

{

con.Open();

SqlCommand empSearchCmd = new SqlCommand("SELECT \* FROM employeeData WHERE empId = @empId", con);

empSearchCmd.Parameters.AddWithValue("@empId", empIdToSearch);

SqlDataReader empReader = empSearchCmd.ExecuteReader();

if (empReader.Read())

{

text\_monthlysalaery.Text = empReader["salary"].ToString();

text\_overtimerate.Text = empReader["Overtime\_Rate"].ToString();

text\_allownce.Text = empReader["allowances"].ToString();

text\_employeeid.Text = empReader["empId"].ToString();

empReader.Close();

}

else

{

MessageBox.Show("No record found with the provided Employee ID.", "Record Not Found", MessageBoxButtons.OK, MessageBoxIcon.Warning);

text\_monthlysalaery.Clear();

text\_overtimerate.Clear();

text\_allownce.Clear();

}

}

}

catch (Exception ex)

{

MessageBox.Show("An error occurred: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void Salary\_Compenent\_Load(object sender, EventArgs e)

{

}

}

}

## Analyze the features of an Integrated Development Environment (IDE)

An application that helps with application development is called an integrated development environment (IDE). All of a programmer's needs may be met by using an integrated development environment (IDE). Typically, an integrated development environment (IDE) will include a debugger, build automation tools, a compiler, and a code editor. However, IDEs like Eclipse, Komodo, and NetBeans exist for more than one language. Python, PHP, Ruby, Javascript, C++, C#, and more... The IDE is compatible with these languages. One or more compatible programmes may include an integrated development system (IDE), or the IDE itself may function as a separate application.

### Debugger

By monitoring the program's actions while it's running, the debugger may assist identify issues. All of the Visual Studio languages and their corresponding libraries are compatible with the debugger. Using the debugger, we can inspect and modify all variables, see all registers, see all instructions generated from source code, and view all memory space used by the application. We can even disrupt the program's execution to do this.

### Build automation tools

By automating tasks that would normally need human intervention, these build automation solutions greatly enhance efficiency. Things like testing and compilation might fall within this category. When a programme contains thousands of lines of code, these build automation tools become invaluable.

### Compiler

With the aid of this compiler tool, source code written in a language that humans can understand and write becomes code that computers can execute.

### Source code editor

Designed for the express purpose of modifying computer programmes' source code, the source code editor is a text editor programme. The programme might exist alone or be a part of a larger web browser or integrated development environment (IDE). Since writing and editing source code is a vital part of each programmer's work, source code editors are also an essential tool for programmers.

### Display the error Message



### Display the error list

### 

### Source code editor

Designed for the express purpose of modifying computer programmes' source code, the source code editor is a text editor programme. As a source code editor, Visual Studio Code lets you change the current document's language, the code page where it is stored, and the character that marks line breaks. While the programme is running, it may also make changes to the source code to fix errors. Complete code must be written by hand, increasing the likelihood of mistakes, in the absence of an integrated development environment (IDE).

### Toolbox

Visual Studio is what comes with it. We may add controls and other objects to Visual Studio projects by displaying their icons in the toolbox. Docking and pinning the Toolbox open or setting it to Auto Hide are also options. may easily transfer toolbox icons to a design view or paste them into a code editor. The core code to build a Toolbox item instance is also added to the current project file by the action. Conventional text editors lack the capabilities of integrated development environments (IDEs), making them useless when it comes to building graphical user interfaces.

### Build in compact SQL server

Also can construct databases right in Visual Studio using the built-in compact Structured Query Language version. No need to use a separate database programme and then export the results to Visual Studio. Because of this, the program's complexity decreases and the effort of the programmer is reduced. Additionally, most programmes need a database to store information about the program's functions.

# Task 04

## 4.1 Design and GUI system for the scenario

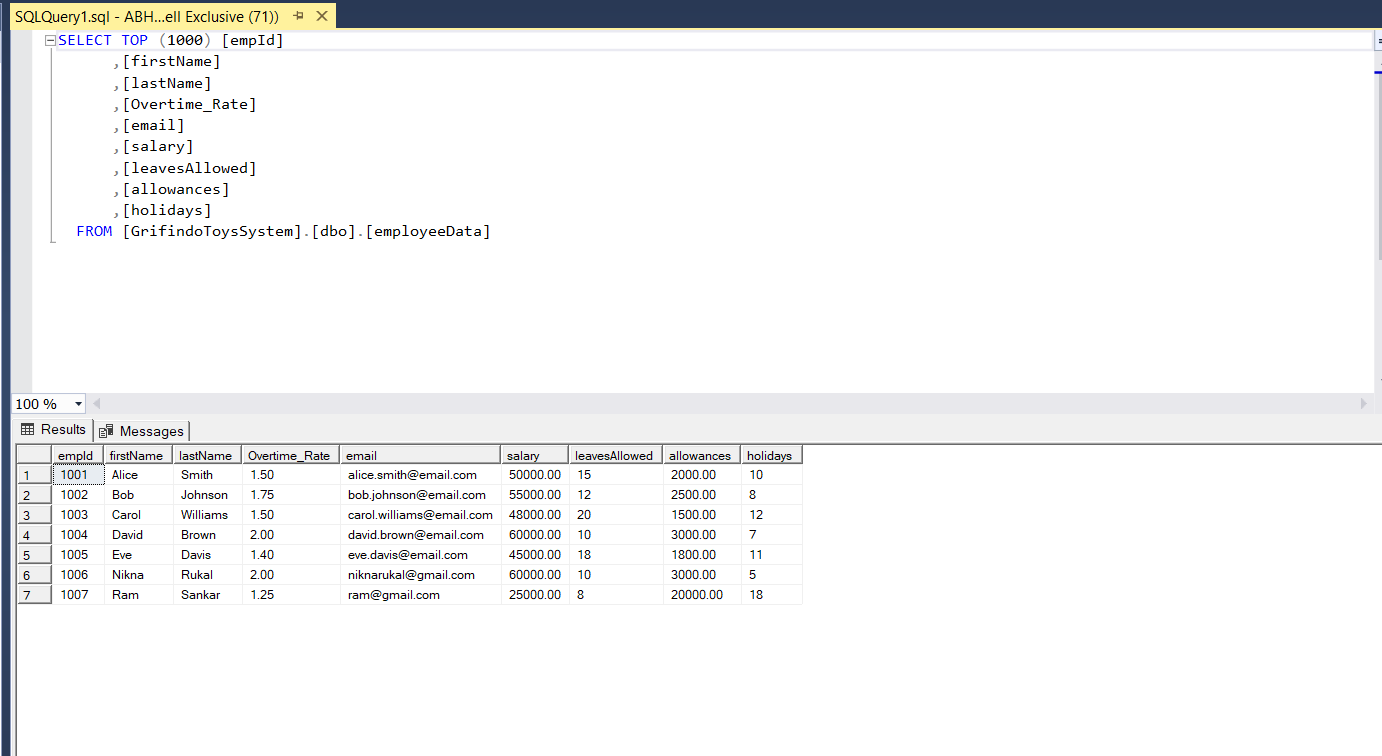


Figure 1 Database for Employee Details

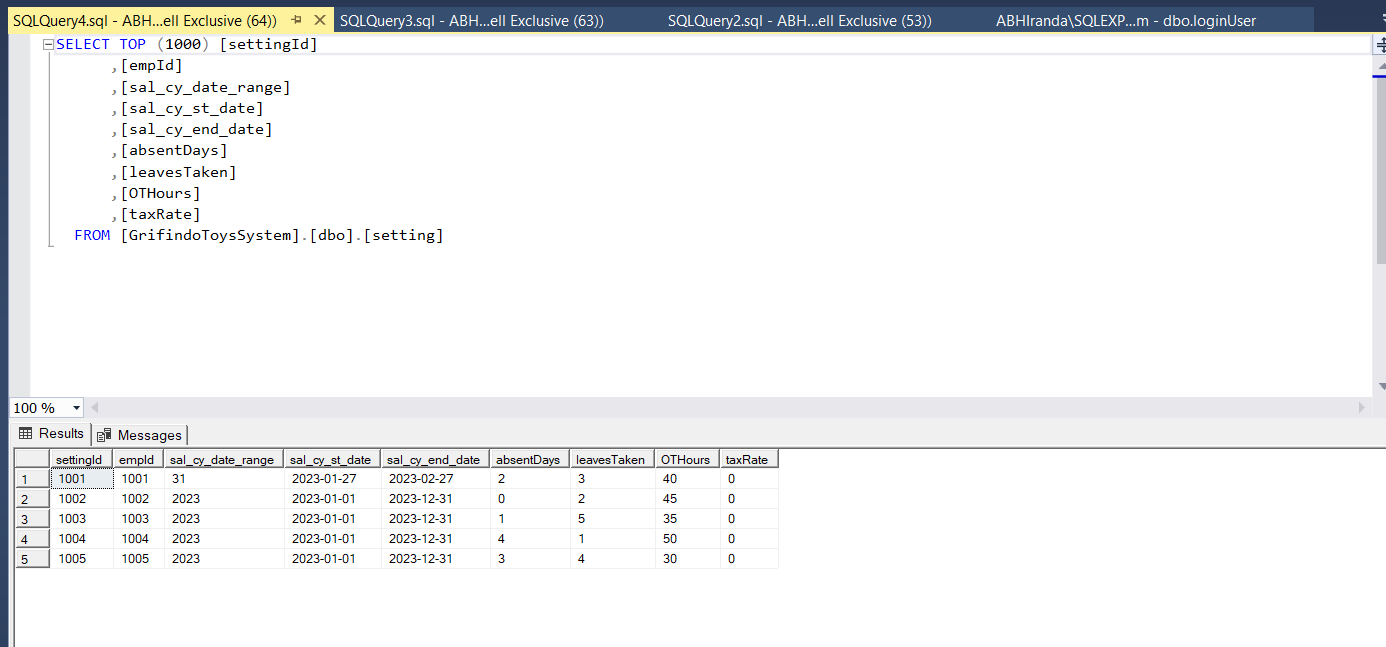
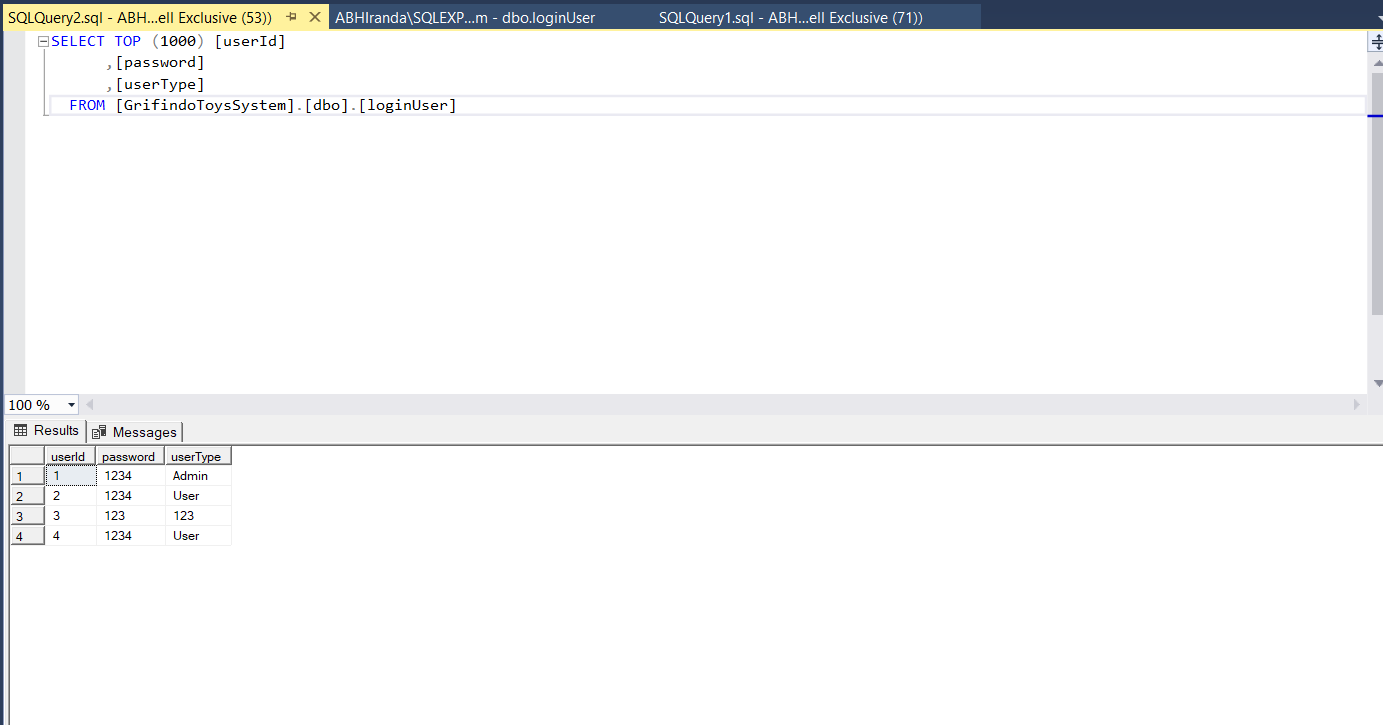


Figure 2 Login user Table

Figure 3 Setting Table

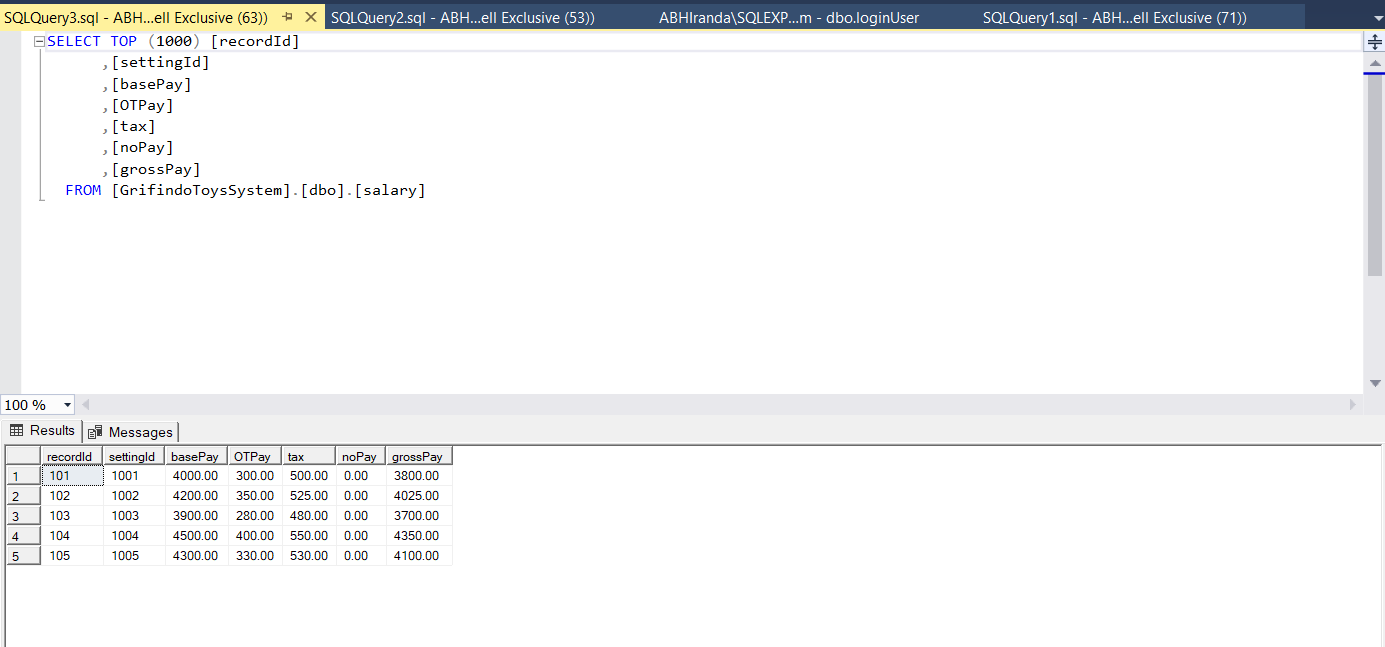


Figure 4 Salary Table



Figure 5 Loading Screen

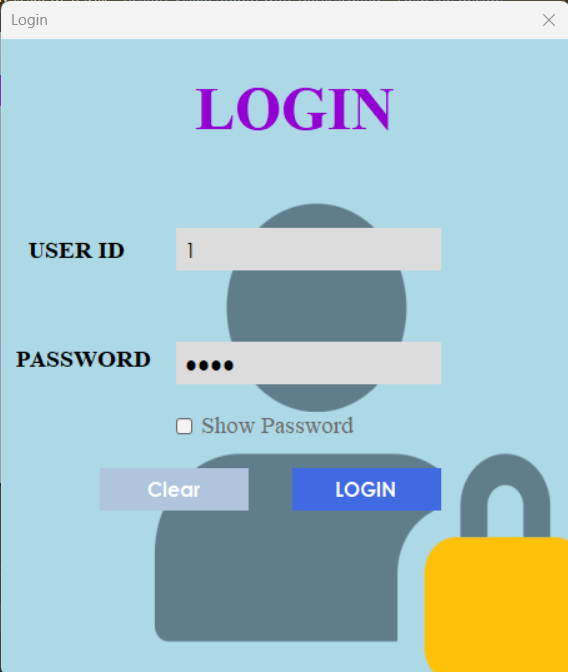


Figure 6 User Login Screen



Figure 7 Payroll System Home Screen

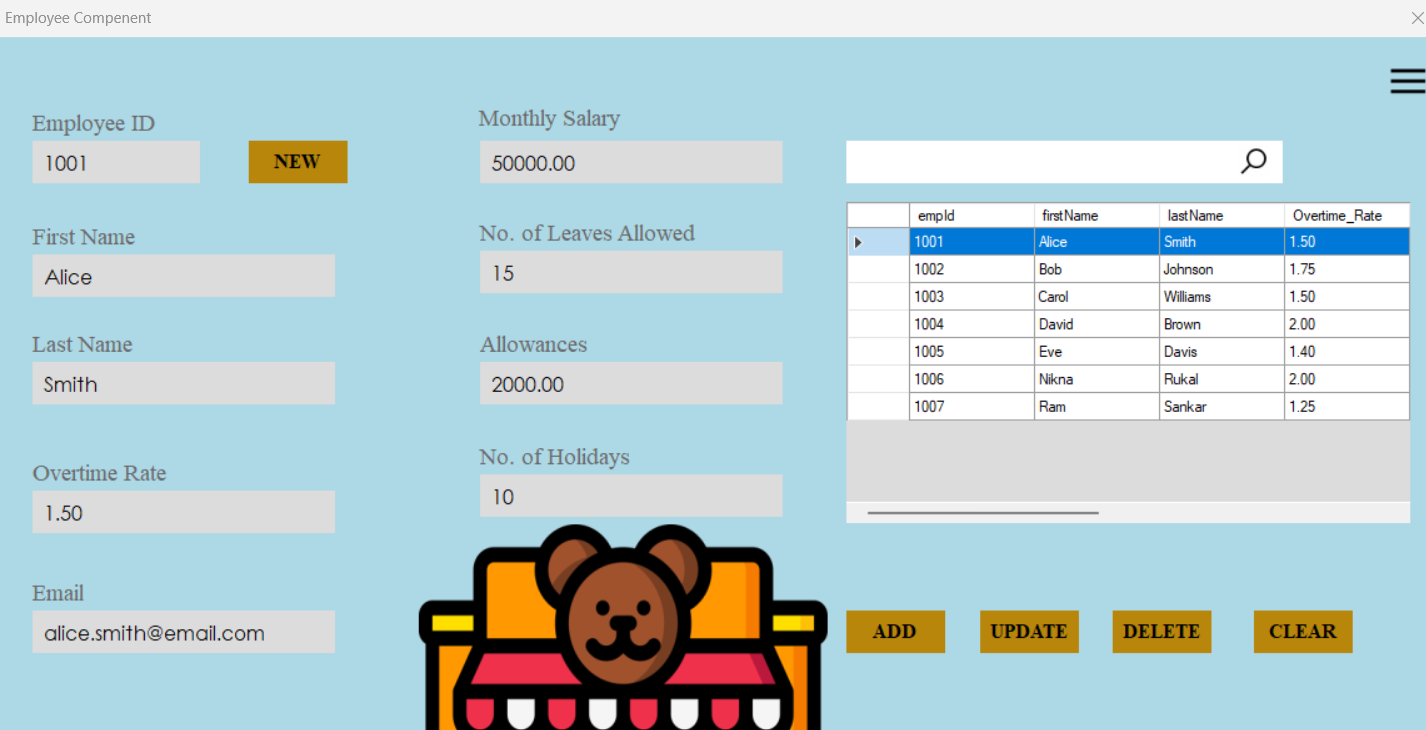


Figure 8 Employee component Screen

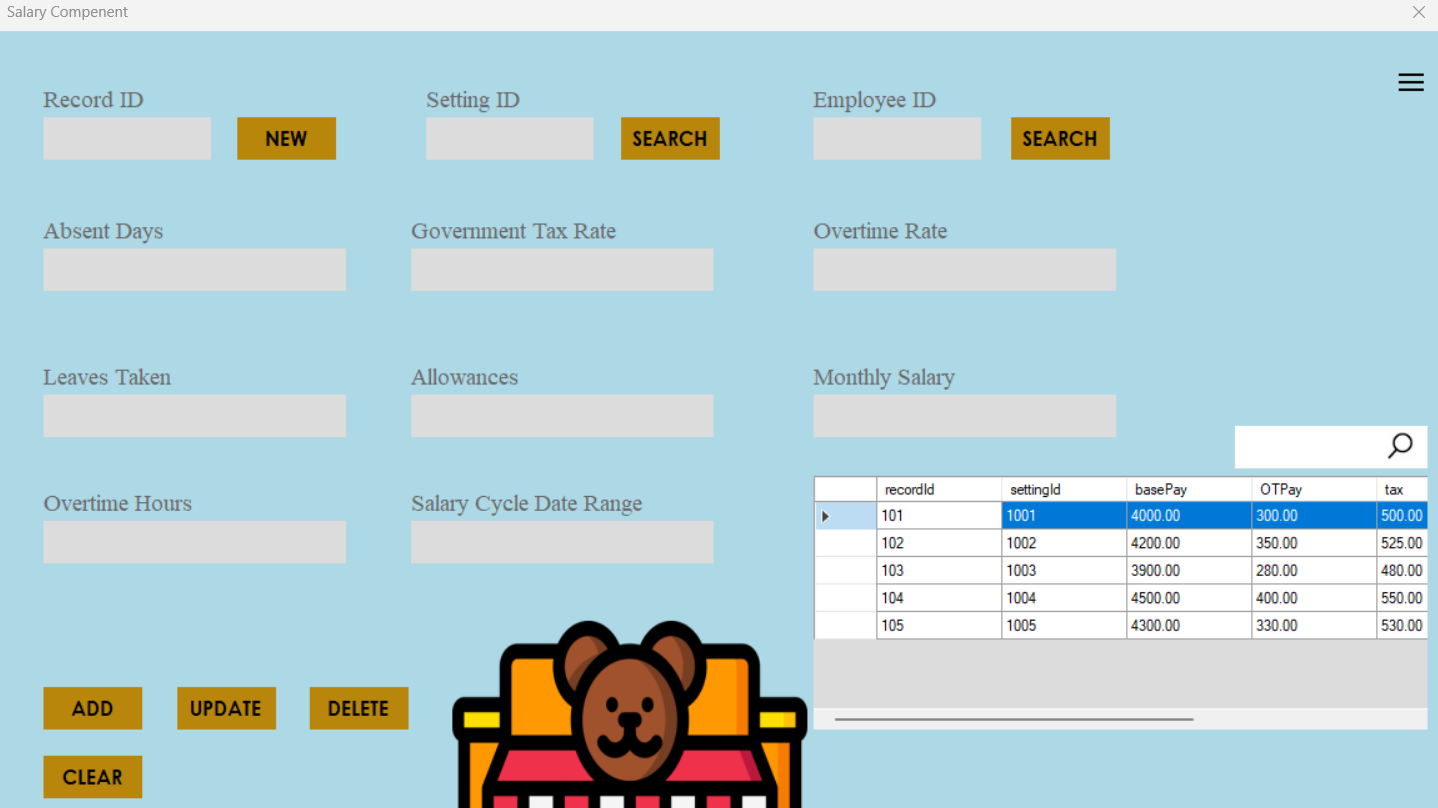


Figure 9 Salary Component Screen

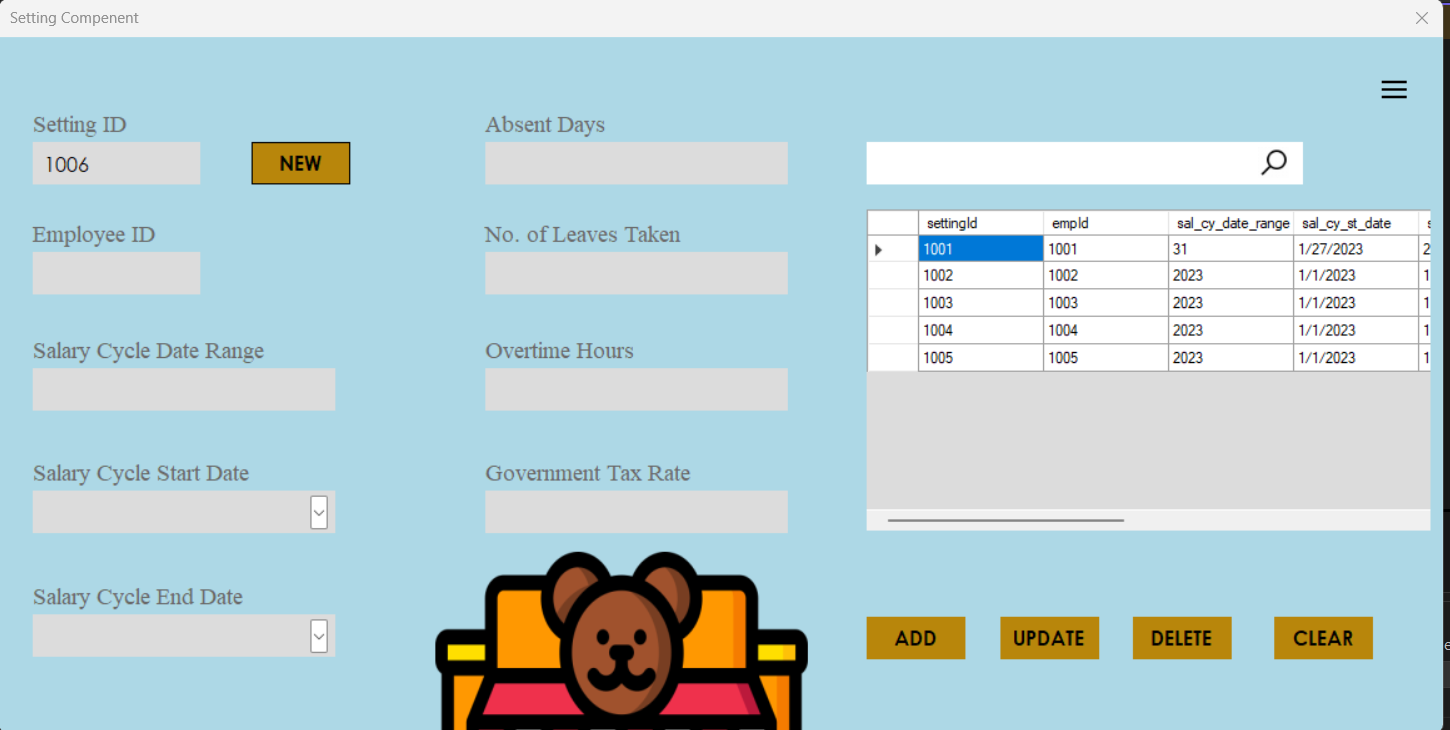


Figure 10 Setting Component Screen

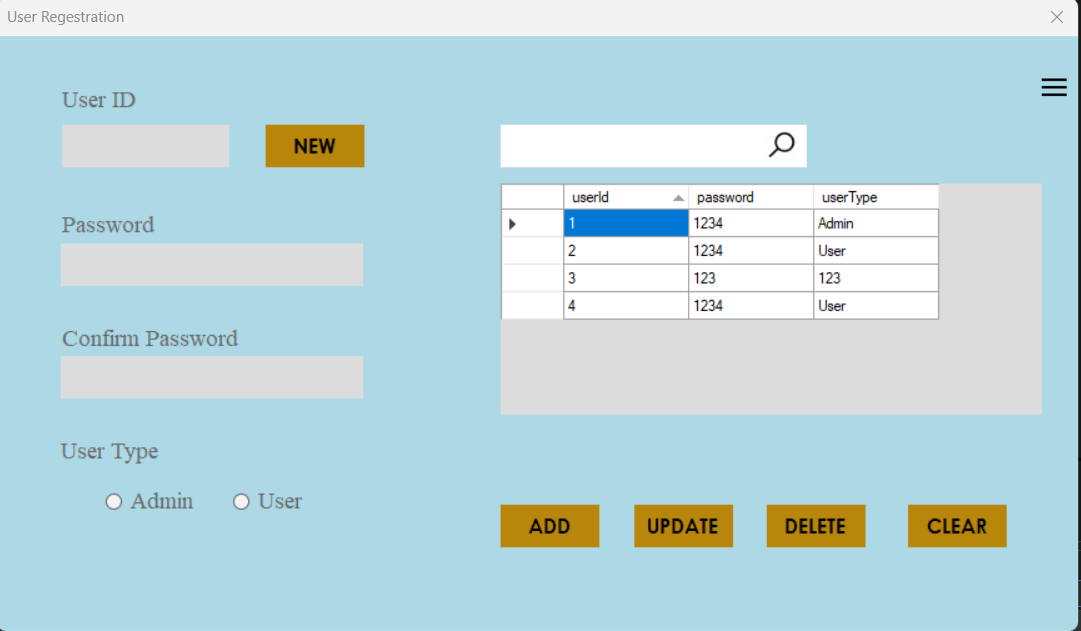


Figure 11 User Registration Screen

## 4.2 Debugging process

When I debug an application, I run it while the debugger is connected. A breakpoint indicates when Visual Studio should halt the running code then may inspect variable values or memory behavior. F5 launches the debugger and pauses at the first breakpoint if the app is not currently running. The Step Into command, represented by F11, advances the program execution one statement at a time.

One can skip through code that he or she is not interested in by hitting F10. Establishing a temporary breakpoint is analogous with using the Run to Click button. Hover over a line of code in the Visual Studio Debugger until the Run to Click (Run execution to here) button appears. The button signifies that execution should go to the line indicated by the button.

It gives keyboard shortcuts for the most actions to help users navigate their app code faster. I pressed F5 until I could get to the line of code where to choose Run to Cursor. Whenever I am modifying code and need to quickly establish a temporary breakpoint, I use this command. The debugger stops at the first breakpoint encountered when executing code. Whenever I wish to exit the debugger and return to the code editor, I hit the red Stop Debugging button rather than Restarting all.

The Autos window displays all variables used on the current or previous line. The variables that are already in scope are displayed in the Locals pane. One may use the Watch window to define a variable to monitor. The Exception Helper is a fantastic feature that may assist developers in debugging issues. When the program throws an exception, the debugger jumps to the line of code where the exception occurred.

The Snapshot Debugger can assist in reducing the time required to resolve issues in production scenarios. At each breakpoint and step event, IntelliTrace step-back captures a snapshot of the program. Snapshots allow developers to return to prior breakpoints or stages and observe the application's previous state. View snapshots while using the Debug toolbar's Step Backward and Step Forward buttons.

## 4.3 Importance of Coding standards

Coding standards are a set of processes that may be created for a certain programming language in order to identify a programming style, techniques, and distinct procedures. These procedures might be for different components of the software developed in that language. A coding standard ensures that all developers working on the project adhere to particular principles. It also guarantees that the code is understandable and that adequate consistency is maintained. The advantages provided by coding standards assist developers in expertly crafting any software solution.

Unmanaged and disorganized code might make it simpler for hackers to get access. The improved code quality reflects the developers' intelligence and professional attitude to their clients. Every software development organization must optimize their development time. Implementing the proper coding standards aids in the creation of already optimized code, which eventually aids in the elimination of any errors that may be discovered after the development period has concluded. Following coding standards in the source code is critical for reducing debugging time and providing the best output to end users. Coding standards aid in the creation of code that is simple to maintain as well as execute.

### Code standard for the group

Coding standards are crucial when building applications as a team. Guidelines for example coding can aid software development teams in producing uniform, easily-understood code. Setting such standards might be tricky if not executed well, but can provide huge dividends for the whole team if done correctly. In addition to assisting developers with the software's functionality, team coding standards lower the likelihood of project failure. By adhering to certain standards, teams may lessen the likelihood of project failure and increase efficiency. The ability to track the progress of other team members who are working on other aspects of the same application process is another significant advantage of working in a team. Teams that adhere to good coding standards are able to seamlessly transition between them, ensuring that all members are up-to-date on the development process. It is possible to make modifications to the code as required when coding standards are followed. In reality, a team may greatly benefit from adhering to code standards.

### Coding standard for individual

When we operate alone, coding standards are really crucial. This is more like how a person would create a programme than how a team would do it. By adhering to coding standards, one may better understand the code and, in the event that development must halt, pass the code on to another person for completion of the project. On the other hand, if the project can be handed off to another programmer, it won't be a huge hassle since the code is straightforward to grasp if it follows a single pattern. Adherence to coding standards aids in personal development by allowing the developer to adjust to new circumstances and pursue more contemporary approaches. Modifying the system in a manner that anybody might understand is possible if suitable coding standards are followed. Lastly, it is critical for individuals to adhere to the code standards.