# Design and Implementation of a virtual PunchClock

**Ruchira Kumarasinghe**[†]
**Department of Mathematics and Computer Science**
**University of Wisconsin-Superior**
**rkumaras@students.uwsuper.edu**

**Victor Piotrowski**
**Department of Mathematics and Computer Science**
**University of Wisconsin-Superior**
**vpiotrow@uwsuper.edu**

## Abstract

This paper describes a software development process of a *PunchClock*, a .Net application for managing timesheets of student employees. The application is based upon Windows Forms, C# and ADO.NET. It consists of a desktop component that connects to a database server in a secure way. The application contains basic Punch Clock functions in a user mode, and restricted management features in an administrative mode.

[†] Student presenter.

## Introduction

The Department of Mathematics and Computer Science at the University of Wisconsin-Superior has been using a commercial PunchClock application to manage time sheets of student employees. This application had a steep price tag, but, more importantly, it was saving its data in an unencrypted, plain text file on a local hard drive. It also required a server component to be running on a different machine.

We decided to develop our own version of the PunchClock using a new Microsoft .NET framework, C# language, and Visual Studio .NET. All computers in UW-S student computer labs are PC's running Windows XP, therefore cross-platform capability was not a concern. The power of C# language, the simplicity of developing Windows Forms in Visual Studio .NET, and a collection of powerful objects such as ADO.NET for a database connectivity were main reasons for choosing .NET technology. The fact that you can convert a windows application to a web-based application with a relatively little work was another argument for using .NET framework.

This paper describes the software development process of a .Net-based version of the PunchClock. The application consists of a desktop module that connects to a database server in a secure way. The application contains basic Punch Clock functions in a user mode, and restricted management features in an administrative mode.

## Requirement analysis

After analyzing the problem statement we identified the following list of problems:

1. We need to store and organize the payroll information of individual employees. We should prevent unauthorized access to this information.
2. We need basic and advanced reporting functions to get reports for each pay period with the number of hours each employee worked and to display contact information of employees.
3. We need some sort of management functionality allowing to create, edit, and delete user accounts; correct user errors; update passwords; etc.
4. The application should work in a distributed environment using secure network connections.

## Solution

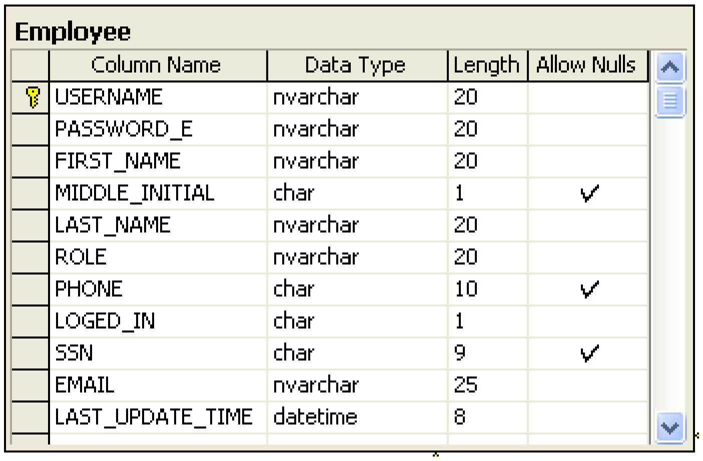
To store data in a secure and reliable manner, we decided to use a relational database hosted on MSSQL server. We decided to create a windows desktop application based on the .NET architecture. We used .NET because of the simplicity of developing Windows Forms in Visual Studio .NET, and a collection of powerful objects such as ADO.NET for database connectivity. A reporting module is done in Crystal Reports which is a part of Visual Studio .NET. We decided to use Crystal Reports because of its ability to create professionally looking reports and the fact that our campus uses Crystal Reports as a reporting standard due to PeopleSoft integration. PunchClock's management functionality is based on two privilege levels:

- **User**: Authorized to view her contact information, payroll reports, salary information.
- **Administrator**: Authorized to create, edit, and delete user accounts; modify the data; change employee passwords; print single or multiple payroll reports.

## Database Design

After normalization, the database consists of four linked tables:

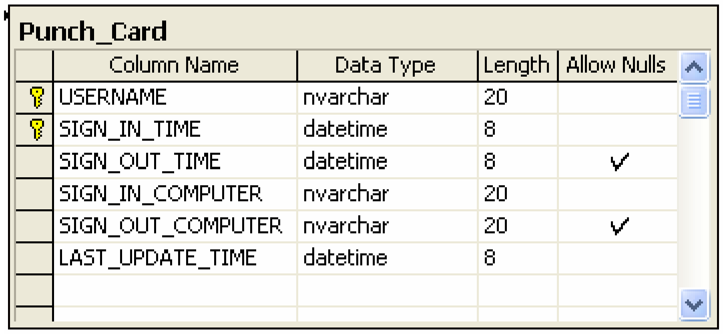
1. **Employee Table**
   Employee table consists of 11 columns. These columns store the employee's name, a unique user name, encrypted password, role (Admin, User), phone number, social security number (SSN), email address, last update time for the record, and the status of the employee, whether she is logged in (Y or N). User name is the primary key of this table.



Employee

| | Column Name | Data Type | Length | Allow Nulls |
|---|---|---|---|---|
| 🔑 | USERNAME | nvarchar | 20 | |
| | PASSWORD_E | nvarchar | 20 | |
| | FIRST_NAME | nvarchar | 20 | |
| | MIDDLE_INITIAL | char | 1 | ✓ |
| | LAST_NAME | nvarchar | 20 | |
| | ROLE | nvarchar | 20 | |
| | PHONE | char | 10 | ✓ |
| | LOGED_IN | char | 1 | |
| | SSN | char | 9 | ✓ |
| | EMAIL | nvarchar | 25 | |
| | LAST_UPDATE_TIME | datetime | 8 | |

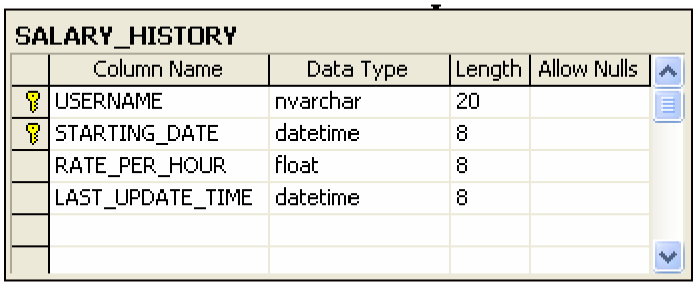
2. **Punch Card Table**
   Punch_Card table contains the punch-in and -out time for every employee. The table consists of six columns: user name; sign-in and sign-out times; the sign-in computer name and the sign-out computer name; and the last time the record was updated. The computer name field contains a computer name or an IP address of the computer that was used to punch-in and punch-out. The last update time will be used to monitor data inconsistency. User and sign-in time will be the primary keys in this table.



**Punch_Card**

| | Column Name | Data Type | Length | Allow Nulls |
|---|---|---|---|---|
| 🔑 | USERNAME | nvarchar | 20 | |
| 🔑 | SIGN_IN_TIME | datetime | 8 | |
| | SIGN_OUT_TIME | datetime | 8 | ✓ |
| | SIGN_IN_COMPUTER | nvarchar | 20 | |
| | SIGN_OUT_COMPUTER | nvarchar | 20 | ✓ |
| | LAST_UPDATE_TIME | datetime | 8 | |

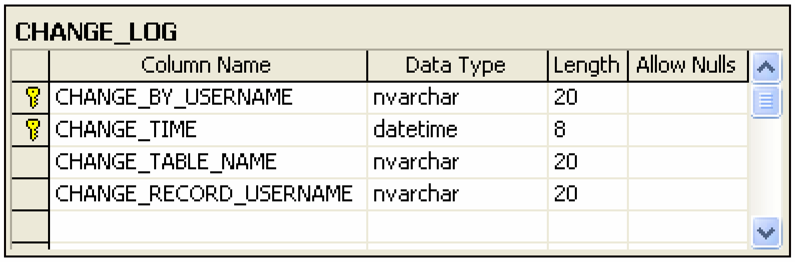
3. **Salary History Table**
   The Salary_History table has four columns: user name, starting date, salary (rate per hour), and the last update time. This table will keep track of the employee's salary over time. User and starting date fields will act as the primary keys in this table.



**SALARY_HISTORY**

| | Column Name | Data Type | Length | Allow Nulls |
|---|---|---|---|---|
| 🔑 | USERNAME | nvarchar | 20 | |
| 🔑 | STARTING_DATE | datetime | 8 | |
| | RATE_PER_HOUR | float | 8 | |
| | LAST_UPDATE_TIME | datetime | 8 | |

4. **Change Log Table**
   The Change_Log table keeps records of any changes made by the administrators. The table consists of four columns: change by username (administrator's username), time the change was made, name of the table the change was made, and the username of the employee the change was made to. Change by username and the change time act as the primary keys for this table.

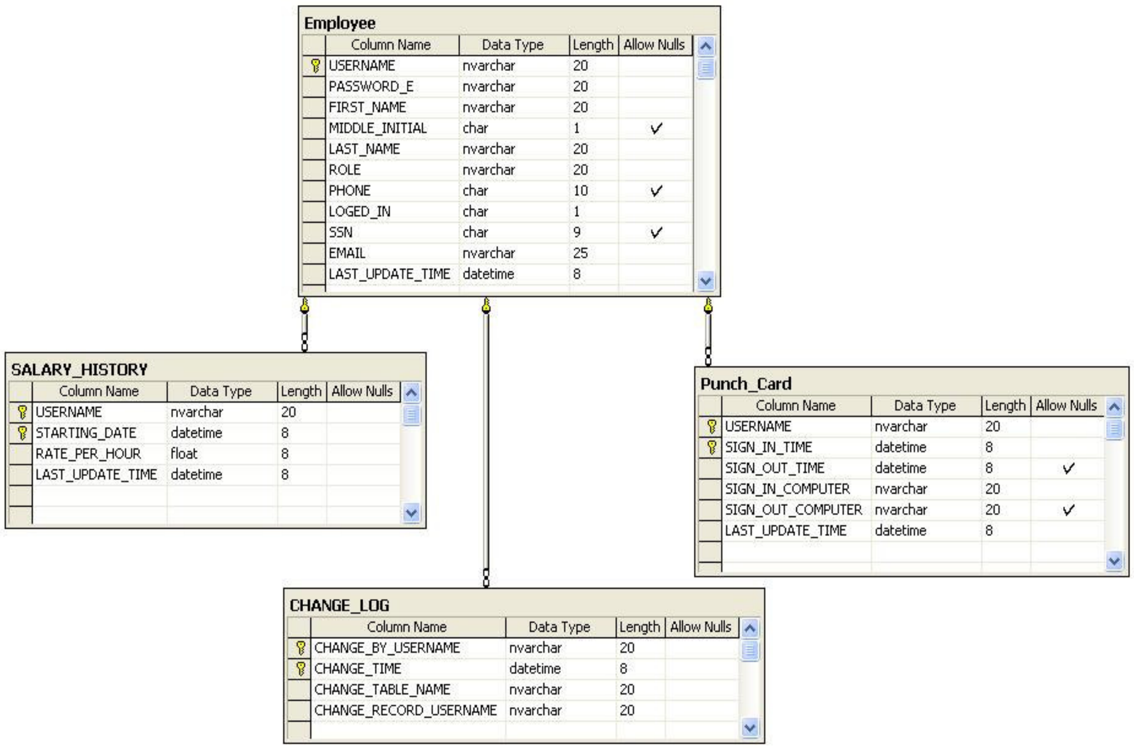


**CHANGE_LOG**

| | Column Name | Data Type | Length | Allow Nulls |
|---|---|---|---|---|
| 🔑 | CHANGE_BY_USERNAME | nvarchar | 20 | |
| 🔑 | CHANGE_TIME | datetime | 8 | |
| | CHANGE_TABLE_NAME | nvarchar | 20 | |
| | CHANGE_RECORD_USERNAME | nvarchar | 20 | |

## Database Schema

PunchClock database contains the following relations:



**Employee**

| | Column Name | Data Type | Length | Allow Nulls |
|---|---|---|---|---|
| 🔑 | USERNAME | nvarchar | 20 | |
| | PASSWORD_E | nvarchar | 20 | |
| | FIRST_NAME | nvarchar | 20 | |
| | MIDDLE_INITIAL | char | 1 | ✓ |
| | LAST_NAME | nvarchar | 20 | |
| | ROLE | nvarchar | 20 | |
| | PHONE | char | 10 | ✓ |
| | LOGED_IN | char | 1 | |
| | SSN | char | 9 | ✓ |
| | EMAIL | nvarchar | 25 | |
| | LAST_UPDATE_TIME | datetime | 8 | |

**SALARY_HISTORY**

| | Column Name | Data Type | Length | Allow Nulls |
|---|---|---|---|---|
| 🔑 | USERNAME | nvarchar | 20 | |
| 🔑 | STARTING_DATE | datetime | 8 | |
| | RATE_PER_HOUR | float | 8 | |
| | LAST_UPDATE_TIME | datetime | 8 | |

**Punch_Card**

| | Column Name | Data Type | Length | Allow Nulls |
|---|---|---|---|---|
| 🔑 | USERNAME | nvarchar | 20 | |
| 🔑 | SIGN_IN_TIME | datetime | 8 | |
| | SIGN_OUT_TIME | datetime | 8 | ✓ |
| | SIGN_IN_COMPUTER | nvarchar | 20 | |
| | SIGN_OUT_COMPUTER | nvarchar | 20 | ✓ |
| | LAST_UPDATE_TIME | datetime | 8 | |

**CHANGE_LOG**

| | Column Name | Data Type | Length | Allow Nulls |
|---|---|---|---|---|
| 🔑 | CHANGE_BY_USERNAME | nvarchar | 20 | |
| 🔑 | CHANGE_TIME | datetime | 8 | |
| | CHANGE_TABLE_NAME | nvarchar | 20 | |
| | CHANGE_RECORD_USERNAME | nvarchar | 20 | |

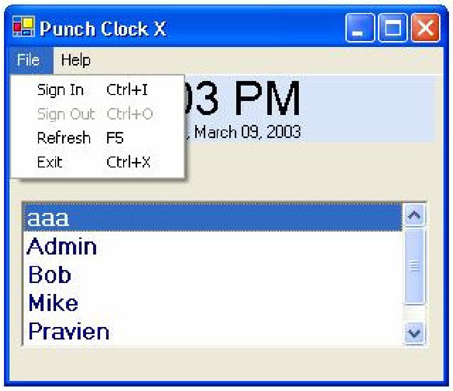
| Foreign Key | Relationship Tables | Relationship |
|---|---|---|
| FK_SALARY_HISTORY_USERNAME | Employee – Salary History | One to Many |
| FK_PUNCH_CARD_USERNAME | Employee – Punch Card | One to Many |
| FK_CHANGE_LOG_USERNAME | Employee – Change Log | One to Many |

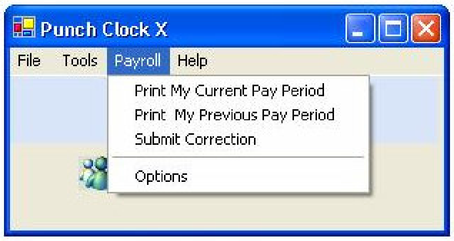
The database may be created during an installation process. If a user selects create database option, the application will connect to the database server and it will create the PunchClock database as shown in a code segment below:

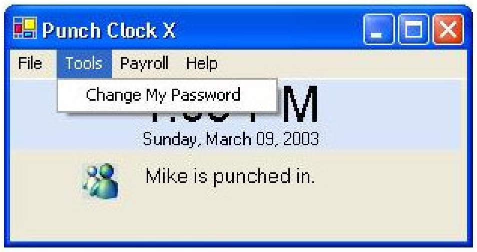
```
CREATE TABLE punchClockX.DBO.CHANGE_LOG
(
        CHANGE_BY_USERNAME      nvarchar(20) NOT NULL,
        CHANGE_TIME             datetime NOT NULL,
        CHANGE_TABLE_NAME       nvarchar(20) NOT NULL,
        CHANGE_RECORD_USERNAME  nvarchar(20) NOT NULL

        CONSTRAINT PK_CHANGE_LOG PRIMARY KEY (CHANGE_BY_USERNAME, CHANGE_TIME)
        CONSTRAINT FK_CHANGE_LOG_USERNAME FOREIGN KEY (CHANGE_BY_USERNAME)
                REFERENCES  punchClockX.DBO.EMPLOYEE (USERNAME)
)
```

## User Interface

The Punch clock application has a single logon screen. The screen displays usernames of all the employees in the database and a user can login by selecting her username from the list. The application will validate the user against the *Employee* table in the database and will display the user screen or the administrator screen depending on the privilege level of the user.







Mike is punched in.

## User Level Functionality

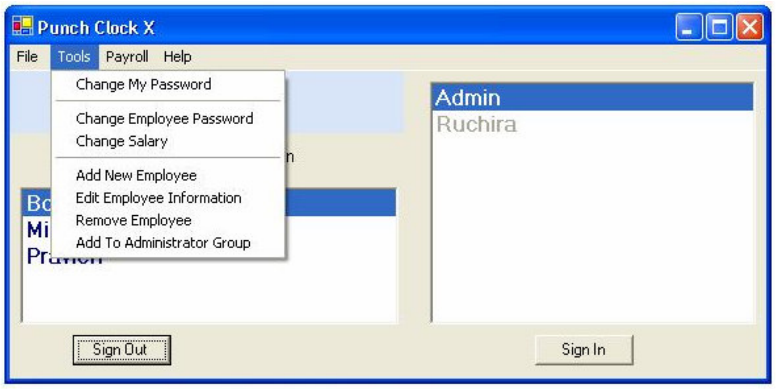
When the user is signed-in she can change her password using the Change Password screen. She has to enter the old password and the new password. Also, she can get an unofficial report of the current or previous pay period by selecting the appropriate options from the payroll menu. She can also send an email to all the administrators by using the submit correction option in the same menu.

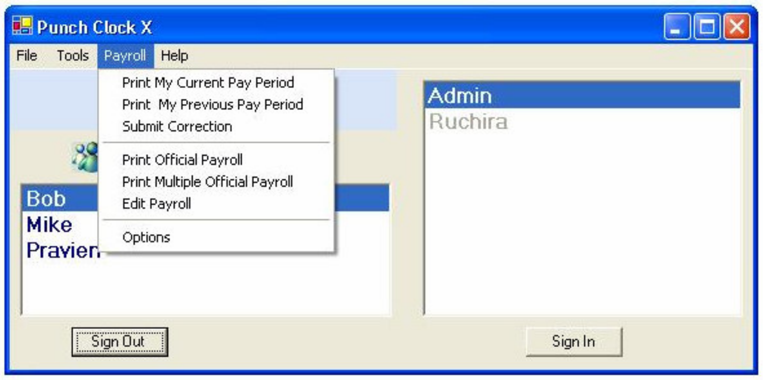
## Administrator Level Functionality

When an administrator is signed-in she can see the form with more functionality than a regular user. She would see a list of users that are currently signed-in on the left and signed-out on the right. She can sign-in or -out any user by selecting a username from the list. Administrator can change an employee's salary and add any existing employee to the Administrators group by selecting an appropriate option from the tools menu. By selecting Change Employee Password option form the Tools menu the administrator can change any employee's password. The password is encrypted and saved in the database. The Administrator can add new employee, edit employee information, and remove employee by selecting the options form the Tools menu.

An administrator has all the payroll functionality that a regular user has and more. She can print official payroll information for a single or multiple employees by selecting the

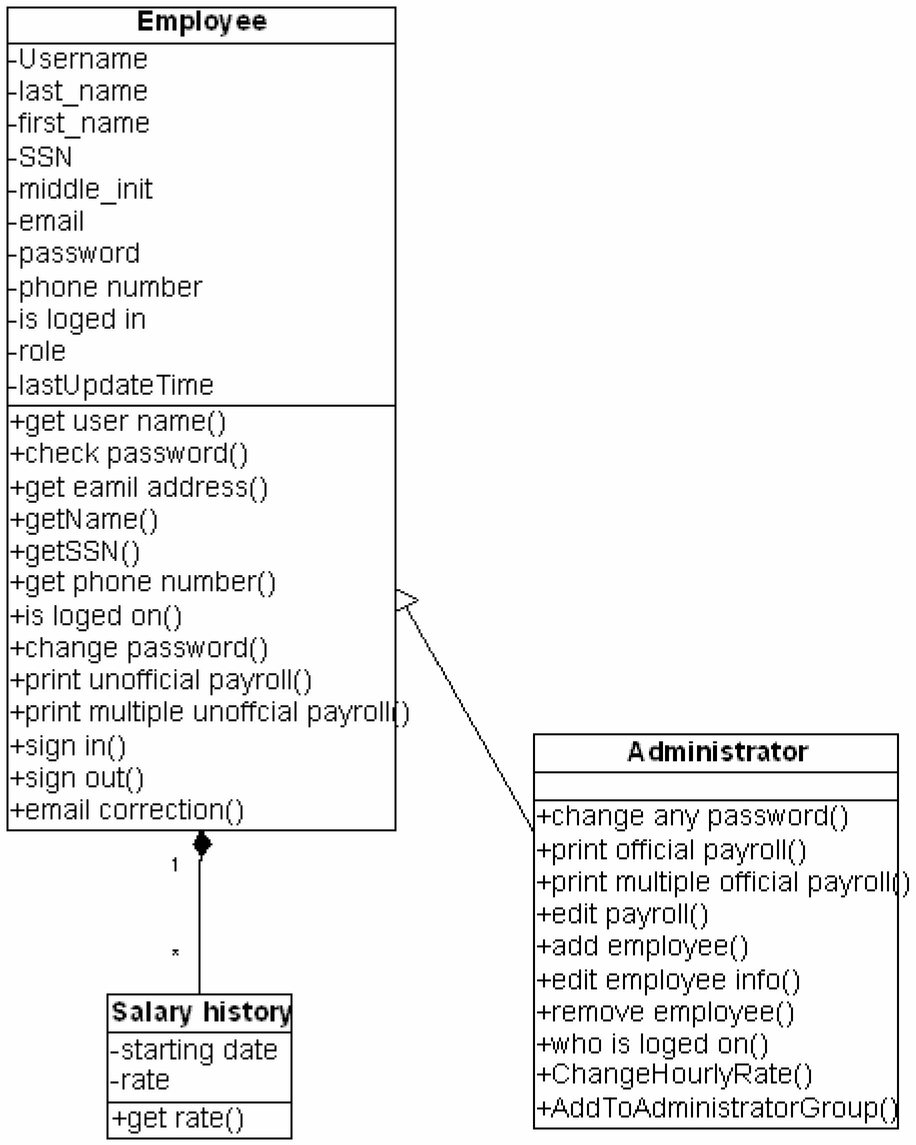
Print Official Payroll and Print Multiple Official Payroll options form the Payroll menu. When an administrator is viewing payroll information she can choose to send an email to an employee with his payroll information. If there is an error in the payroll information (for example, an employee forgot to sign-out) the administrator can correct it by selecting the Edit Payroll option from the Payroll menu.





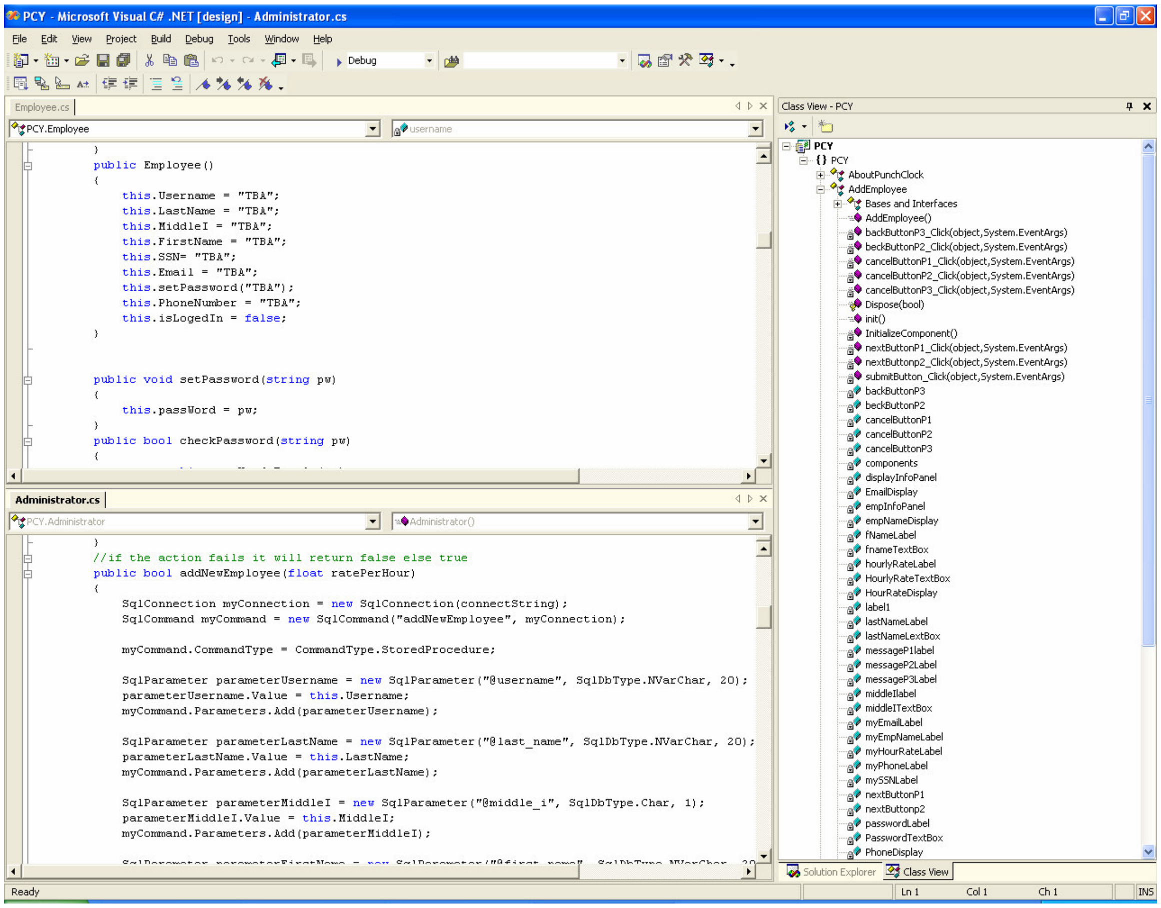
## Object Model

The class Employee contains the employee information and all the methods to implement the user level functionality. The class Administrator extends the class Employee. Salary History Class has an aggregation relationship with Employee. Employee Class has an array of Salary History Objects. The diagram below summarizes the object design:



## Implementation

Coding several thousand lines in C# went quite smoothly. Although it is possible to create Windows Forms using only the command-line interface, in practice it is much easier and faster to use Visual Studio.NET. Therefore, we generated all GUI elements in Visual Studio Forms Designer. In addition, we had to write few SQL stored procedures and to create data binding for several forms. Regarding performance, it seems that MSSQL server ADO.NET data provider is about 20% faster than other data providers we experimented with.


PCY - Microsoft Visual C# .NET [design] - Administrator.cs
File Edit View Project Build Debug Tools Window Help
Debug
Employee.cs
PCY.Employee
username

```
        }
        public Employee()
        {
            this.Username = "TBA";
            this.LastName = "TBA";
            this.MiddleI = "TBA";
            this.FirstName = "TBA";
            this.SSN= "TBA";
            this.Email = "TBA";
            this.setPassword("TBA");
            this.PhoneNumber = "TBA";
            this.isLogedIn = false;
        }


        public void setPassword(string pw)
        {
            this.passWord = pw;
        }
        public bool checkPassword(string pw)
        {
```

Administrator.cs
PCY.Administrator
Administrator()

```
        }
        //if the action fails it will return false else true
        public bool addNewEmployee(float ratePerHour)
        {
            SqlConnection myConnection = new SqlConnection(connectString);
            SqlCommand myCommand = new SqlCommand("addNewEmployee", myConnection);

            myCommand.CommandType = CommandType.StoredProcedure;

            SqlParameter parameterUsername = new SqlParameter("@username", SqlDbType.NVarChar, 20);
            parameterUsername.Value = this.Username;
            myCommand.Parameters.Add(parameterUsername);

            SqlParameter parameterLastName = new SqlParameter("@last_name", SqlDbType.NVarChar, 20);
            parameterLastName.Value = this.LastName;
            myCommand.Parameters.Add(parameterLastName);

            SqlParameter parameterMiddleI = new SqlParameter("@middle_i", SqlDbType.Char, 1);
            parameterMiddleI.Value = this.MiddleI;
            myCommand.Parameters.Add(parameterMiddleI);
```

```
PCY
  {} PCY
    AboutPunchClock
    AddEmployee
      Bases and Interfaces
      AddEmployee()
      backButtonP3_Click(object,System.EventArgs)
      beckButtonP2_Click(object,System.EventArgs)
      cancelButtonP1_Click(object,System.EventArgs)
      cancelButtonP2_Click(object,System.EventArgs)
      cancelButtonP3_Click(object,System.EventArgs)
      Dispose(bool)
      init()
      InitializeComponent()
      nextButtonP1_Click(object,System.EventArgs)
      nextButtonp2_Click(object,System.EventArgs)
      submitButton_Click(object,System.EventArgs)
      backButtonP3
      beckButtonP2
      cancelButtonP1
      cancelButtonP2
      cancelButtonP3
      components
      displayInfoPanel
      EmailDisplay
      empInfoPanel
      empNameDisplay
      fNameLabel
      fnameTextBox
      hourlyRateLabel
      HourlyRateTextBox
      HourRateDisplay
      label1
      lastNameLabel
      lastNameLextBox
      messageP1label
      messageP2Label
      messageP3Label
      middleIlabel
      middleITextBox
      myEmailLabel
      myEmpNameLabel
      myHourRateLabel
      myPhoneLabel
      mySSNLabel
      nextButtonP1
      nextButtonp2
      passwordLabel
      PasswordTextBox
      PhoneDisplay
```

## Future Work

As of writing of this paper, we are in a testing phase and we hope to deploy a pilot program within 4 weeks, and we plan to start a production deployment in 8 weeks. Currently, the objective of using the PunchClock program is to automate student timesheet's reporting. We hope that the PunchClock database can be further integrated with a student payroll system so the entire process will be paperless. Another objective would be to create ASP.NET version of the PunchClock and to deploy it as a web-based application.

The URL for this project is [http://ruchira.cemert.org/punchclock](http://ruchira.cemert.org/punchclock).

## References

1. Stiefel, Oberg. Application Development Using C# and .NET. Prentice Hall, 2002.
2. Sceppa David. *ADO.NET*. Microsoft Press, 2002.