Application Development - COMP1551
Phạm Thanh Trúc – GCD220368 – 001308734

## Contents

## Task 1:

### Description of the Desktop Information System

The education center is transitioning from a traditional paper-based record-keeping system to a more efficient and reliable desktop information system. This new system is designed to handle and manage data for three primary user groups: teaching staff, administration, and students. Each group has specific data requirements that the system will process and store securely.

For teaching staff, the system will manage personal details such as name, telephone, and email, along with role-specific information like salary and the subjects they teach. For administration personnel, the system will capture similar personal details and additional information such as salary, employment type (full-time or part-time), and working hours. For students, the system will record their personal information and educational details, including the names of their current and previously studied subjects.

The system will offer robust functionality to streamline data management. Users will be able to add new records, view all existing records, filter records by user group, search record, edit current records, and delete outdated or incorrect records. This will ensure that the education center maintains accurate and up-to-date information at all times.

The desktop information system will be built to operate on Microsoft Windows, ensuring compatibility with the education center's existing infrastructure. The application will run on standard desktop hardware, making it accessible and easy to implement without requiring specialized equipment.

Overall, this system aims to enhance the efficiency and accuracy of data management within the education center. By providing a user-friendly interface and comprehensive data handling capabilities, it will reduce administrative workload, minimize errors associated with manual data entry, and facilitate quick access to important information. This transition to a digital system marks a significant step forward in modernizing the education center's operations, aligning with contemporary data management practices.

## Software Requirements Specification

### 1. Introduction

**Purpose**
The purpose of this document is to outline the requirements for developing a desktop information system for an education center. This system aims to replace the existing paper-based record-keeping process with a digital solution to enhance efficiency, accuracy, and data accessibility.

**Project Scope**
The project involves designing, implementing, and testing a desktop application that manages data for teaching staff, administration personnel, and students. The system will offer functionalities such as adding, viewing, searching, editing, and deleting records. It will run on Microsoft Windows and require standard desktop hardware.

### 2. Overall Description

**Product**
The desktop information system will manage user data for teaching staff, administration, and students. It will store personal details (name, telephone, email, role) and role-specific information (e.g., salary, subjects taught, employment type). The system will provide a user-friendly interface for data management tasks including add new records, view all existing records, filter records by user group, search record, edit current records, and delete outdated or incorrect records.

**Users**

- The CEO: who owns and manages the education center.
- Administration Personnel: Users who handle administrative tasks and operations.
- Teaching Staff: Users who teach and manage student education.
- Students: Users who are enrolled in the education center.

**Operational Environment**
The system will operate on Microsoft Windows OS and will be compatible with standard desktop hardware. It will function in an office environment with typical network and security setups.

### 3. System Features

**Description**
The system will provide functionalities to manage user data efficiently. It will allow users to add new records, view all existing records, filter records by user group, search record, edit records, and delete records.

**Functional Requirements**

- **Add New Data**: Users can input new records for teaching staff, administration, and students.
- **View All Data**: Users can view a comprehensive list of all records in the system.
- **View Data by Group**: Users can filter and view records based on user groups (teaching staff, administration, students).
- **Search Data**: Users can search existing records.
- **Edit Existing Data**: Users can update details of existing records.
- **Delete Existing Data**: Users can remove outdated or incorrect records from the system.

## 4. User Interface Requirements

The system will feature a graphical user interface (GUI) that allows users to interact with the application through visually intuitive elements. The interface will include a menu for various operations, buttons, lables, textboxes. Users will be able to perform tasks such as adding, viewing, searching, editing, and deleting records using interactive forms and dialogs. The GUI will provide real-time feedback through pop-up messages, status indicators, and tooltips to inform users of successful operations or errors.

## 5. Platform Requirements

The system will run on Microsoft Windows operating system and will be compatible with standard desktop hardware. It does not require specialized hardware or software beyond what is typically available in a standard office environment.

## 6. Quality Attributes

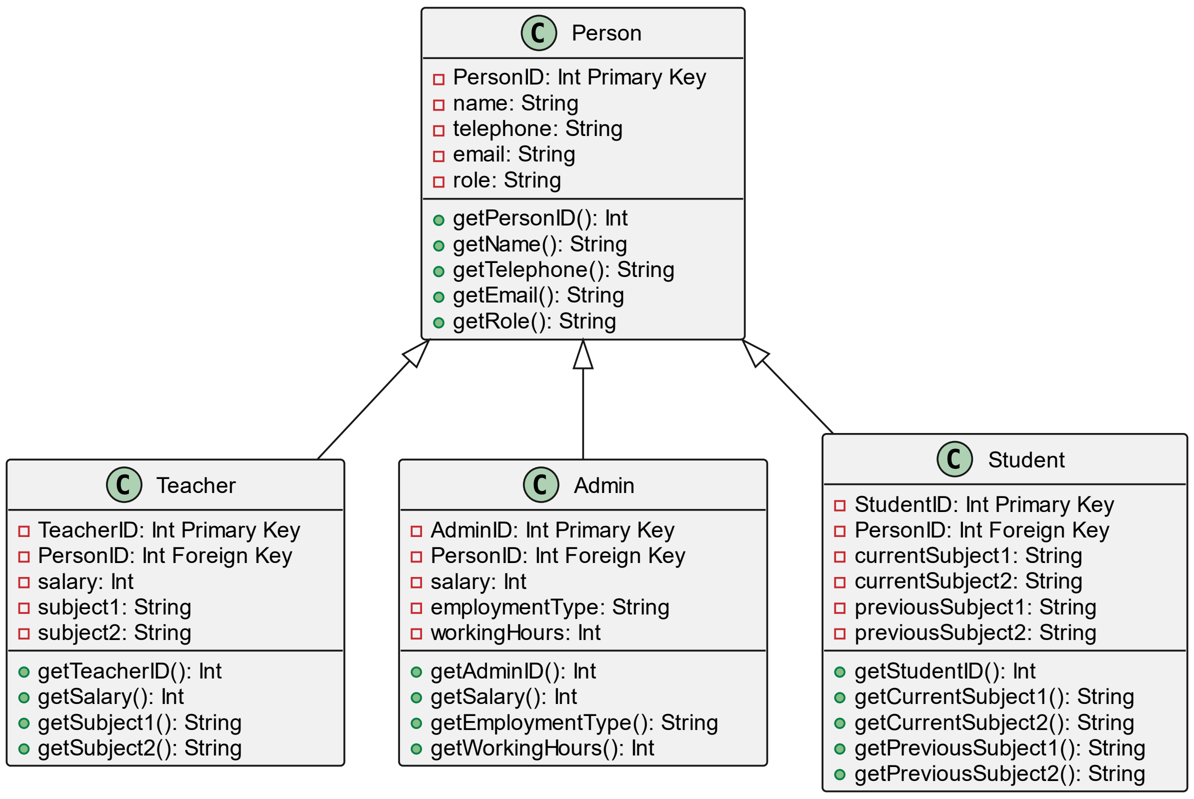
**Performance**
The system will be optimized for quick data processing and retrieval. It will handle a large number of records efficiently without significant lag or delay.

**Security**
User data will be protected with appropriate access controls to ensure only authorized personnel can add, view, edit, or delete records. Data integrity will be maintained to prevent unauthorized modifications.

**Safety**
Data validation mechanisms will be in place to ensure that only valid data is entered into the system. Error messages and prompts will guide users to correct input errors, ensuring data consistency and reliability.

## Task 2: UML Diagrams

### Class Diagram



The class diagram represents an inheritance hierarchy with a base class Person and three derived classes: Teacher, Admin, and Student.

**1. Person (Base Class):**

This class serves as the base class for other specific roles (Teacher, Admin, Student). It includes common attributes like name, telephone, email, and role, along with methods to retrieve these attributes.

**2. Teacher (Derived from Person):**

This class represents a teacher and extends the Person class by adding attributes specific to a teacher, such as salary and subjects they teach, along with methods to retrieve these attributes.

**3. Admin (Derived from Person):**

This class represents an admin role and extends the Person class by adding attributes such as salary, employmentType, and workingHours, along with methods to retrieve these attributes.

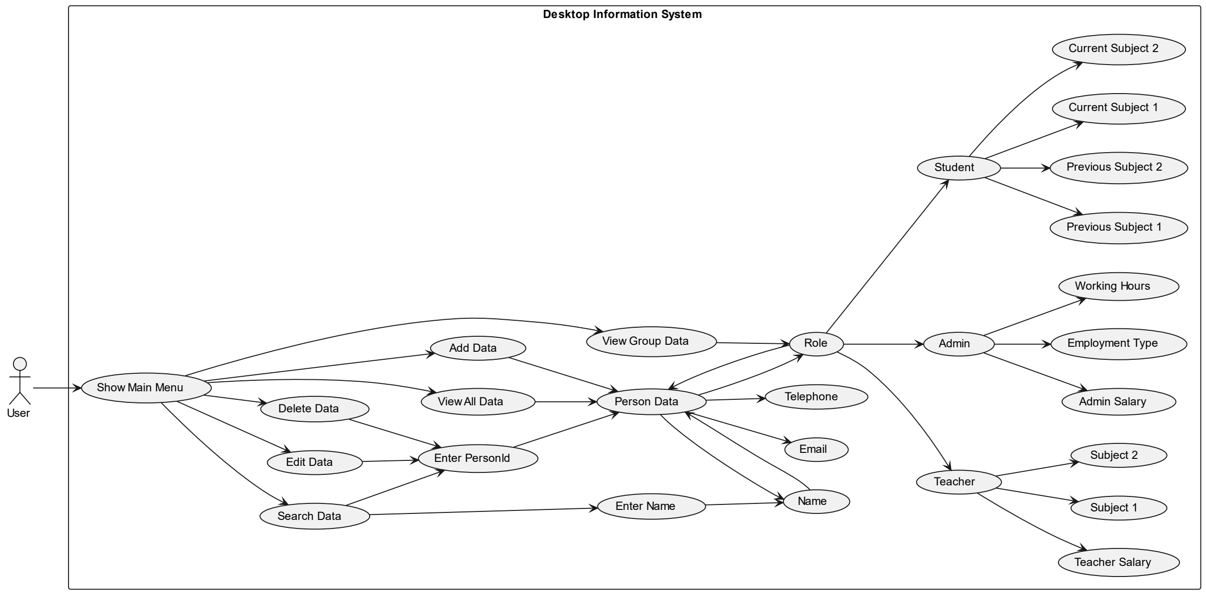
**4. Student (Derived from Person):**

This class represents a student and extends the Person class by including attributes related to the subjects they are currently taking and those they took previously. Methods are provided to retrieve this information.

**General Structure:**

- The Person class acts as a common foundation for the derived classes, which add specialized attributes and methods to represent different roles in the education center.
- The design promotes code reusability and modularity by allowing the shared attributes and methods to be defined in the Person class and inherited by the Teacher, Admin, and Student classes.

## Use Case Diagram



When users interact with Desktop Information System, they see the main menu:
Add data:

- They need to add Person Data with “Name”, “Telephone”, “Email”, “Role”.
- Then if “Role” is “Teacher”, they need to add Teacher Data with “Teacher Salary”, “Subjects”.
- if “Role” is “Admin”, they need to add Admin Data with “Admin Salary”, “Employment Type”, “Working Hours”.
- if “Role” is “Student”, they need to add Student Data with “Current Subjects”, “Previous Subjects”.

View all data:

- Users are showed all Person data.
- When they click on a specific person, it will show more info based on that person’s role.

View group data:

- When users choose which group data they want to view, the system will get and merge data from both Person data and specific Group data “Teacher”, “Admin” or “Student”.

Search data:

- Users need to enter PersonId or Name.
- Then the system will look that PersonId or Name in Person data and show detail info of that person which also included with Role data.

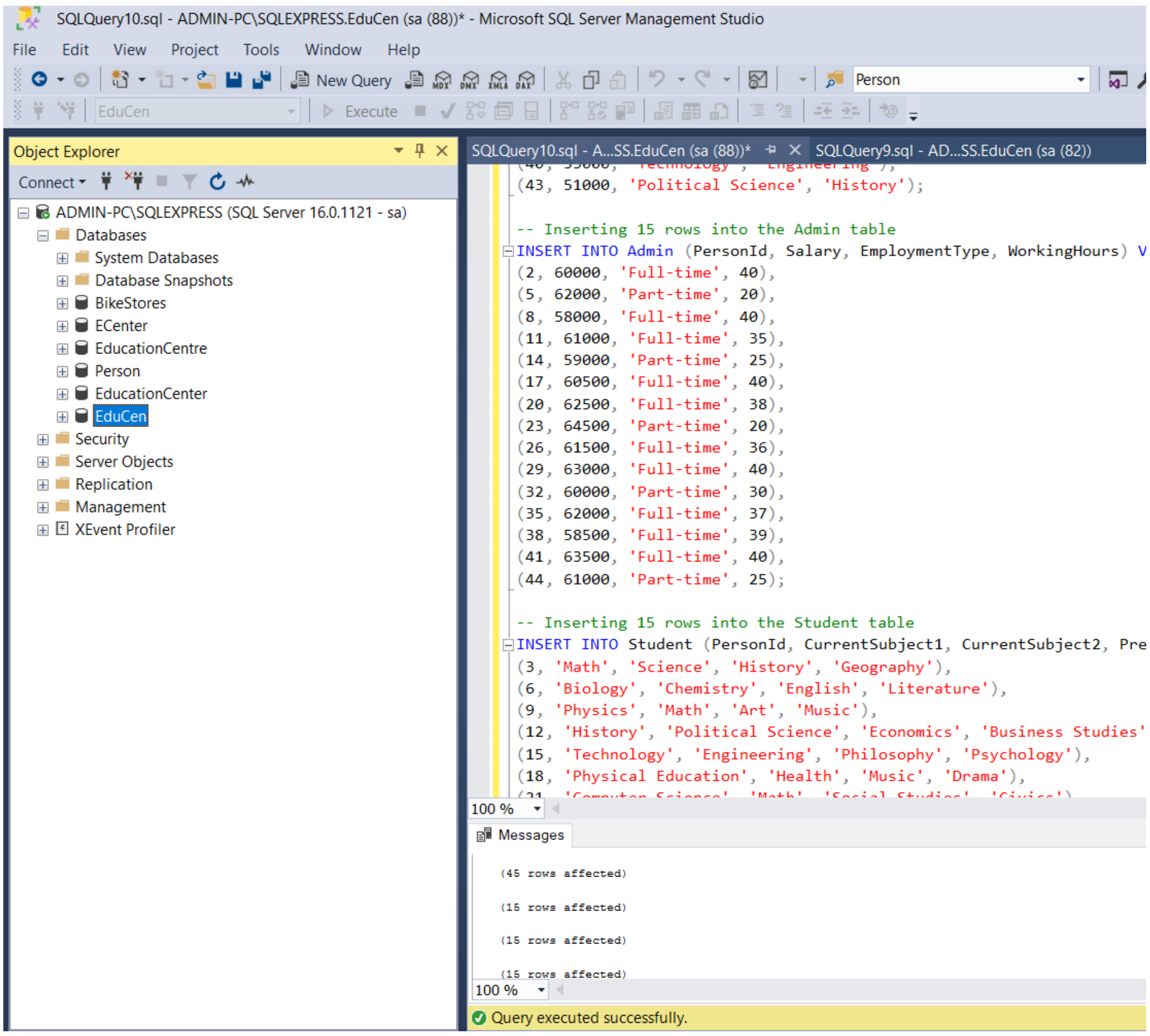
Edit data:

- Users need to enter PersonId.
- Users need to enter in which fields (also fields in Role data) of that person they want to edit.
- Delete data:
- Users need to enter PersonId.
- Then the system will look that PersonId in Person data and delete that person in Person data and in Role data.

## Task 3: C# Console Application

### Database

Establish database “EducationCenter”, which can store unknown number of objects.

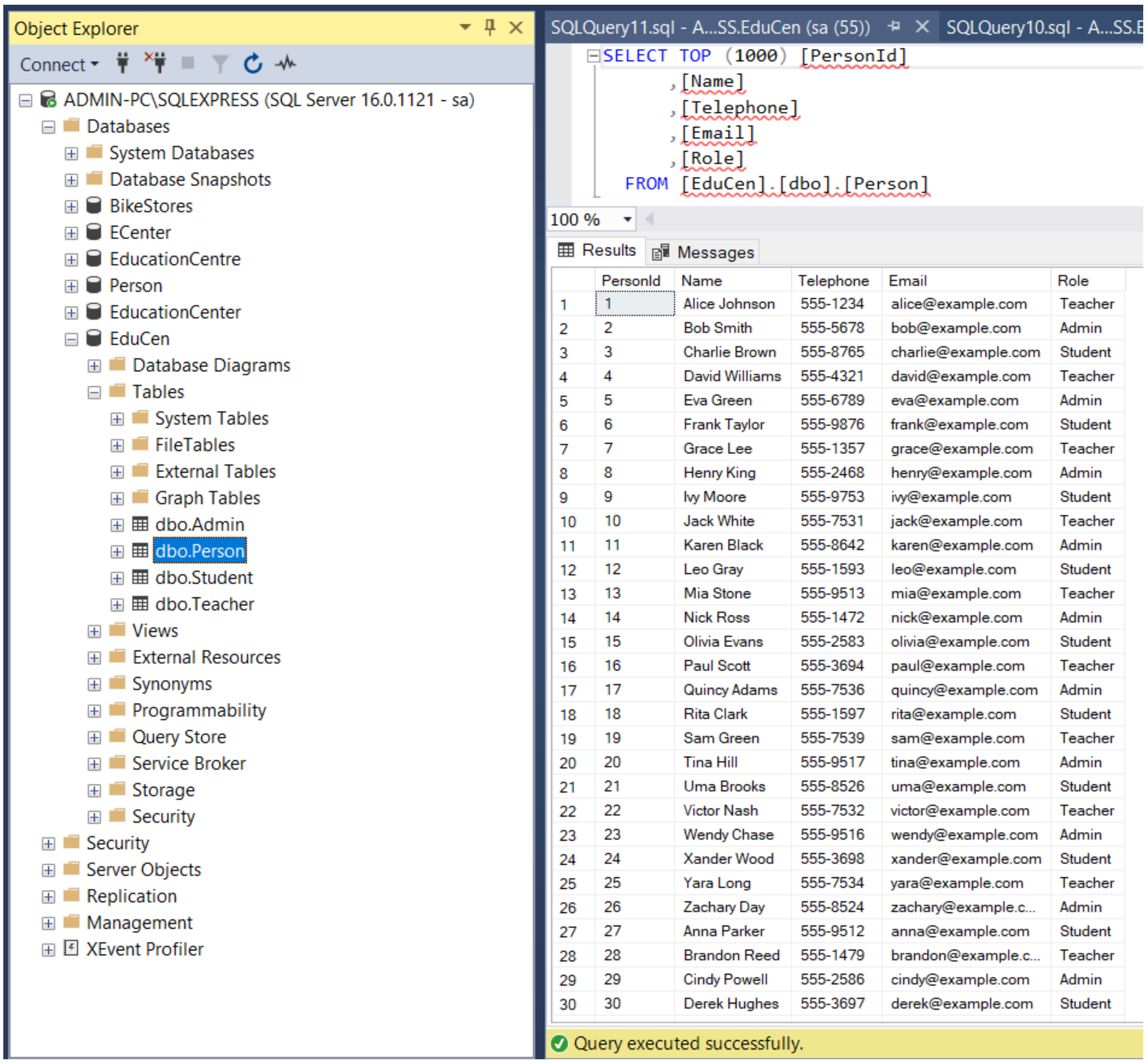


```sql
(40, 55000, 'Technology', 'Engineering'),
(43, 51000, 'Political Science', 'History');

-- Inserting 15 rows into the Admin table
INSERT INTO Admin (PersonId, Salary, EmploymentType, WorkingHours) V
(2, 60000, 'Full-time', 40),
(5, 62000, 'Part-time', 20),
(8, 58000, 'Full-time', 40),
(11, 61000, 'Full-time', 35),
(14, 59000, 'Part-time', 25),
(17, 60500, 'Full-time', 40),
(20, 62500, 'Full-time', 38),
(23, 64500, 'Part-time', 20),
(26, 61500, 'Full-time', 36),
(29, 63000, 'Full-time', 40),
(32, 60000, 'Part-time', 30),
(35, 62000, 'Full-time', 37),
(38, 58500, 'Full-time', 39),
(41, 63500, 'Full-time', 40),
(44, 61000, 'Part-time', 25);

-- Inserting 15 rows into the Student table
INSERT INTO Student (PersonId, CurrentSubject1, CurrentSubject2, Pre
(3, 'Math', 'Science', 'History', 'Geography'),
(6, 'Biology', 'Chemistry', 'English', 'Literature'),
(9, 'Physics', 'Math', 'Art', 'Music'),
(12, 'History', 'Political Science', 'Economics', 'Business Studies'
(15, 'Technology', 'Engineering', 'Philosophy', 'Psychology'),
(18, 'Physical Education', 'Health', 'Music', 'Drama'),
(21, 'Computer Science', 'Math', 'Social Studies', 'Civics'),
```

```
(45 rows affected)

(15 rows affected)

(15 rows affected)

(15 rows affected)
```

Query executed successfully.

Person table, same apply for other tables


Object Explorer
Connect
SQLQuery11.sql - A...SS.EduCen (sa (55))
SQLQuery10.sql - A...SS.E
100 %
Results
Messages

- ADMIN-PC\SQLEXPRESS (SQL Server 16.0.1121 - sa)
  - Databases
    - System Databases
    - Database Snapshots
    - BikeStores
    - ECenter
    - EducationCentre
    - Person
    - EducationCenter
    - EduCen
      - Database Diagrams
      - Tables
        - System Tables
        - FileTables
        - External Tables
        - Graph Tables
        - dbo.Admin
        - dbo.Person
        - dbo.Student
        - dbo.Teacher
      - Views
      - External Resources
      - Synonyms
      - Programmability
      - Query Store
      - Service Broker
      - Storage
      - Security
  - Security
  - Server Objects
  - Replication
  - Management
  - XEvent Profiler

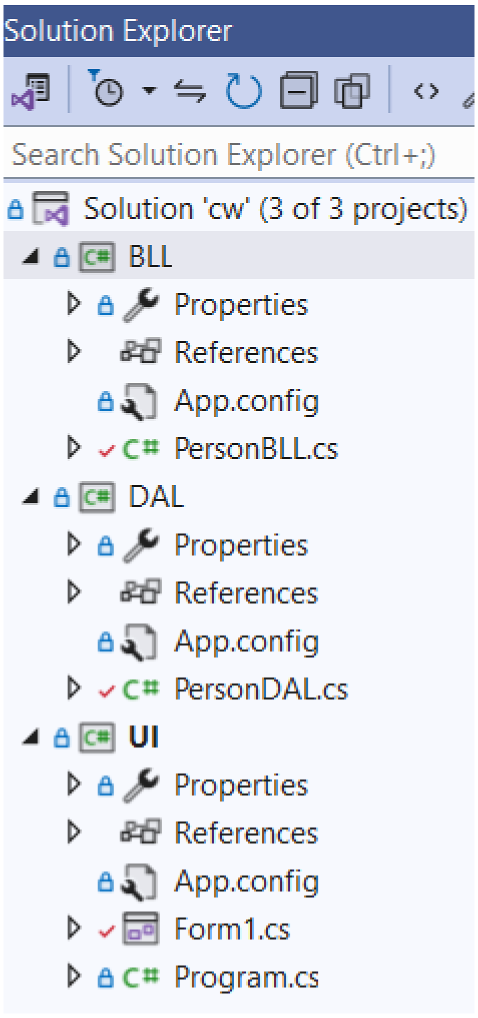
```sql
SELECT TOP (1000) [PersonId]
      ,[Name]
      ,[Telephone]
      ,[Email]
      ,[Role]
  FROM [EduCen].[dbo].[Person]
```

| | PersonId | Name | Telephone | Email | Role |
|---|---|---|---|---|---|
| 1 | 1 | Alice Johnson | 555-1234 | alice@example.com | Teacher |
| 2 | 2 | Bob Smith | 555-5678 | bob@example.com | Admin |
| 3 | 3 | Charlie Brown | 555-8765 | charlie@example.com | Student |
| 4 | 4 | David Williams | 555-4321 | david@example.com | Teacher |
| 5 | 5 | Eva Green | 555-6789 | eva@example.com | Admin |
| 6 | 6 | Frank Taylor | 555-9876 | frank@example.com | Student |
| 7 | 7 | Grace Lee | 555-1357 | grace@example.com | Teacher |
| 8 | 8 | Henry King | 555-2468 | henry@example.com | Admin |
| 9 | 9 | Ivy Moore | 555-9753 | ivy@example.com | Student |
| 10 | 10 | Jack White | 555-7531 | jack@example.com | Teacher |
| 11 | 11 | Karen Black | 555-8642 | karen@example.com | Admin |
| 12 | 12 | Leo Gray | 555-1593 | leo@example.com | Student |
| 13 | 13 | Mia Stone | 555-9513 | mia@example.com | Teacher |
| 14 | 14 | Nick Ross | 555-1472 | nick@example.com | Admin |
| 15 | 15 | Olivia Evans | 555-2583 | olivia@example.com | Student |
| 16 | 16 | Paul Scott | 555-3694 | paul@example.com | Teacher |
| 17 | 17 | Quincy Adams | 555-7536 | quincy@example.com | Admin |
| 18 | 18 | Rita Clark | 555-1597 | rita@example.com | Student |
| 19 | 19 | Sam Green | 555-7539 | sam@example.com | Teacher |
| 20 | 20 | Tina Hill | 555-9517 | tina@example.com | Admin |
| 21 | 21 | Uma Brooks | 555-8526 | uma@example.com | Student |
| 22 | 22 | Victor Nash | 555-7532 | victor@example.com | Teacher |
| 23 | 23 | Wendy Chase | 555-9516 | wendy@example.com | Admin |
| 24 | 24 | Xander Wood | 555-3698 | xander@example.com | Student |
| 25 | 25 | Yara Long | 555-7534 | yara@example.com | Teacher |
| 26 | 26 | Zachary Day | 555-8524 | zachary@example.c... | Admin |
| 27 | 27 | Anna Parker | 555-9512 | anna@example.com | Student |
| 28 | 28 | Brandon Reed | 555-1479 | brandon@example.c... | Teacher |
| 29 | 29 | Cindy Powell | 555-2586 | cindy@example.com | Admin |
| 30 | 30 | Derek Hughes | 555-3697 | derek@example.com | Student |

Query executed successfully.

## Code C#

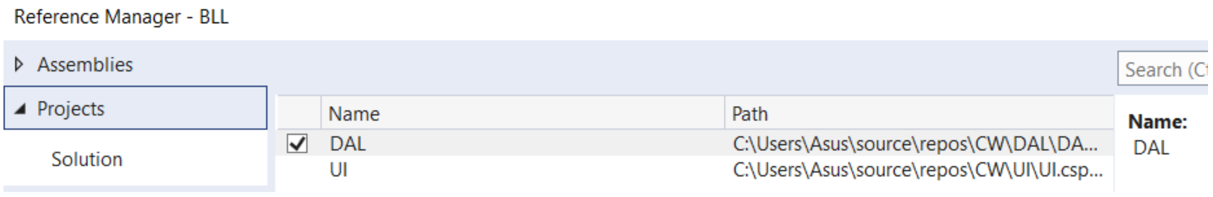
I built a 3-tier architecture


Solution Explorer

Search Solution Explorer (Ctrl+;)

```
Solution 'cw' (3 of 3 projects)
  BLL
    Properties
    References
    App.config
    PersonBLL.cs
  DAL
    Properties
    References
    App.config
    PersonDAL.cs
  UI
    Properties
    References
    App.config
    Form1.cs
    Program.cs
```

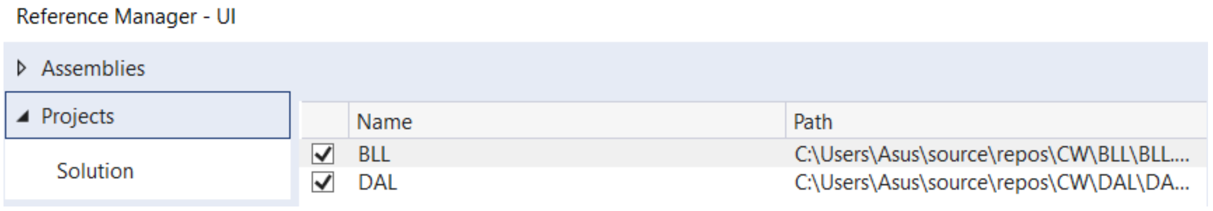
BLL was added reference with DAL


Reference Manager - BLL

| | Name | Path |
|---|---|---|
| ✓ | DAL | C:\Users\Asus\source\repos\CW\DAL\DA... |
| | UI | C:\Users\Asus\source\repos\CW\UI\UI.csp... |

**Name:** DAL

UI was added references with BLL,  DAL


Reference Manager - UI

| | Name | Path |
|---|---|---|
| ✓ | BLL | C:\Users\Asus\source\repos\CW\BLL\BLL.... |
| ✓ | DAL | C:\Users\Asus\source\repos\CW\DAL\DA... |

*Code in Form1.cs:*

```
using System;
using System.Linq;
using System.Windows.Forms;
```

```
namespace CW
{
    public partial class Form1 : Form
    {
        // Boolean flags to track if specific fields have been changed by the
user
        private bool isNameChanged = false;
        private bool isTelephoneChanged = false;
        private bool isEmailChanged = false;
        private bool isRoleChanged = false;

        private bool isSalaryChanged = false;
        private bool isSubject1Changed = false;
        private bool isSubject2Changed = false;

        private bool isSalaryAChanged = false;
        private bool isEmploymentTypeChanged = false;
        private bool isWorkingHoursChanged = false;

        private bool isCurrentSubject1Changed = false;
        private bool isCurrentSubject2Changed = false;
        private bool isPreviousSubject1Changed = false;
        private bool isPreviousSubject2Changed = false;

        // Constructor initializes the form and attaches event handlers
        public Form1()
        {
            InitializeComponent();

            // Event handler for when the selected row in the DataGridView
changes
            this.PersonDGV.SelectionChanged += new
System.EventHandler(this.PersonDGV_SelectionChanged);

            // Attach event handlers to TextBox TextChanged events to track
changes
            nameTB.TextChanged += nameTB_TextChanged;
            telephoneTB.TextChanged += telephoneTB_TextChanged;
            emailTB.TextChanged += emailTB_TextChanged;

            teacherLb.TextChanged += teacherLb_TextChanged;
            adminLb.TextChanged += adminLb_TextChanged;
            stulb.TextChanged += stulb_TextChanged;

            salaryTB.TextChanged += salaryTB_TextChanged;
            subject1TB.TextChanged += subject1TB_TextChanged;
            subject2TB.TextChanged += subject2TB_TextChanged;
            salaryAtb.TextChanged += salaryAtb_TextChanged;
            emptyptb.TextChanged += emptyptb_TextChanged;
            worhoutb.TextChanged += worhoutb_TextChanged;

            currentsubject1tb.TextChanged += currentsubject1tb_TextChanged;
            currentsubject2tb.TextChanged += currentsubject2tb_TextChanged;
            previoussubject1tb.TextChanged += previoussubject1tb_TextChanged;
            previoussubject2tb.TextChanged += previoussubject2tb_TextChanged;
        }

        // Event handler methods that set the corresponding flag to true when
a TextBox value changes
        private void nameTB_TextChanged(object sender, EventArgs e) =>
isNameChanged = true;
        private void telephoneTB_TextChanged(object sender, EventArgs e) =>
isTelephoneChanged = true;
        private void emailTB_TextChanged(object sender, EventArgs e) =>
isEmailChanged = true;

        private void teacherLb_TextChanged(object sender, EventArgs e) =>
isRoleChanged = true;
        private void adminLb_TextChanged(object sender, EventArgs e) =>
isRoleChanged = true;
        private void stulb_TextChanged(object sender, EventArgs e) =>
isRoleChanged = true;

        private void salaryTB_TextChanged(object sender, EventArgs e) =>
isSalaryChanged = true;
        private void subject1TB_TextChanged(object sender, EventArgs e) =>
isSubject1Changed = true;
        private void subject2TB_TextChanged(object sender, EventArgs e) =>
isSubject2Changed = true;

        private void salaryAtb_TextChanged(object sender, EventArgs e) =>
isSalaryAChanged = true;
        private void emptyptb_TextChanged(object sender, EventArgs e) =>
isEmploymentTypeChanged = true;
        private void worhoutb_TextChanged(object sender, EventArgs e) =>
isWorkingHoursChanged = true;

        private void currentsubject1tb_TextChanged(object sender, EventArgs e)
=> isCurrentSubject1Changed = true;
        private void currentsubject2tb_TextChanged(object sender, EventArgs e)
=> isCurrentSubject2Changed = true;
        private void previoussubject1tb_TextChanged(object sender, EventArgs
e) => isPreviousSubject1Changed = true;
        private void previoussubject2tb_TextChanged(object sender, EventArgs
e) => isPreviousSubject2Changed = true;
```

```
        // Method that runs when the form loads, setting up the DataGridView
with data
        private void Form1_Load(object sender, EventArgs e)
        {
            try
            {
                // Create an instance of the business logic layer and set the
data source of the DataGridView
                PersonBLL p = new PersonBLL();
                this.PersonDGV.DataSource = p.GetPersons();
            }
            catch
            {
                // Display an error message if something goes wrong during
data loading
                MessageBox.Show("Error Occurred");
            }
        }


        // This method handles the SelectionChanged event of the PersonDGV
DataGridView.
        // It triggers when the selection of rows changes in the DataGridView.
        private void PersonDGV_SelectionChanged(object sender, EventArgs e)
        {
            // Check if at least one row is selected
            if (PersonDGV.SelectedRows.Count > 0)
            {
                // Get the selected row
                DataGridViewRow selectedRow = PersonDGV.SelectedRows[0];

                // Retrieve the PersonId and Role values from the selected row
                var personId = selectedRow.Cells["PersonId"].Value;
                var personRole = selectedRow.Cells["Role"].Value;

                // Create an instance of the PersonBLL class to interact with
the business logic layer
                PersonBLL p = new PersonBLL();

                // Set the DataSource of PersonDGV to the data returned by
GetPersonInfo method
                this.PersonDGV.DataSource = p.GetPersonInfo(personId,
personRole);
            }
        }

        // This method handles the Click event of the LoadDataInfoBtn button.
        // It is used to load person data based on the input provided in the
LoadDataInfoTxtBox.
        private void LoadDataInfoBtn_Click(object sender, EventArgs e)
        {
            try
            {
                // Validate that the input textbox is not empty or whitespace
                if (string.IsNullOrWhiteSpace(this.LoadDataInfoTxtBox.Text))
                {
                    // Display a message if the input is invalid
                    MessageBox.Show("Please enter a valid PersonId or Name.");
                    return;
                }

                // Create an instance of the PersonBLL class to interact with
the business logic layer
                PersonBLL p = new PersonBLL();

                // Check if the input is numeric (i.e., a Person ID)
                if (this.LoadDataInfoTxtBox.Text.All(Char.IsDigit))
                {
                    // Load data based on Person ID
                    this.PersonDGV.DataSource =
p.GetPerson(Convert.ToInt16(this.LoadDataInfoTxtBox.Text));
                }
                else
                {
                    // Load data based on Person Name
                    this.PersonDGV.DataSource =
p.GetPersons(this.LoadDataInfoTxtBox.Text);
                }
            }
            catch
            {
                // Display an error message if an exception occurs
                MessageBox.Show("Error Occurred");
            }
        }

        // This method handles the Click event of the ViewAllDataBtn button.
        // It loads all person data into the DataGridView.
        private void ViewAllDataBtn_Click(object sender, EventArgs e)
        {
            try
            {
                // Create an instance of the PersonBLL class to interact with
the business logic layer
                PersonBLL p = new PersonBLL();
                // Load all person data
                this.PersonDGV.DataSource = p.GetPersons();
            }
            catch
            {
                // Display an error message if an exception occurs
                MessageBox.Show("Error Occurred");
            }
        }

        // This method handles the Click event of the ViewTeacherBtn button.
        // It loads all teacher data into the DataGridView.
        private void ViewTeacherBtn_Click(object sender, EventArgs e)
        {
            try
            {
                // Create an instance of the PersonBLL class to interact with
the business logic layer
                PersonBLL p = new PersonBLL();

                // Load teacher data
                this.PersonDGV.DataSource = p.GetTeachers();
            }
            catch
            {
                // Display an error message if an exception occurs
                MessageBox.Show("Error Occurred");
            }
        }

        // This method handles the Click event of the ViewAdminBtn button.
        // It loads all administrator data into the DataGridView.
        private void ViewAdminBtn_Click(object sender, EventArgs e)
        {
            try
            {
                // Create an instance of the PersonBLL class to interact with
the business logic layer
                PersonBLL p = new PersonBLL();

                // Load administrator data
                this.PersonDGV.DataSource = p.GetAdmins();
            }
            catch
            {
                // Display an error message if an exception occurs
                MessageBox.Show("Error Occurred");
            }
        }
        // Event handler for the "View Student" button click
        private void ViewStudentBtn_Click(object sender, EventArgs e)
        {
            try
            {
                // Create an instance of the PersonBLL class
                PersonBLL p = new PersonBLL();

                // Retrieve the list of students and bind it to the
DataGridView
                this.PersonDGV.DataSource = p.GetStudents();
            }
            catch
            {
                // Show an error message if something goes wrong
                MessageBox.Show("Error Occurred");
            }
        }

        // Event handler for the "Add Teacher" button click
        private void AddTeacherBtn_Click(object sender, EventArgs e)
        {
            try
            {
                // Collect input data from TextBoxes and other UI elements
                string name = nameTB.Text;
                string telephone = telephoneTB.Text;
                string email = emailTB.Text;
                string role = teacherLb.Text;
                int salary = Convert.ToInt32(salaryTB.Text);
                string subject1 = subject1TB.Text;
                string subject2 = subject2TB.Text;

                // Create an instance of the PersonBLL class
                PersonBLL p = new PersonBLL();

                // Add the person to the database and get the generated
PersonId
                int personId = p.AddPerson(name, telephone, email, role);

                // Use the PersonId to add the teacher-specific details
                p.AddTeacher(personId, salary, subject1, subject2);

                // Refresh the DataGridView to display the updated list of
teachers
                this.PersonDGV.DataSource = p.GetTeachers();
                // Show a success message upon completion
                MessageBox.Show("Teacher added successfully");
            }
            catch (Exception ex)
            {
                // Show an error message if something goes wrong, including
the exception message
                MessageBox.Show("Error Occurred: " + ex.Message);
            }
        }

        // Event handler for the "Add Admin" button click
        private void AddAdminBtn_Click(object sender, EventArgs e)
        {
            try
            {
                // Collect input data from TextBoxes and other UI elements
                string name = nameTB.Text;
                string telephone = telephoneTB.Text;
                string email = emailTB.Text;
                string role = adminLb.Text;
                int salary = Convert.ToInt32(salaryAtb.Text);
                string employmenttype = emptyptb.Text;
                int workinghours = Convert.ToInt32(worhoutb.Text);

                // Create an instance of the PersonBLL class
                PersonBLL p = new PersonBLL();

                // Add the person to the database and get the generated
PersonId
                int personId = p.AddPerson(name, telephone, email, role);

                // Use the PersonId to add the admin-specific details
                p.AddAdmin(personId, salary, employmenttype, workinghours);

                // Refresh the DataGridView to display the updated list of
admins
                this.PersonDGV.DataSource = p.GetAdmins();

                // Show a success message upon completion
                MessageBox.Show("Admin added successfully");
            }
            catch (Exception ex)
            {
                // Show an error message if something goes wrong, including
the exception message
                MessageBox.Show("Error Occurred: " + ex.Message);
            }
        }
        // Event handler for adding a student when the "Add Student" button is
clicked
        private void addStudentBtn_Click(object sender, EventArgs e)
        {
            try
            {
                // Retrieve input data from text boxes
                string name = nameTB.Text;
                string telephone = telephoneTB.Text;
                string email = emailTB.Text;
                string role = stulb.Text;
                string currentsubject1 = currentsubject1tb.Text;
                string currentsubject2 = currentsubject2tb.Text;
                string previoussubject1 = previoussubject1tb.Text;
                string previoussubject2 = previoussubject2tb.Text;

                // Create a new PersonBLL object to interact with the business
logic layer
                PersonBLL p = new PersonBLL();

                // Add the person to the database and retrieve the generated
PersonId
                int personId = p.AddPerson(name, telephone, email, role);

                // Add the student with the provided PersonId and subject
information
                p.AddStudent(personId, currentsubject1, currentsubject2,
previoussubject1, previoussubject2);

                // Update the DataGridView to display the list of students
                this.PersonDGV.DataSource = p.GetStudents();

                // Display a success message to the user
                MessageBox.Show("Student added successfully");
            }
            catch (Exception ex)
            {
                // Handle any errors that occur during the process and display
an error message
                MessageBox.Show("Error Occurred: " + ex.Message);
            }
        }

        // Event handler for editing a teacher when the "Edit Teacher" button
is clicked
        private void editTeacherBtn_Click(object sender, EventArgs e)
        {
            try
            {
                // Validate that the PersonId is provided and is numeric
                if (string.IsNullOrWhiteSpace(editIDtb.Text) ||
!int.TryParse(editIDtb.Text, out int personId))
                {
                    MessageBox.Show("Please enter a valid PersonId.");
                    return;
                }

                // Retrieve updated data from text boxes based on whether
fields have been changed
                string name = isNameChanged ? nameTB.Text : null;
                string telephone = isTelephoneChanged ? telephoneTB.Text :
null;
                string email = isEmailChanged ? emailTB.Text : null;
                string role = isRoleChanged ? teacherLb.Text : null;
                int salary = Convert.ToInt32(salaryTB.Text);
                string subject1 = isSubject1Changed ? subject1TB.Text : null;
                string subject2 = isSubject2Changed ? subject2TB.Text : null;

                // Create a new PersonBLL object to interact with the business
logic layer
                PersonBLL p = new PersonBLL();

                // Update the person's basic details in the database
                p.EditPerson(personId, name, telephone, email, role);

                // Update the teacher's specific details, including salary and
subjects
                p.EditTeacher(personId, salary, subject1, subject2);

                // Ensure at least one field is provided for updating,
otherwise notify the user
                if (!isNameChanged && !isTelephoneChanged && !isEmailChanged
&& !isRoleChanged && !isSalaryChanged && !isSubject1Changed &&
!isSubject2Changed)
                {
                    MessageBox.Show("No fields to update.");
                    return;
                }

                // Update the DataGridView to display the list of teachers
                this.PersonDGV.DataSource = p.GetTeachers();

                // Display a success message to the user
                MessageBox.Show("Teacher edited successfully");
            }
            catch (Exception ex)
            {
                // Handle any errors that occur during the process and display
an error message
                MessageBox.Show("Error Occurred: " + ex.Message);
            }
        }
        private void editAdminBtn_Click(object sender, EventArgs e)
        {
            try
            {
                // Validate the input for the admin's PersonId, ensuring it's
not empty and contains only numeric characters.
                if (string.IsNullOrWhiteSpace(editIdAdminTb.Text) ||
!int.TryParse(editIdAdminTb.Text, out int personId))
                {
                    MessageBox.Show("Please enter a valid PersonId.");
                    return; // Exit if validation fails.
                }

                // Initialize variables with input values if they are marked
as changed, otherwise set to null.
                string name = isNameChanged ? nameTB.Text : null;
                string telephone = isTelephoneChanged ? telephoneTB.Text :
null;
                string email = isEmailChanged ? emailTB.Text : null;
                string role = isRoleChanged ? adminLb.Text : null;
                int salary = Convert.ToInt32(salaryAtb.Text); // Convert
salary input to int.
                string employmenttype = isEmploymentTypeChanged ?
emptyptb.Text : null;
                int workinghours = Convert.ToInt32(worhoutb.Text); // Convert
working hours input to int.

                // Create an instance of PersonBLL to handle business logic.
                PersonBLL p = new PersonBLL();

                // Call method to edit general person details.
                p.EditPerson(personId, name, telephone, email, role);

                // Call method to edit specific admin details.
                p.EditAdmin(personId, salary, employmenttype, workinghours);

                // If no fields are provided for updating, display a message
and exit.
                if (name == null && telephone == null && email == null && role
== null && salary == 0 && employmenttype == null && workinghours == 0)
```

```
                {
                    MessageBox.Show("No fields to update.");
                    return;
                }

                // Update the DataGridView to reflect changes to the list of
admins.
                this.PersonDGV.DataSource = p.GetAdmins();

                MessageBox.Show("Admin edited successfully"); // Notify the
user of success.
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error Occurred: " + ex.Message); // Display
any exceptions that occur.
            }
        }

        private void editStudentBtn_Click(object sender, EventArgs e)
        {
            try
            {
                // Validate the input for the student's PersonId, ensuring
it's not empty and contains only numeric characters.
                if (string.IsNullOrWhiteSpace(editStuIdTb.Text) ||
!int.TryParse(editStuIdTb.Text, out int personId))
                {
                    MessageBox.Show("Please enter a valid PersonId.");
                    return; // Exit if validation fails.
                }

                // Initialize variables with input values if they are marked
as changed, otherwise set to null.
                string name = isNameChanged ? nameTB.Text : null;
                string telephone = isTelephoneChanged ? telephoneTB.Text :
null;
                string email = isEmailChanged ? emailTB.Text : null;
                string role = isRoleChanged ? stulb.Text : null;
                string currentsubject1 = isCurrentSubject1Changed ?
currentsubject1tb.Text : null;
                string currentsubject2 = isCurrentSubject2Changed ?
currentsubject2tb.Text : null;
                string previoussubject1 = isPreviousSubject1Changed ?
previoussubject1tb.Text : null;
                string previoussubject2 = isPreviousSubject2Changed ?
previoussubject2tb.Text : null;
```

```
                // Create an instance of PersonBLL to handle business logic.
                PersonBLL p = new PersonBLL();

                // Call method to edit general person details.
                p.EditPerson(personId, name, telephone, email, role);

                // Call method to edit specific student details.
                p.EditStudent(personId, currentsubject1, currentsubject2,
previoussubject1, previoussubject2);

                // If no fields are provided for updating, display a message
and exit.
                if (name == null && telephone == null && email == null && role
== null && currentsubject1 == null && currentsubject2 == null &&
previoussubject1 == null && previoussubject2 == null)
                {
                    MessageBox.Show("No fields to update.");
                    return;
                }

                // Update the DataGridView to reflect changes to the list of
students.
                this.PersonDGV.DataSource = p.GetStudents();

                MessageBox.Show("Student edited successfully"); // Notify the
user of success.
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error Occurred: " + ex.Message); // Display
any exceptions that occur.
            }
        }

        private void deleteBtn_Click(object sender, EventArgs e)
        {
            try
            {
                // Validate the input for the PersonId to be deleted, ensuring
it's not empty and contains only numeric characters.
                if (string.IsNullOrWhiteSpace(deleteTb.Text) ||
!int.TryParse(deleteTb.Text, out int personId))
                {
                    MessageBox.Show("Please enter a valid PersonId.");
                    return; // Exit if validation fails.
                }

                // Create an instance of PersonBLL to handle business logic.
                PersonBLL p = new PersonBLL();

                // Call method to delete the person from the system.
                p.DeletePerson(personId);

                // Update the DataGridView to reflect changes to the list of
persons.
                this.PersonDGV.DataSource = p.GetPersons();

                MessageBox.Show("Person deleted successfully"); // Notify the
user of success.
            }
            catch
            {
                MessageBox.Show("Error Occurred"); // Display a general error
message in case of exceptions.
            }
        }
    }
}
```

*Code in PersonBLL.cs:*

```
using System;
using System.Collections.Generic;
using System.Data;

// Aliases for different roles to avoid namespace collisions.
using Person = CW.PersonDAL.Person;
using Teacher = CW.PersonDAL.Teacher;
using Admin = CW.PersonDAL.Admin;
using Student = CW.PersonDAL.Student;

namespace CW
{
    // Business Logic Layer (BLL) class for managing Person entities.
    public class PersonBLL
    {
        // List to store person objects.
        private List<Person> persons = new List<Person>();
        // Data Access Layer (DAL) instance to interact with the database.
        private PersonDAL personDAL = new PersonDAL();

        static void Main(string[] args)
        {
            // Instantiate PersonBLL and demonstrate its methods.
            PersonBLL personBLL = new PersonBLL();
            // Retrieve and display all persons from the database.
            DataTable allPersons = personBLL.GetPersons();
            Console.WriteLine("All persons data loaded.");

            // Retrieve and display person data by ID.
            DataTable personById = personBLL.GetPerson(1);
            Console.WriteLine("Person with ID 1 data loaded.");

            // Retrieve and display person data by Name.
            DataTable personByName = personBLL.GetPersons("Tom");
            Console.WriteLine("Person with Name Tom data loaded.");
        }

        // Method to retrieve all persons from the database.
        public DataTable GetPersons()
        {
            try
            {
                PersonDAL objdal = new PersonDAL(); // DAL instance.
                return objdal.Read(); // Fetch and return all persons.
            }
            catch
            {
                // Rethrow the exception for the calling method to handle.
                throw;
            }
        }

        // Method to retrieve a person by ID.
        public DataTable GetPerson(Int16 ID)
        {
            try
            {
                PersonDAL objdal = new PersonDAL(); // DAL instance.
                return objdal.Read(ID); // Fetch and return person by ID.
            }
            catch
            {
                // Rethrow the exception for the calling method to handle.
                throw;
            }
        }

        // Method to retrieve persons by their name.
        public DataTable GetPersons(String NAME)
        {
            try
            {
                PersonDAL objdal = new PersonDAL(); // DAL instance.
                return objdal.ReadByName(NAME); // Fetch and return persons by
name.
            }
            catch
            {
                // Rethrow the exception for the calling method to handle.
                throw;
            }
        }

        // Method to retrieve specific person information based on ID and
role.
        public DataTable GetPersonInfo(object personId, object personRole)
        {
            try
            {
                PersonDAL objdal = new PersonDAL(); // DAL instance.
                return objdal.ReadPersonInfo(personId, personRole); // Fetch
and return person info by ID and role.
            }
            catch
            {
                // Rethrow the exception for the calling method to handle.
                throw;
            }
        }

        // Method to add a new person to the database.
        public int AddPerson(string name, string telephone, string email,
string role)
        {
            try
            {
                PersonDAL objdal = new PersonDAL(); // DAL instance.
                return objdal.CreatePerson(name, telephone, email, role); //
Add a new person and return the new person ID.
            }
            catch
            {
                // Rethrow the exception for the calling method to handle.
                throw;
            }
        }


        // Method to add a new Teacher to the system
        public void AddTeacher(int personId, int salary, string subject1,
string subject2)
```

```
        {
            try
            {
                // Initialize a new Teacher object with provided details
                Teacher teacher = new Teacher
                {
                    PersonId = personId,
                    Salary = salary,
                    Subject1 = subject1,
                    Subject2 = subject2
                };

                // Instantiate the data access layer object
                PersonDAL objdal = new PersonDAL();

                // Call the method to create a new Teacher record in the
database
                objdal.CreateTeacher(personId, salary, subject1, subject2);
            }
            catch (Exception ex)
            {
                // Handle any exceptions that occur during the process and
throw a custom exception message
                throw new Exception("Error adding teacher: " + ex.Message);
            }
        }

        // Method to edit an existing Teacher's information
        public void EditTeacher(int personId, int salary, string subject1,
string subject2)
        {
            try
            {
                // Initialize a Teacher object with updated details
                Teacher teacher = new Teacher
                {
                    PersonId = personId,
                    Salary = salary,
                    Subject1 = subject1,
                    Subject2 = subject2
                };

                // Instantiate the data access layer object
                PersonDAL objdal = new PersonDAL();

                // Call the method to update the Teacher's record in the
database
                objdal.UpdateTeacher(personId, salary, subject1, subject2);
            }
            catch (Exception ex)
            {
                // Handle any exceptions that occur during the process and
throw a custom exception message
                throw new Exception("Error editing teacher: " + ex.Message);
            }
        }

        // Method to edit an existing Admin's information
        public void EditAdmin(int personId, int salary, string employmenttype,
int workinghours)
        {
            try
            {
                // Initialize an Admin object with updated details
                Admin admin = new Admin
                {
                    PersonId = personId,
                    Salary = salary,
                    EmploymentType = employmenttype,
                    WorkingHours = workinghours
                };

                // Instantiate the data access layer object
                PersonDAL objdal = new PersonDAL();

                // Call the method to update the Admin's record in the
database
                objdal.UpdateAdmin(personId, salary, employmenttype,
workinghours);
            }
            catch (Exception ex)
            {
                // Handle any exceptions that occur during the process and
throw a custom exception message
                throw new Exception("Error editing admin: " + ex.Message);
            }
        }

        // Method to edit an existing Student's information
        public void EditStudent(int personId, string currentsubject1, string
currentsubject2, string previoussubject1, string previoussubject2)
        {
            try
            {
                // Initialize a Student object with updated details
                Student student = new Student
                {
                    PersonId = personId,
                    CurrentSubject1 = currentsubject1,
                    CurrentSubject2 = currentsubject2,
                    PreviousSubject1 = previoussubject1,
                    PreviousSubject2 = previoussubject2
                };

                // Instantiate the data access layer object
                PersonDAL objdal = new PersonDAL();

                // Call the method to update the Student's record in the
database
                objdal.UpdateStudent(personId, currentsubject1,
currentsubject2, previoussubject1, previoussubject2);
            }
            catch (Exception ex)
            {
                // Handle any exceptions that occur during the process and
throw a custom exception message
                throw new Exception("Error editing student: " + ex.Message);
            }
        }

        // Method to add a new Admin to the system
        public void AddAdmin(int personId, int salary, string employmenttype,
int workinghours)
        {
            try
            {
                // Initialize a new Admin object with provided details
                Admin admin = new Admin
                {
                    PersonId = personId,
                    Salary = salary,
                    EmploymentType = employmenttype,
                    WorkingHours = workinghours
                };

                // Instantiate the data access layer object
                PersonDAL objdal = new PersonDAL();

                // Call the method to create a new Admin record in the
database
                objdal.CreateAdmin(personId, salary, employmenttype,
workinghours);
            }
            catch (Exception ex)
            {
                // Handle any exceptions that occur during the process and
throw a custom exception message
                throw new Exception("Error adding admin: " + ex.Message);
            }
        }

        // Adds a new student to the system with the given details.
        public void AddStudent(int personId, string currentsubject1, string
currentsubject2, string previoussubject1, string previoussubject2)
        {
            try
            {
                // Create a new Student object with the provided details.
                Student student = new Student
                {
                    PersonId = personId,
                    CurrentSubject1 = currentsubject1,
                    CurrentSubject2 = currentsubject2,
                    PreviousSubject1 = previoussubject1,
                    PreviousSubject2 = previoussubject2
                };

                // Instantiate the data access layer (DAL) to interact with
the database.
                PersonDAL objdal = new PersonDAL();

                // Use the DAL to create a new student record in the database.
                objdal.CreateStudent(personId, currentsubject1,
currentsubject2, previoussubject1, previoussubject2);
            }
            catch (Exception ex)
            {
                // Catch any exceptions and throw a new exception with a
descriptive message.
                throw new Exception("Error adding student: " + ex.Message);
            }
        }

        // Method to retrieve a list of teachers from the data access layer
        public List<Teacher> GetTeachers()
        {
            try
            {
                // Instantiate the data access layer (DAL) to interact with
the database.
                PersonDAL objdal = new PersonDAL();
                // Use the DAL to read teacher records from the database.
                DataTable teacherTable = objdal.ReadTeachers();

                // Initialize a list to hold the teacher objects.
                List<Teacher> teachers = new List<Teacher>();

                // Iterate through each row in the DataTable and create a
Teacher object.
                foreach (DataRow row in teacherTable.Rows)
                {
                    Teacher teacher = new Teacher
                    {
                        PersonId = Convert.ToInt32(row["PersonId"]),
                        Name = row["Name"].ToString(),
                        Telephone = row["Telephone"].ToString(),
                        Email = row["Email"].ToString(),
                        Role = row["Role"].ToString(),
                        Salary = Convert.ToInt32(row["Salary"]),
                        Subject1 = row["Subject1"].ToString(),
                        Subject2 = row["Subject2"].ToString()
                    };

                    // Add the Teacher object to the list.
                    teachers.Add(teacher);
                }

                // Return the list of teachers.
                return teachers;
            }
            catch (Exception ex)
            {
                // Catch any exceptions and throw a new exception with a
descriptive message.
                throw new Exception("Error retrieving teachers: " +
ex.Message);
            }
        }

        // Method to retrieve a list of admins from the data access layer
        public List<Admin> GetAdmins()
        {
            try
            {
                // Instantiate the data access layer (DAL) to interact with
the database.
                PersonDAL objdal = new PersonDAL();

                // Use the DAL to read admin records from the database.
                DataTable adminTable = objdal.ReadAdmins();

                // Initialize a list to hold the admin objects.
                List<Admin> admins = new List<Admin>();

                // Iterate through each row in the DataTable and create an
Admin object.
                foreach (DataRow row in adminTable.Rows)
                {
                    Admin admin = new Admin
                    {
                        PersonId = Convert.ToInt32(row["PersonId"]),
                        Name = row["Name"].ToString(),
                        Telephone = row["Telephone"].ToString(),
                        Email = row["Email"].ToString(),
                        Role = row["Role"].ToString(),
                        Salary = Convert.ToInt32(row["Salary"]),
                        EmploymentType = row["EmploymentType"].ToString(),
                        WorkingHours = Convert.ToInt32(row["WorkingHours"])
                    };

                    // Add the Admin object to the list.
                    admins.Add(admin);
                }

                // Return the list of admins.
                return admins;
            }
            catch (Exception ex)
            {
                // Catch any exceptions and throw a new exception with a
descriptive message.
                throw new Exception("Error retrieving admins: " + ex.Message);
            }
        }

        // Method to retrieve a list of students from the data access layer
        public List<Student> GetStudents()
        {
            try
            {
                // Create an instance of the data access layer class to
interact with the database
                PersonDAL objdal = new PersonDAL();

                // Retrieve the data table containing student information from
the database
                DataTable studentTable = objdal.ReadStudents();
                // Create a list to hold the student objects
                List<Student> students = new List<Student>();

                // Iterate through each row in the data table
                foreach (DataRow row in studentTable.Rows)
                {
                    // Create a new student object and populate it with data
from the current row
                    Student student = new Student
                    {
                        PersonId = Convert.ToInt32(row["PersonId"]), //
Convert PersonId to integer
                        Name = row["Name"].ToString(),              // Get
student name
                        Telephone = row["Telephone"].ToString(),     // Get
student telephone
                        Email = row["Email"].ToString(),             // Get
student email
                        Role = row["Role"].ToString(),              // Get
student role
                        CurrentSubject1 = row["CurrentSubject1"].ToString(),
// Get current subject 1
                        CurrentSubject2 = row["CurrentSubject2"].ToString(),
// Get current subject 2
                        PreviousSubject1 = row["PreviousSubject1"].ToString(),
// Get previous subject 1
                        PreviousSubject2 =
row["PreviousSubject2"].ToString()  // Get previous subject 2
                    };

                    // Add the newly created student object to the list
                    students.Add(student);
                }

                // Return the list of students
                return students;
            }
            catch (Exception ex)
            {
                // Catch any exceptions and throw a new exception with a
custom message
                throw new Exception("Error retrieving students: " +
ex.Message);
            }
        }

        // Method to edit the details of a person based on their ID
        public int EditPerson(int personId, string name = null, string 
telephone = null, string email = null, string role = null)
        {
            try
            {
                // Create an instance of the data access layer class to 
interact with the database
                PersonDAL objdal = new PersonDAL();

                // Update the person's details and return the result of the 
operation
                return objdal.UpdatePerson(personId, name, telephone, email, 
role);
            }
            catch
            {
                // Catch any exceptions and rethrow them
                throw;
            }
        }

        // Method to delete a person based on their ID
        public void DeletePerson(int personId)
        {
            try
            {
                // Create an instance of the data access layer class to 
interact with the database
                PersonDAL objdal = new PersonDAL();

                // Delete the person and handle any necessary operations
                objdal.DeletePerson(personId);
            }
            catch
            {
                // Catch any exceptions and rethrow them
                throw;
            }
        }
    }
}
```

*Code in PersonDAL.cs:*

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.SqlClient;
using System.Linq;

namespace CW
{
    public class PersonDAL
    {
        static void Main(string[] args)
        {
            // Create an instance of the PersonDAL class to access data
operations
            PersonDAL personDAL = new PersonDAL();

            // Retrieve all persons from the database
            DataTable allPersons = personDAL.Read();
            Console.WriteLine("All persons data loaded.");
        }

        // Connection string to connect to the SQL Server database
        public string ConString = "Data Source=ADMIN-PC\\SQLEXPRESS;Initial
Catalog=EduCen;Persist Security Info=True;User
ID=sa;Password=zxc;TrustServerCertificate=True";

        // SqlConnection object to manage the connection to the database
        SqlConnection con = new SqlConnection();

        // DataTable to hold the results of database queries
        DataTable dt = new DataTable();

        // Method to retrieve all records from the Person table, ordered by
PersonId in descending order
        public DataTable Read()
        {
            // Set the connection string for the SqlConnection object
            con.ConnectionString = ConString;

            // Open the connection if it is closed
            if (ConnectionState.Closed == con.State)
                con.Open();

            // Define the SQL query to select all persons
            SqlCommand cmd = new SqlCommand("SELECT * FROM Person ORDER BY
PersonId DESC", con);
            try
            {
                // Execute the query and load the results into the DataTable
                SqlDataReader rd = cmd.ExecuteReader();
                dt.Load(rd);
                return dt;
            }
            catch
            {
                // Rethrow the exception for further handling
                throw;
            }
        }

        // Method to retrieve a specific person based on their PersonId
        public DataTable Read(Int16 id)
        {
            // Set the connection string for the SqlConnection object
            con.ConnectionString = ConString;

            // Open the connection if it is closed
            if (ConnectionState.Closed == con.State)
                con.Open();

            // Define the SQL query to select a person by their PersonId
            SqlCommand cmd = new SqlCommand("select * from Person where
PersonId= " + id, con);
            try
            {
                // Execute the query and load the results into the DataTable
                SqlDataReader rd = cmd.ExecuteReader();
                dt.Load(rd);
                return dt;
            }
            catch
            {
                // Rethrow the exception for further handling
                throw;
            }
        }

        // Method to retrieve persons based on their name, using a LIKE query
for partial matches
        public DataTable ReadByName(String Name)
        {
            // Set the connection string for the SqlConnection object
            con.ConnectionString = ConString;

            // Open the connection if it is closed
            if (ConnectionState.Closed == con.State)
                con.Open();

            // Define the SQL query to select persons where the name matches
the parameter
```

```
            SqlCommand cmd = new SqlCommand("SELECT * FROM Person WHERE Name
LIKE @Name", con);

            // Add the Name parameter with wildcard characters for partial
matching
            cmd.Parameters.AddWithValue("@Name", "%" + Name + "%");

            try
            {
                // Execute the query and load the results into a new DataTable
                SqlDataReader rd = cmd.ExecuteReader();
                DataTable dt = new DataTable();
                dt.Load(rd);
                return dt;
            }
            catch (Exception ex)
            {
                // Handle or log the exception and provide a custom error
message
                throw new Exception("Error reading person data by name with
LIKE.", ex);
            }
        }

        // Method to retrieve all records from the Teacher table, joining with
the Person table
        public DataTable ReadTeachers()
        {
            // Set the connection string for the SqlConnection object
            con.ConnectionString = ConString;

            // Open the connection if it is closed
            if (ConnectionState.Closed == con.State)
                con.Open();

            // Define the SQL query to join Teacher and Person tables
            string query = @"
                SELECT
                    Teacher.TeacherId,
                    Teacher.PersonId,
                    Person.Name,
                    Person.Telephone,
                    Person.Email,
                    Person.Role,
                    Teacher.Salary,
                    Teacher.Subject1,
                    Teacher.Subject2
                FROM
```

```
                    Teacher
                INNER JOIN
                    Person ON Teacher.PersonId = Person.PersonId
                ORDER BY PersonId DESC";

            // Create a SqlCommand object with the query
            SqlCommand cmd = new SqlCommand(query, con);
            try
            {
                // Execute the query and load the results into the DataTable
                SqlDataReader rd = cmd.ExecuteReader();
                dt.Load(rd);
                return dt;
            }
            catch
            {
                // Rethrow the exception for further handling
                throw;
            }
        }

        /// <summary>
        /// Reads information about a person based on their ID and role.
        /// The role determines which table to query (e.g., Teacher, Admin,
Student).
        /// </summary>
        /// <param name="personId">The ID of the person whose information is
to be retrieved.</param>
        /// <param name="personRole">The role of the person (e.g., Teacher,
Admin, Student).</param>
        /// <returns>A DataTable containing the person's
information.</returns>
        public DataTable ReadPersonInfo(object personId, object personRole)
        {
            // Set the connection string for the database connection
            con.ConnectionString = ConString;

            // Open the connection if it's closed
            if (ConnectionState.Closed == con.State)
            {
                con.Open();
            }

            // Validate and sanitize the personRole parameter to prevent SQL
injection
            string[] allowedRoles = { "Teacher", "Admin", "Student" };
            if (!allowedRoles.Contains(personRole.ToString()))
            {
                throw new ArgumentException("Invalid role specified.");
            }

            // Construct the SQL query string dynamically with table aliases
            string query = $"SELECT * FROM [{personRole}] AS RoleTable " +
                           $"INNER JOIN Person AS P ON RoleTable.PersonId =
P.PersonId " +
                           $"WHERE P.PersonId = @PersonId";

            using (SqlCommand cmd = new SqlCommand(query, con))
            {
                // Add the personId parameter to the SQL query
                cmd.Parameters.AddWithValue("@PersonId", personId);

                using (SqlDataAdapter da = new SqlDataAdapter(cmd))
                {
                    dt = new DataTable();
                    // Fill the DataTable with the results of the query
                    da.Fill(dt);
                }
            }

            // Close the connection after the operation is complete
            con.Close();
            return dt;
        }

        /// <summary>
        /// Retrieves a list of all admins along with their personal and
employment details.
        /// </summary>
        /// <returns>A DataTable containing information about
admins.</returns>
        public DataTable ReadAdmins()
        {
            // Set the connection string for the database connection
            con.ConnectionString = ConString;

            // Open the connection if it's closed
            if (ConnectionState.Closed == con.State)
                con.Open();

            // SQL query to get details of all admins joined with personal
information
            string query = @"
        SELECT
            Admin.AdminId,
            Admin.PersonId,
            Person.Name,
            Person.Telephone,
            Person.Email,
            Person.Role,
            Admin.Salary,
            Admin.EmploymentType,
            Admin.WorkingHours
        FROM
            Admin
        INNER JOIN
            Person ON Admin.PersonId = Person.PersonId
        ORDER BY PersonId DESC";

            SqlCommand cmd = new SqlCommand(query, con);
            try
            {
                SqlDataReader rd = cmd.ExecuteReader();
                // Load the query results into the DataTable
                dt.Load(rd);
                return dt;
            }
            catch
            {
                // Rethrow any exceptions that occur
                throw;
            }
        }

        /// <summary>
        /// Retrieves a list of all students along with their personal and
academic details.
        /// </summary>
        /// <returns>A DataTable containing information about
students.</returns>
        public DataTable ReadStudents()
        {
            // Set the connection string for the database connection
            con.ConnectionString = ConString;

            // Open the connection if it's closed
            if (ConnectionState.Closed == con.State)
                con.Open();

            // SQL query to get details of all students joined with personal
information
            string query = @"
        SELECT
            Student.StudentId,
            Student.PersonId,
            Person.Name,
            Person.Telephone,
            Person.Email,
            Person.Role,
            Student.CurrentSubject1,
            Student.CurrentSubject2,
            Student.PreviousSubject1,
            Student.PreviousSubject2
        FROM
            Student
        INNER JOIN
            Person ON Student.PersonId = Person.PersonId
        ORDER BY PersonId DESC";

            SqlCommand cmd = new SqlCommand(query, con);
            try
            {
                SqlDataReader rd = cmd.ExecuteReader();
                // Load the query results into the DataTable
                dt.Load(rd);
                return dt;
            }
            catch
            {
                // Rethrow any exceptions that occur
                throw;
            }
        }

        /// <summary>
        /// Creates a new person in the database and returns the newly created
person's ID.
        /// </summary>
        /// <param name="name">The name of the person.</param>
        /// <param name="telephone">The telephone number of the
person.</param>
        /// <param name="email">The email address of the person.</param>
        /// <param name="role">The role of the person (e.g., teacher,
student).</param>
        /// <returns>The ID of the newly created person.</returns>
        public int CreatePerson(string name, string telephone, string email,
string role)
        {
            // Set the connection string and open the connection if it is
closed
            con.ConnectionString = ConString;
            if (ConnectionState.Closed == con.State)
```

```
                con.Open();

            // SQL query to insert a new person and return the generated
PersonId
            string query = "INSERT INTO Person (Name, Telephone, Email, Role)
OUTPUT INSERTED.PersonId VALUES (@Name, @Telephone, @Email, @Role)";
            SqlCommand cmd = new SqlCommand(query, con);

            // Add parameters to prevent SQL injection
            cmd.Parameters.AddWithValue("@Name", name);
            cmd.Parameters.AddWithValue("@Telephone", telephone);
            cmd.Parameters.AddWithValue("@Email", email);
            cmd.Parameters.AddWithValue("@Role", role);

            try
            {
                // Execute the query and return the new person's ID
                int personId = (int)cmd.ExecuteScalar();
                return personId;
            }
            catch (Exception ex)
            {
                // Handle and rethrow exceptions that occur during execution
                throw new Exception("Error creating person", ex);
            }
            finally
            {
                // Ensure the connection is closed
                con.Close();
            }
        }

        /// <summary>
        /// Creates a new teacher record in the database based on the given
person ID.
        /// </summary>
        /// <param name="personId">The ID of the person who will be a
teacher.</param>
        /// <param name="salary">The salary of the teacher.</param>
        /// <param name="subject1">The first subject the teacher
teaches.</param>
        /// <param name="subject2">The second subject the teacher
teaches.</param>
        public void CreateTeacher(int personId, int salary, string subject1,
string subject2)
        {
            // Set the connection string and open the connection if it is
closed
```

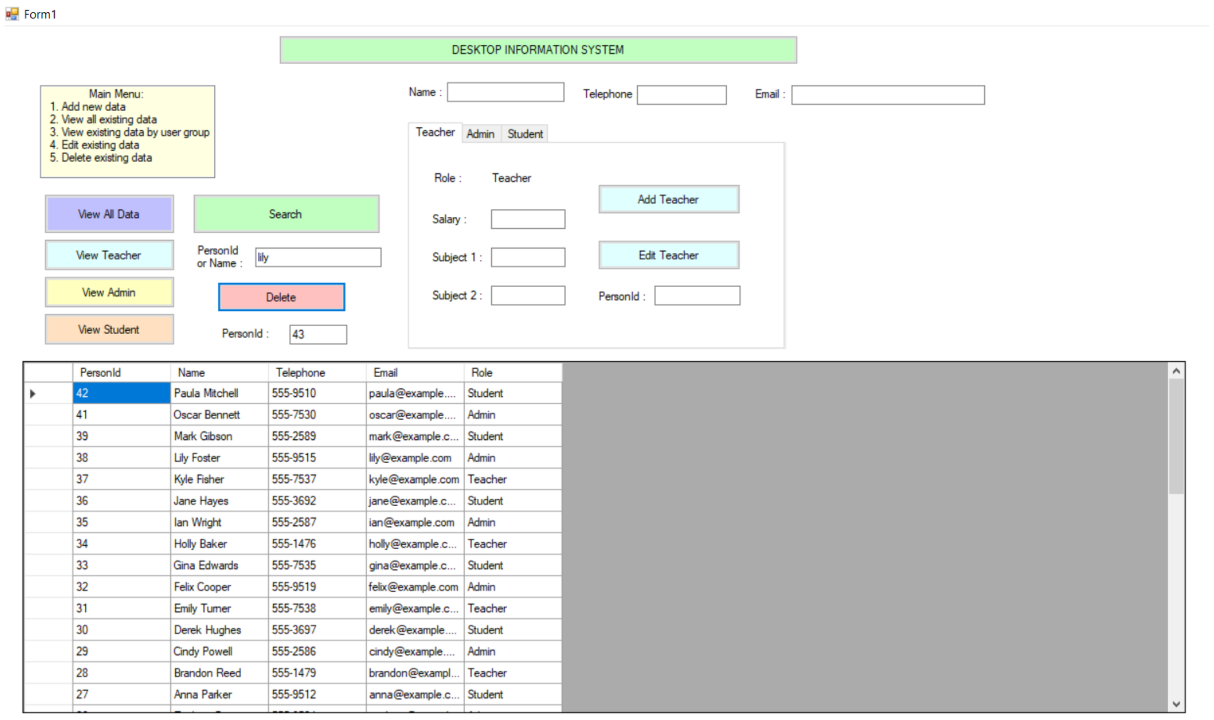
```
            con.ConnectionString = ConString;
            if (ConnectionState.Closed == con.State)
                con.Open();

            // SQL query to insert a new teacher record
            string query = "INSERT INTO Teacher (PersonId, Salary, Subject1, 
Subject2) VALUES (@PersonId, @Salary, @Subject1, @Subject2)";
            SqlCommand cmd = new SqlCommand(query, con);

            // Add parameters to prevent SQL injection
            cmd.Parameters.AddWithValue("@PersonId", personId);
            cmd.Parameters.AddWithValue("@Salary", salary);
            cmd.Parameters.AddWithValue("@Subject1", subject1);
            cmd.Parameters.AddWithValue("@Subject2", subject2);

            try
            {
                // Execute the query without returning any results
                cmd.ExecuteNonQuery();
            }
            catch (Exception ex)
            {
                // Handle and rethrow exceptions that occur during execution
                throw new Exception("Error creating teacher", ex);
            }
            finally
            {
                // Ensure the connection is closed
                con.Close();
            }
        }

        /// <summary>
        /// Updates the details of an existing person based on the provided 
person ID.
        /// </summary>
        /// <param name="personId">The ID of the person to update.</param>
        /// <param name="name">The new name of the person (optional).</param>
        /// <param name="telephone">The new telephone number of the person 
(optional).</param>
        /// <param name="email">The new email address of the person 
(optional).</param>
        /// <param name="role">The new role of the person (optional).</param>
        /// <returns>The number of rows affected by the update.</returns>
        public int UpdatePerson(int personId, string name = null, string 
telephone = null, string email = null, string role = null)
        {
            // Set the connection string and open the connection if it is
closed
            con.ConnectionString = ConString;
            if (ConnectionState.Closed == con.State)
                con.Open();

            // Build the update query dynamically based on provided parameters
            List<string> updateFields = new List<string>();
            if (!string.IsNullOrEmpty(name)) updateFields.Add("Name = @Name");
            if (!string.IsNullOrEmpty(telephone)) updateFields.Add("Telephone
= @Telephone");
            if (!string.IsNullOrEmpty(email)) updateFields.Add("Email =
@Email");
            if (!string.IsNullOrEmpty(role)) updateFields.Add("Role = @Role");

            // Handle case where no fields are provided for update
            if (updateFields.Count == 0)
            {
                return 0; // No rows updated
            }

            // SQL query to update the person record
            string query = $"UPDATE Person SET {string.Join(", ",
updateFields)} WHERE PersonId = @PersonId";
            SqlCommand cmd = new SqlCommand(query, con);

            // Add parameters to prevent SQL injection
            cmd.Parameters.AddWithValue("@PersonId", personId);
            if (!string.IsNullOrEmpty(name))
cmd.Parameters.AddWithValue("@Name", name);
            if (!string.IsNullOrEmpty(telephone))
cmd.Parameters.AddWithValue("@Telephone", telephone);
            if (!string.IsNullOrEmpty(email))
cmd.Parameters.AddWithValue("@Email", email);
            if (!string.IsNullOrEmpty(role))
cmd.Parameters.AddWithValue("@Role", role);

            try
            {
                // Execute the update query and return the number of affected
rows
                int rowsAffected = cmd.ExecuteNonQuery();
                return rowsAffected;
            }
            catch (Exception ex)
            {
                // Handle and rethrow exceptions that occur during execution
                throw new Exception("Error updating person", ex);
            }
            finally
            {
                // Ensure the connection is closed
                con.Close();
            }
        }

        /// <summary>
        /// Updates the details of an existing teacher based on the provided
person ID.
        /// </summary>
        /// <param name="personId">The ID of the teacher to update.</param>
        /// <param name="salary">The new salary of the teacher
(optional).</param>
        /// <param name="subject1">The new first subject the teacher teaches
(optional).</param>
        /// <param name="subject2">The new second subject the teacher teaches
(optional).</param>
        public void UpdateTeacher(int personId, int? salary = null, string
subject1 = null, string subject2 = null)
        {
            // Set the connection string and open the connection if it is
closed
            con.ConnectionString = ConString;
            if (ConnectionState.Closed == con.State)
                con.Open();

            // Build the update query dynamically based on provided parameters
            List<string> updateFields = new List<string>();
            if (salary.HasValue) updateFields.Add("Salary = @Salary");
            if (!string.IsNullOrEmpty(subject1)) updateFields.Add("Subject1 =
@Subject1");
            if (!string.IsNullOrEmpty(subject2)) updateFields.Add("Subject2 =
@Subject2");

            // Handle case where no fields are provided for update
            if (updateFields.Count == 0)
            {
                return; // No fields to update
            }

            // SQL query to update the teacher record
            string query = $"UPDATE Teacher SET {string.Join(", ",
updateFields)} WHERE PersonId = @PersonId";
            SqlCommand cmd = new SqlCommand(query, con);

            // Add parameters to prevent SQL injection
            cmd.Parameters.AddWithValue("@PersonId", personId);
            if (salary.HasValue) cmd.Parameters.AddWithValue("@Salary",
salary.Value);
            if (!string.IsNullOrEmpty(subject1))
cmd.Parameters.AddWithValue("@Subject1", subject1);
            if (!string.IsNullOrEmpty(subject2))
cmd.Parameters.AddWithValue("@Subject2", subject2);

            try
            {
                // Execute the update query without returning any results
                cmd.ExecuteNonQuery();
            }
            catch (Exception ex)
            {
                // Handle and rethrow exceptions that occur during execution
                throw new Exception("Error updating teacher: " + ex.Message,
ex);
            }
            finally
            {
                // Ensure the connection is closed
                con.Close();
            }
        }

        // Method to update the details of an Admin in the database
        public void UpdateAdmin(int personId, int? salary = null, string
employmenttype = null, int? workinghours = null)
        {
            con.ConnectionString = ConString;
            // Open connection if it's not already open
            if (ConnectionState.Closed == con.State)
                con.Open();

            List<string> updateFields = new List<string>();
            // Add fields to be updated based on provided parameters
            if (salary.HasValue) updateFields.Add("Salary = @Salary");
            if (!string.IsNullOrEmpty(employmenttype))
updateFields.Add("EmploymentType = @EmploymentType");
            if (workinghours.HasValue) updateFields.Add("WorkingHours =
@WorkingHours");

            // Handle case when no fields are provided
            if (updateFields.Count == 0)
            {
                return; // No fields to update
            }
            // Construct SQL query with dynamic fields
            string query = $"UPDATE Admin SET {string.Join(", ", 
updateFields)} WHERE PersonId = @PersonId";
            SqlCommand cmd = new SqlCommand(query, con);
            // Add parameters to SQL command
            cmd.Parameters.AddWithValue("@PersonId", personId);
            if (salary.HasValue) cmd.Parameters.AddWithValue("@Salary", 
salary.Value);
            if (!string.IsNullOrEmpty(employmenttype)) 
cmd.Parameters.AddWithValue("@EmploymentType", employmenttype);
            if (workinghours.HasValue) 
cmd.Parameters.AddWithValue("@WorkingHours", workinghours.Value);

            try
            {
                // Execute the query to update the Admin
                cmd.ExecuteNonQuery();
            }
            catch (Exception ex)
            {
                // Throw a detailed exception if something goes wrong
                throw new Exception("Error updating Admin: " + ex.Message, 
ex);
            }
            finally
            {
                // Ensure the connection is closed
                con.Close();
            }
        }

        // Method to update the details of a Student in the database
        public void UpdateStudent(int personId, string currentSubject1 = null, 
string currentSubject2 = null, string previousSubject1 = null, string 
previousSubject2 = null)
        {
            con.ConnectionString = ConString;
            // Open connection if it's not already open
            if (ConnectionState.Closed == con.State)
                con.Open();

            List<string> updateFields = new List<string>();
            // Add fields to be updated based on provided parameters
            if (!string.IsNullOrEmpty(currentSubject1)) 
updateFields.Add("CurrentSubject1 = @CurrentSubject1");
            if (!string.IsNullOrEmpty(currentSubject2)) 
updateFields.Add("CurrentSubject2 = @CurrentSubject2");
```

```
            if (!string.IsNullOrEmpty(previousSubject1))
updateFields.Add("PreviousSubject1 = @PreviousSubject1");
            if (!string.IsNullOrEmpty(previousSubject2))
updateFields.Add("PreviousSubject2 = @PreviousSubject2");

            // Handle case when no fields are provided
            if (updateFields.Count == 0)
            {
                return; // No fields to update
            }

            // Construct SQL query with dynamic fields
            string query = $"UPDATE Student SET {string.Join(", ",
updateFields)} WHERE PersonId = @PersonId";
            SqlCommand cmd = new SqlCommand(query, con);
            // Add parameters to SQL command
            cmd.Parameters.AddWithValue("@PersonId", personId);
            if (!string.IsNullOrEmpty(currentSubject1))
cmd.Parameters.AddWithValue("@CurrentSubject1", currentSubject1);
            if (!string.IsNullOrEmpty(currentSubject2))
cmd.Parameters.AddWithValue("@CurrentSubject2", currentSubject2);
            if (!string.IsNullOrEmpty(previousSubject1))
cmd.Parameters.AddWithValue("@PreviousSubject1", previousSubject1);
            if (!string.IsNullOrEmpty(previousSubject2))
cmd.Parameters.AddWithValue("@PreviousSubject2", previousSubject2);

            try
            {
                // Execute the query to update the Student
                cmd.ExecuteNonQuery();
            }
            catch (Exception ex)
            {
                // Throw a detailed exception if something goes wrong
                throw new Exception("Error updating student: " + ex.Message,
ex);
            }
            finally
            {
                // Ensure the connection is closed
                con.Close();
            }
        }

        // Method to create a new Admin record in the database
        public void CreateAdmin(int personId, int salary, string
employmenttype, int workinghours)
        {
            con.ConnectionString = ConString;
            // Open connection if it's not already open
            if (ConnectionState.Closed == con.State)
                con.Open();

            // SQL query to insert a new Admin record
            string query = "INSERT INTO Admin (PersonId, Salary,
EmploymentType, WorkingHours) VALUES (@PersonId, @Salary, @EmploymentType,
@WorkingHours)";
            SqlCommand cmd = new SqlCommand(query, con);
            // Add parameters to SQL command
            cmd.Parameters.AddWithValue("@PersonId", personId);
            cmd.Parameters.AddWithValue("@Salary", salary);
            cmd.Parameters.AddWithValue("@EmploymentType", employmenttype);
            cmd.Parameters.AddWithValue("@WorkingHours", workinghours);

            try
            {
                // Execute the query to create a new Admin record
                cmd.ExecuteNonQuery();
            }
            catch (Exception ex)
            {
                // Throw a detailed exception if something goes wrong
                throw new Exception("Error creating Admin", ex);
            }
            finally
            {
                // Ensure the connection is closed
                con.Close();
            }
        }

        // Method to create a new Student record in the database
        public void CreateStudent(int personId, string currentsubject1, string
currentsubject2, string previoussubject1, string previoussubject2)
        {
            con.ConnectionString = ConString;
            // Open connection if it's not already open
            if (ConnectionState.Closed == con.State)
                con.Open();

            // SQL query to insert a new Student record
            string query = "INSERT INTO Student (PersonId, CurrentSubject1,
CurrentSubject2, PreviousSubject1, PreviousSubject2) VALUES (@PersonId,
@CurrentSubject1, @CurrentSubject2, @PreviousSubject1, @PreviousSubject2)";
            SqlCommand cmd = new SqlCommand(query, con);
            // Add parameters to SQL command
            cmd.Parameters.AddWithValue("@PersonId", personId);
            cmd.Parameters.AddWithValue("@CurrentSubject1", currentsubject1);
            cmd.Parameters.AddWithValue("@CurrentSubject2", currentsubject2);
            cmd.Parameters.AddWithValue("@PreviousSubject1",
previoussubject1);
            cmd.Parameters.AddWithValue("@PreviousSubject2",
previoussubject2);

            try
            {
                // Execute the query to create a new Student record
                cmd.ExecuteNonQuery();
            }
            catch (Exception ex)
            {
                // Throw a detailed exception if something goes wrong
                throw new Exception("Error creating Student", ex);
            }
            finally
            {
                // Ensure the connection is closed
                con.Close();
            }
        }

        // Method to delete a person from the database based on their unique
identifier.
        public void DeletePerson(int personId)
        {
            // Set the connection string for the SQL connection.
            con.ConnectionString = ConString;

            // Open the connection if it is currently closed.
            if (ConnectionState.Closed == con.State)
                con.Open();

            // SQL query to delete a person record where the PersonId matches
the provided value.
            string query = "DELETE FROM Person WHERE PersonId = @PersonId";

            // Create a SqlCommand object with the query and connection.
            SqlCommand cmd = new SqlCommand(query, con);

            // Add the personId parameter to the SQL command to avoid SQL
injection.
            cmd.Parameters.AddWithValue("@PersonId", personId);

            try
            {
                // Execute the SQL command to delete the person record.
                cmd.ExecuteNonQuery();
            }
            catch (Exception ex)
            {
                // Handle any errors that occur during the execution of the
SQL command.
                throw new Exception("Error deleting person", ex);
            }
            finally
            {
                // Ensure the SQL connection is closed even if an error
occurs.
                con.Close();
            }
        }

        // Base class representing a person with common properties.
        public class Person
        {
            // Unique identifier for the person.
            public int PersonId { get; set; }

            // Full name of the person.
            public string Name { get; set; }

            // Contact telephone number of the person.
            public string Telephone { get; set; }

            // Email address of the person.
            public string Email { get; set; }

            // Role of the person within the organization (e.g., Teacher,
Student).
            public string Role { get; set; }
        }

        // Derived class representing a teacher with additional properties.
        public class Teacher : Person
        {
            // Salary of the teacher.
            public int Salary { get; set; }

            // Primary subject taught by the teacher.
            public string Subject1 { get; set; }

            // Secondary subject taught by the teacher.
            public string Subject2 { get; set; }
        }

        // Derived class representing an admin with additional properties.
        public class Admin : Person
        {
            // Salary of the admin.
            public int Salary { get; set; }

            // Employment type of the admin (e.g., Full-time, Part-time).
            public string EmploymentType { get; set; }

            // Number of working hours per week for the admin.
            public int WorkingHours { get; set; }
        }

        // Derived class representing a student with additional properties.
        public class Student : Person
        {
            // Current subject the student is enrolled in (first subject).
            public string CurrentSubject1 { get; set; }

            // Current subject the student is enrolled in (second subject).
            public string CurrentSubject2 { get; set; }

            // Previous subject the student was enrolled in (first subject).
            public string PreviousSubject1 { get; set; }

            // Previous subject the student was enrolled in (second subject).
            public string PreviousSubject2 { get; set; }
        }
    }
}
```

## *Run code and show GUI*

### View All Data

Click “View All Data” button to view all people

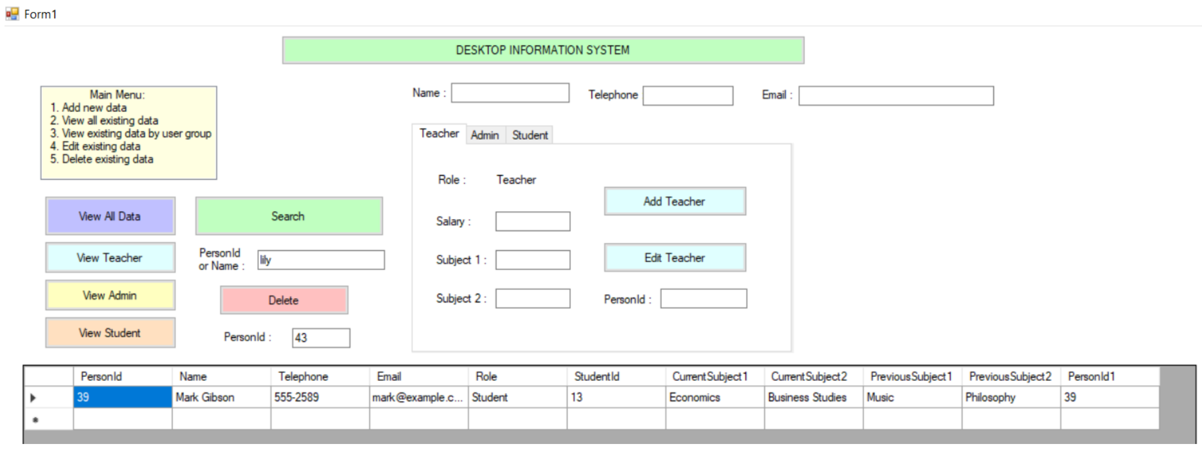


Main Menu:
1. Add new data
2. View all existing data
3. View existing data by user group
4. Edit existing data
5. Delete existing data

Name :
Telephone
Email :
Role : Teacher
Salary :
Subject 1 :
Subject 2 :
PersonId :
PersonId or Name : lily
PersonId : 43

| | PersonId | Name | Telephone | Email | Role |
|---|---|---|---|---|---|
| ▸ | 42 | Paula Mitchell | 555-9510 | paula@example.... | Student |
| | 41 | Oscar Bennett | 555-7530 | oscar@example.... | Admin |
| | 39 | Mark Gibson | 555-2589 | mark@example.c... | Student |
| | 38 | Lily Foster | 555-9515 | lily@example.com | Admin |
| | 37 | Kyle Fisher | 555-7537 | kyle@example.com | Teacher |
| | 36 | Jane Hayes | 555-3692 | jane@example.c... | Student |
| | 35 | Ian Wright | 555-2587 | ian@example.com | Admin |
| | 34 | Holly Baker | 555-1476 | holly@example.c... | Teacher |
| | 33 | Gina Edwards | 555-7535 | gina@example.c... | Student |
| | 32 | Felix Cooper | 555-9519 | felix@example.com | Admin |
| | 31 | Emily Turner | 555-7538 | emily@example.c... | Teacher |
| | 30 | Derek Hughes | 555-3697 | derek@example.... | Student |
| | 29 | Cindy Powell | 555-2586 | cindy@example.... | Admin |
| | 28 | Brandon Reed | 555-1479 | brandon@exampl... | Teacher |
| | 27 | Anna Parker | 555-9512 | anna@example.c... | Student |

Click into a specific person to view detail info based on the person’s role



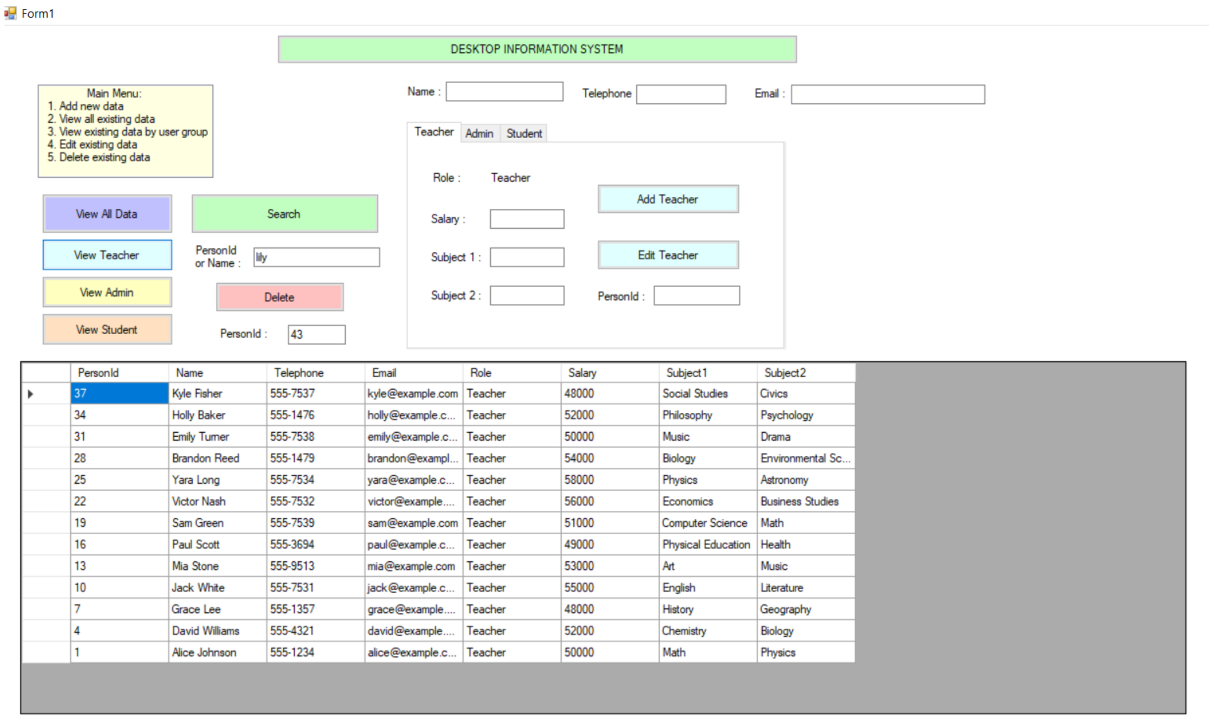
Main Menu:
1. Add new data
2. View all existing data
3. View existing data by user group
4. Edit existing data
5. Delete existing data

Name :
Telephone
Email :
Role : Teacher
Salary :
Subject 1 :
Subject 2 :
PersonId :
PersonId or Name : lily
PersonId : 43

| | PersonId | Name | Telephone | Email | Role | StudentId | CurrentSubject1 | CurrentSubject2 | PreviousSubject1 | PreviousSubject2 | PersonId1 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ▸ | 39 | Mark Gibson | 555-2589 | mark@example.c... | Student | 13 | Economics | Business Studies | Music | Philosophy | 39 |
| * | | | | | | | | | | | |

## View Group Data

Click “View Teacher” button to view all teachers

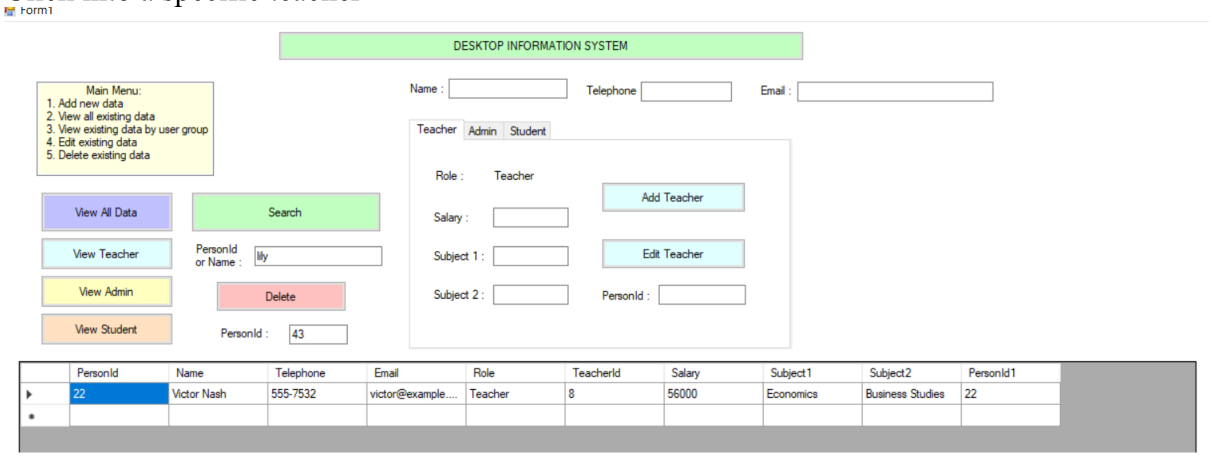


Main Menu:
1. Add new data
2. View all existing data
3. View existing data by user group
4. Edit existing data
5. Delete existing data

Name :
Telephone
Email :
PersonId or Name : lily
PersonId : 43
Role : Teacher
Salary :
Subject 1 :
Subject 2 :
PersonId :

| PersonId | Name | Telephone | Email | Role | Salary | Subject1 | Subject2 |
|---|---|---|---|---|---|---|---|
| 37 | Kyle Fisher | 555-7537 | kyle@example.com | Teacher | 48000 | Social Studies | Civics |
| 34 | Holly Baker | 555-1476 | holly@example.c... | Teacher | 52000 | Philosophy | Psychology |
| 31 | Emily Turner | 555-7538 | emily@example.c... | Teacher | 50000 | Music | Drama |
| 28 | Brandon Reed | 555-1479 | brandon@exampl... | Teacher | 54000 | Biology | Environmental Sc... |
| 25 | Yara Long | 555-7534 | yara@example.c... | Teacher | 58000 | Physics | Astronomy |
| 22 | Victor Nash | 555-7532 | victor@example.... | Teacher | 56000 | Economics | Business Studies |
| 19 | Sam Green | 555-7539 | sam@example.com | Teacher | 51000 | Computer Science | Math |
| 16 | Paul Scott | 555-3694 | paul@example.c... | Teacher | 49000 | Physical Education | Health |
| 13 | Mia Stone | 555-9513 | mia@example.com | Teacher | 53000 | Art | Music |
| 10 | Jack White | 555-7531 | jack@example.c... | Teacher | 55000 | English | Literature |
| 7 | Grace Lee | 555-1357 | grace@example.... | Teacher | 48000 | History | Geography |
| 4 | David Williams | 555-4321 | david@example.... | Teacher | 52000 | Chemistry | Biology |
| 1 | Alice Johnson | 555-1234 | alice@example.c... | Teacher | 50000 | Math | Physics |

Click into a specific teacher

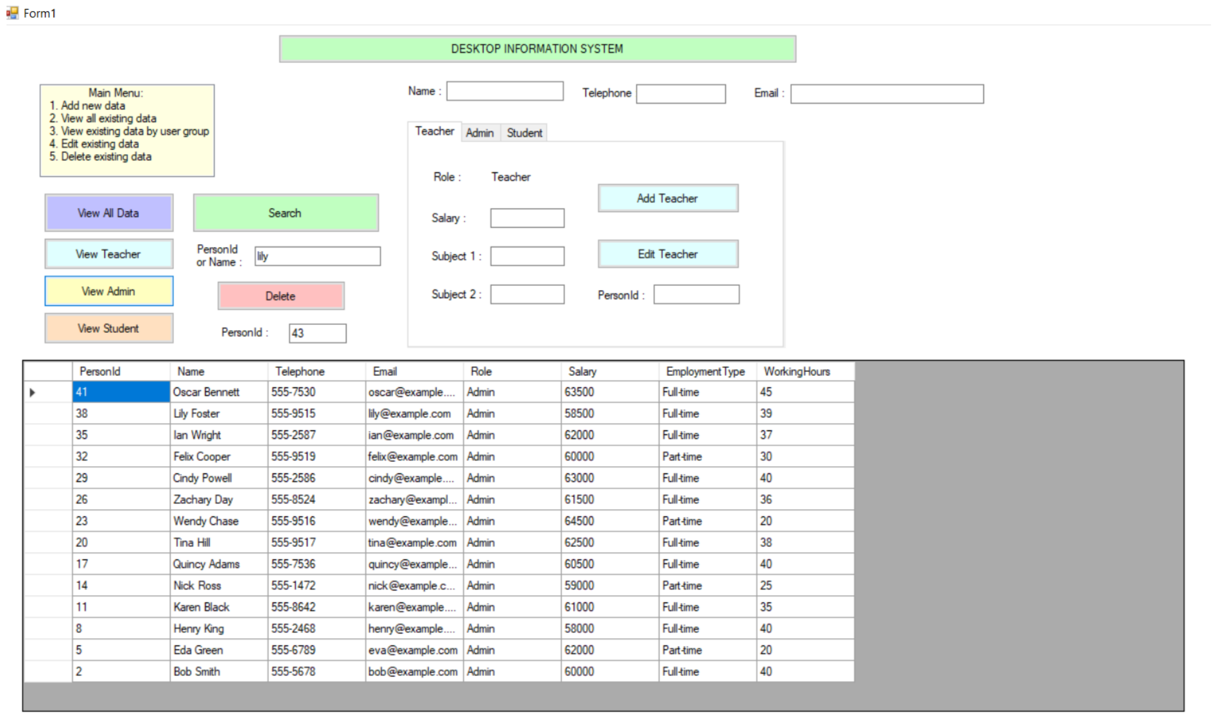


Main Menu:
1. Add new data
2. View all existing data
3. View existing data by user group
4. Edit existing data
5. Delete existing data

Name :
Telephone
Email :
PersonId or Name : lily
PersonId : 43
Role : Teacher
Salary :
Subject 1 :
Subject 2 :
PersonId :

| PersonId | Name | Telephone | Email | Role | TeacherId | Salary | Subject1 | Subject2 | PersonId1 |
|---|---|---|---|---|---|---|---|---|---|
| 22 | Victor Nash | 555-7532 | victor@example.... | Teacher | 8 | 56000 | Economics | Business Studies | 22 |
| | | | | | | | | | |

View all Admins



Main Menu:
1. Add new data
2. View all existing data
3. View existing data by user group
4. Edit existing data
5. Delete existing data

Role : Teacher

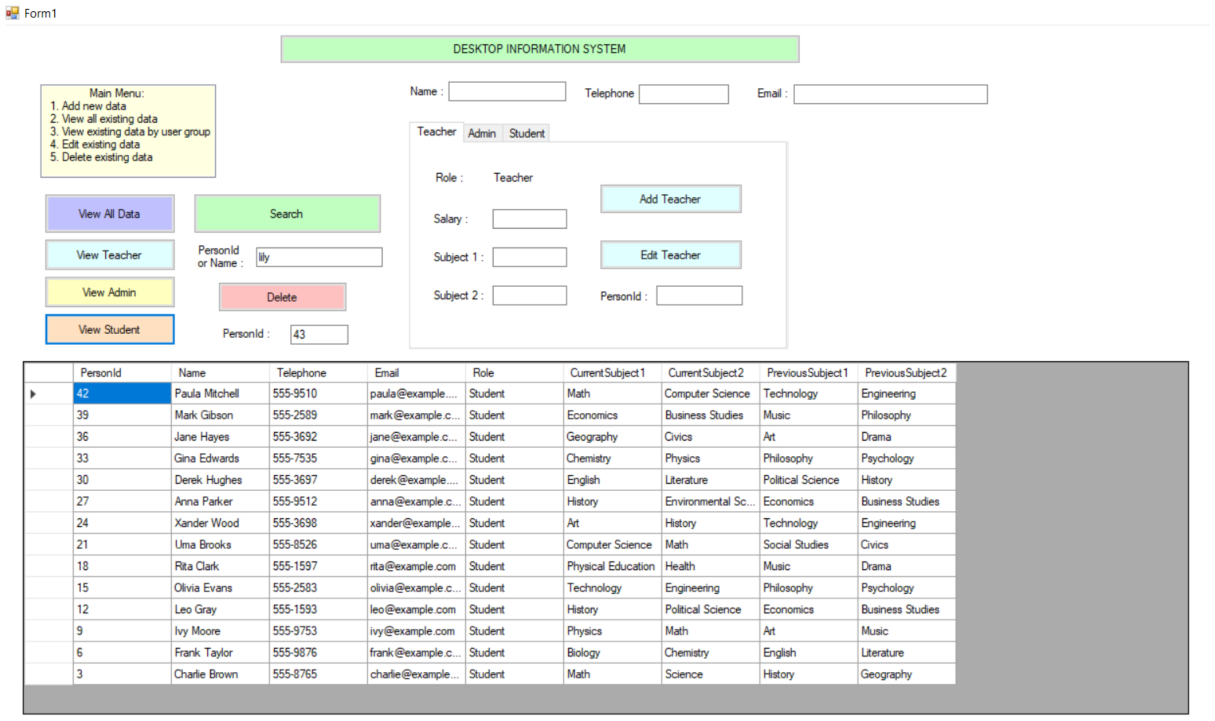
PersonId or Name : lily

PersonId : 43

PersonId :

| PersonId | Name | Telephone | Email | Role | Salary | EmploymentType | WorkingHours |
|---|---|---|---|---|---|---|---|
| 41 | Oscar Bennett | 555-7530 | oscar@example.... | Admin | 63500 | Full-time | 45 |
| 38 | Lily Foster | 555-9515 | lily@example.com | Admin | 58500 | Full-time | 39 |
| 35 | Ian Wright | 555-2587 | ian@example.com | Admin | 62000 | Full-time | 37 |
| 32 | Felix Cooper | 555-9519 | felix@example.com | Admin | 60000 | Part-time | 30 |
| 29 | Cindy Powell | 555-2586 | cindy@example.... | Admin | 63000 | Full-time | 40 |
| 26 | Zachary Day | 555-8524 | zachary@exampl... | Admin | 61500 | Full-time | 36 |
| 23 | Wendy Chase | 555-9516 | wendy@example... | Admin | 64500 | Part-time | 20 |
| 20 | Tina Hill | 555-9517 | tina@example.com | Admin | 62500 | Full-time | 38 |
| 17 | Quincy Adams | 555-7536 | quincy@example... | Admin | 60500 | Full-time | 40 |
| 14 | Nick Ross | 555-1472 | nick@example.c... | Admin | 59000 | Part-time | 25 |
| 11 | Karen Black | 555-8642 | karen@example.... | Admin | 61000 | Full-time | 35 |
| 8 | Henry King | 555-2468 | henry@example.... | Admin | 58000 | Full-time | 40 |
| 5 | Eda Green | 555-6789 | eva@example.com | Admin | 62000 | Part-time | 20 |
| 2 | Bob Smith | 555-5678 | bob@example.com | Admin | 60000 | Full-time | 40 |

View all students



Main Menu:
1. Add new data
2. View all existing data
3. View existing data by user group
4. Edit existing data
5. Delete existing data

Role : Teacher

PersonId or Name : lily

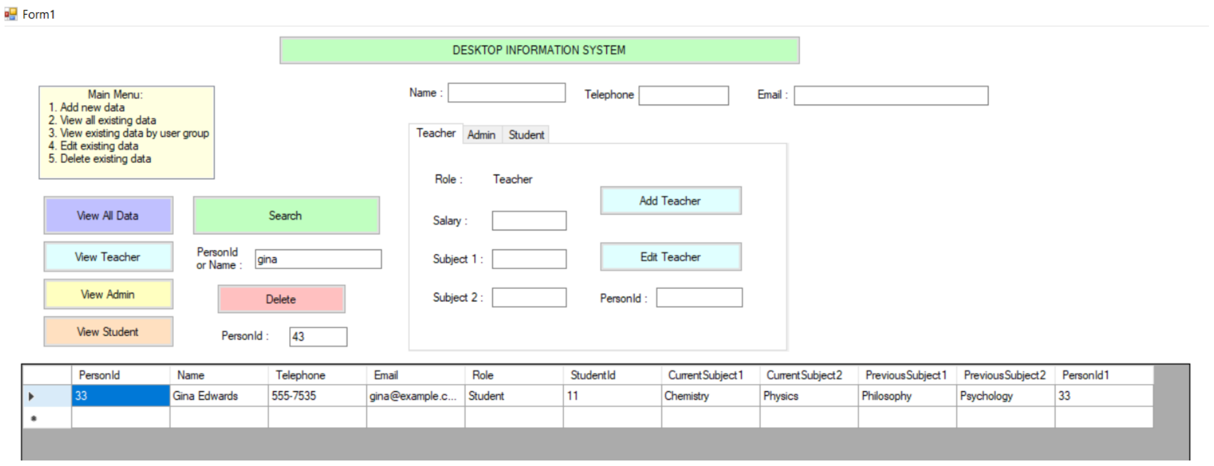
PersonId : 43

PersonId :

| PersonId | Name | Telephone | Email | Role | CurrentSubject1 | CurrentSubject2 | PreviousSubject1 | PreviousSubject2 |
|---|---|---|---|---|---|---|---|---|
| 42 | Paula Mitchell | 555-9510 | paula@example.... | Student | Math | Computer Science | Technology | Engineering |
| 39 | Mark Gibson | 555-2589 | mark@example.c... | Student | Economics | Business Studies | Music | Philosophy |
| 36 | Jane Hayes | 555-3692 | jane@example.c... | Student | Geography | Civics | Art | Drama |
| 33 | Gina Edwards | 555-7535 | gina@example.c... | Student | Chemistry | Physics | Philosophy | Psychology |
| 30 | Derek Hughes | 555-3697 | derek@example.... | Student | English | Literature | Political Science | History |
| 27 | Anna Parker | 555-9512 | anna@example.c... | Student | History | Environmental Sc... | Economics | Business Studies |
| 24 | Xander Wood | 555-3698 | xander@example... | Student | Art | History | Technology | Engineering |
| 21 | Uma Brooks | 555-8526 | uma@example.c... | Student | Computer Science | Math | Social Studies | Civics |
| 18 | Rita Clark | 555-1597 | rita@example.com | Student | Physical Education | Health | Music | Drama |
| 15 | Olivia Evans | 555-2583 | olivia@example.c... | Student | Technology | Engineering | Philosophy | Psychology |
| 12 | Leo Gray | 555-1593 | leo@example.com | Student | History | Political Science | Economics | Business Studies |
| 9 | Ivy Moore | 555-9753 | ivy@example.com | Student | Physics | Math | Art | Music |
| 6 | Frank Taylor | 555-9876 | frank@example.c... | Student | Biology | Chemistry | English | Literature |
| 3 | Charlie Brown | 555-8765 | charlie@example... | Student | Math | Science | History | Geography |

## Search Data

Search by name



Main Menu:
1. Add new data
2. View all existing data
3. View existing data by user group
4. Edit existing data
5. Delete existing data

PersonId or Name : gina

PersonId : 43

Name : Telephone Email :

Role : Teacher

Salary :

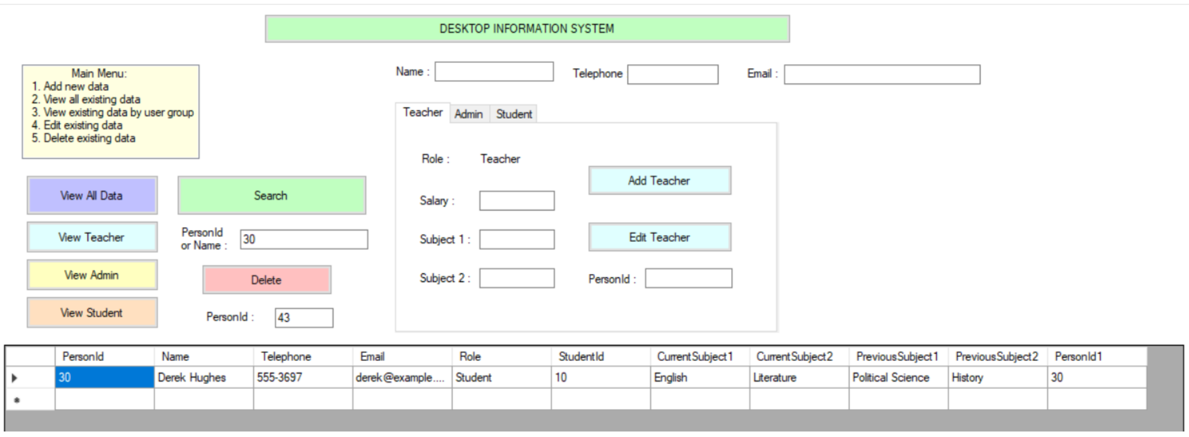
Subject 1 :

Subject 2 :

PersonId :

| | PersonId | Name | Telephone | Email | Role | StudentId | CurrentSubject1 | CurrentSubject2 | PreviousSubject1 | PreviousSubject2 | PersonId1 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ▸ | 33 | Gina Edwards | 555-7535 | gina@example.c... | Student | 11 | Chemistry | Physics | Philosophy | Psychology | 33 |
| * | | | | | | | | | | | |

Search by PersonId



Main Menu:
1. Add new data
2. View all existing data
3. View existing data by user group
4. Edit existing data
5. Delete existing data

PersonId or Name : 30

PersonId : 43

Name : Telephone Email :

Role : Teacher

Salary :

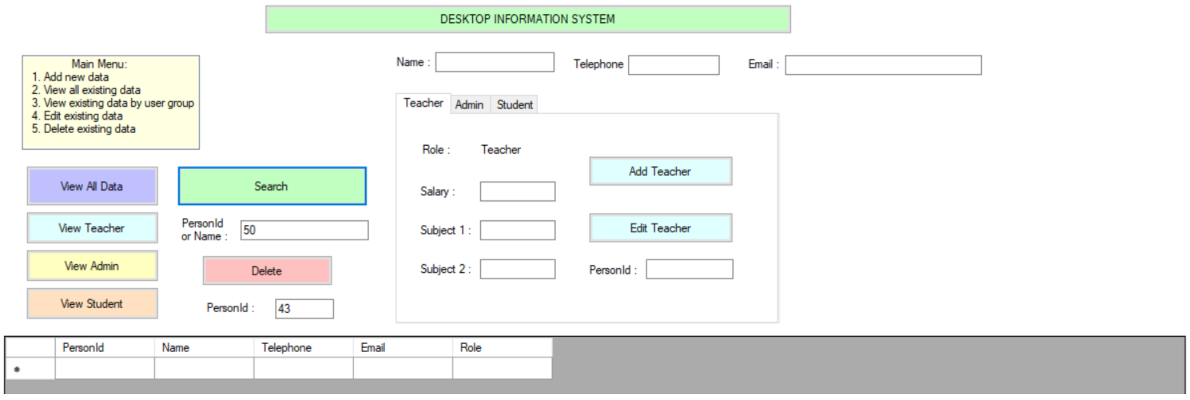
Subject 1 :

Subject 2 :

PersonId :

| | PersonId | Name | Telephone | Email | Role | StudentId | CurrentSubject1 | CurrentSubject2 | PreviousSubject1 | PreviousSubject2 | PersonId1 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ▸ | 30 | Derek Hughes | 555-3697 | derek@example.... | Student | 10 | English | Literature | Political Science | History | 30 |
| * | | | | | | | | | | | |

If enter invalid PersonId or Name, it shows nothing



Main Menu:
1. Add new data
2. View all existing data
3. View existing data by user group
4. Edit existing data
5. Delete existing data

PersonId or Name : 50

PersonId : 43

Name : Telephone Email :

Role : Teacher

Salary :

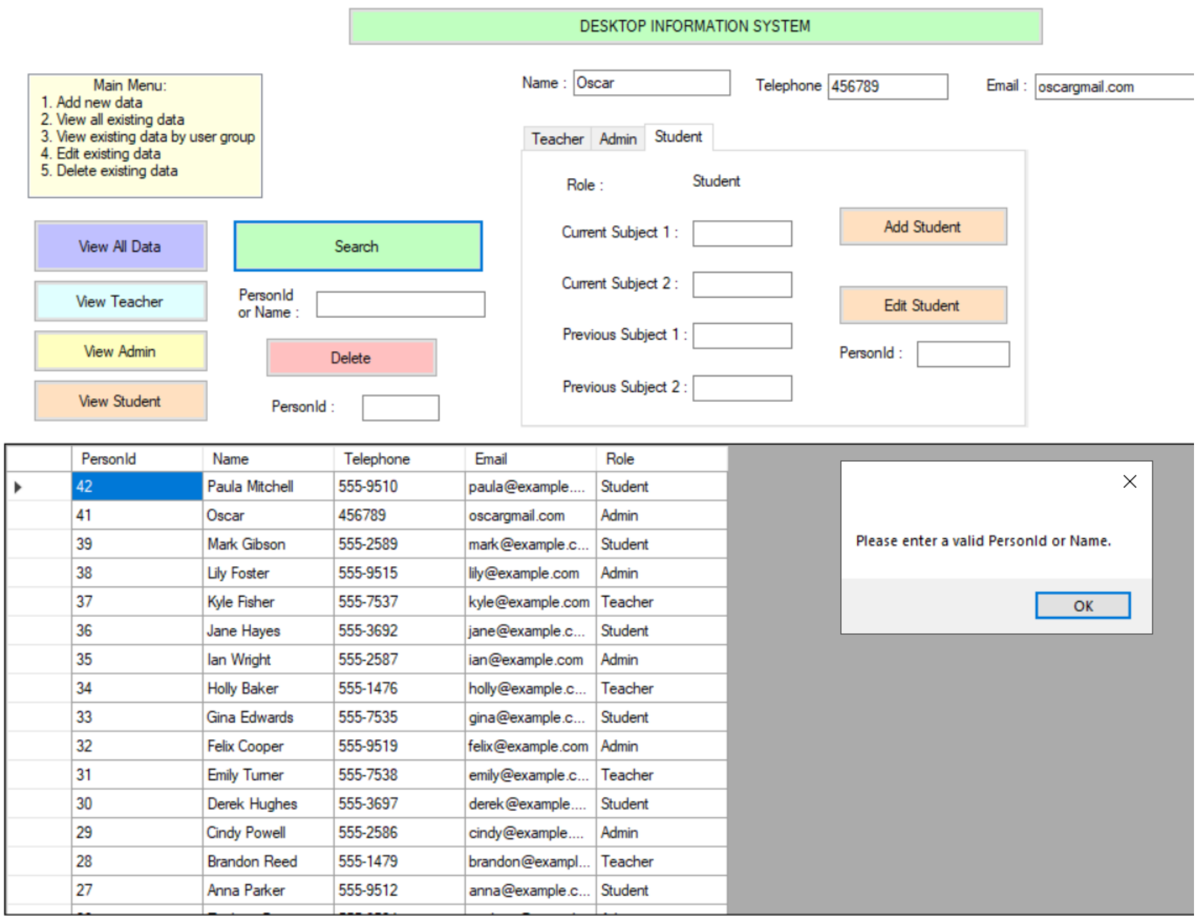
Subject 1 :

Subject 2 :

PersonId :

| | PersonId | Name | Telephone | Email | Role |
|---|---|---|---|---|---|
| * | | | | | |

If user forgot to enter PersonId or Name before clicking “Search” button, it will show



Name : Oscar
Telephone 456789
Email : oscargmail.com

PersonId or Name :

PersonId :

Role : Student

Current Subject 1 :

Current Subject 2 :

Previous Subject 1 :

Previous Subject 2 :

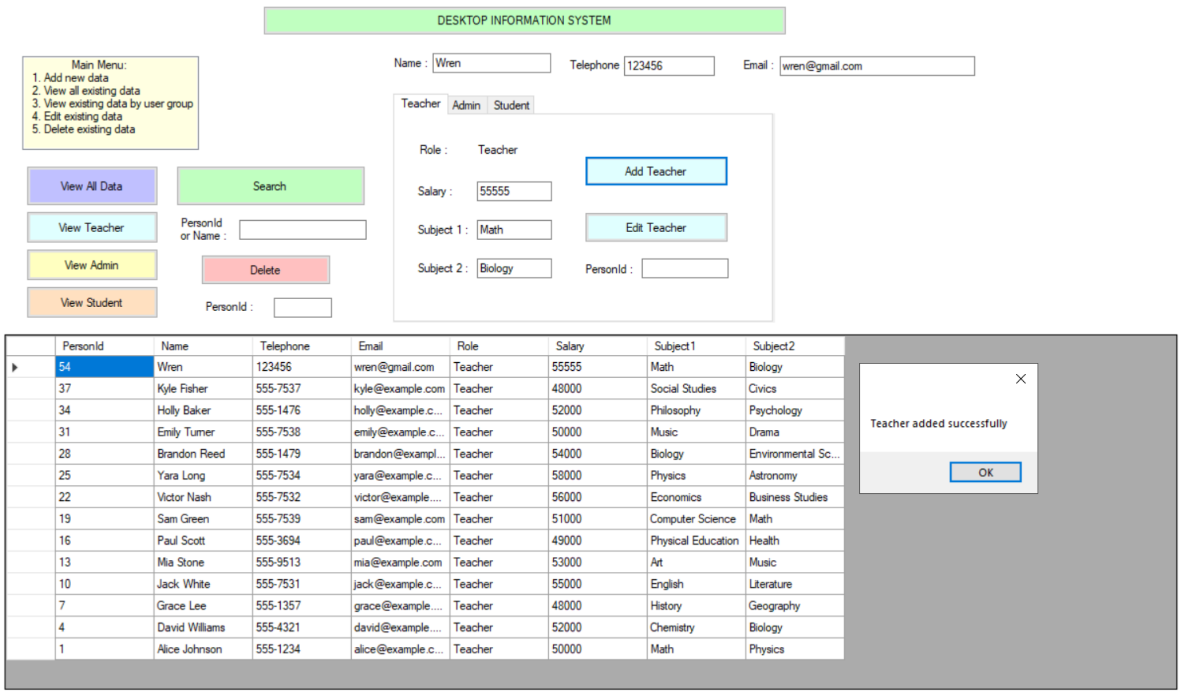
PersonId :

| | PersonId | Name | Telephone | Email | Role |
|---|---|---|---|---|---|
| ▸ | 42 | Paula Mitchell | 555-9510 | paula@example.... | Student |
| | 41 | Oscar | 456789 | oscargmail.com | Admin |
| | 39 | Mark Gibson | 555-2589 | mark@example.c... | Student |
| | 38 | Lily Foster | 555-9515 | lily@example.com | Admin |
| | 37 | Kyle Fisher | 555-7537 | kyle@example.com | Teacher |
| | 36 | Jane Hayes | 555-3692 | jane@example.c... | Student |
| | 35 | Ian Wright | 555-2587 | ian@example.com | Admin |
| | 34 | Holly Baker | 555-1476 | holly@example.c... | Teacher |
| | 33 | Gina Edwards | 555-7535 | gina@example.c... | Student |
| | 32 | Felix Cooper | 555-9519 | felix@example.com | Admin |
| | 31 | Emily Turner | 555-7538 | emily@example.c... | Teacher |
| | 30 | Derek Hughes | 555-3697 | derek@example.... | Student |
| | 29 | Cindy Powell | 555-2586 | cindy@example.... | Admin |
| | 28 | Brandon Reed | 555-1479 | brandon@exampl... | Teacher |
| | 27 | Anna Parker | 555-9512 | anna@example.c... | Student |

Please enter a valid PersonId or Name.

## Add Data

### Add teacher



Name : Wren
Telephone 123456
Email : wren@gmail.com

PersonId or Name :

PersonId :

Role : Teacher

Salary : 55555

Subject 1 : Math

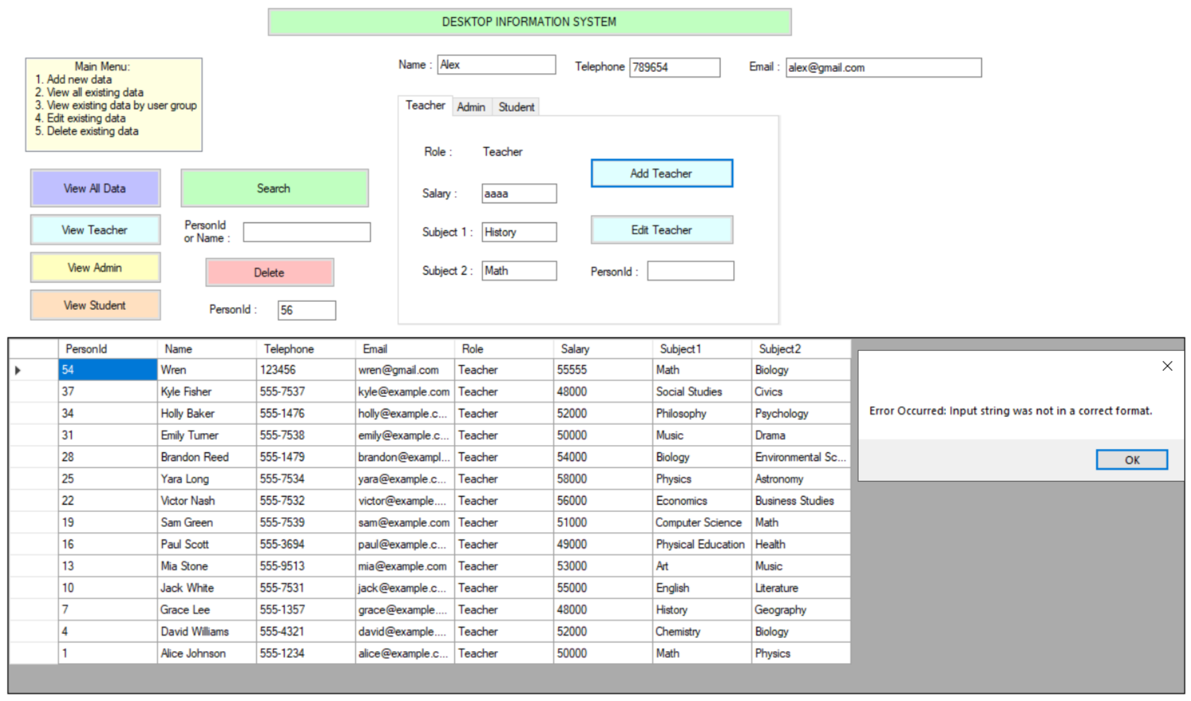
Subject 2 : Biology

PersonId :

| | PersonId | Name | Telephone | Email | Role | Salary | Subject1 | Subject2 |
|---|---|---|---|---|---|---|---|---|
| ▸ | 54 | Wren | 123456 | wren@gmail.com | Teacher | 55555 | Math | Biology |
| | 37 | Kyle Fisher | 555-7537 | kyle@example.com | Teacher | 48000 | Social Studies | Civics |
| | 34 | Holly Baker | 555-1476 | holly@example.c... | Teacher | 52000 | Philosophy | Psychology |
| | 31 | Emily Turner | 555-7538 | emily@example.c... | Teacher | 50000 | Music | Drama |
| | 28 | Brandon Reed | 555-1479 | brandon@exampl... | Teacher | 54000 | Biology | Environmental Sc... |
| | 25 | Yara Long | 555-7534 | yara@example.c... | Teacher | 58000 | Physics | Astronomy |
| | 22 | Victor Nash | 555-7532 | victor@example.... | Teacher | 56000 | Economics | Business Studies |
| | 19 | Sam Green | 555-7539 | sam@example.com | Teacher | 51000 | Computer Science | Math |
| | 16 | Paul Scott | 555-3694 | paul@example.c... | Teacher | 49000 | Physical Education | Health |
| | 13 | Mia Stone | 555-9513 | mia@example.com | Teacher | 53000 | Art | Music |
| | 10 | Jack White | 555-7531 | jack@example.c... | Teacher | 55000 | English | Literature |
| | 7 | Grace Lee | 555-1357 | grace@example.... | Teacher | 48000 | History | Geography |
| | 4 | David Williams | 555-4321 | david@example.... | Teacher | 52000 | Chemistry | Biology |
| | 1 | Alice Johnson | 555-1234 | alice@example.c... | Teacher | 50000 | Math | Physics |

Teacher added successfully

It showed error message box when user entered incorrect format



Name : Alex Telephone 789654 Email : alex@gmail.com

PersonId or Name :

PersonId : 56

Role : Teacher

Salary : aaaa

Subject 1 : History

Subject 2 : Math

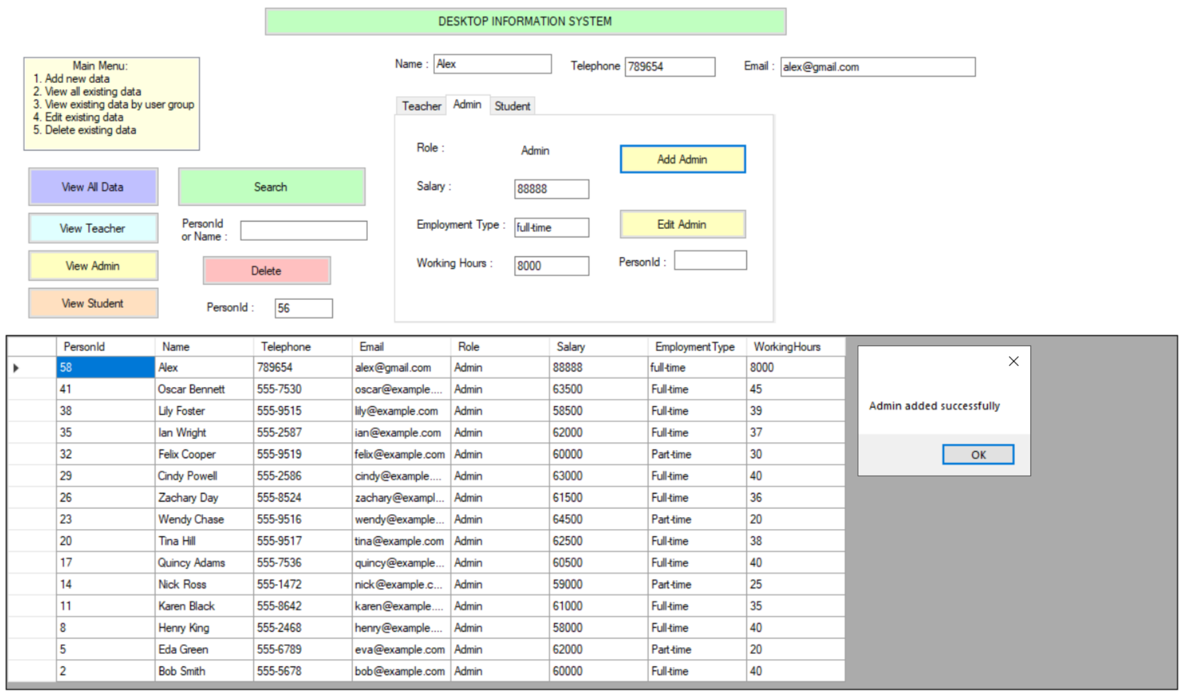
PersonId :

| PersonId | Name | Telephone | Email | Role | Salary | Subject1 | Subject2 |
|---|---|---|---|---|---|---|---|
| 54 | Wren | 123456 | wren@gmail.com | Teacher | 55555 | Math | Biology |
| 37 | Kyle Fisher | 555-7537 | kyle@example.com | Teacher | 48000 | Social Studies | Civics |
| 34 | Holly Baker | 555-1476 | holly@example.c... | Teacher | 52000 | Philosophy | Psychology |
| 31 | Emily Turner | 555-7538 | emily@example.c... | Teacher | 50000 | Music | Drama |
| 28 | Brandon Reed | 555-1479 | brandon@exampl... | Teacher | 54000 | Biology | Environmental Sc... |
| 25 | Yara Long | 555-7534 | yara@example.c... | Teacher | 58000 | Physics | Astronomy |
| 22 | Victor Nash | 555-7532 | victor@example.... | Teacher | 56000 | Economics | Business Studies |
| 19 | Sam Green | 555-7539 | sam@example.com | Teacher | 51000 | Computer Science | Math |
| 16 | Paul Scott | 555-3694 | paul@example.c... | Teacher | 49000 | Physical Education | Health |
| 13 | Mia Stone | 555-9513 | mia@example.com | Teacher | 53000 | Art | Music |
| 10 | Jack White | 555-7531 | jack@example.c... | Teacher | 55000 | English | Literature |
| 7 | Grace Lee | 555-1357 | grace@example.... | Teacher | 48000 | History | Geography |
| 4 | David Williams | 555-4321 | david@example.... | Teacher | 52000 | Chemistry | Biology |
| 1 | Alice Johnson | 555-1234 | alice@example.c... | Teacher | 50000 | Math | Physics |

Error Occurred: Input string was not in a correct format.

OK

Add Admin



Name : Alex Telephone 789654 Email : alex@gmail.com

PersonId or Name :

PersonId : 56

Role : Admin

Salary : 88888

Employment Type : full-time

Working Hours : 8000

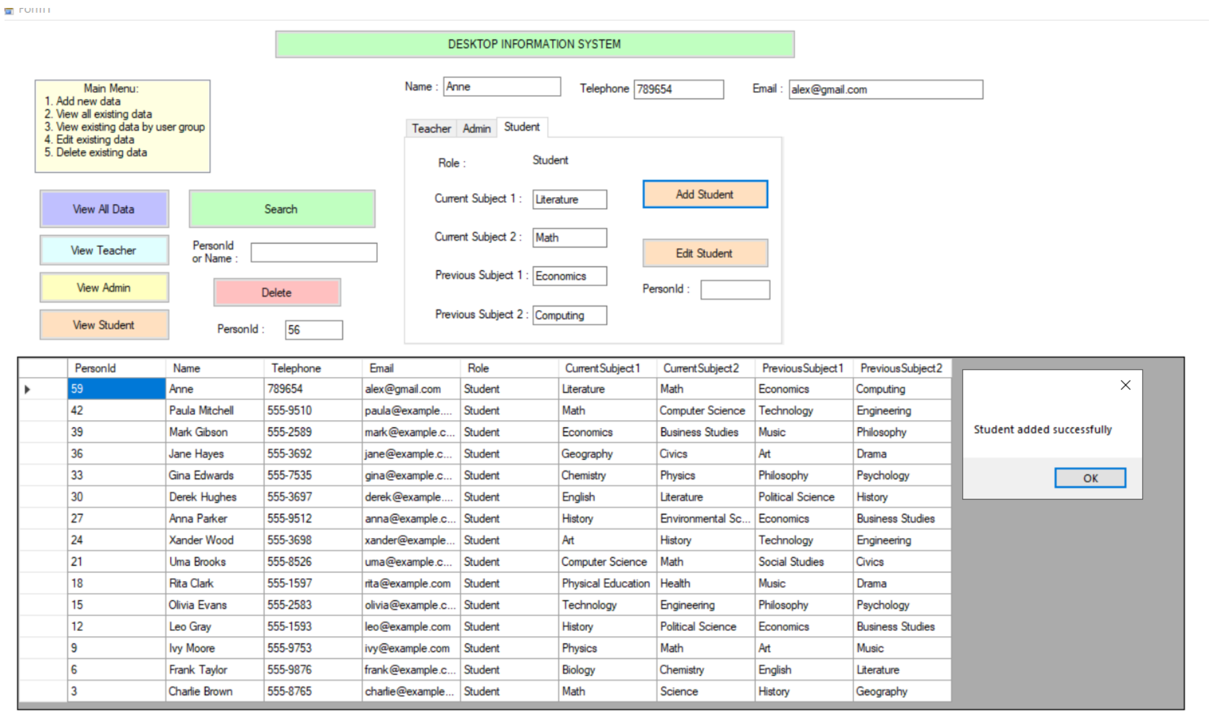
PersonId :

| PersonId | Name | Telephone | Email | Role | Salary | EmploymentType | WorkingHours |
|---|---|---|---|---|---|---|---|
| 58 | Alex | 789654 | alex@gmail.com | Admin | 88888 | full-time | 8000 |
| 41 | Oscar Bennett | 555-7530 | oscar@example.... | Admin | 63500 | Full-time | 45 |
| 38 | Lily Foster | 555-9515 | lily@example.com | Admin | 58500 | Full-time | 39 |
| 35 | Ian Wright | 555-2587 | ian@example.com | Admin | 62000 | Full-time | 37 |
| 32 | Felix Cooper | 555-9519 | felix@example.com | Admin | 60000 | Part-time | 30 |
| 29 | Cindy Powell | 555-2586 | cindy@example.... | Admin | 63000 | Full-time | 40 |
| 26 | Zachary Day | 555-8524 | zachary@exampl... | Admin | 61500 | Full-time | 36 |
| 23 | Wendy Chase | 555-9516 | wendy@example... | Admin | 64500 | Part-time | 20 |
| 20 | Tina Hill | 555-9517 | tina@example.com | Admin | 62500 | Full-time | 38 |
| 17 | Quincy Adams | 555-7536 | quincy@example... | Admin | 60500 | Full-time | 40 |
| 14 | Nick Ross | 555-1472 | nick@example.c... | Admin | 59000 | Part-time | 25 |
| 11 | Karen Black | 555-8642 | karen@example.... | Admin | 61000 | Full-time | 35 |
| 8 | Henry King | 555-2468 | henry@example.... | Admin | 58000 | Full-time | 40 |
| 5 | Eda Green | 555-6789 | eva@example.com | Admin | 62000 | Part-time | 20 |
| 2 | Bob Smith | 555-5678 | bob@example.com | Admin | 60000 | Full-time | 40 |

Admin added successfully

OK

Add Student



DESKTOP INFORMATION SYSTEM

Main Menu:
1. Add new data
2. View all existing data
3. View existing data by user group
4. Edit existing data
5. Delete existing data

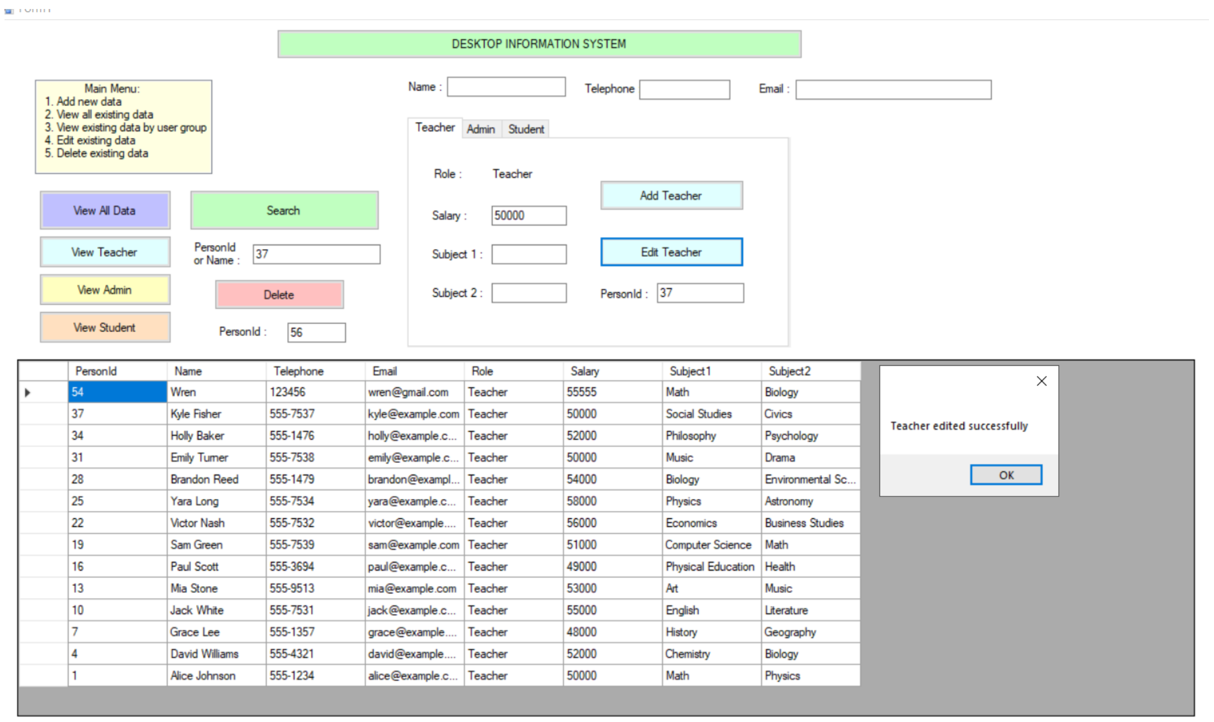
|  |  |
|---|---|
| Name : | Anne |
| Telephone | 789654 |
| Email : | alex@gmail.com |
| Role : | Student |
| Current Subject 1 : | Literature |
| Current Subject 2 : | Math |
| Previous Subject 1 : | Economics |
| Previous Subject 2 : | Computing |
| PersonId : | |
| PersonId or Name : | |
| PersonId : | 56 |

| PersonId | Name | Telephone | Email | Role | CurrentSubject1 | CurrentSubject2 | PreviousSubject1 | PreviousSubject2 |
|---|---|---|---|---|---|---|---|---|
| 59 | Anne | 789654 | alex@gmail.com | Student | Literature | Math | Economics | Computing |
| 42 | Paula Mitchell | 555-9510 | paula@example.... | Student | Math | Computer Science | Technology | Engineering |
| 39 | Mark Gibson | 555-2589 | mark@example.c... | Student | Economics | Business Studies | Music | Philosophy |
| 36 | Jane Hayes | 555-3692 | jane@example.c... | Student | Geography | Civics | Art | Drama |
| 33 | Gina Edwards | 555-7535 | gina@example.c... | Student | Chemistry | Physics | Philosophy | Psychology |
| 30 | Derek Hughes | 555-3697 | derek@example.... | Student | English | Literature | Political Science | History |
| 27 | Anna Parker | 555-9512 | anna@example.c... | Student | History | Environmental Sc... | Economics | Business Studies |
| 24 | Xander Wood | 555-3698 | xander@example... | Student | Art | History | Technology | Engineering |
| 21 | Uma Brooks | 555-8526 | uma@example.c... | Student | Computer Science | Math | Social Studies | Civics |
| 18 | Rita Clark | 555-1597 | rita@example.com | Student | Physical Education | Health | Music | Drama |
| 15 | Olivia Evans | 555-2583 | olivia@example.c... | Student | Technology | Engineering | Philosophy | Psychology |
| 12 | Leo Gray | 555-1593 | leo@example.com | Student | History | Political Science | Economics | Business Studies |
| 9 | Ivy Moore | 555-9753 | ivy@example.com | Student | Physics | Math | Art | Music |
| 6 | Frank Taylor | 555-9876 | frank@example.c... | Student | Biology | Chemistry | English | Literature |
| 3 | Charlie Brown | 555-8765 | charlie@example... | Student | Math | Science | History | Geography |

Student added successfully

## Edit Data

Enter PersonId “59” and Edit Teacher



DESKTOP INFORMATION SYSTEM

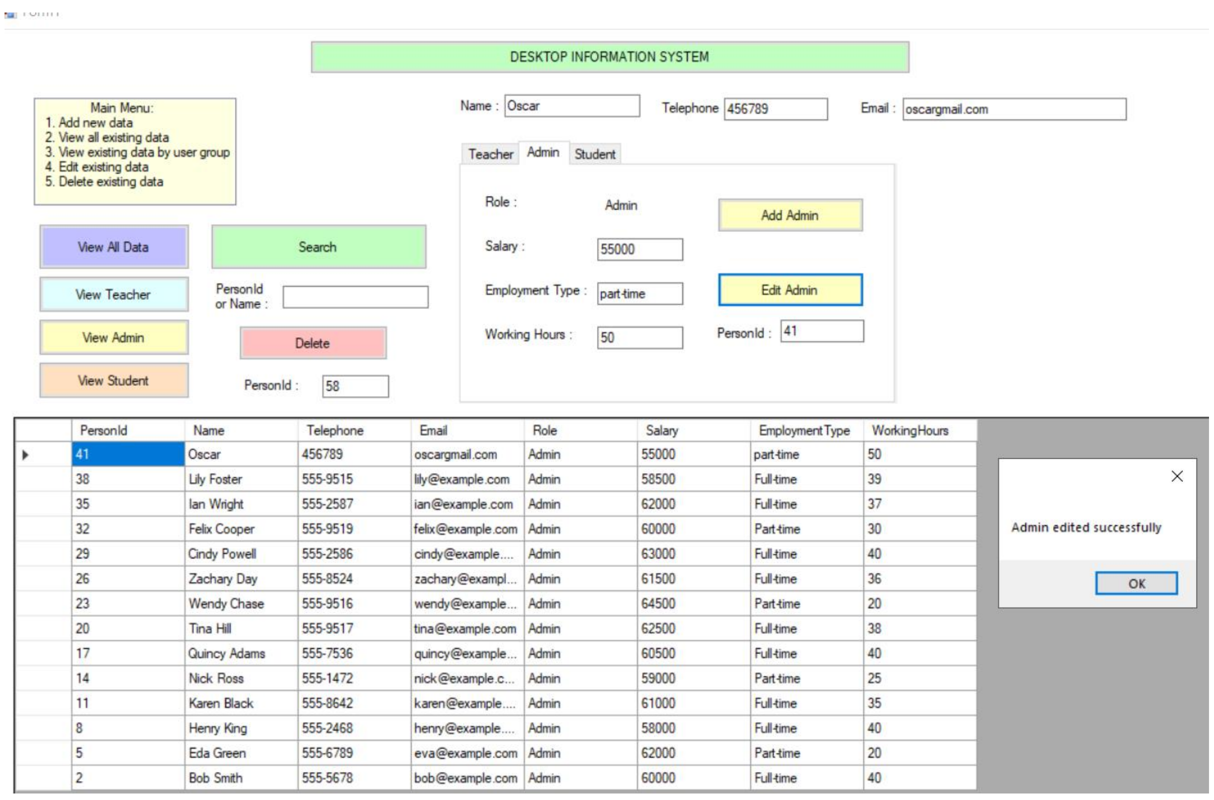
Main Menu:
1. Add new data
2. View all existing data
3. View existing data by user group
4. Edit existing data
5. Delete existing data

|  |  |
|---|---|
| Name : | |
| Telephone | |
| Email : | |
| Role : | Teacher |
| Salary : | 50000 |
| Subject 1 : | |
| Subject 2 : | |
| PersonId : | 37 |
| PersonId or Name : | 37 |
| PersonId : | 56 |

| PersonId | Name | Telephone | Email | Role | Salary | Subject1 | Subject2 |
|---|---|---|---|---|---|---|---|
| 54 | Wren | 123456 | wren@gmail.com | Teacher | 55555 | Math | Biology |
| 37 | Kyle Fisher | 555-7537 | kyle@example.com | Teacher | 50000 | Social Studies | Civics |
| 34 | Holly Baker | 555-1476 | holly@example.c... | Teacher | 52000 | Philosophy | Psychology |
| 31 | Emily Turner | 555-7538 | emily@example.c... | Teacher | 50000 | Music | Drama |
| 28 | Brandon Reed | 555-1479 | brandon@exampl... | Teacher | 54000 | Biology | Environmental Sc... |
| 25 | Yara Long | 555-7534 | yara@example.c... | Teacher | 58000 | Physics | Astronomy |
| 22 | Victor Nash | 555-7532 | victor@example.... | Teacher | 56000 | Economics | Business Studies |
| 19 | Sam Green | 555-7539 | sam@example.com | Teacher | 51000 | Computer Science | Math |
| 16 | Paul Scott | 555-3694 | paul@example.c... | Teacher | 49000 | Physical Education | Health |
| 13 | Mia Stone | 555-9513 | mia@example.com | Teacher | 53000 | Art | Music |
| 10 | Jack White | 555-7531 | jack@example.c... | Teacher | 55000 | English | Literature |
| 7 | Grace Lee | 555-1357 | grace@example.... | Teacher | 48000 | History | Geography |
| 4 | David Williams | 555-4321 | david@example.... | Teacher | 52000 | Chemistry | Biology |
| 1 | Alice Johnson | 555-1234 | alice@example.c... | Teacher | 50000 | Math | Physics |

Teacher edited successfully

Enter PersonId “59” and Edit Admin



Name : Oscar
Telephone 456789
Email : oscargmail.com

PersonId or Name :
PersonId : 58

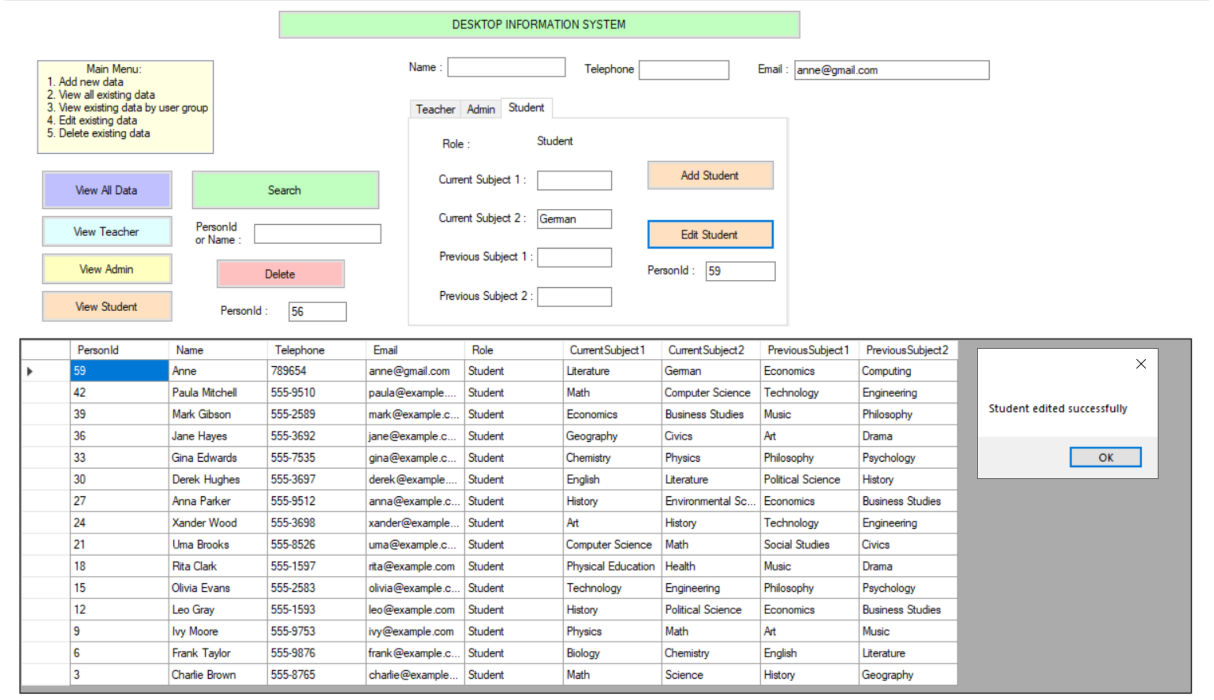
Role : Admin
Salary : 55000
Employment Type : part-time
Working Hours : 50
PersonId : 41

| PersonId | Name | Telephone | Email | Role | Salary | EmploymentType | WorkingHours |
|---|---|---|---|---|---|---|---|
| 41 | Oscar | 456789 | oscargmail.com | Admin | 55000 | part-time | 50 |
| 38 | Lily Foster | 555-9515 | lily@example.com | Admin | 58500 | Full-time | 39 |
| 35 | Ian Wright | 555-2587 | ian@example.com | Admin | 62000 | Full-time | 37 |
| 32 | Felix Cooper | 555-9519 | felix@example.com | Admin | 60000 | Part-time | 30 |
| 29 | Cindy Powell | 555-2586 | cindy@example.... | Admin | 63000 | Full-time | 40 |
| 26 | Zachary Day | 555-8524 | zachary@exampl... | Admin | 61500 | Full-time | 36 |
| 23 | Wendy Chase | 555-9516 | wendy@example... | Admin | 64500 | Part-time | 20 |
| 20 | Tina Hill | 555-9517 | tina@example.com | Admin | 62500 | Full-time | 38 |
| 17 | Quincy Adams | 555-7536 | quincy@example... | Admin | 60500 | Full-time | 40 |
| 14 | Nick Ross | 555-1472 | nick@example.c... | Admin | 59000 | Part-time | 25 |
| 11 | Karen Black | 555-8642 | karen@example.... | Admin | 61000 | Full-time | 35 |
| 8 | Henry King | 555-2468 | henry@example... | Admin | 58000 | Full-time | 40 |
| 5 | Eda Green | 555-6789 | eva@example.com | Admin | 62000 | Part-time | 20 |
| 2 | Bob Smith | 555-5678 | bob@example.com | Admin | 60000 | Full-time | 40 |

Admin edited successfully

OK

Enter PersonId “59” and Edit Student



Name :
Telephone
Email : anne@gmail.com

PersonId or Name :
PersonId : 56

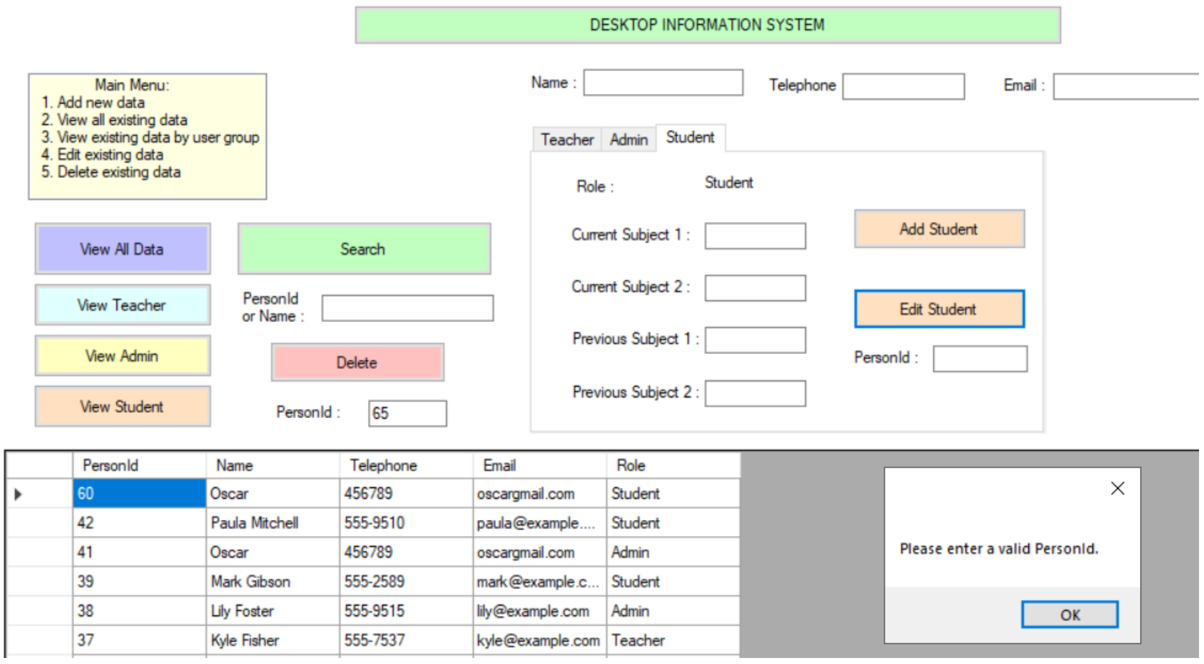
Role : Student
Current Subject 1 :
Current Subject 2 : German
Previous Subject 1 :
Previous Subject 2 :
PersonId : 59

| PersonId | Name | Telephone | Email | Role | CurrentSubject1 | CurrentSubject2 | PreviousSubject1 | PreviousSubject2 |
|---|---|---|---|---|---|---|---|---|
| 59 | Anne | 789654 | anne@gmail.com | Student | Literature | German | Economics | Computing |
| 42 | Paula Mitchell | 555-9510 | paula@example.... | Student | Math | Computer Science | Technology | Engineering |
| 39 | Mark Gibson | 555-2589 | mark@example.c... | Student | Economics | Business Studies | Music | Philosophy |
| 36 | Jane Hayes | 555-3692 | jane@example.c... | Student | Geography | Civics | Art | Drama |
| 33 | Gina Edwards | 555-7535 | gina@example.c... | Student | Chemistry | Physics | Philosophy | Psychology |
| 30 | Derek Hughes | 555-3697 | derek@example.... | Student | English | Literature | Political Science | History |
| 27 | Anna Parker | 555-9512 | anna@example.c... | Student | History | Environmental Sc... | Economics | Business Studies |
| 24 | Xander Wood | 555-3698 | xander@example... | Student | Art | History | Technology | Engineering |
| 21 | Uma Brooks | 555-8526 | uma@example.c... | Student | Computer Science | Math | Social Studies | Civics |
| 18 | Rita Clark | 555-1597 | rita@example.com | Student | Physical Education | Health | Music | Drama |
| 15 | Olivia Evans | 555-2583 | olivia@example.c... | Student | Technology | Engineering | Philosophy | Psychology |
| 12 | Leo Gray | 555-1593 | leo@example.com | Student | History | Political Science | Economics | Business Studies |
| 9 | Ivy Moore | 555-9753 | ivy@example.com | Student | Physics | Math | Art | Music |
| 6 | Frank Taylor | 555-9876 | frank@example.c... | Student | Biology | Chemistry | English | Literature |
| 3 | Charlie Brown | 555-8765 | charlie@example... | Student | Math | Science | History | Geography |

Student edited successfully

OK

If user forgot to enter PersonId before clicking “Edit” button, it will show



PersonId : 65

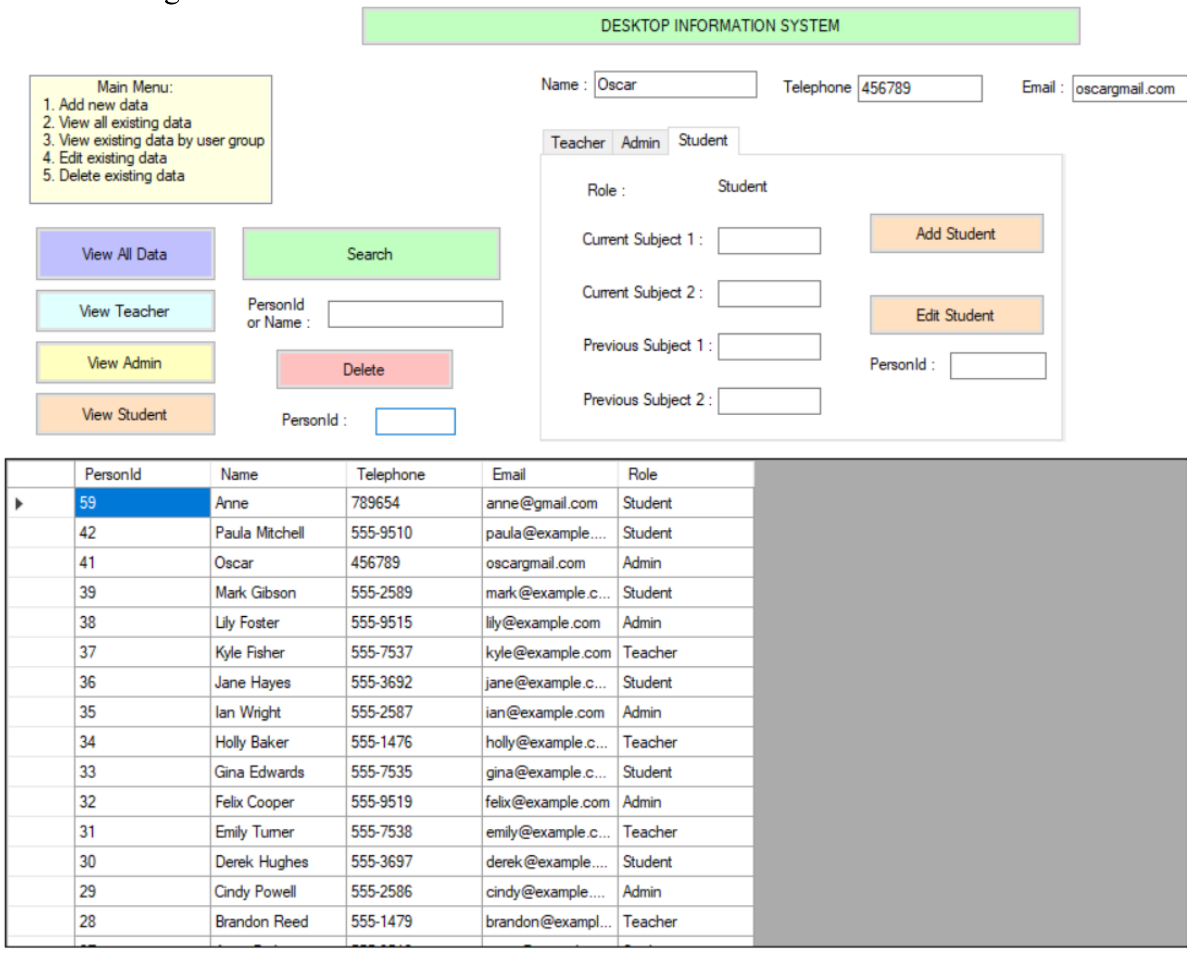
Role : Student

| PersonId | Name | Telephone | Email | Role |
|---|---|---|---|---|
| 60 | Oscar | 456789 | oscargmail.com | Student |
| 42 | Paula Mitchell | 555-9510 | paula@example.... | Student |
| 41 | Oscar | 456789 | oscargmail.com | Admin |
| 39 | Mark Gibson | 555-2589 | mark@example.c... | Student |
| 38 | Lily Foster | 555-9515 | lily@example.com | Admin |
| 37 | Kyle Fisher | 555-7537 | kyle@example.com | Teacher |

Please enter a valid PersonId.

## Delete Data

Enter PersonId and Delete a person
Before deleting

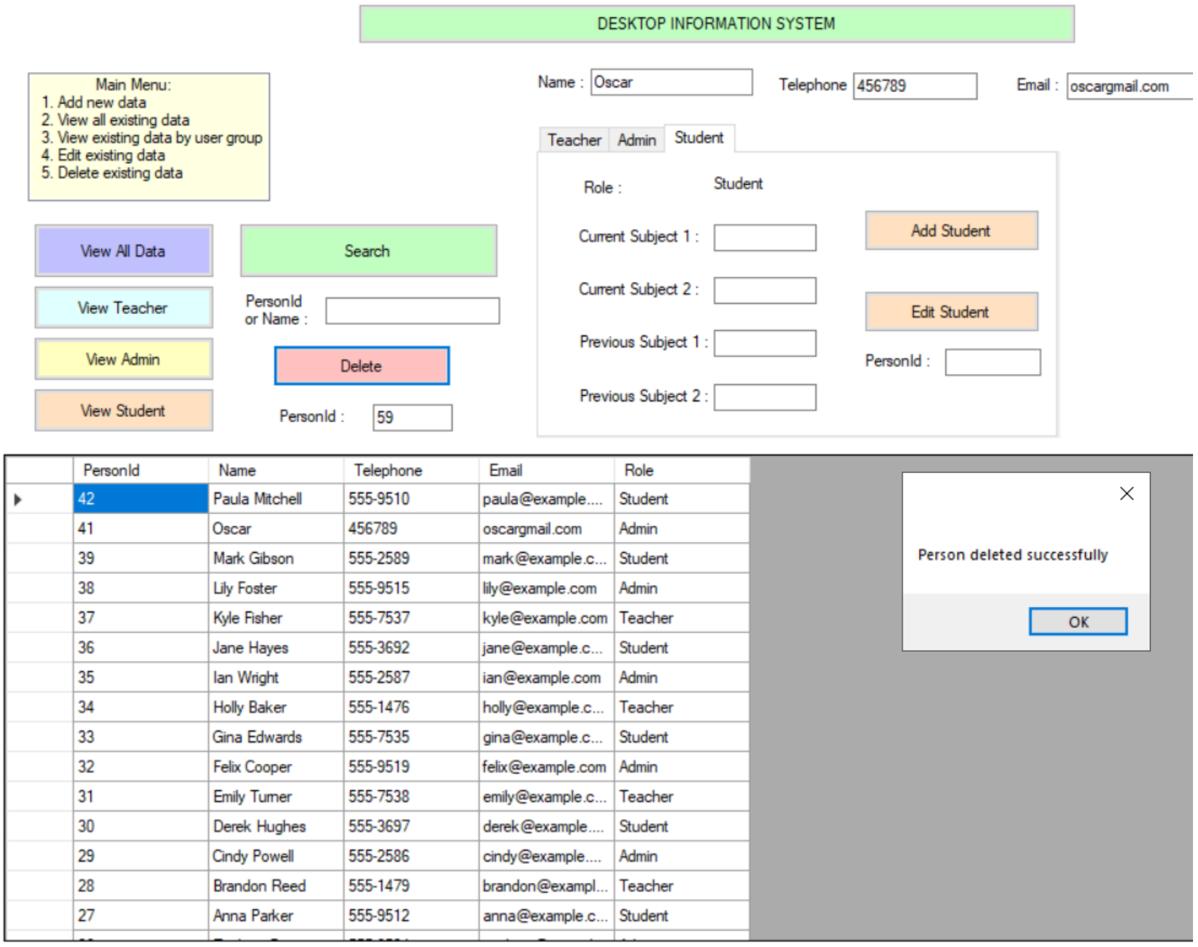


Name : Oscar
Telephone 456789
Email : oscargmail.com

Role : Student

| PersonId | Name | Telephone | Email | Role |
|---|---|---|---|---|
| 59 | Anne | 789654 | anne@gmail.com | Student |
| 42 | Paula Mitchell | 555-9510 | paula@example.... | Student |
| 41 | Oscar | 456789 | oscargmail.com | Admin |
| 39 | Mark Gibson | 555-2589 | mark@example.c... | Student |
| 38 | Lily Foster | 555-9515 | lily@example.com | Admin |
| 37 | Kyle Fisher | 555-7537 | kyle@example.com | Teacher |
| 36 | Jane Hayes | 555-3692 | jane@example.c... | Student |
| 35 | Ian Wright | 555-2587 | ian@example.com | Admin |
| 34 | Holly Baker | 555-1476 | holly@example.c... | Teacher |
| 33 | Gina Edwards | 555-7535 | gina@example.c... | Student |
| 32 | Felix Cooper | 555-9519 | felix@example.com | Admin |
| 31 | Emily Turner | 555-7538 | emily@example.c... | Teacher |
| 30 | Derek Hughes | 555-3697 | derek@example.... | Student |
| 29 | Cindy Powell | 555-2586 | cindy@example.... | Admin |
| 28 | Brandon Reed | 555-1479 | brandon@exampl... | Teacher |

I enter PersonId “59” and clicked “Delete” button



Main Menu:
1. Add new data
2. View all existing data
3. View existing data by user group
4. Edit existing data
5. Delete existing data

PersonId or Name :

PersonId : 59

Name : Oscar

Telephone 456789

Email : oscargmail.com

Role : Student

Current Subject 1 :

Current Subject 2 :

Previous Subject 1 :

Previous Subject 2 :

PersonId :

| PersonId | Name | Telephone | Email | Role |
|---|---|---|---|---|
| 42 | Paula Mitchell | 555-9510 | paula@example.... | Student |
| 41 | Oscar | 456789 | oscargmail.com | Admin |
| 39 | Mark Gibson | 555-2589 | mark@example.c... | Student |
| 38 | Lily Foster | 555-9515 | lily@example.com | Admin |
| 37 | Kyle Fisher | 555-7537 | kyle@example.com | Teacher |
| 36 | Jane Hayes | 555-3692 | jane@example.c... | Student |
| 35 | Ian Wright | 555-2587 | ian@example.com | Admin |
| 34 | Holly Baker | 555-1476 | holly@example.c... | Teacher |
| 33 | Gina Edwards | 555-7535 | gina@example.c... | Student |
| 32 | Felix Cooper | 555-9519 | felix@example.com | Admin |
| 31 | Emily Turner | 555-7538 | emily@example.c... | Teacher |
| 30 | Derek Hughes | 555-3697 | derek@example.... | Student |
| 29 | Cindy Powell | 555-2586 | cindy@example.... | Admin |
| 28 | Brandon Reed | 555-1479 | brandon@exampl... | Teacher |
| 27 | Anna Parker | 555-9512 | anna@example.c... | Student |

Person deleted successfully