

## Лабораторная работа № 32

**Тема:** Разработка приложения по технологии Entity Framework: расширение функционала: поиск, добавление и редактирование данных.

**Программное обеспечение:** MS Visual Studio 2015

**Цель:** научиться реализовывать поиск данных, добавление и редактирование данных в приложении Database First Entity Framework.

**Время на выполнение:** 2 часа.

**Задание 1:** Реализовать поиск данных по таблице Абитуриенты.

**Порядок выполнения:**

1. Откройте проект, созданный в предыдущей лабораторной работе.
2. Если поле для ввода не пустое, тогда при нажатии кнопки поиска будем передавать в *dataGridView1* данные из *sheetabiturients*, отфильтрованные по значениям одного из столбцов, выбор которого осуществляется через *comboBox1*, и обновлять состояние *label1*. Сброс фильтра будет происходить по нажатию кнопки поиска с пустым полем ввода:

```
private void button1_Click(object sender, EventArgs e)
{
    if (textBox1.Text != "")
    {
        switch (comboBox1.SelectedIndex)
        {
            case 0:
                dataGridView1.DataSource =
                    sheetabiturients.Where(p => p.Код_абитуриента.ToString() ==
                    textBox1.Text.ToString())
                    .ToList(); break;
            case 1:
                dataGridView1.DataSource =
                    sheetabiturients.Where(p =>
                    p.Фамилия.ToString().Contains(textBox1.Text.ToString()))
                    .ToList(); break;
        }
    }
    else
    {
        dataGridView1.DataSource = sheetabiturients;
    }
    dataGridView1.Update();
    if (dataGridView1.RowCount == 0) label1.Visible = true;
    else label1.Visible = false;
}
```

3. В данном примере реализован поиск через запросы LINQ по столбцам: Код\_абитуриента и Фамилия. Добавьте в данный пример поиск по остальным столбцам таблицы Абитуриенты.
4. Проверьте разработанный функционал приложения: поиск данных по всем столбцам таблицы Абитуриенты.

**Задание 2:** Добавить возможность редактирования и добавления записей в таблице Абитуриенты.

**Порядок выполнения:**

1. Создадим две отдельные формы FormEditAbiturient и FormAddAbiturient.

Для добавления нового абитуриента будем просто создавать новую форму FormAddAbiturient addAbit. Понятно, что такая форма может быть открыта только одна. Можно сделать ее модальной, т.е. блокировать работу с начальной формой: addAbit.ShowDialog();

2. Однако считается, что пользователю, во время ее заполнения, может потребоваться просмотр информации из списка абитуриентов, поэтому необходимо проверять, сколько форм уже открыто, и, если уже есть одна – передавать управление ей. В таком случае button3\_Click выглядит следующим образом:

```
private void button3_Click(object sender, EventArgs e)
{
    if (Application.OpenForms.Count == 1)
    {
        FormAddAbiturient addAbit = new FormAddAbiturient();
        addAbit.Owner = this; addAbit.Show();
    }
    else Application.OpenForms[1].Focus();
}
```

Передача данных между формами осуществляется с использованием свойства «Родитель». Важно помнить, что форма с индексом 1 – основное рабочее окно регистратора. Отличие формы редактирования в том, что в нее мы должны передать объект типа Абитуриенты, поля которого будем редактировать:

```
private void button2_Click(object sender, EventArgs e)
{
    if (dataGridView1.SelectedCells.Count == 1)
    {
        if (Application.OpenForms.Count == 1)
        {
            Абитуриенты item = sheetabiturients
                .First(w => w.Код_абитуриента.ToString() == dataGridView1
                    .SelectedCells[0]
                    .OwningRow
                    .Cells[0]
                    .Value
                    .ToString());
            FormEditAbiturient edAbit = new FormEditAbiturient(item);
            edAbit.Owner = this;
            edAbit.Show();
        }
        else Application.OpenForms[1].Focus();
    }
}
```

Условие первой строки необходимо для устранения неопределенности в случае выделения пользователем ячеек нескольких строк: невозможно редактировать сразу две записи в одной форме.

Теперь перейдем к дизайну форм и их логике. Две данные формы будут абсолютно

идентичны по дизайну, поэтому можно просто скопировать элементы из конструктора одной в конструктор другой. Единственное ограничение – Код\_абитуриента в уже существующей записи редактировать запрещено, это первичный ключ.

3. Добавим на форму шесть текстовых полей с поясняющими надписями, соответствующие столбцам таблицы (полям записи, объекта типа Абитуриенты), одну кнопку и label1 для вывода ошибок (см. Рисунок 2):

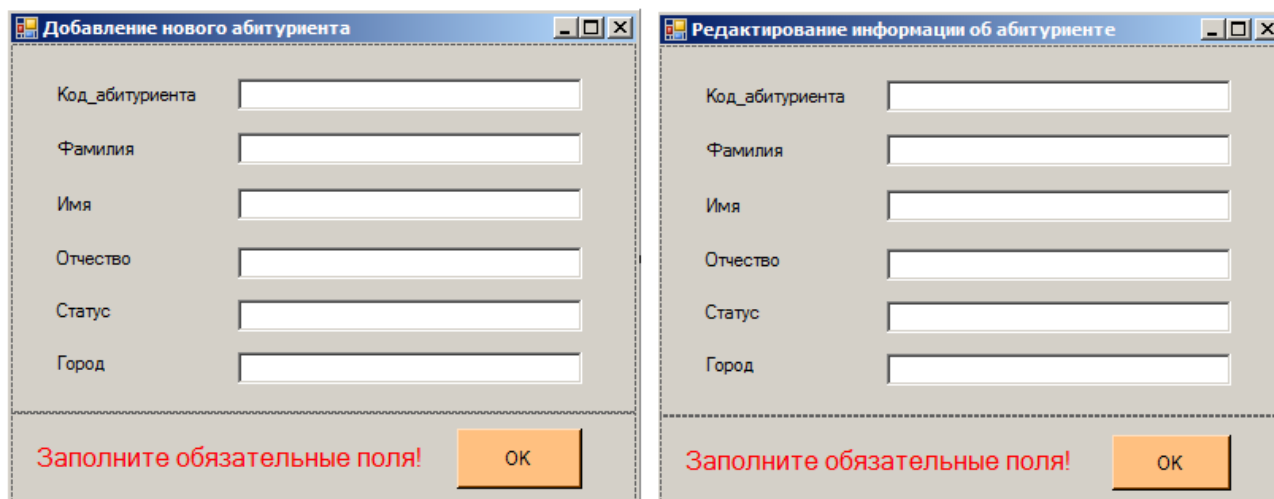


Рисунок 1 – Формы для добавления и редактирования записи абитуриента  
Для текстового поля Код\_абитуриента в свойстве Enabled укажите False.

4. Определим в классе формы объект Абитуриенты item и дополним конструктор:

```
public partial class FormEditAbiturient : Form
{
    Абитуриенты item;
    ссылка:1
    public FormEditAbiturient(Абитуриенты itm)
    {
        item = itm;
        InitializeComponent();
        label7.Visible = false;
        textBox1.Text = item.Код_абитуриента.ToString();
        textBox2.Text = item.Фамилия.ToString();
        textBox3.Text = item.Имя.ToString();
        textBox4.Text = item.Отчество.ToString();
        textBox5.Text = item.Статус.ToString();
        textBox6.Text = item.Город.ToString();
    }
}
```

5. В код, обрабатывающий нажатие кнопки ОК, добавим необходимые проверки ввода:

```
if (textBox2.Text.Trim() == "" || textBox3.Text.Trim() == ""
    || textBox4.Text.Trim() == "")
{
    label7.Visible = true;
    label7.Text = "Заполните поля \'Фамилия\', \'Имя\' и \'Отчество\'";
}
else if (textBox2.Text.ToCharArray()
    .All(w => ((w >= 'a' && w <= 'я')
    || (w >= 'А' && w <= 'Я') || w == ' ')) == false)
{
    label7.Visible = true;
    label7.Text = "Поле \'ФИО\' может содержать только кириллицу и пробелы";
}
else { }
```

6. Внутри оператора else добавляем следующий код, «обернутый» в конструкцию try-catch:

```
var result = ((Form1)Owner).db.Абитуриенты
.SingleOrDefault(w => w.Код_абитуриента == item.Код_абитуриента);
result.Фамилия = textBox2.Text.ToString();
result.Имя = textBox3.Text.ToString();
result.Отчество = textBox4.Text.ToString();
result.Статус = textBox5.Text.ToString();
result.Город = textBox6.Text.ToString();
((Form1)Owner).db.SaveChanges();
((Form1)Owner).sheetabiturients = ((Form1)Owner).db.Абитуриенты
.OrderBy(o => o.Код_абитуриента).ToList();
((Form1)Owner).dataGridView1.DataSource =
((Form1)Owner).sheetabiturients;
this.Close();
```

Здесь создается новый объект result, который на самом деле будет иметь тип Абитуриенты, и присваиваются его полям соответствующие значения из текстовых полей формы. После этого сохраняются изменения в экземпляре DbContext, обновляются данные в таблице Form1 и закрывается текущая форма. Использование неявной типизации переменной result здесь не является обязательным.

7. В форме добавления новой карточки пациента код поменяется незначительно. Не будет создаваться объект абитуриенты item, а сразу result:

```
public partial class FormAddAbiturient : Form
{
    Абитуриенты result = new Абитуриенты();
    ссылка: 1
    public FormAddAbiturient()
    {
        InitializeComponent();
        label7.Visible = false;
    }
}
```

8. Проверка вводимых данных будет иметь следующий вид:

```
if (textBox2.Text.Trim() == ""
    || textBox3.Text.Trim() == ""
    || textBox4.Text.Trim() == ""
    || textBox5.Text.Trim() == "")
{
    label7.Visible = true;
    label7.Text = "Заполните поля \'Фамилия\', \'Имя\', \'Отчество\' и \'Статус\'";
}
else if (textBox2.Text.ToCharArray()
.All(w => ((w >= 'a' && w <= 'я')
|| (w >= 'А' && w <= 'Я')
|| w == ' ')) == false)
{
    label7.Visible = true;
    label7.Text = "Поля \'ФИО\' может содержать только кириллицу и пробелы";
}
```

9. Внутри конструкции try-catch добавится заполнение полей таблицы Абитуриенты:

```
result.Фамилия = textBox2.Text;  
result.Имя = textBox3.Text;  
result.Отчество = textBox4.Text;  
result.Статус = textBox5.Text;  
result.Город = textBox6.Text;
```

10. А после заполнения полей – добавление объекта Абитуриенты result в экземпляр класса pr116\_IvanovaEntities:

```
((Form1)Owner).db.Абитуриенты.Add(result);
```

11. Сохранение изменений в DbContext, обновление списка абитуриентов и код внутри операторов catch идентичны рассмотренному ранее в форме редактирования информации об абитуриенте.

12. Проверьте разработанный функционал приложения: добавление и редактирование данных.