

## Комп'ютерний практикум №9

**Тема:** Автономна модель зберігання даних, Entity Framework, LINQ.

**Мета:** отримати поняття що таке автономна модель зберігання даних та як вона використовується; оглянути можливості Entity Framework та мови інтегрованих запитів LINQ, навчитись користуватись цими технологіями.

**Навички що будуть здобуті:** вміння отримувати доступ до створеної бази даних за допомогою можливостей, що надають автономна модель зберігання даних, Entity Framework та технологія LINQ.

**Зміст:**

- 1) Робота з автономною моделлю зберігання даних.
- 2) Робота з Entity Framework.
- 3) Робота з LINQ.

**Контрольні питання до роботи:**

1. Визначення і призначення об'єктів автономної моделі зберігання даних.
2. Об'єкт DataSet. Визначення, складові, як використовується.
3. Об'єкти DataTable, DataRow, DataColumn, DataView, Constraint, DataRelation. Визначення, складові, як використовуються.
4. Об'єкт SqlDataAdapter. Визначення, як використовується.
5. Об'єкт SqlCommandBuilder. Визначення, як використовується.
6. В чому полягає сенс використання збережених процедур?
7. Як вивести і підключити спеціальний бар для виконання операцій з таблицею BindingNavigator?
8. Що таке технологія Entity Framework, як вона працює?
9. Що таке контекст даних у технології Entity Framework?
10. Що таке лямбда-вираз? Який у них механізм роботи?
11. Чи можна використовувати оператори ref і out з лямбда виразами?
12. Як у лямбда виразах організована робота з параметрами що передаються? Чи може їх буди більше одного чи не бути взагалі?
13. Технологія LINQ. Визначення, різновиди.
14. Структура запиту LINQ. Ключові слова.
15. Механізм мінімізації запитів LINQ.
16. Фільтрація та сортування у обох напрямках у LINQ.
17. Використання агрегатних операцій у LINQ.
18. Перетворення об'єктів та робота з колекціями у LINQ.
19. Механізм розширення методів Extension Methods.

**Хід виконання роботи:**

1. Розберемо другий спосіб отримання доступу до бази даних, пов'язаний з використанням об'єктів SqlDataAdapter і DataSet. SqlDataReader, розглянутий у попередньому комп'ютерному практикумі, читав інформацію отриману з бази даних порядково, на відміну від нього DataSet представляє сховище даних з якими можна працювати незалежно від наявності підключення. SqlDataAdapter в свою чергу слугує для завантаження даних у DataSet. Щоб отримати з його допомогою дані потрібно організувати підключення та виконати команду SELECT:

SqlDataAdapter adapter = new SqlDataAdapter(sql, connection);

Або:

```
SqlDataAdapter adapter = new SqlDataAdapter(sql,
connectionString);
```

- Створимо новий проект Windows Forms. У ньому в файл App.config після startup додамо наступний код. Це ще один спосіб збереження строк підключення до БД, найбільш зручний у нашому випадку. Строк підключення може бути декілька:

```
<connectionStrings>
  <add name="Laba2_2Connection" connectionString="Data
Source=.\SQLEXPRESS; Initial Catalog=Laba2_2;Integrated
Security=True"
      providerName="System.Data.SqlClient"/>
  <add name="Laba6Connection" connectionString="Data
Source=.\SQLEXPRESS; Initial Catalog=Laba6;Integrated
Security=True"
      providerName="System.Data.SqlClient"/>
</connectionStrings>
```

- Додамо на форму DataGridView, а у посилання проекту бібліотеку System.Configuration. Підключимо її за допомогою using у файлі що містить код форми. Також підключимо бібліотеку System.Data.SqlClient та додамо наступний код:

```
string connectionString =
ConfigurationManager.ConnectionStrings["Laba2_2Connection"].Conn
ectionString;
string sql = "SELECT * FROM Student";
SqlDataAdapter adapter;
DataSet ds;

public Form1()
{
    InitializeComponent();

    using (SqlConnection connection = new
SqlConnection(connectionString))
    {
        //Инициализируем объект DataAdapter
        adapter = new SqlDataAdapter(sql, connection);
        //Инициализируем объект DataSet
        ds = new DataSet();
        //Заполняем DataSet
        adapter.Fill(ds);
        //Отображаем данные
        dataGridView1.DataSource = ds.Tables[0];
    }
}
```

- Отриманий код заповнює DataSet, а за його допомогою і DataGridView, даними таблиці Student. Ускладнимо приклад додавши посторінковий перегляд. Для цього додамо дві нових змінні:

```
int pSize = 14; //Размер страницы
int pNumber = 0; //Текущая страница
```

Ускладнимо заповнення DataSet даними та встановимо відповідність назв таблиць:

```
//Выбор сразу всей строки
```

```

dataGridView1.SelectionMode =
DataGridViewSelectionMode.FullRowSelect;
//Запрет добавления новых строк
dataGridView1.AllowUserToAddRows = false;
//Запрет редактирования таблицы
dataGridView1.ReadOnly = true;

using (SqlConnection connection = new
SqlConnection(connectionString))
{
    adapter = new SqlDataAdapter(GetSql(), connection);
    ds = new DataSet();
    adapter.Fill(ds, "Student");
    dataGridView1.DataSource = ds.Tables[0];
}

```

Додамо метод що буде змінювати запит до БД в залежності від вибраної сторінки:

```

private string GetSql()
{
    /*
        //Работать эта конструкция будет строго для SQL Server 2012
        (11.x) и более поздних версий
        return "SELECT StudentId, Surname, Name, Patronymic,
        Birthday, RecordBook, BirthdayCity, StudyGroupId, Stipendia " +
            "FROM(SELECT *, row_number() OVER(ORDER BY
        StudentId) as Nom FROM Student) as s " +
            "ORDER BY Nom OFFSET ((" + pNumber + ") * " + pSize
        + ") " + "ROWS FETCH NEXT " + pSize + "ROWS ONLY";
    */
    //Актуально для более ранних версий:
    return "SELECT StudentId, Surname, Name, Patronymic,
    Birthday, RecordBook, BirthdayCity, StudyGroupId, Stipendia " +
        "FROM(SELECT *, row_number() OVER(ORDER BY
    StudentId) as Nom FROM Student) as s " +
        "WHERE Nom BETWEEN " + (pSize * pNumber + 1) + " AND
    " + (pSize * pNumber + pSize) + " ORDER BY StudentId";
}

```

Додамо на форму дві нових кнопки та їх обробники:

```

//Обработчик кнопки Влево
private void button1_Click(object sender, EventArgs e)
{
    if (pNumber == 0) return;
    pNumber--;

    using (SqlConnection connection = new
SqlConnection(connectionString))
    {
        adapter = new SqlDataAdapter(GetSql(), connection);
        ds.Tables["Student"].Rows.Clear(); //Чистим таблицу
        Student нашего DataSet
        adapter.Fill(ds, "Student"); //И заполняем по новой
        конкретно её
    }
}

```

```

    }
}

//Обработчик кнопки Вправо
private void button2_Click(object sender, EventArgs e)
{
    if (ds.Tables["Student"].Rows.Count < pSize) return;
    pNumber++;

    using (SqlConnection connection = new
SqlConnection(connectionString))
    {
        adapter = new SqlDataAdapter(GetSql(), connection);
        ds.Tables["Student"].Rows.Clear();
        adapter.Fill(ds, "Student");
    }
}

```

5. DataSet працює з даними отриманими з БД автономно. Щоб зберегти зміни треба викликати відповідний метод SqlDataAdapter що має назву Update. Для внесення змін він використовує команди InsertCommand, UpdateCommand та DeleteCommand. Їх SQL вирази можна визначити самостійно або скористатись класом SqlCommandBuilder що генерує потрібні команди автоматично.

Створимо кнопку що додасть у базу даних новий, готовий запис найпростішим способом та дозволить подивитись на структуру команд SqlCommandBuilder. Зрозуміло що замість констант завжди можна підставити дані з полів вводу форми:

```

//Добавить строку
private void button3_Click(object sender, EventArgs e)
{
    using (SqlConnection connection = new
SqlConnection(connectionString))
    {
        //Связываем адаптер с конкретно этим подключением. Не
        обязательно каждый раз создавать новый адаптер
        adapter.SelectCommand = new SqlCommand(GetSql(),
connection);

        //Добавляем новую строку в таблицу датасета, а поскольку
        он связан с dataGridView, то и туда
        DataRow newRow = ds.Tables[0].NewRow();
        newRow["Surname"] = "Сидореко";
        newRow["Name"] = "Петро";
        newRow["Patronymic"] = "Осипович";
        newRow["StudyGroupId"] = 1;
        newRow["RecordBook"] = "IK-11-59";
        newRow["BirthdayCity"] = "Полтава";
        newRow["Birthday"] = "1979-12-06";
        newRow["Stipendia"] = 650;
        ds.Tables[0].Rows.Add(newRow);
    }
}

```

```

        //Создаем объект SqlCommandBuilder для того, чтобы можно
        было пользоваться обновлением и обновляем конкретно эту таблицу
        SqlCommandBuilder commandBuilder = new
        SqlCommandBuilder(adapter);
        adapter.Update(ds.Tables[0]);

        //Перезагружаем данные чтобы обновился StudentId который
        под автоинкрементом
        ds.Tables[0].Clear();
        adapter.Fill(ds.Tables[0]);

        //Смотрим на то, что из себя представляют встроенные
        команды

        Console.WriteLine(commandBuilder.GetUpdateCommand().CommandText)
        ;

        Console.WriteLine(commandBuilder.GetInsertCommand().CommandText)
        ;

        Console.WriteLine(commandBuilder.GetDeleteCommand().CommandText)
        ;
    }
}

```

6. Описаний вище метод додавання даних має один недолік — він потребує перезавантаження усіх даних таблиці хоча не вистачає лише id студента. Щоб позбутися його додамо нову кнопку а в її обробнику пропишемо наступний код.

```

//Добавить строку процедурой
private void button4_Click(object sender, EventArgs e)
{
    using (SqlConnection connection = new
    SqlConnection(connectionString))
    {
        adapter.SelectCommand = new SqlCommand(GetSql(),
        connection);

        //Устанавливаем команду на вставку
        adapter.InsertCommand = new SqlCommand("AddStudent",
        connection);
        //Указываем что это будет хранимая процедура
        adapter.InsertCommand.CommandType =
        CommandType.StoredProcedure;
        //Добавляем параметры
        adapter.InsertCommand.Parameters.Add(new
        SqlParameter("@surname", SqlDbType.VarChar, 20, "Surname"));
        adapter.InsertCommand.Parameters.Add(new
        SqlParameter("@name", SqlDbType.VarChar, 15, "Name"));
        adapter.InsertCommand.Parameters.Add(new
        SqlParameter("@patronymic", SqlDbType.VarChar, 20,
        "Patronymic"));
    }
}

```

```

        adapter.InsertCommand.Parameters.Add(new
SqlParameter("@studyGroupId", SqlDbType.Int, 0,
"StudyGroupId"));
        adapter.InsertCommand.Parameters.Add(new
SqlParameter("@recordBook", SqlDbType.VarChar, 10,
"RecordBook"));
        adapter.InsertCommand.Parameters.Add(new
SqlParameter("@birthdayCity", SqlDbType.VarChar, 10,
"BirthdayCity"));
        adapter.InsertCommand.Parameters.Add(new
SqlParameter("@birthday", SqlDbType.Date, 0, "Birthday"));
        adapter.InsertCommand.Parameters.Add(new
SqlParameter("@stipendia", SqlDbType.Money, 0, "Stipendia"));

//Добавляем выходной параметр для Id
SqlParameter parameter =
adapter.InsertCommand.Parameters.Add("@studentId",
SqlDbType.Int, 0, "StudentId");
parameter.Direction = ParameterDirection.Output;

//Добавим новую строку в таблицу
DataRow newRow = ds.Tables[0].NewRow();
newRow["Surname"] = "Петренко";
newRow["Name"] = "Дмитро";
newRow["Patronymic"] = "Васильович";
newRow["StudyGroupId"] = 1;
newRow["RecordBook"] = "IK-11-59";
newRow["BirthdayCity"] = "Бахмут";
newRow["Birthday"] = "1979-06-06";
newRow["Stipendia"] = 330;
ds.Tables[0].Rows.Add(newRow);

//По итогу получим StudentId чт и подтвердит
использование хранимой процедуры
adapter.Update(ds.Tables[0]);
ds.Tables[0].AcceptChanges();
    }
}

```

7. Цей код перевизначає команду додавання даних до таблиці і замість згенерованої автоматично запускається наша власна команда що складається з збереженої процедури AddStudent. При додаванні запису вона повертає його Id. Створимо цю збережену процедуру у SQL Server Menagement Studio:

```

CREATE PROCEDURE AddStudent
@surname varchar(20),
@name varchar(15),
@patronymic varchar(20),
@studyGroupId int,
@recordBook varchar(10),
@birthdayCity varchar(10),
@birthday date,
@stipendia money,

```

```

@studentId int out
AS
    INSERT INTO Student (Surname, Name, Patronymic,
StudyGroupId, RecordBook, BirthdayCity, Birthday, Stipendia)
    VALUES (@surname, @name, @patronymic, @studyGroupId,
@recordBook, @birthdayCity, @birthday, @stipendia)

    SET @studentId=SCOPE_IDENTITY()
GO

```

8. Отже ми отримали простий додаток що може перемикаати сторінки отриманих рядків та додавати рядки до бази даних двома способами. Ми розглянули використання збережених процедур та перевизначення готових команд призначених для автоматичної роботи з базою даних.

StudentId	Surname	Name	Patronymic	StudyGroupId	RecordBook	BirthdayCity	B
1	Стасюк	Назар	Леонідович	1	IK-11-01	Київ	05
2	Сиденко	Катя	Леонідівна	1	IK-11-02	Тернопіль	06
3	Сиденко	Катя	Леонідівна	1	IK-11-02	Київ	07
4	Мельниченко	Максим	Леонідович	2	IK-12-01	Чернігів	10
5	Карнага	Ярослав		2	IK-12-03	Київ	14
6	Карнага	Ярослава	Ігорівна	2	IK-12-03	Москва	14
7	Волокита	Артем	Миколайович	3	IK-23-01	Київ	09
8	Саливоненко	Наталка	Анатоліївна	3	IK-23-02	Київ	09
9	Краснонос	Наталка	Іванівна	4	IK-24-01	Черкаси	12
10	Персиков	Іван	Петрович	4	IK-24-03	Київ	23
11	Перончиков	Михайло		4	IK-24-04	Луцьк	12
12	Пасичников	Михайло	Петрович	4	IK-24-04	Луцьк	12
13	Масливець	Артем	Петрович	4	IK-24-04	Київ	12
14	Карнага	Ярослав	Ігоревич	2	IK-12-12	Ялта	14

9. Створимо нарешті повноцінну форму для роботи з таблицею що дозволить додавати, видаляти та редагувати записи. Для цього використаємо нову вкладку TabControl у тій самій формі, або нову форму. Розмістимо там DataGridView та 3 кнопки. Заповнимо другу таблицю при створенні. Нехай для різноманітності це буде Employee:

```

string sql2 = "SELECT * FROM Employee";

//Заполнение второй таблички
dataGridView2.SelectionMode =
System.Windows.Forms.DataGridViewSelectionMode.FullRowSelect;
dataGridView2.AllowUserToAddRows = false;
using (SqlConnection connection = new
SqlConnection(connectionString))
{
    adapter = new SqlDataAdapter(sql2, connection);
    adapter.Fill(ds, "Employee");
    dataGridView2.DataSource = ds.Tables[1];
    dataGridView2.Columns["EmployeeId"].ReadOnly = true;
}

```



10. Додамо обробники кнопок додавання та видалення рядків за бази даних:

```
//Добавить строку
private void button7_Click(object sender, EventArgs e)
{
    //Добавляем новую пустую строку по той же схеме что и в
    //таблице и добавляем её туда
    DataRow row = ds.Tables[1].NewRow();
    ds.Tables[1].Rows.Add(row);

    //Выделяем созданную строку
    dataGridView2.ClearSelection();
    dataGridView2.Rows[dataGridView2.Rows.Count - 1].Selected =
true;
    dataGridView2.FirstDisplayedScrollingRowIndex =
dataGridView2.Rows.Count - 1;
}

//Удалить строку
private void button6_Click(object sender, EventArgs e)
{
    //Удаляем выделенные строки из dataGridView2
    foreach (DataGridViewRow row in dataGridView2.SelectedRows)
    {
        dataGridView2.Rows.Remove(row);
    }
}
```

11. Створимо обробник кнопки збереження що працює подібною до розглянутої вище процедури:

```
//Сохранить изменения
private void button5_Click(object sender, EventArgs e)
{
    BindingContext[ds, "Employee"].EndCurrentEdit();

    using (SqlConnection connection = new
SqlConnection(connectionString))
    {
        adapter = new SqlDataAdapter(sql2, connection);
        SqlCommandBuilder commandBuilder = new
SqlCommandBuilder(adapter);
        adapter.InsertCommand = new SqlCommand("AddEmployee",
connection);
        adapter.InsertCommand.CommandType =
CommandType.StoredProcedure;
        adapter.InsertCommand.Parameters.Add(new
SqlParameter("@surname", SqlDbType.VarChar, 20, "Surname"));
        adapter.InsertCommand.Parameters.Add(new
SqlParameter("@name", SqlDbType.VarChar, 15, "Name"));
        adapter.InsertCommand.Parameters.Add(new
SqlParameter("@patronymic", SqlDbType.VarChar, 20,
"Patronymic"));
    }
}
```



```

        adapter.InsertCommand.Parameters.Add(new
SqlParameter("@dutiesId", SqlDbType.Int, 0, "DutiesId"));
        adapter.InsertCommand.Parameters.Add(new
SqlParameter("@birthdayCity", SqlDbType.VarChar, 10,
"BirthdayCity"));
        adapter.InsertCommand.Parameters.Add(new
SqlParameter("@birthday", SqlDbType.Date, 0, "Birthday"));
        adapter.InsertCommand.Parameters.Add(new
SqlParameter("@oklad", SqlDbType.Money, 0, "Oklad"));
        adapter.InsertCommand.Parameters.Add(new
SqlParameter("@nadbavka", SqlDbType.Money, 0, "Nadbavka"));
        adapter.InsertCommand.Parameters.Add(new
SqlParameter("@dataVuplatu", SqlDbType.Date, 0, "DataVuplatu"));

        SqlParameter parameter =
adapter.InsertCommand.Parameters.Add("@employeeId",
SqlDbType.Int, 0, "EmployeeId");
        parameter.Direction = ParameterDirection.Output;

        adapter.Update(ds.Tables[1]);
    }
}

```

Саму збережену процедуру:

```

CREATE PROCEDURE AddEmployee
    @surname varchar(20),
    @name varchar(15),
    @patronymic varchar(20),
    @dutiesId int,
    @birthdayCity varchar(10),
    @birthday date,
    @oklad money,
    @nadbavka money,
    @dataVuplatu date,
    @employeeId int out
AS
    INSERT INTO Employee (Surname, Name, Patronymic, DutiesId,
    BirthdayCity, Birthday, Oklad, Nadbavka, DataVuplatu)
    VALUES (@surname, @name, @patronymic, @dutiesId,
    @birthdayCity, @birthday, @oklad, @nadbavka, @dataVuplatu)

    SET @employeeId=SCOPE_IDENTITY()
GO

```

12. Отриманий код дозволяє вільно працювати з таблицею, додавати нові рядки, видаляти, змінювати їх та після зберігати зміни.
13. Додамо сюди механізм запитання чи були здійснені зміни у базі даних. Для цього додамо змінну:

```
bool isChanges = false;
```

Підпишемося на події DataTable після заповнення таблиці початковими даними:

```

//Спрашиває перед закритием если есть изменения
ds.Tables[1].RowChanged += delegate { isChanges = true; };

```

```
ds.Tables[1].RowDeleted += delegate { isChanges = true; };
ds.Tables[1].TableNewRow += delegate { isChanges = true; };
```

Додамо запитання при закритті форми:

```
private void Form1_FormClosing(object sender,
FormClosingEventArgs e)
{
    if (isChanges && MessageBox.Show("Хотите закрыть
форму не сохранив изменения?", "Внимание!",
MessageBoxButtons.YesNo, MessageBoxIcon.Information) ==
DialogResult.No)
    {
        e.Cancel = true;
    }
}
```

Та зміну isChanges при збереженні:

```
isChanges = false;
```

14. Тепер щоб форма роботи з таблицею була взагалі повноцінною добавимо на неї TextBox, ComboBox та DateTimePicker що потрібні для відображення усіх полів таблиці Empolyee:

15. Та прив'яжемо поля цих елементів до створеного DataSet (ds.Tables[1]). Це дозволить оновлювати інформацію у TextBox, ComboBox та DateTimePicker при виборі рядків DataGridView.

```
//Привязка полей ввода к датасету
textBox1.Enabled = false;
textBox1.DataBindings.Add("Text", ds.Tables[1], "EmployeeId");
textBox2.DataBindings.Add("Text", ds.Tables[1], "Surname",
true);
textBox3.DataBindings.Add("Text", ds.Tables[1], "Name", true);
textBox4.DataBindings.Add("Text", ds.Tables[1], "Patronymic",
true);
using (SqlConnection connection = new
SqlConnection(connectionString))
{
```

```

        adapter = new SqlDataAdapter("SELECT d.DutiesId, p.Name + '
' + s.Name AS Nam FROM Duties AS d " +
                                   "JOIN Position AS p ON
p.PositionId = d.PositionId " +
                                   "JOIN Subdivision AS s ON
s.SubdivisionId = d.SubdivisionId", connection);
        adapter.Fill(ds, "Duties");
        comboBox1.DataSource = ds.Tables[2];
        comboBox1.DisplayMember = "Nam";
        comboBox1.ValueMember = "DutiesId";
        comboBox1.DataBindings.Add("SelectedValue", ds.Tables[1],
"DutiesId", true);
    }
    textBox5.DataBindings.Add("Text", ds.Tables[1], "BirthdayCity",
true);
    dateTimePicker1.DataBindings.Add("Value", ds.Tables[1],
"Birthday", true);
    textBox6.DataBindings.Add("Text", ds.Tables[1], "Oklad", true);
    textBox7.DataBindings.Add("Text", ds.Tables[1], "Nadbavka",
true);
    dateTimePicker2.DataBindings.Add("Value", ds.Tables[1],
"DataVuplatu", true);

```

16. Щоб дозволити зберігати інформацію змінену у доданих елементах управління змінимо обробник натискання на кнопку «Збереження» додавши перед створенням підключення наступний код:

```
BindingContext[ds, "Employee"].EndCurrentEdit();
```

Додамо його також перед логікою обробника закриття форми,

17. Отже в результаті ми маємо повноцінну форму що дозволяє працювати з таблицею, додавати дані, змінювати, видаляти. Причому інформація виводиться не лише у DataGridView, а і у прив'язані елементи управління один з яких працює не просто з Id зв'язаної таблиці, а з конкретними іменами посад та кафедр співробітників.

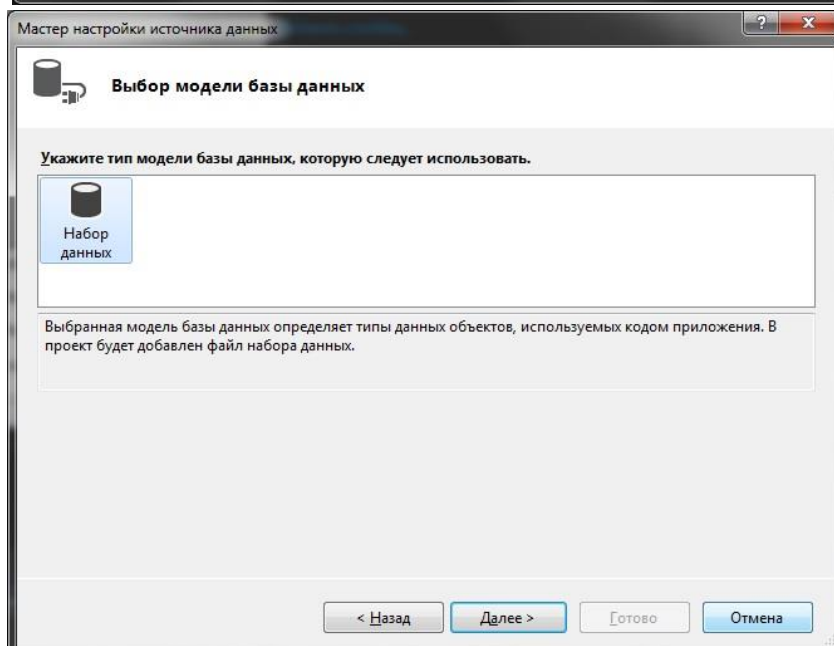
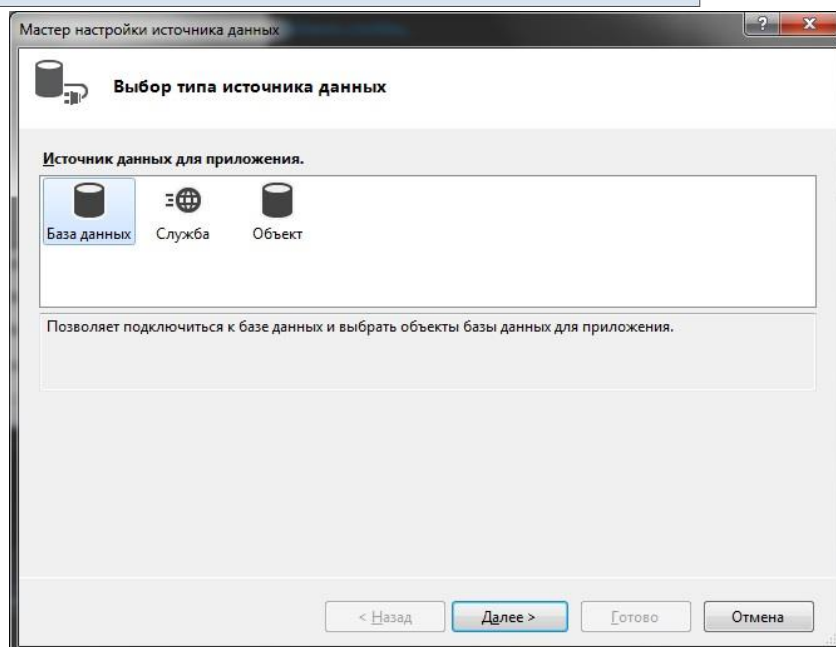
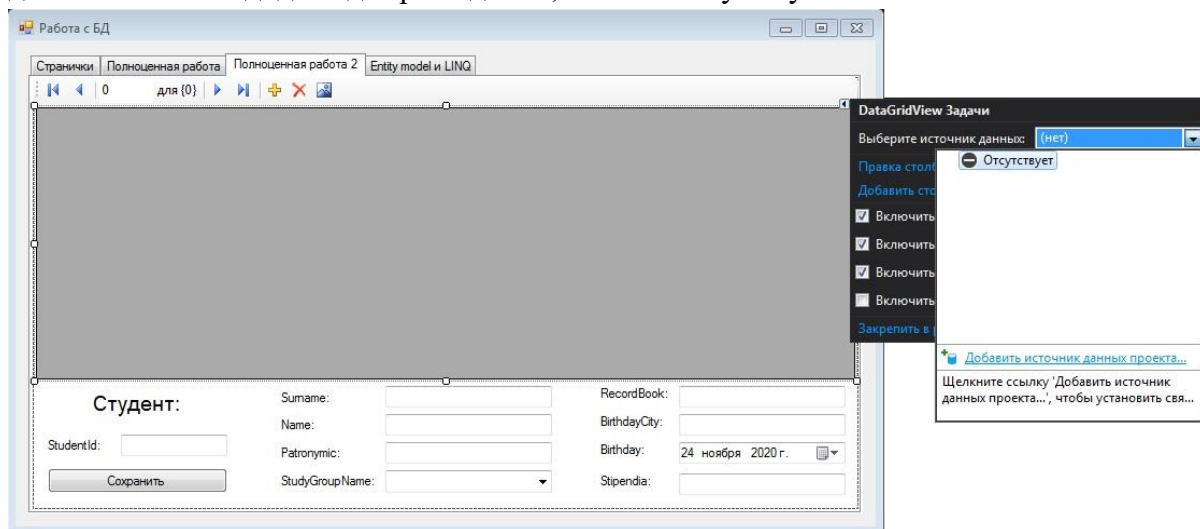
The screenshot displays a Windows application window titled "Работа с БД". It features a tabbed interface with tabs for "Странички", "Полноценная работа", "Полноценная работа 2", and "Entity model и LINQ". The "Полноценная работа" tab is active, showing a data grid with the following columns: EmployeeId, Surname, Name, Patronymic, DutiesId, BirthdayCity, and Birthday. The grid contains 11 rows of employee data.

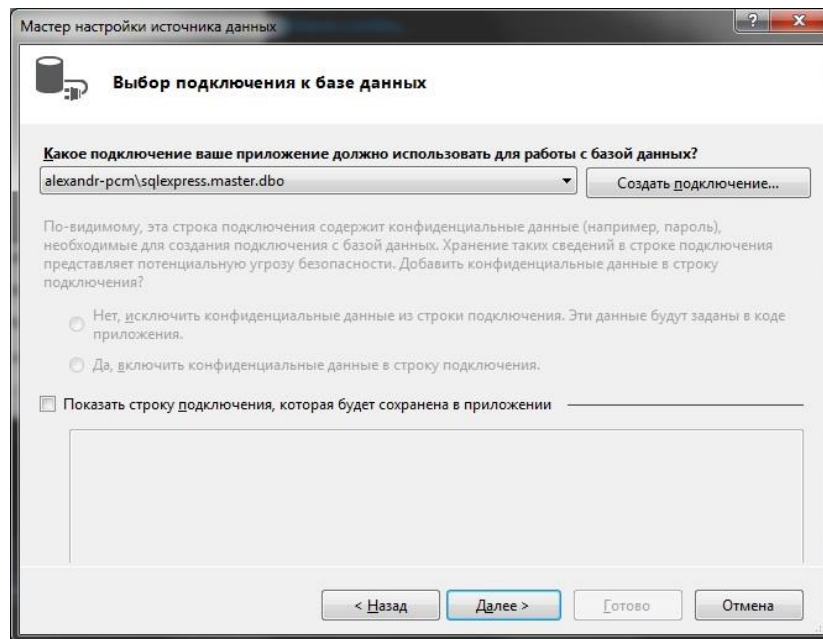
Below the grid is a form titled "Преподаватель:" for editing an employee's details. The form includes the following fields and controls:

- Surname:** Петренко
- Name:** Максим
- Patronymic:** Леонідович
- BirthdayCity:** Київ
- Birthday:** 05.01.1992
- Oklad:** 4000.5666
- Nadbavka:** 300.0000
- Duties:** A dropdown menu is open, showing a list of job positions and departments:
  - Професор Кафедра 1
  - Доцент Кафедра 1
  - Асистент Кафедра 1
  - Професор Кафедра 1
  - Методист Відділ 1
  - Інспектор Відділ 1
  - Начальник Департаменту
  - Лаборант Сектор 1
  - Доцент Кафедра 2
  - Асистент Кафедра 2
  - Професор Кафедра 2
  - Лаборант старший Сектор 2

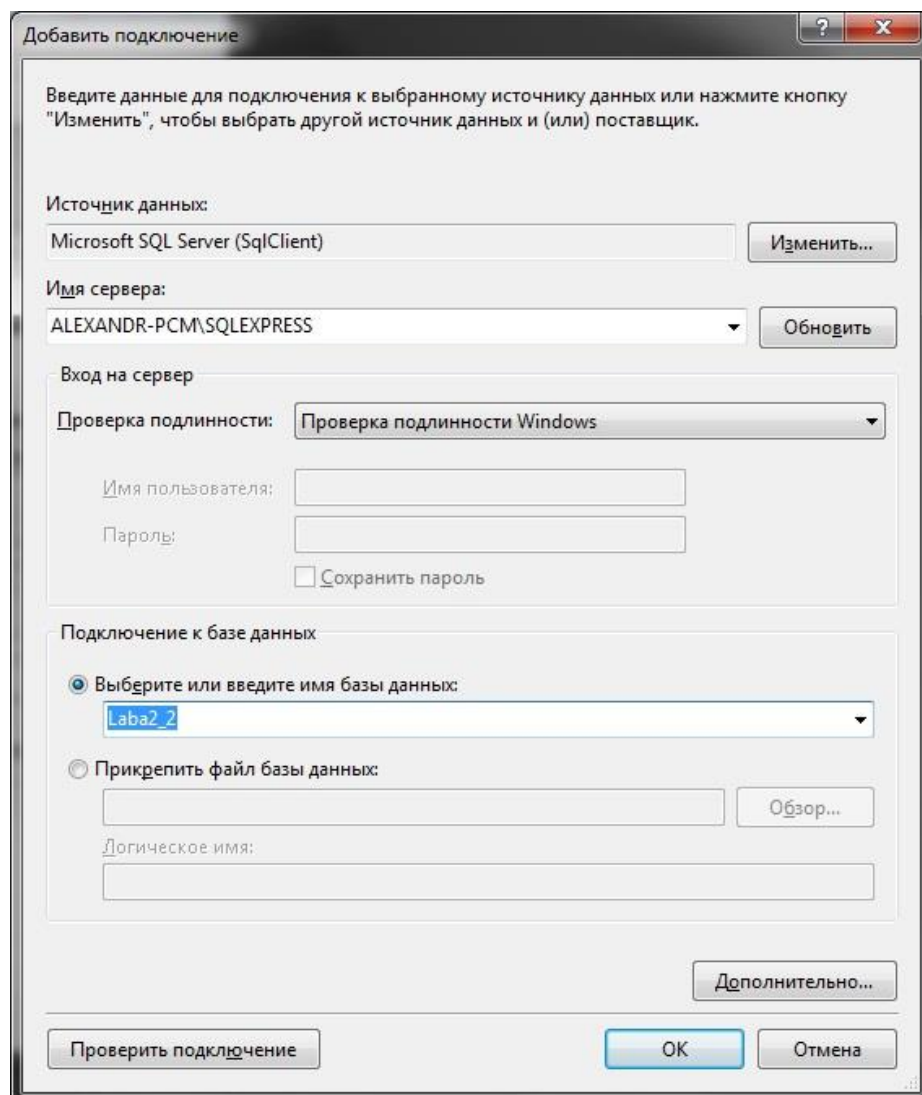
At the bottom of the form are four buttons: "Добавить", "Сохранить", "Удалить", and "Отмена".

18. Аналогічного результату можна добитися і більш легким шляхом. Для цього потрібно до DataGridView додати джерело даних, власне нашу базу:

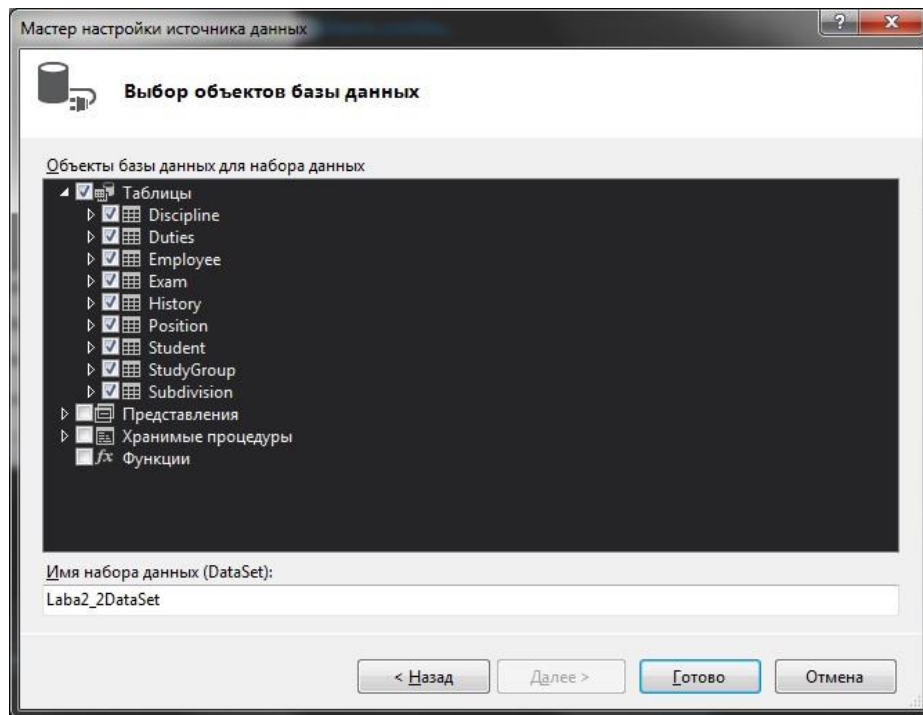




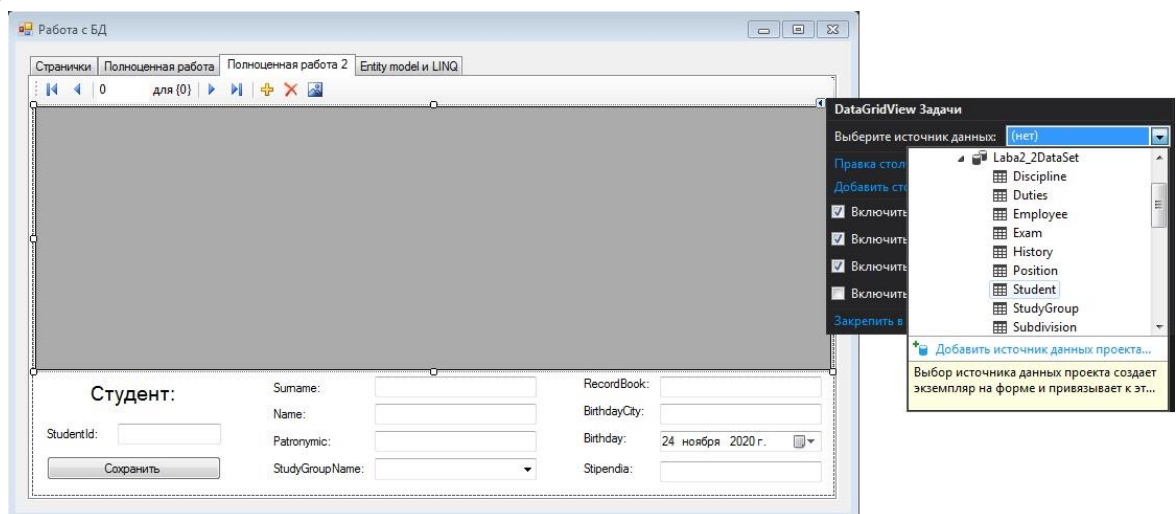
19. Потім налаштувати підключення де ім'я серверу копіюємо з Microsoft SQL Management Studio, а назву бази даних обираємо з випадаючого списку коли з'єднання буде встановлено:



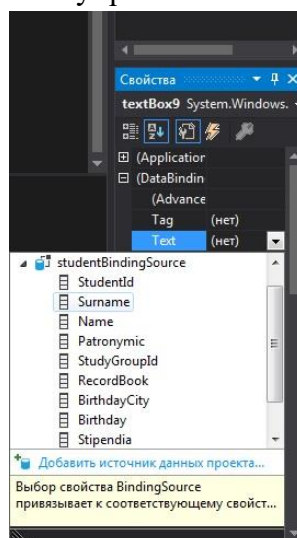
20. Далі потрібно обрати усі необхідні таблиці та натиснути на кнопку «Готово»:



21. Буде створено DataSet який і використовуватиметься далі як джерело даних. Фактично весь той код що ми писали може бути створений автоматично через вибір джерела даних у DataGridView:

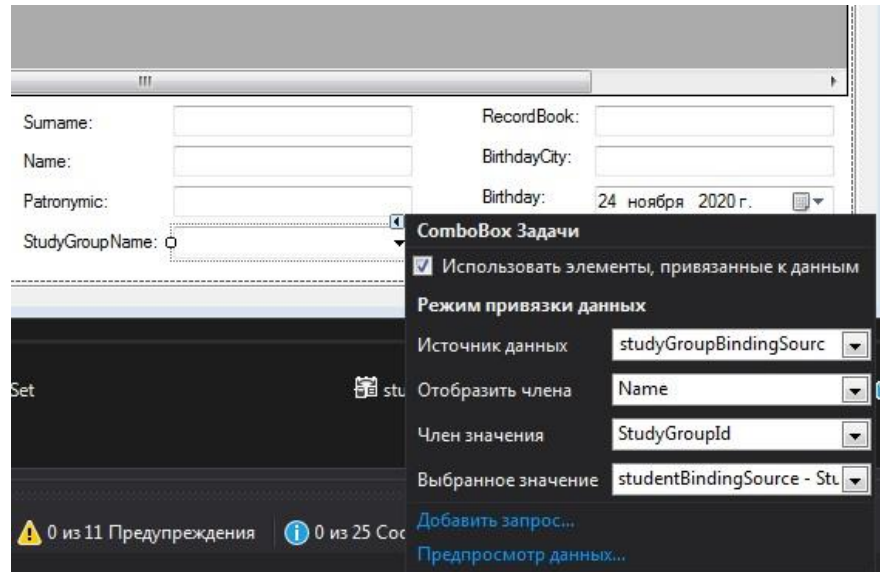


Через прив'язку полів Text елементів управління TextBox та DateTimePicker:





Поля SelectedValue для ComboBox:



22. Залишається лише додати BindingNavigator, записати в його властивість BindingSource значення studentBindingSource, та додати кнопку збереження змін у таблиці з наступним кодом:

```
private void toolStripButton1_Click(object sender, EventArgs e)
{
    this.Validate();
    this.studentBindingSource.EndEdit();
    this.studentTableAdapter.Update(laba2_2DataSet.Student);
}
```

23. Не менш простою виглядає і робота з базою даних за допомогою Entity Framework. Суть цієї технології полягає в створенні моделі бази даних з окремих класів, з можливістю вільного маніпулювання ними та легким збереженням змін у базу даних. В якості прикладу роздивимось дві таблиці Game Ganre цієї гри створених у Entity Framework:

```
public class Ganre
{
    public int Id { get; set; }
    public string Name { get; set; }
    public ICollection<Game> Games { get; set; }
    public Ganre()
    {
        Games = new List<Game>();
    }
}

public class Game
{
    public int Id { get; set; }
    public string Name { get; set; }
    public int Price { get; set; }
    public int GanreId { get; set; }
    public Ganre Ganre { get; set; }
}
```

24. Гра має посилання на свій жанр, а жанр на список ігор що його використовують. Саму модель створює клас Context в якому і містяться усі колекції даних:



```

public class GameContext : DbContext
{
    public GameContext() : base("DefaultConnection")
    { }

    public DbSet<Game> Games { get; set; }
    public DbSet<Ganre> Ganres { get; set; }
}

```

25. Ініціалізуємо та заповнимо цю базу даних:

```

Laba2_2Entities db = new Laba2_2Entities();

//Добавление записей в БД
using (GameContext db = new GameContext())
{
    Ganre g1 = new Ganre { Name = "Стратегия" };
    Ganre g2 = new Ganre { Name = "Шутер" };
    Ganre g3 = new Ganre { Name = "Ролевая" };
    Ganre g4 = new Ganre { Name = "Аркада" };
    db.Ganres.Add(g1);
    db.Ganres.Add(g2);
    db.Ganres.Add(g3);
    db.Ganres.Add(g4);
    //Сохранение изменений
    db.SaveChanges();

    Game gg1 = new Game { Name = "Counter - Strike", Price =
200, Ganre = g2 };
    Game gg2 = new Game { Name = "StarCraft II", Price = 914,
Ganre = g1 };
    Game gg3 = new Game { Name = "Total War : Attila", Price =
633, Ganre = g2 };
    Game gg4 = new Game { Name = "Kingdom Come Deliverance",
Price = 1083, Ganre = g3 };
    db.Games.AddRange(new List<Game> { gg1, gg2, gg3, gg4 });
    db.SaveChanges();
}

```

26. Додамо метод що дозволяє переглянути те, що міститься у створеній моделі:

```

private void WriteEntity()
{
    using (GameContext db = new GameContext())
    {
        var games = db.Games.ToList();
        foreach (var g in games)
            Console.WriteLine("{0} {1} {2} {3}", g.Id, g.Name,
g.Price, g.GanreId);
        Console.WriteLine();

        var ganres = db.Ganres.ToList();
        foreach (var g in ganres)
            Console.WriteLine("{0} {1}", g.Id, g.Name);
        Console.WriteLine("\n");
    }
}

```

```

        /*foreach (Ganre g in db.Ganres.Include(g => g.Games))
        {
            Console.WriteLine("Жанр: {0}", g.Name);
            foreach (Game gg in g.Games)
            {
                Console.WriteLine("{0} - {1}", gg.Id, gg.Name,
gg.Price, g.Name);
            }
        }*/
    }
}

```

27. Додамо приклад зміни записів та їх видалення:

```

//Изменение записей и удаление
Game gg5, gg6;
using (GameContext db = new GameContext())
{
    // получаем первый объект
    gg5 = db.Games.FirstOrDefault();
    gg6 = db.Games.Find(3);
    Game gg7 = db.Games.Find(4);
    gg5.Price = 500;
    gg6.Ganre = db.Ganres.FirstOrDefault();
    db.Games.Remove(gg7);
    db.SaveChanges();
}
WriteEntity();

```

Причому для різних контекстів сеансів редагування:

```

//По же самое, но для данных из другого контекста
using (GameContext db = new GameContext())
{
    if (gg5 != null)
    {
        gg5.Price = 1000;
        db.Entry(gg5).State = EntityState.Modified;
        db.SaveChanges();
    }
    if (gg6 != null)
    {
        db.Entry(gg6).State = EntityState.Deleted;
        db.SaveChanges();
    }
}
WriteEntity();

```

28. Ну і видалимо усі записи з бази:

```

//Удаление всех записей
using (GameContext db = new GameContext())
{
    //Удаление всех записей с сохранением нумерации
    db.Games.RemoveRange(db.Games);
    db.Ganres.RemoveRange(db.Ganres);
}

```

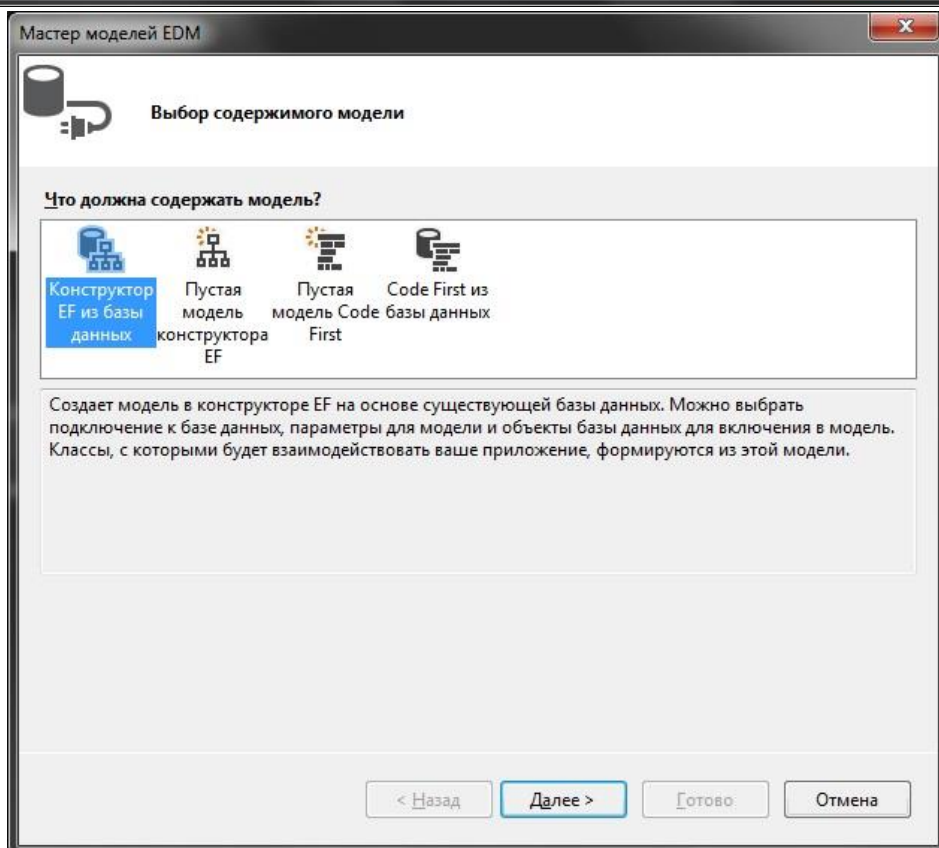
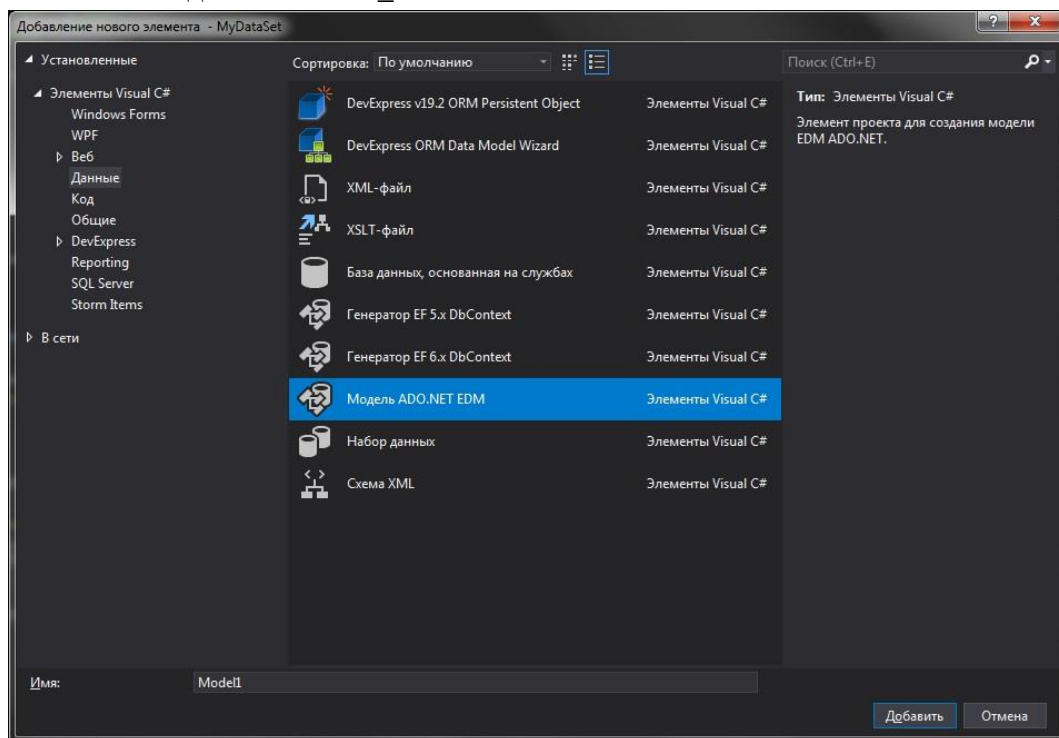
```
db.SaveChanges();
```

```
//Полное удаление
```

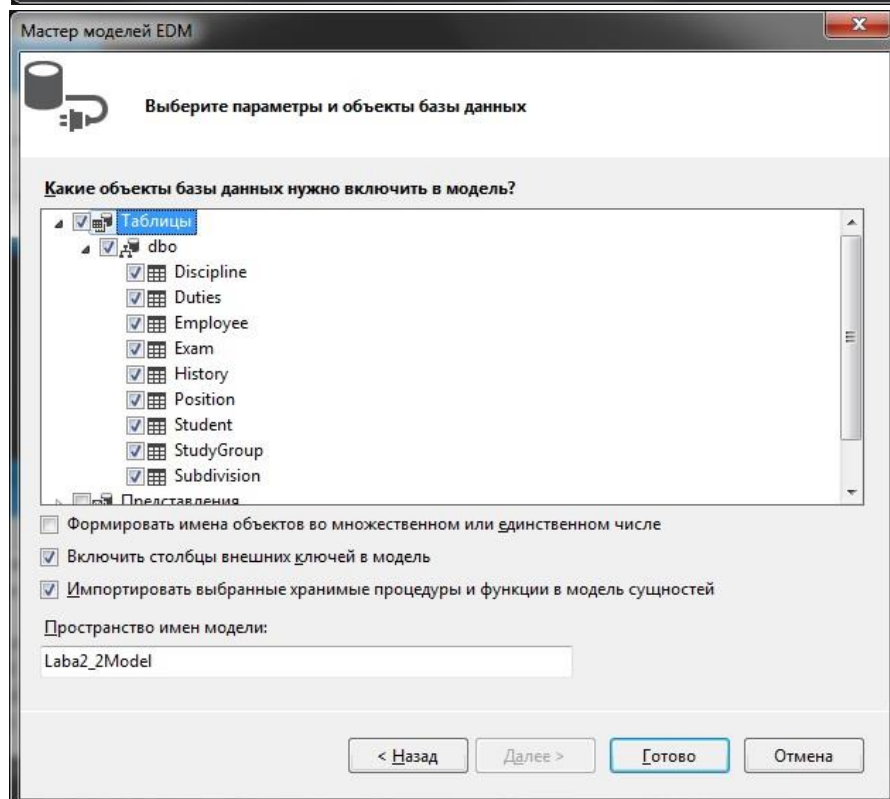
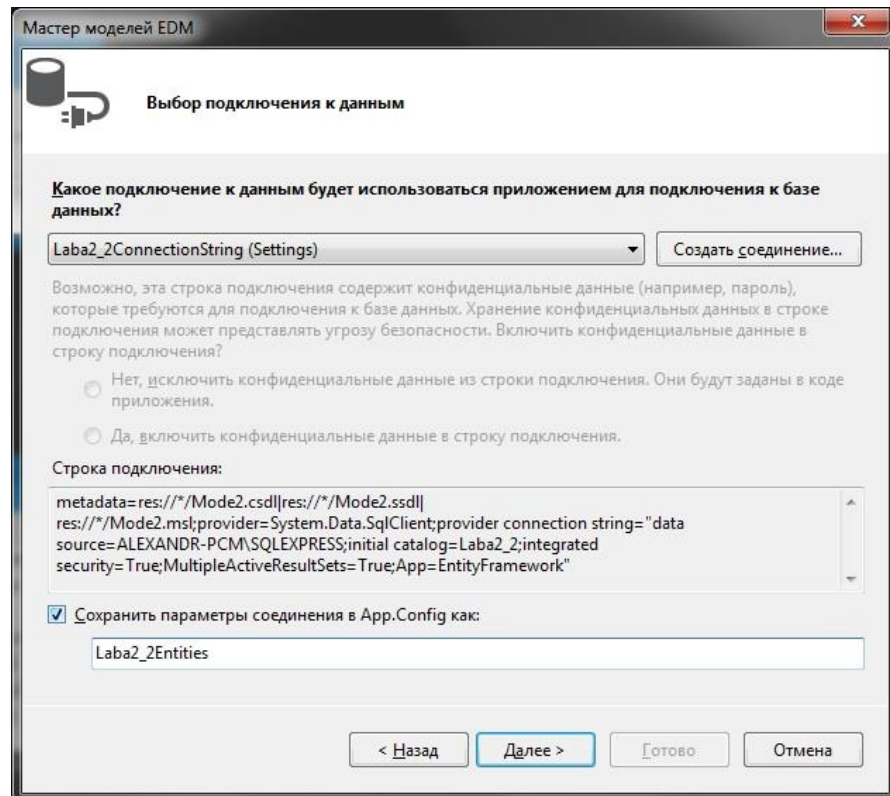
```
db.Database.ExecuteSqlCommand("TRUNCATE TABLE [Games]");
```

```
db.Database.ExecuteSqlCommand("DELETE FROM Ganres; DBCC  
CHECKIDENT ('Ganres',RESEED, 0)");  
}
```

29. Якщо додати до проекту модель бази даних з минулих лабораторних робіт ми зможемо робити те саме і з даними Laba2\_2:



30. При створенні моделі підключення створюється аналогічним чином до вже розглянутого:



31. Після чого можна завантажити дані у DataGridView з тільки но створеної моделі:

```
dataGridView4.DataSource = db.StudyGroup.ToList();
```

Та як варіант створити кілька запитів до бази даних за допомогою LINQ. Перший завантажить у dataGridView4 більш зручний варіант StudyGroup:

```
private void button9_Click(object sender, EventArgs e)
{
```

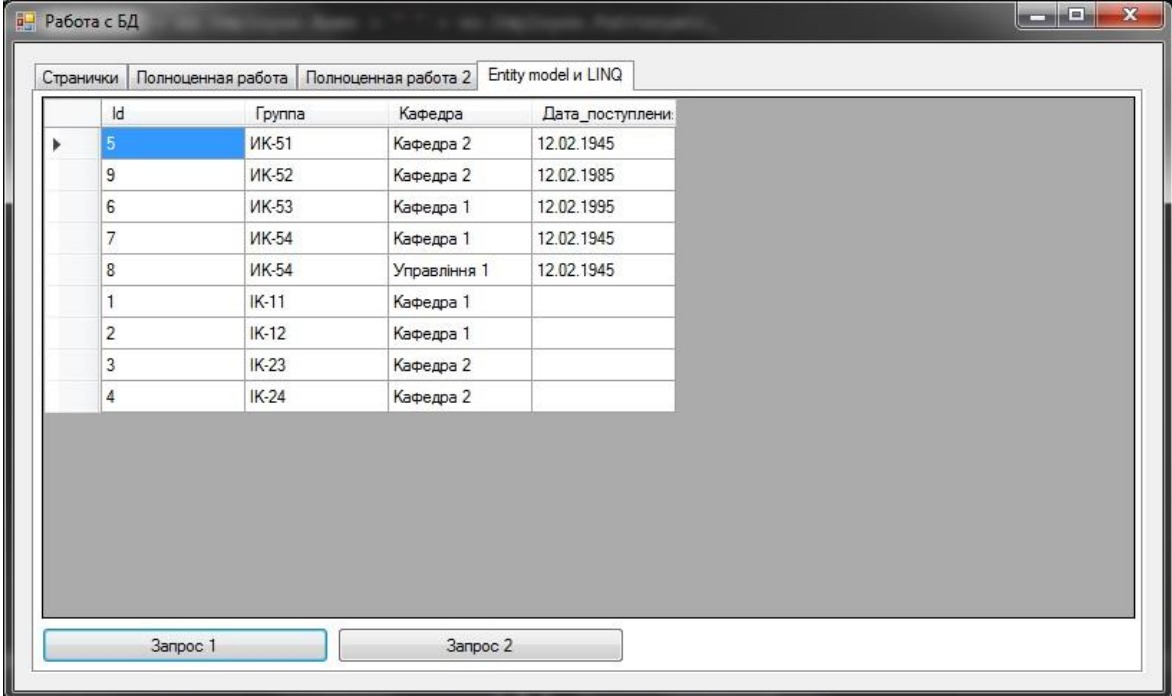
```

var query = from g in db.StudyGroup
            //join s in db.Subdivision on
g.SubdivisionId equals s.SubdivisionId
            orderby g.Name
            select new
            {
                Id = g.StudyGroupId,
                Группа = g.Name,
                //Кафедра = s.Name,
                Кафедра = g.Subdivision.Name,
                Дата_поступления = g.DataVstup
            };

dataGridView4.DataSource = query.ToList();
}

```

А другий завантажить туди таблицю оцінок тих студентів, чий загальний середній бал вищий за середній бал по інституту:



Id	Группа	Кафедра	Дата_поступления
5	ИК-51	Кафедра 2	12.02.1945
9	ИК-52	Кафедра 2	12.02.1985
6	ИК-53	Кафедра 1	12.02.1995
7	ИК-54	Кафедра 1	12.02.1945
8	ИК-54	Управління 1	12.02.1945
1	ИК-11	Кафедра 1	
2	ИК-12	Кафедра 1	
3	ИК-23	Кафедра 2	
4	ИК-24	Кафедра 2	

```

private void button10_Click(object sender, EventArgs e)
{
    var query = from ex in db.Exam
                join s in db.Student on ex.StudentId equals
s.StudentId
                where s.Exam.Average(n => n.Mark) >
db.Exam.Average(n => n.Mark)
                orderby s.StudentId
                select new
                {
                    Фамилия = ex.Student.Surname,
                    Имя = ex.Student.Name,
                    Отчество = ex.Student.Patronymic,
                    Группа = ex.Student.StudyGroup.Name,
                    Дисциплина = ex.Discipline.Name,

```

```

        Преподаватель = ex.Employee.Surname + " " +
ex.Employee.Name + " " + ex.Employee.Patronymic,
        Оценка = ex.Mark,
        Дата_сдачи = ex.DateExam,
        //m1 = s.Exam.Average(n => n.Mark),
        //m2 = db.Exam.Average(n => n.Mark)
    };

    dataGridView4.DataSource = query.ToList();
}

```

### 32. У якості самостійної роботи:

32.1. Виконати завдання згідно з варіантом (додаток 1).

## Додаток 1

Варіанти виконання самостійної частини роботи:

### Варіант 1: Замоклення у книжковому магазині.

1. Створити повноцінну форму для роботи з таблицею Книга обома варіантами що наведено у комп'ютерному практикумі.
2. За допомогою LINQ створити наступний запит до БД: вивести назву книги, середню кількість замовлень цієї книги, жанр книги, середню кількість замовлень книг даного жанру для книг, чия середня кількість замовлень вище, ніж середня кількість замовлень в даному жанрі.
3. Вивести його результати у DataGridView.

### Варіант 2: Ресторан.

1. Створити повноцінну форму для роботи з таблицею СкладСтрави обома варіантами що наведено у комп'ютерному практикумі.
2. За допомогою LINQ створити наступний запит до БД: вивести ПІБ співробітника, середню суму рахунку співробітника, назву посади, середню суму рахунку по посаді для співробітників, чия середня сума рахунку вище, ніж середня сума рахунку по їх посаді.
3. Вивести його результати у DataGridView.

### Варіант 3: Автосервіс.

1. Створити повноцінну форму для роботи з таблицею Сервіс обома варіантами що наведено у комп'ютерному практикумі.
2. За допомогою LINQ створити наступний запит до БД: вивести державний № автомобілю, середню ціну сервісу, назву марки, середню ціну сервісу марки та країну виробника для автомобілів, чия середня ціна сервісу вище, ніж середня ціну сервісу для їх марки.
3. Вивести його результати у DataGridView.

**Варіант 4: Музична крамниця.**

1. Створити повноцінну форму для роботи з таблицею Інструмент обома варіантами що наведено у комп'ютерному практикумі.
2. За допомогою LINQ створити наступний запит до БД: вивести Назву інструмента, середню кількість проданих екземплярів, назву марки, середню кількість проданих екземплярів марки та рік виробництва для інструментів, чия середня кількість проданих екземплярів вище, ніж середня кількість проданих екземплярів для їх марки.
3. Вивести його результати у DataGridView.

**Варіант 5: Поліклініка.**

1. Створити повноцінну форму для роботи з таблицею Прийом обома варіантами що наведено у комп'ютерному практикумі.
2. За допомогою LINQ створити наступний запит до БД: вивести ПІБ співробітника, середній термін лікування, назву посади, середній термін лікування по посаді для співробітників, що ставили середній термін лікування більше, ніж середній термін лікування для їх посади.
3. Вивести його результати у DataGridView.