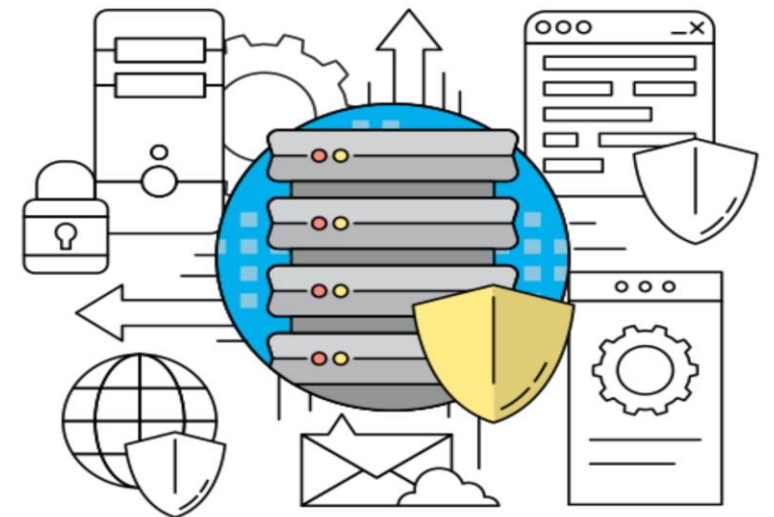


# Распределенные информационные системы

# Лекция №10

# Проектирование РИС

# Фаза 5



## Результаты фаз 3 и 4 (см. предыдущие лекции)

---

- **Архитектура системы AirLogger многоуровневая** (уровень сервера БД, уровень сервера приложений и уровень клиента).
- **В качестве базовой архитектуры используем ASP.Net CORE Architecture**
- Сервер баз данных **MS SQL Server** (Материалы для скачивания <https://www.microsoft.com/ru-ru/sql-server/sql-server-downloads>)
- **Архитектура данных распределенной базы данных** (логический уровень) представлена **ORM-моделью**, содержащей 12 взаимосвязанных сущностей
- **Реплицируемой** сущностью является **UseException**
- **Клиенты** в системе двух типов – «толстый» **desktop-клиент** «Комплексный проект студента», являющийся источником данных, и «тонкий» **ASP.Net Web-клиент** «Логгер-консоль», предназначенный для мониторинга исключений и регистрации инцидентов
- На стороне desktop-клиента формируются фрагменты реплицируемой базы данных, для чего desktop-клиент использует **MS SQL Server Express**
- **Базовые платформы: Windows 10** (допустима Windows 7), **IDE – MS Visual Studio 2019 .NET Core 3**

## 5. Проектирование приложений доступа к данным

---

### На данной фазе:

- В зависимости от архитектуры системы выбирают и обосновывают IDE,
- разрабатывают ограничения на доступ к данным (на уровне клиентов),
- определяют эшелоны защиты,
- разрабатывают приложения доступа к данным,
- выбирают средства защиты информации СЗИ

На этапе формирования архитектуры системы было получено, что система AirLogger будет включать в свой состав два типа клиентов «толстый» и «тонкий»

Поскольку система AirLogger в основе архитектуры базируется на **ORM-подходах**, то приложения **доступа к данным строятся на базовых платформах: Windows 10** (допустима Windows 7), **IDE – MS Visual Studio 2019 .NET Core 3**

## 5. Проектирование приложений доступа к данным

---

Поскольку **источниками данных**, которые будут храниться в распределенной базе данных, **являются «толстые» desktop-клиенты системы AirLogger**, то **сначала спроектируем** именно **desktop-клиент**.

Напомним, что в системе AirLogger функционально desktop-клиент представляет собой **Windows Form проект**, объединяющий лабораторные работы студентов по дисциплинам, связанным с информационной безопасностью и **фиксирующий исключения** при его эксплуатации.

Следуя концепции визуального проектирования, первоначально **проектируется интерфейс приложения** и согласуется с заказчиком.

Проектируем приложения не для себя (разработчика), а для пользователя (заказчика) !!!

## 5. Проектирование приложений доступа к данным

---

Проектируем приложения не для себя, а для пользователя (заказчика), поэтому перечислим несколько (не все) правила разработки визуальных интерфейсов:

1. Интерфейс должен быть **интуитивно понятным**
2. Если используются элементы управления, размещенные на форме, то они **дублируются пунктами главного меню**
3. Формы **не перегружают** информацией и элементами управления
4. Чем **меньше пользователю** придется **вводить информацию**, тем лучше
5. Отдельно взятая форма должна быть **настолько простой, насколько это возможно** и объединять логически выделенный функционал
6. **Доступность** тех или иных элементов на форме **должна определяться текущими** в данный момент **действиями пользователя**
7. Для полей ввода предусматривают **значения «по умолчанию»**
8. **Дизайн форм** и размещение элементов на форме определяется корпоративными стандартами заказчика, либо (если таковых нет) **общепринятыми шаблонами**

## 5. Проектирование приложений доступа к данным

---

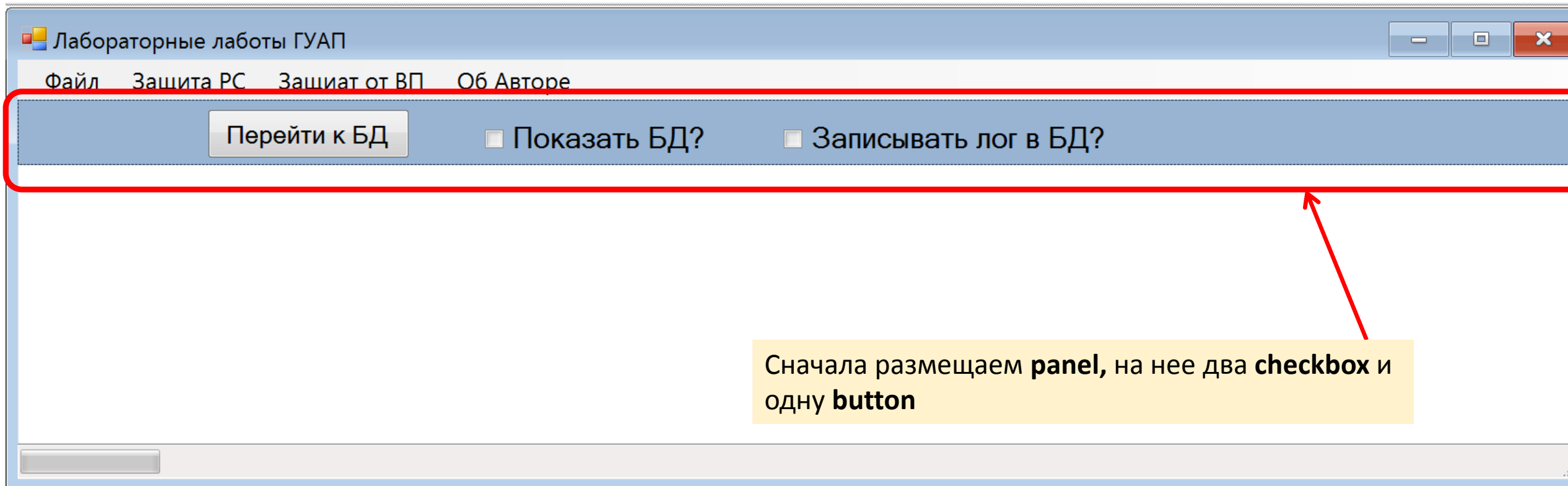
### Также отметим:

1. До настоящего момента исключения в проекте записывались в **лог-журнал, представляющий из себя текст**, отображаемый на главной форме и записываемый, с целью хранения в файл.
2. Но такое локальное (файловое) хранилище **не позволяет** организовать **эффективную репликацию** распределенных данных.
3. В системе AirLogger должны **быть «собраны» исключения всех студенческих проектов** => локальные хранилища нужно реплицировать.
4. Для этого нужна **распределенная база данных**, архитектуру которой мы уже разработали на фазе 4.
5. Разработку функционала **по взаимодействию desktop-клиента с базой данных** будем производить, используя подход **«Code First»**, а ASP.Net Core WEB API (**тонкого**) клиента, используя подход **«Model First»**

## 5. Проектирование приложений доступа к данным

Интерфейс главной формы мы согласовали на занятиях.

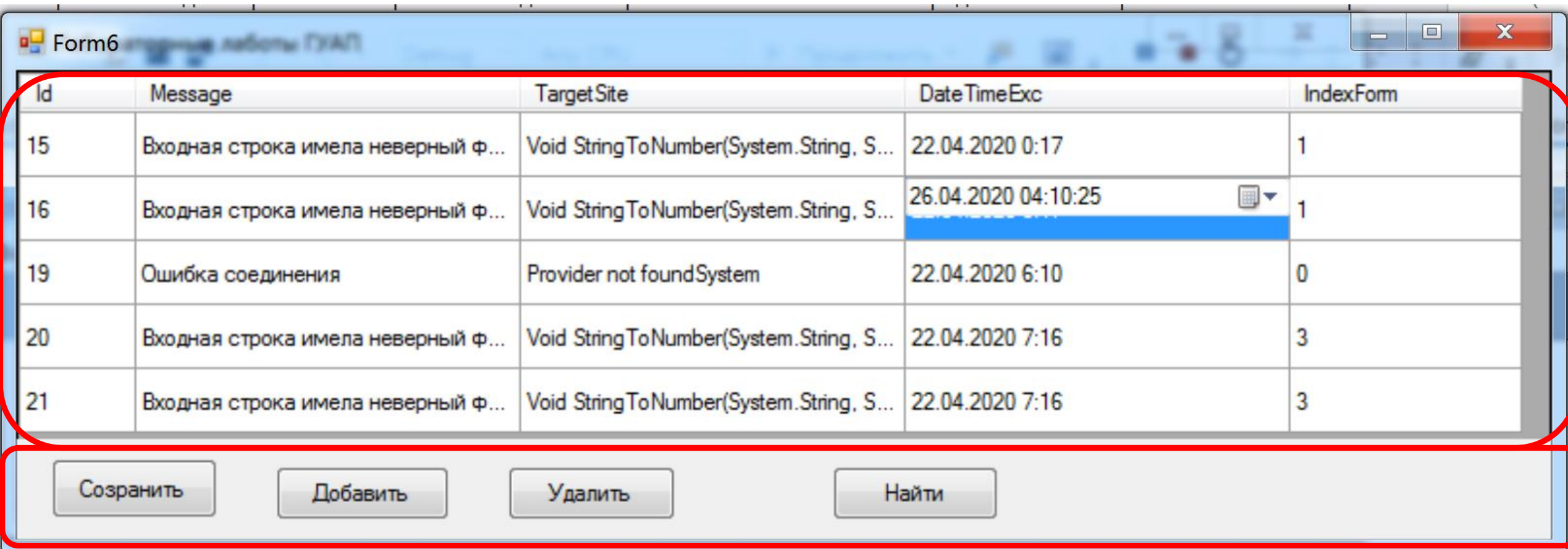
Теперь в него необходимо **добавить элементы управления**, позволяющие обеспечить **взаимодействие с базой данных**



Кнопка «Перейти к БД» должна появляться, когда активирован checkbox «Показать БД?», и становиться невидимой (или недоступной) в противном случае

## 5. Проектирование приложений доступа к данным

При нажатии на кнопку «Перейти к БД», должна открываться **новая форма**, позволяющая **выполнять основные операции с базой данных**



The screenshot shows a Windows application window titled "Form6" with a standard Windows title bar (minimize, maximize, close buttons). The main area contains a table with 5 columns: "Id", "Message", "Target Site", "Date Time Exc", and "Index Form". The table has 6 rows of data. The row with Id 16 is highlighted in blue. Below the table is a panel with four buttons: "Созрнить", "Добавить", "Удалить", and "Найти". A red rectangular border highlights the table and the button panel.

Id	Message	Target Site	Date Time Exc	Index Form
15	Входная строка имела неверный ф...	Void StringToNumber(System.String, S...	22.04.2020 0:17	1
16	Входная строка имела неверный ф...	Void StringToNumber(System.String, S...	26.04.2020 04:10:25	1
19	Ошибка соединения	Provider not foundSystem	22.04.2020 6:10	0
20	Входная строка имела неверный ф...	Void StringToNumber(System.String, S...	22.04.2020 7:16	3
21	Входная строка имела неверный ф...	Void StringToNumber(System.String, S...	22.04.2020 7:16	3

Buttons: Созрнить, Добавить, Удалить, Найти



## 5. Проектирование приложений доступа к данным

**Добавление** данных рекомендуется выполнять на **отдельной форме**

The screenshot displays a Windows application titled 'Лабораторные работы ГУАП'. The main window has a menu bar with 'Файл', 'Защита PC', 'Защита от ВП', and 'Об Авторе'. Below the menu is a toolbar with a 'Перейти к БД' button and two checkboxes: 'Показать БД?' and 'Записывать лог в БД?'. The main area contains a table with the following data:

Id	Message	TargetSite	Date TimeExc	IndexForm
15	Входная строка имела неверный ф...	Void StringToNumber(System.String, S...	22.04.2020 0:17	1
16	Входная строка имела неверный ф...	Void StringToNumber(System.String, S...	26.04.2020 04:10:25	1
19	Ошибка соединения			0
20	Входная строка имела неверный ф...			3
21	Входная строка имела неверный ф...			3

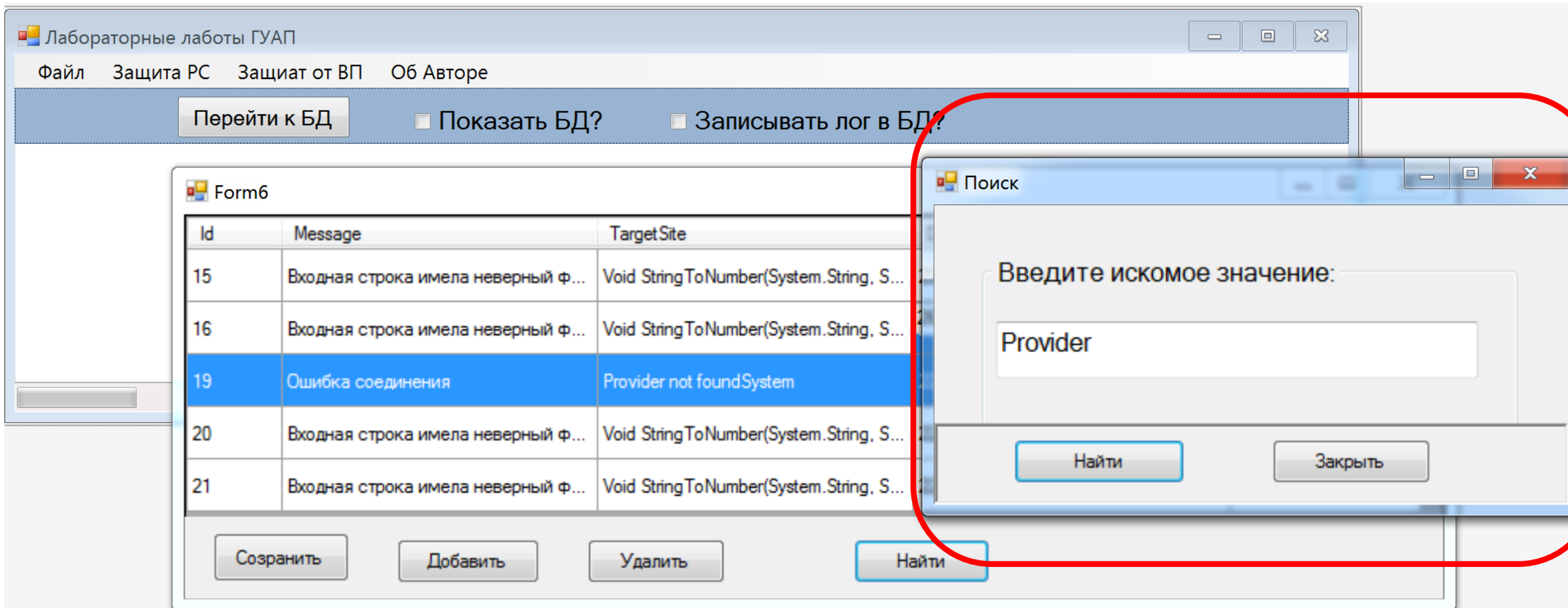
Below the table are two buttons: 'Созрнить' and 'Добавить'. A red circle highlights a dialog box titled 'Добавление записи' (Add record) which is open over the table. The dialog contains the following fields:

- ID\_UserExc: 22
- MessageExc: (empty)
- TargetSiteExc: (empty)
- DataTimeExc: 26.04.2020 04:22:14
- IndexForm: 0

At the bottom of the dialog are two buttons: 'Добавить' and 'Закрыть'.

## 5. Проектирование приложений доступа к данным

Также **поиск** данных рекомендуется выполнять на **отдельной форме**



## 5. Проектирование приложений доступа к данным

---

### Шаг 1. Добавление пакета EF Core

Теперь, когда понятно, как выглядит визуально интерфейс, переходим непосредственно к реализации подхода «**Code First**».

В надежде на то, что все ознакомились с этим материалом

<https://metanit.com/sharp/entityframeworkcore/1.1.php>

и имеют представление, что такое **Entity Framework Core**, начнем разрабатывать наше приложение доступа к данным.

Вначале нам надо добавить пакет EF Core, чтобы воспользоваться его функционалом. Для этого перейдем в проекте к пакетному менеджеру NuGet.

Здесь мы ищем пакет для конкретной СУБД (MS SQL Server) - пакет **Microsoft.EntityFrameworkCore.SqlServer**.

Также, если нам еще надо создать базу данных, мы должны добавить через NuGet и второй пакет **Microsoft.EntityFrameworkCore.Tools**

Browse

Installed

Updates

Search (Ctrl+L)

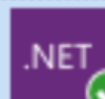


Include prerelease

Package source: nuget.org



## NuGet Package Manager: HelloApp

**Microsoft.EntityFrameworkCore.SqlServer** by Micro v3.0.0

Microsoft SQL Server database provider for Entity Framework Core.

**Microsoft.EntityFrameworkCore.Tools** by Microsoft v3.0.0

Entity Framework Core Tools for the NuGet Package Manager Console in Visual Studio.



## Microsoft.EntityFrameworkCore.SqlServer nuget.org

Installed: 3.0.0

Uninstall

Version: 3.0.0

Update



Options

## Description

Microsoft SQL Server database provider for Entity Framework Core.

Version: 3.0.0

Author(s): Microsoft

License: Apache-2.0

Date published: Monday, September 23, 2019  
(9/23/2019)

Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any licenses to, third-party packages.



Do not show this again

## Шаг 2. Формирование модели данных

---

Итак, необходимые пакеты добавлены. Теперь мы можем их использовать.

Нам надо **определить модель**, которая будет описывать данные, **порождаемые клиентским приложением**.

Размещать элементы модели данных будем **в отдельном файле нашего проекта**, поэтому в «Обозревателе решений» выделяем наш проект (в моем случае **Name\_my**) -> **Добавить** -> **Создать элемент** -> **Файл с текстом программы** -> в качестве имени введем **MyDBNames**

Form7.cs [Конструктор]MyDBNames.cs

ContextUserExceptions

{ get; set; }

set; }

: DbContext

Us

Создать элемент...Ctrl+Shift+A

Существующий элемент...Shift+Alt+A

Создать папку

Клиент REST API...

Ссылка...

Веб-ссылка...

Ссылка на службу...

Подключенная служба

Анализатор...

Форма (Windows Forms)...

User Control (Windows Forms)...

Компонент...

Класс...Shift+Alt+C

Обозреватель решений

Обозреватель решений — поиск (Ctrl)

Решение "Name\_my" (проекты: 1 из

C# Name\_mv

Собрать

Пересобрать

Очистить

Анализ и очистка кода

Опубликовать...

Открыть элемент как корень обозревателя

Новое представление: Обозреватель решений

Добавить

Управление пакетами NuGet...

Назначить автозагружаемым проектом

Отладка

Инициализировать интерактивное окно с проектом

Система управления версиями

ВырезатьCtrl+X

ВставитьCtrl+V

УдалитьDel

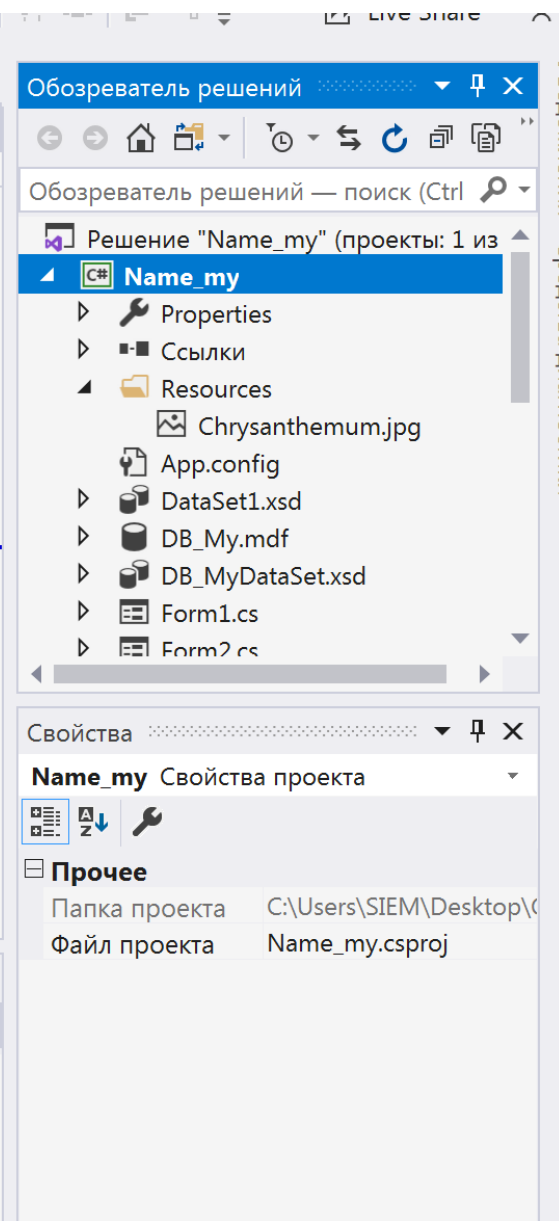
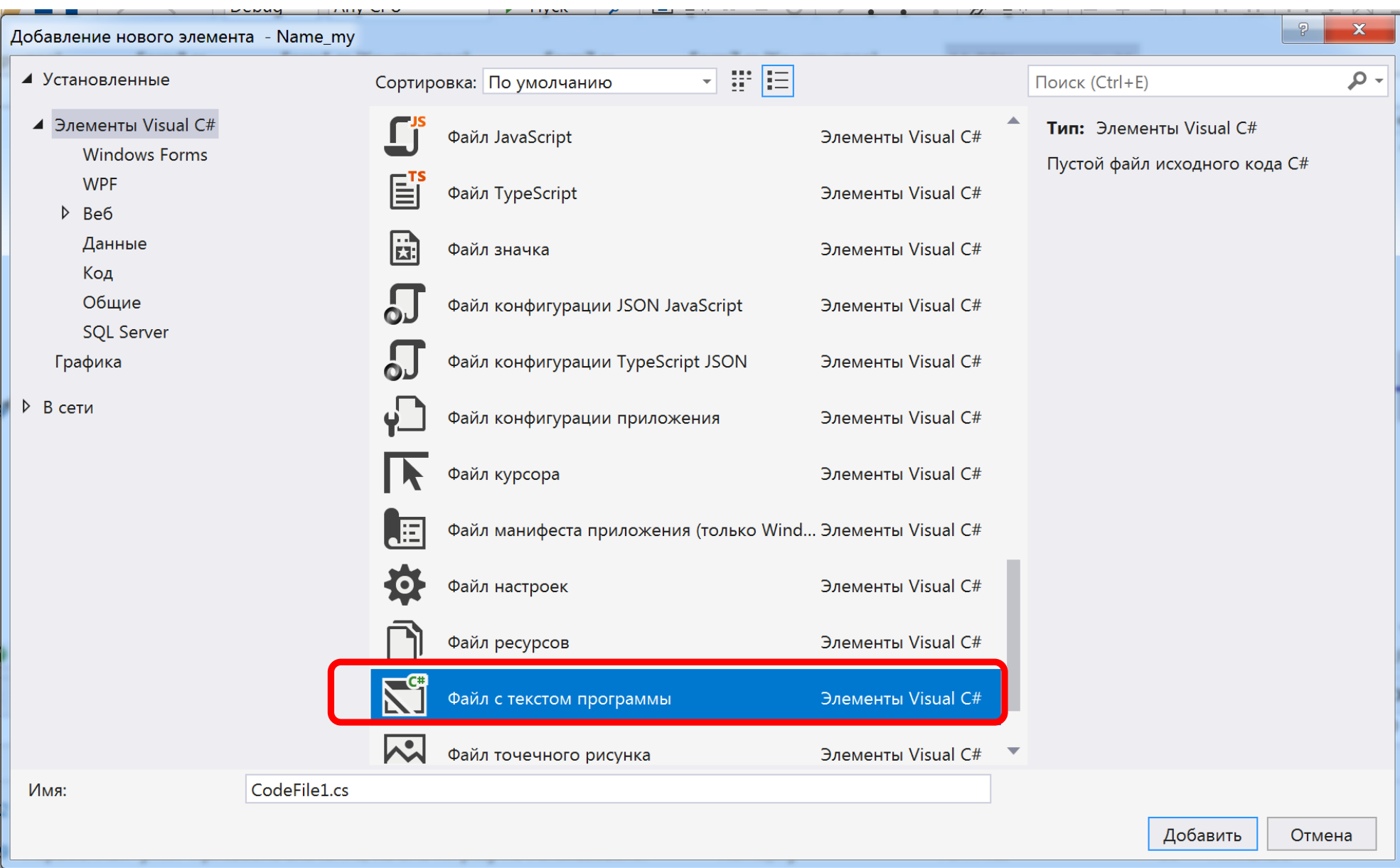
Переименовать

Выгрузить проект

Загрузить зависимости проекта

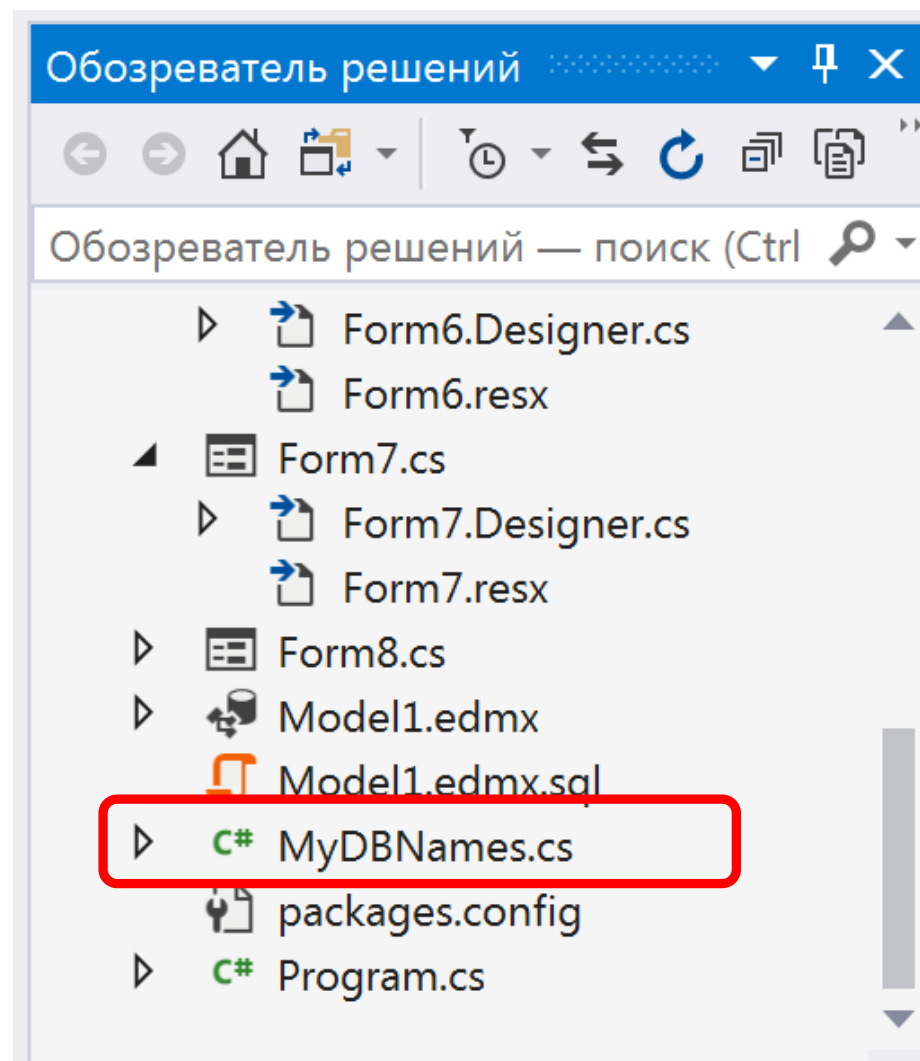
Открыть папку в проводнике

СвойстваAlt+ВВОД



## Шаг 2. Формирование модели данных

В результате к проекту добавлен **пустой файл MyDBNames.cs**, перейти к нему всегда можно в «Обозревателе решений»





## Шаг 2. Формирование модели данных

---

В файле **MyDBNames.cs** потребуются следующие пространства имен

```
using System;  
using System.ComponentModel.DataAnnotations;  
using Microsoft.EntityFrameworkCore;
```

Также в файле **MyDBNames.cs** создадим **свое пространство имен для работы с базой данных**

```
namespace MyDBNames.Scheme
```

И в этом пространстве создадим два класса:

```
public class UserException
```

Это обычный класс, который содержит несколько свойств. Каждое свойство будет сопоставляться с отдельным столбцом в таблице из бд.

```
public class ApplicationContext : DbContext
```

Это специальный класс для взаимодействия с базой данных в Entity Framework Core - **контекст данных**

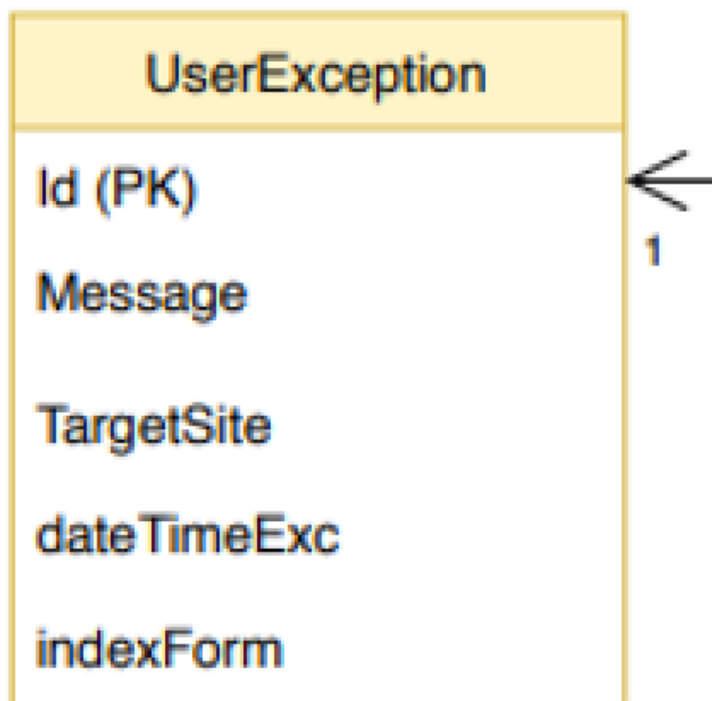
## Шаг 2. Формирование модели данных

Обратим внимание, что свойства класса **UserException** определяются структурой сущности **UserException**, полученной нами ранее на фазе 4

```
public class UserException
{
    public int Id { get; set; }
    public string Message { get; set; }
    public string TargetSite { get; set; }

    [DisplayFormat(DataFormatString =
        "{0:dd.MM.yyyy hh:mm:ss}", ApplyFormatInEditMode =
        true)]
    public DateTime DateTimeExc { get; set; }

    public int IndexForm { get; set; }
}
```



## Шаг 2. Формирование модели данных

В любом приложении, работающем с БД через Entity Framework, нужен **контекст данных** (класс производный от DbContext).

В данном случае таким контекстом является класс ApplicationContext.

```
public class ApplicationContext : DbContext
```

```
{
```

```
    public DbSet<UserException> UserExceptions { get; set; }
```

свойство UserExceptions, которое будет хранить набор объектов UserException

```
public ApplicationContext()
```

```
{
```

```
    Database.EnsureCreated();
```

```
}
```

метод Database.EnsureCreated() при создании контекста автоматически проверит наличие базы данных и, если она отсутствует, создаст ее.

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
```

```
{
```

```
optionsBuilder.UseSqlServer("Server=(localdb)\\mssqllocaldb;Database=TestDB_v2;Trusted_Connection=True;");
```

```
}
```

```
}
```

настройка строки подключения для соединения с MS SQL Server

## Шаг 2. Формирование модели данных

---

Основу функциональности Entity Framework Core для работы с MS SQL Server составляют классы, которые располагаются в пространстве имен **Microsoft.EntityFrameworkCore**. Среди всего набора классов этого пространства имен следует выделить следующие:

- **DbContext**: определяет контекст данных, используемый для взаимодействия с базой данных
- **DbSet/DbSet<TEntity>**: представляет набор объектов, которые хранятся в базе данных
- **DbContextOptionsBuilder**: устанавливает параметры подключения

В данном случае мы в качестве сервера будем использовать встроенный localdb ("Server=(localdb)\\mssqllocaldb"), а файл базы данных будет называться TestDB\_v2 ("Database=TestDB\_v2").

По умолчанию у нас нет базы данных. Поэтому в конструкторе класса контекста определен вызов метода Database.EnsureCreated(), который при создании контекста автоматически проверит наличие базы данных и, если она отсутствует, создаст ее.

## Шаг 3. Использование контекста данных

---

**Лог-журнал** у нас комплектовался **на главной форме проекта**, поэтому переходим на главную (первую) форму проекта и **подключаем наше пространство имен**, чтобы записывать исключения в базу данных

```
using MyDBNames.Scheme; //подключаем наши сущности и контекст БД
```

А также определим булевскую переменную needDB, чтобы реагировать на необходимость записывать исключения в базу данных

Ссылка: 11

```
public partial class Form1 : Form
{
    ToolStripLabel dateLabel;
    ToolStripLabel timeLabel;
    ToolStripLabel infoLabel;
    Timer timer;
    public bool needDB;
```

## Шаг 3. Использование контекста данных

---

checkBox1 активирует «Показать БД?» и делает видимой кнопку button1 «перейти к БД»

```
private void checkBox1_CheckedChanged(object sender, EventArgs e)
{
    if (checkBox1.Checked)
    {
        needDB = true;
        button1.Visible = true;
    }
    else
    {
        needDB = false;
        button1.Visible = false;
    }
}
```

## Шаг 3. Использование контекста данных

---

button1 «перейти к БД» открывает немодально новую форму Form6, где будет отображаться база данных

ссылка: 1

```
private void button1_Click(object sender, EventArgs e)
{
    {
        Form6 newForm = new Form6(this);
        newForm.Show();
    }
}
```

## Шаг 3. Использование контекста данных

---

checkbox cbRecordDB активирует «Записывать лог в БД?» и делает видимой кнопку butto1 «перейти к БД»

```
private void cbRecordDB_CheckedChanged(object sender, EventArgs e)
{
    if (cbRecordDB.Checked)
    {
        needDB = true;
        button1.Visible = true;
    }
    else
    {
        needDB = false;
        button1.Visible = false;
    }
}
```



## Шаг 3. Использование контекста данных

**Изменяем метод `addErrorMesssge`**, чтобы при активации checkbox записывать исключения в БД

```
public void addErrorMessage (UserException userExc)//(String msg)
{
    textBox1.Text += ("\r\n" + userExc.DateTimeExc + " form" + userExc.IndexForm);
    textBox1.Text += ("\r\n" + "Исключение:" + userExc.Message);
    textBox1.Text += ("\r\n" + "Метод: " + userExc.TargetSite);
    textBox1.Text += ("\r\n" + "_____");

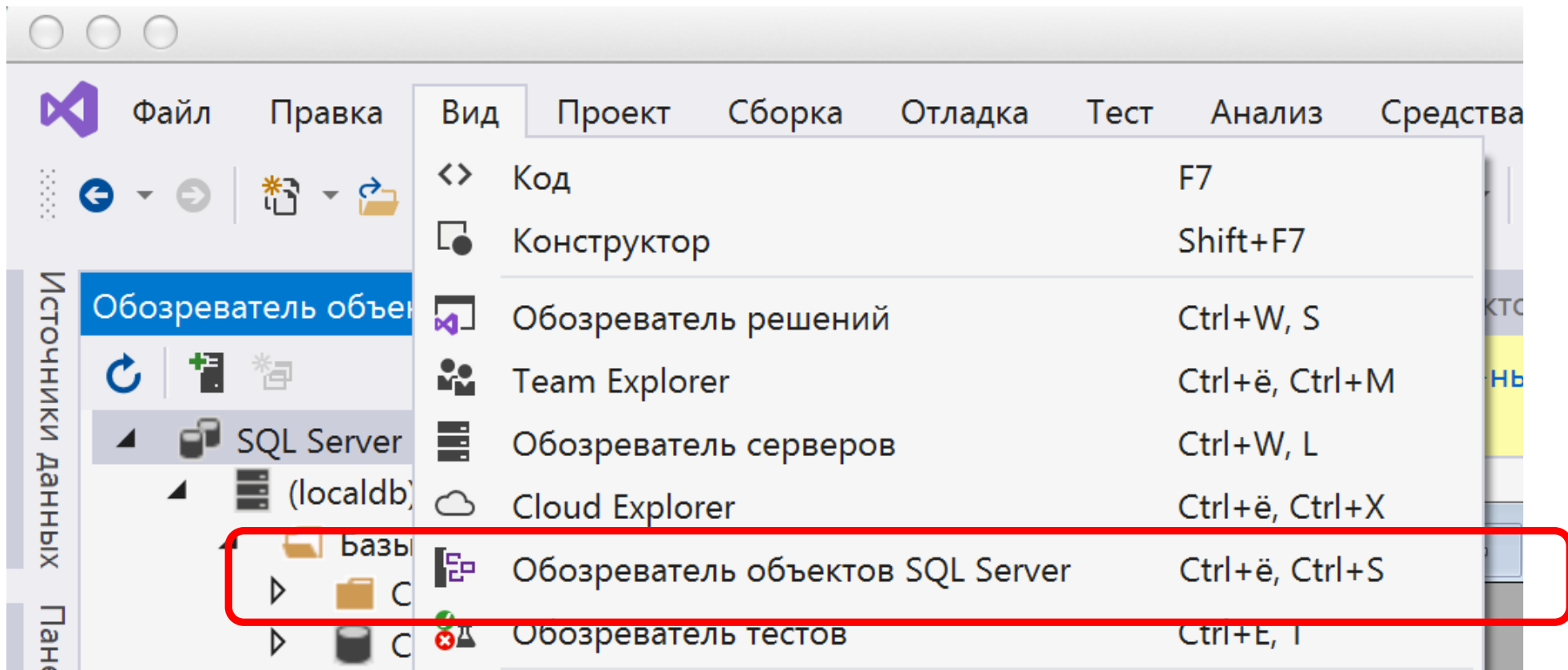
    if (needDB)
    {
        using (MyDBNames.Scheme.ApplicationContext db = new MyDBNames.Scheme.ApplicationContext())
        {
            // создаем объект userException
            UserException userException = new UserException { Message = userExc.Message,
                TargetSite = userExc.TargetSite, DateTimeExc = DateTime.Now, IndexForm = userExc.IndexForm };
            // добавляем его в бд
            db.UserExceptions.Add(userException);
            db.SaveChanges();
        }
    }
}
```

С главной формой закончили 😊

## Шаг 3. Использование контекста данных

Сейчас мы можем, сохранив проект, запустить его на выполнение и, если установить галочку в checkbox «Записывать в БД?», все исключения, будут не только отображаться на главной форме. Но и записываться в базу данных.

Наш desktop-клиент еще не способен отобразить эту базу данных. Но ее уже можно посмотреть в «**Обозревателе объектов SQL-server**»



# Шаг 3. Использование контекста данных

ФайлПравкаВидПроектСборкаОтладкаТестАнализСредстваРасширени

DebugAny CPUПуск

Обозреватель объектов SQL Server

SQL Server

- (localdb)\MSSQLLocalDB (SQL Server 13.0.4001 - SIEM-PC\SIEM)
  - Базы данных
    - Системные базы данных
    - C:\USERS\SIEM\DESKTOP\C#\ПРОЕКТЫ\NAME\_MY\BIN\C
    - C:\USERS\SIEM\DESKTOP\C#\ПРОЕКТЫ\NAME\_MY\DB\_M
    - C:\USERS\SIEM\DOCUMENTS\DB\_MY.MDF
    - helloappdb
    - TestDB
    - TestDB\_v2
  - Таблицы
    - Системные таблицы
    - Внешние таблицы
    - dbo.UserExceptions
      - Столбцы
        - Id (PK, int, not null)
        - Message (nvarchar(max), null)
        - TargetSite (nvarchar(max), null)
        - DateTimeExc (datetime2(7), not null)
        - IndexForm (int, not null)
    - Ключи
    - Ограничения
    - Триггеры
    - Индексы
    - Статистика

Источники данныхПанель элементовОбозреватель объектов SQL ServerОбозреват

ФайлПравкаВидПроектСборкаОтладкаSQLТестАнализСредства

DebugAny CPUПуск

dbo.UserExceptions [Данные]DB\_MyDataSet.xsdMyDBNames.csForm1.cs

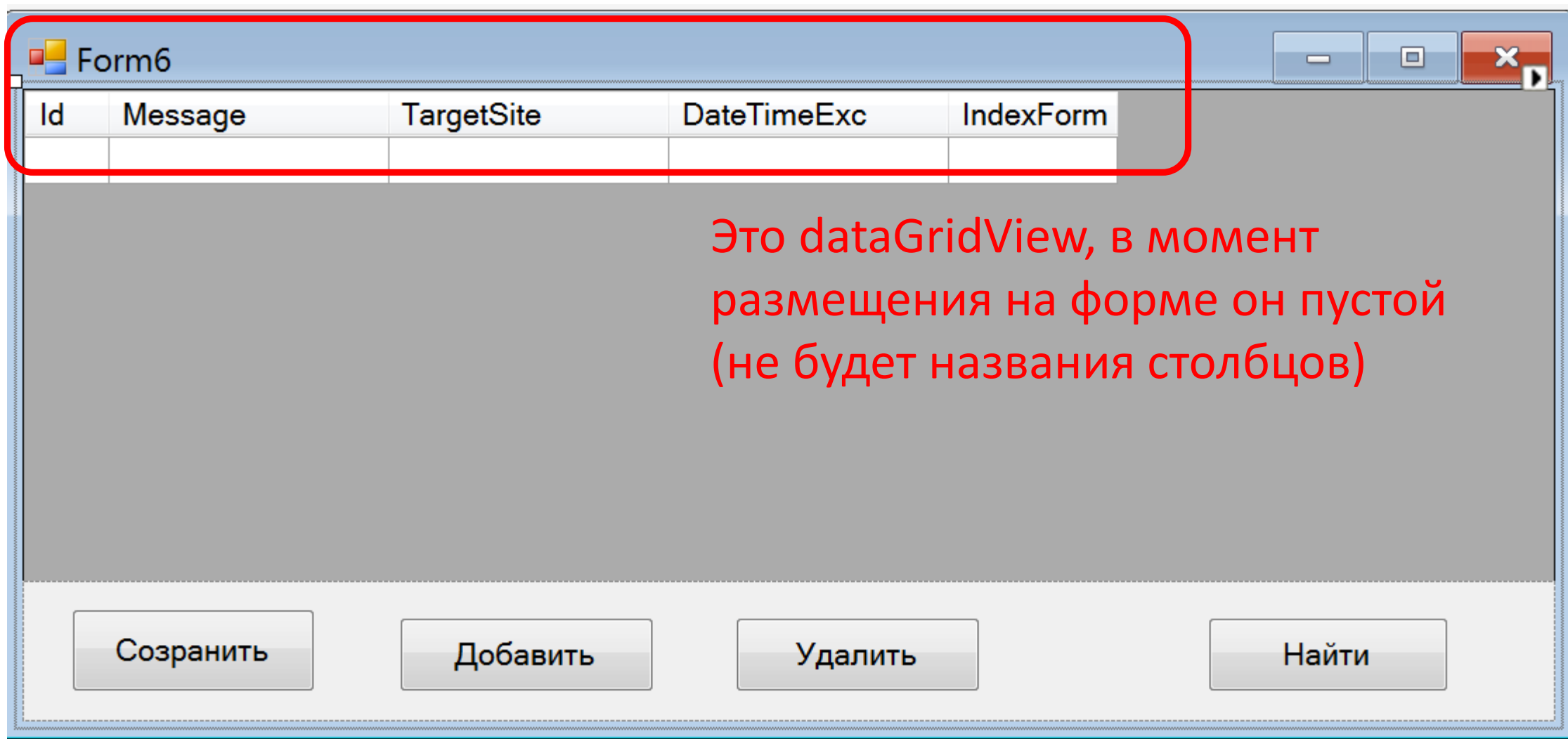
Максимальное количество строк: 1000

	Id	Message	TargetSite	DateTimeExc	IndexForm
▶	15	Входная строк...	Void StringTo...	22.04.2020 0:17...	1
	16	Входная строк...	Void StringTo...	22.04.2020 0:17...	1
	19	dfgdfgdgdg		22.04.2020 6:10...	0
	20	Входная строк...	Void StringTo...	22.04.2020 7:16...	3
	21	Входная строк...	Void StringTo...	22.04.2020 7:16...	3
*	NULL	NULL	NULL	NULL	NULL

Источники данныхПанель элеме

## Шаг 4. Операции с набором данных

Переходим на форму, **предназначенную для работы с БД**, и размещаем на нее необходимые компоненты (отступим от концепции Code-First и воспользуемся функционалом IDE MSVS 2019 Core)



Form6

Id	Message	TargetSite	DateTimeExc	IndexForm
----	---------	------------	-------------	-----------

Это dataGridView, в момент размещения на форме он пустой (не будет названия столбцов)

Созрнить    Добавить    Удалить    Найти

## Шаг 4. Операции с набором данных

Открываем список «**Выберите источник данных**» и нажимаем на ссылку «**Добавить источник данных проекта**»:

The screenshot shows a Windows Forms application window titled "Form6". Inside the window is a **DataGridView** control with the following columns: **Id**, **Message**, **TargetSite**, **DateTimeExc**, and **IndexForm**. The grid is currently empty. Below the grid are four buttons: **Созранить** (Save), **Добавить** (Add), **Удалить** (Delete), and **Найти** (Find). A right-click context menu is open over the **DataGridView**. The menu title is **DataGridView Задачи**. The first item is "Выберите источник данных:" (Select data source:), which is currently set to **dataSet11BindingSource**. Below this are several options: **Правка столбцов** (Edit columns), **Добавить столбец** (Add column), and a list of checkboxes for including data, primary key, unique, and visible columns. At the bottom of the menu, the option **Добавить источник данных проекта...** (Add project data source...) is highlighted with a red rectangle. A tooltip is visible for this option, stating: "В данный момент данные привязаны к 'dataSet11BindingSource'." (At the moment, the data is bound to 'dataSet11BindingSource').

Id	Message	TargetSite	DateTimeExc	IndexForm
----	---------	------------	-------------	-----------

Buttons: Созранить, Добавить, Удалить, Найти

Context Menu: DataGridView Задачи

Выберите источник данных: dataSet11BindingSource

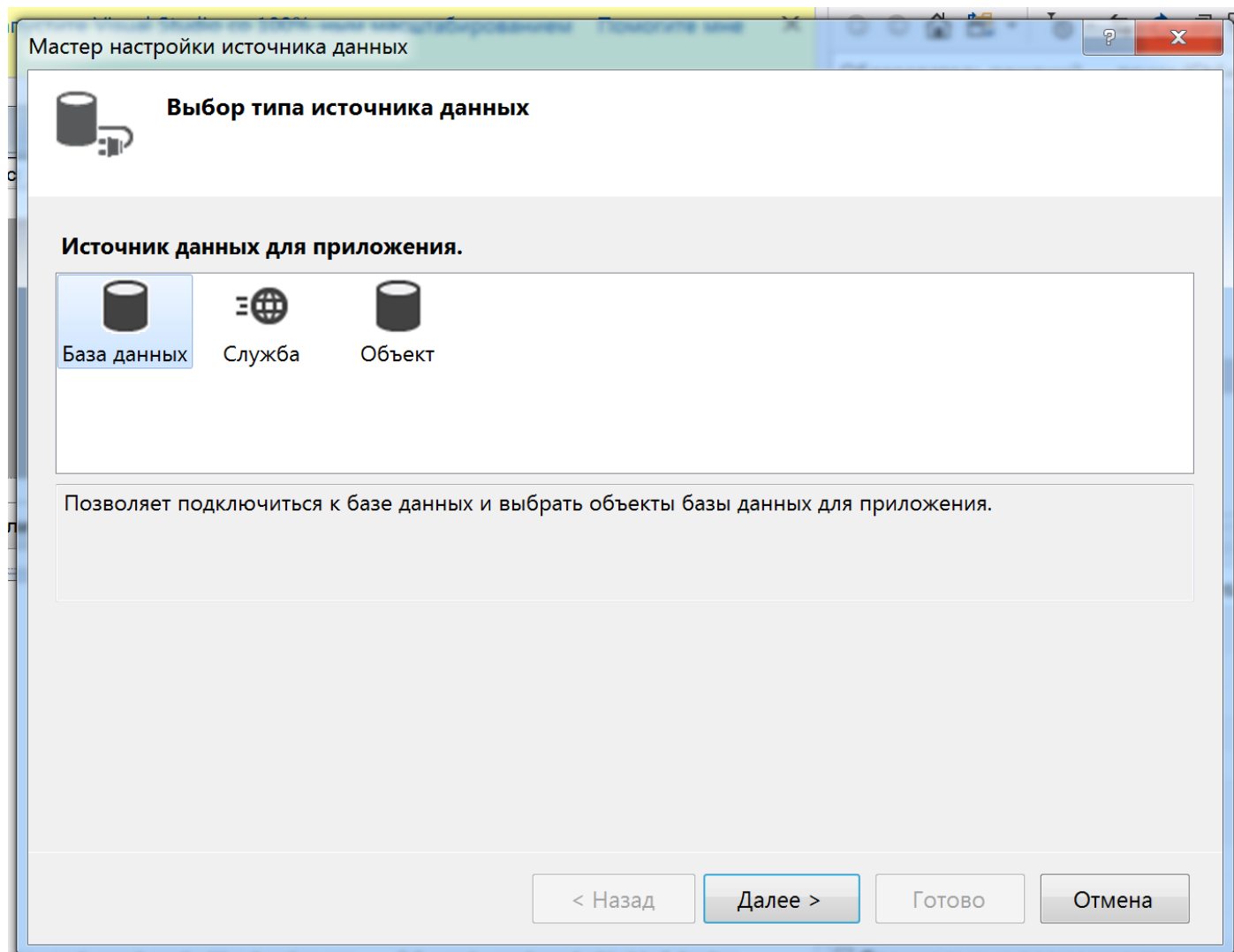
- Отсутствует
- dataSet11BindingSource
- userExceptionsSetBindingSource
- dBMyDataSetBindingSource
- Другие источники данных

Добавить источник данных проекта...

В данный момент данные привязаны к 'dataSet11BindingSource'.

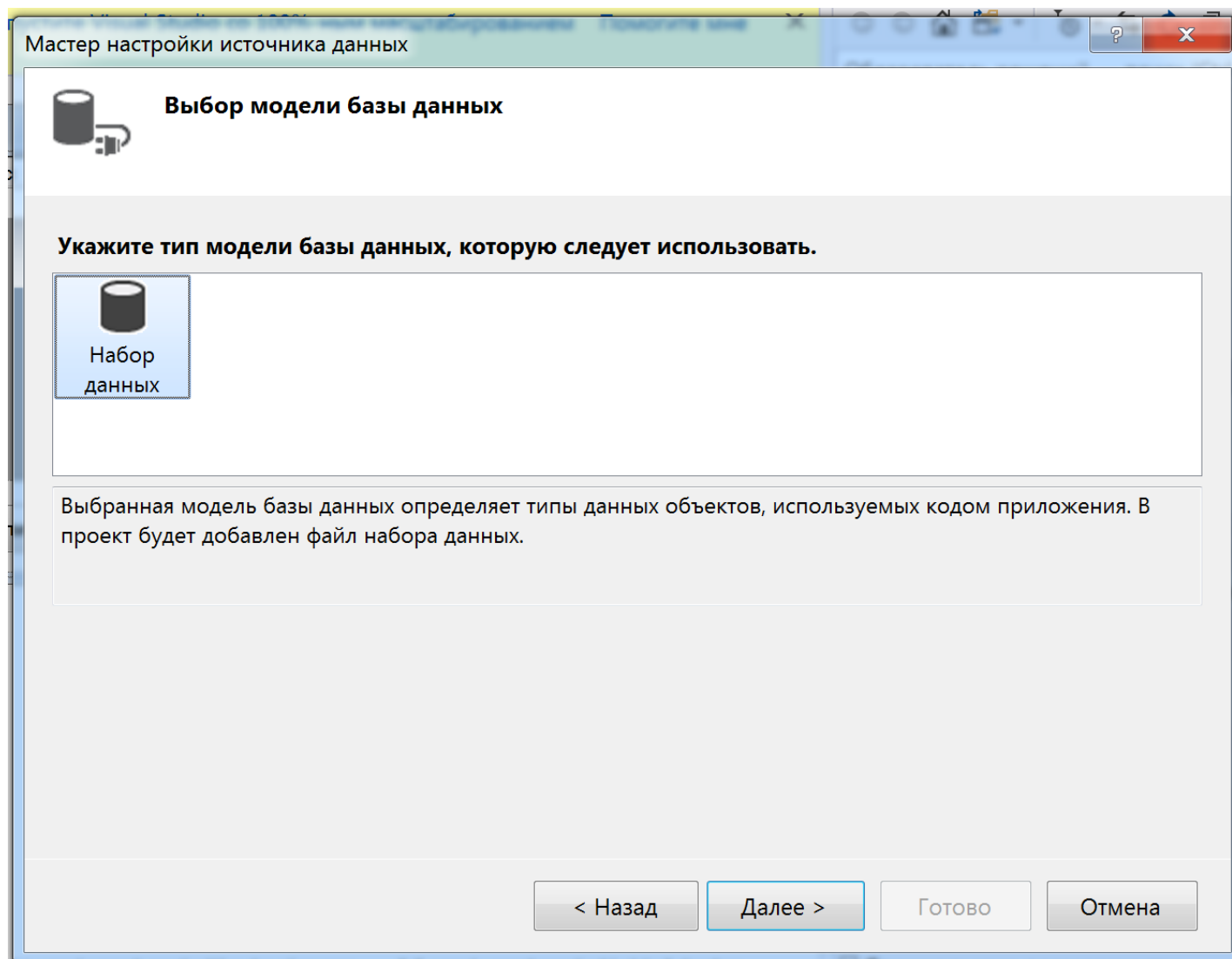
## Шаг 4. Операции с набором данных

Откроется «**Мастер настройки источника данных**». В качестве источника выбираем «**База данных**» и идем далее:



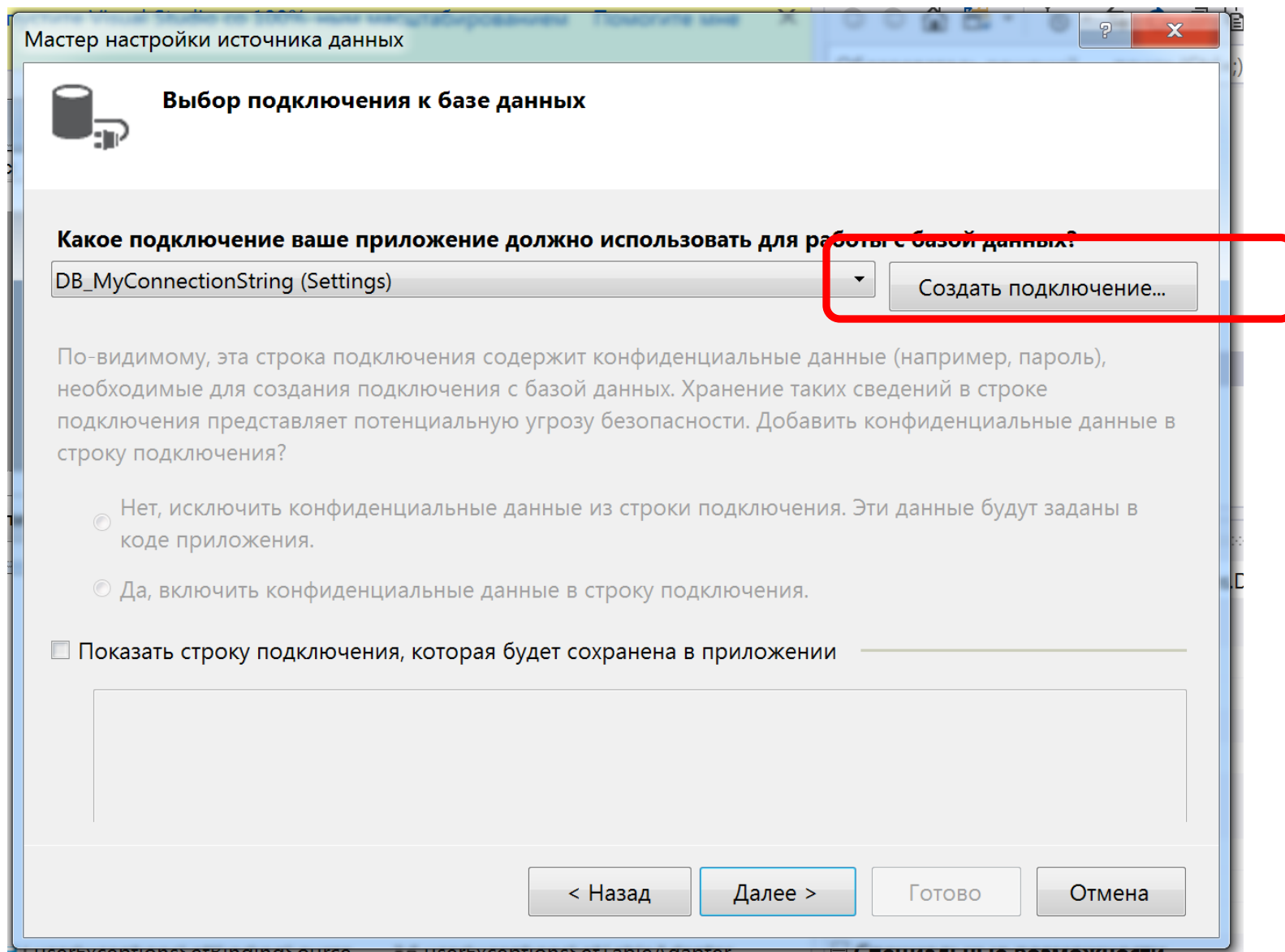
## Шаг 4. Операции с набором данных

В следующем окне идем далее:



## Шаг 4. Операции с набором данных

В окне «Выбор подключения к базе данных» нажимаем на «Создать подключение»:





## Шаг 4. Операции с набором данных

Добавить подключение

Введите данные для подключения к выбранному источнику данных или нажмите кнопку "Изменить", чтобы выбрать другой источник данных и (или) поставщик.

Источник данных:

Файл базы данных Microsoft SQL Server (SqlClient) Изменить...

Имя файла базы данных (новой или существующей):

Обзор...

Вход на сервер

☒ Использовать проверку подлинности Windows

☐ Использовать аутентификацию SQL Server

Имя пользователя:

Пароль:

☐ Сохранить пароль

Дополнительно...

Проверить подключение ОК Отмена

Сменить источник данных



Источник данных:

Microsoft SQL Server

Oracle Database

Источник данных Microsoft ODBC

Файл базы данных Microsoft Access

Файл базы данных Microsoft SQL Server

<другое>

Описание

Используйте этот вариант для подключения к Microsoft SQL Server 2005 или более поздней версии или к Microsoft SQL Azure с помощью поставщика данных .NET Framework для SQL Server.

Поставщик данных:

Поставщик данных .NET Framework для ▼

☐ Всегда использовать этот вариант

OK

Отмена

## Добавить подключение

Введите данные для подключения к выбранному источнику данных или нажмите кнопку "Изменить", чтобы выбрать другой источник данных и (или) поставщик.

Источник данных:

Microsoft SQL Server (SqlClient)

Изменить...

Имя сервера:

(localdb)\mssqllocaldb

Обновить

Вход на сервер

Проверка подлинности:

Проверка подлинности Windows

Имя пользователя:

Пароль:

☐ Сохранить пароль

Подключение к базе данных

☒ Выберите или введите имя базы данных:

C:\USERS\SIEM\DESKTOP\C#\_ПРОЕКТЫ\NAME\_MY\BIN\DEBUG\DB\_MY.MDF  
☐ C:\USERS\SIEM\DESKTOP\C#\_ПРОЕКТЫ\NAME\_MY\DB\_MY.MDF  
C:\USERS\SIEM\DOCUMENTS\DB\_MY.MDF  
helloappdb  
master  
model  
msdb  
tempdb  
TestDB  
TestDB\_v2

Дополнительно...

Проверить подключение

OK

Отмена

Источник данных:

Microsoft SQL Server (SqlClient)

Имя сервера:

(localdb)\mssqllocaldb

Вход на сервер

Вводим имя сервера (сейчас мы работаем с **(localdb)\mssqllocal.db**) и выбираем нашу базу данных **TestDB v2**

C:\USERS\SIEM\DESKTOP\C#\_ПРОЕКТЫ\NAME\_MY\BIN\DEI  
☐ C:\USERS\SIEM\DESKTOP\C#\_ПРОЕКТЫ\NAME\_MY\DB\_MY.  
C:\USERS\SIEM\DOCUMENTS\DB\_MY.MDF  
helloappdb  
master  
model  
msdb  
tempdb  
TestDB  
TestDB\_v2



## Выбор подключения к базе данных

Какое подключение ваше приложение должно использовать для работы с базой данных?

TestDB\_v2ConnectionString (Settings)

Создать подключение...

По-видимому, эта строка подключения содержит конфиденциальные данные (например, пароль), необходимые для создания подключения с базой данных. Хранение таких сведений в строке подключения представляет потенциальную угрозу безопасности. Добавить конфиденциальные данные в строку подключения?

- ☐ Нет, исключить конфиденциальные данные из строки подключения. Эти данные будут заданы в коде приложения.
- ☐ Да, включить конфиденциальные данные в строку подключения.

☒ Показать строку подключения, которая будет сохранена в приложении

```
Data Source=(localdb)\MSSQLLocalDB;Initial Catalog=TestDB_v2;Integrated Security=True;Connect  
Timeout=30;Encrypt=False;TrustServerCertificate=False
```

< Назад

Далее >

Готово

Отмена



## Выбор объектов базы данных

Объекты базы данных для набора данных

- ☒ Таблицы
  - ☒ UserExceptions
- ☐ Представления
- ☐ Хранимые процедуры
- ☐ Функции

Имя набора данных (DataSet):

TestDB\_v2DataSet

< Назад

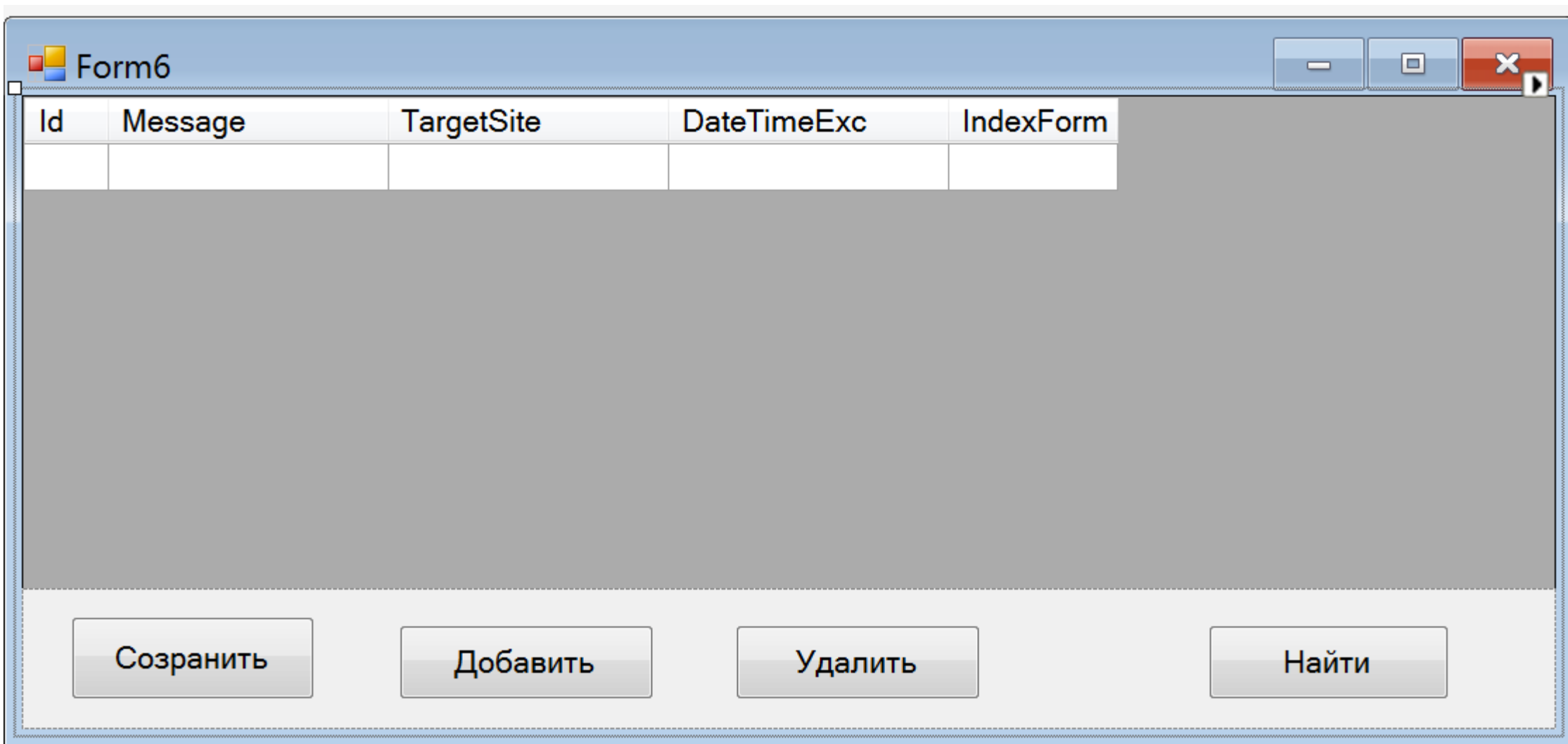
Далее >

Готово

Отмена

## Шаг 4. Операции с набором данных

В итоге видим шаблон своей таблицы с исключениями



The screenshot shows a Windows application window titled "Form6". Inside the window, there is a table with five columns: "Id", "Message", "TargetSite", "DateTimeExc", and "IndexForm". The table has one empty row below the header. Below the table, there are four buttons: "Созранить" (Save), "Добавить" (Add), "Удалить" (Delete), and "Найти" (Find). The buttons are arranged horizontally at the bottom of the window.

Id	Message	TargetSite	DateTimeExc	IndexForm

Созранить      Добавить      Удалить      Найти

## Шаг 4. Операции с набором данных

После запуска проекта и перехода к форме БД, видим содержимое таблицы **UserExceptions**, теперь закрепим обработчики событий за кнопками

The screenshot shows a Windows application with a menu bar and a database table. The menu bar includes 'Файл', 'Защита РС', 'Защит от ВП', and 'Об Авторе'. A red box highlights the 'Перейти к БД' button. Below the menu bar, there are checkboxes for 'Показать БД?' (checked) and 'Записывать лог в БД?' (unchecked). The database table 'Form6' displays the following data:

Id	Message	TargetSite	DateTimeExc	IndexFom
15	Входная строка имела неверный ф...	Void StringToNumber(System.String, S...	22.04.2020 0:17	1
16	Входная строка имела неверный ф...	Void StringToNumber(System.String, S...	22.04.2020 0:17	1
19	dfgdfgdgdg		22.04.2020 6:10	0
20	Входная строка имела неверный ф...	Void StringToNumber(System.String, S...	22.04.2020 7:16	3
21	Входная строка имела неверный ф...	Void StringToNumber(System.String, S...	22.04.2020 7:16	3

At the bottom of the window, there are four buttons: 'Созранить', 'Добавить', 'Удалить', and 'Найти'.

## Шаг 4. Операции с набором данных

---

Кнопка «**Сохранить**»:

```
private void button1_Click(object sender, EventArgs e)
{
    userExceptionsTableAdapter.Update(dataSet1);
}
```

Кнопка «**Добавить**» открывает модально форму **addForm** для добавления записей:

```
private void button2_Click(object sender, EventArgs e)
{
    addForm af = new addForm();
    af.Owner = this;
    af.ShowDialog();
}
```

Кнопка «**Найти**» открывает модально форму **SearchForm** для поиска записей

```
private void button3_Click(object sender, EventArgs e)
{
    SearchForm sf = new SearchForm();
    sf.Owner = this;
    sf.Show();
}
```



## Шаг 4. Операции с набором данных

---

Кнопка «**Удалить**» :

```
private void button4_Click(object sender, EventArgs e)
{
    DialogResult dr = MessageBox.Show("Удалить запись?", "Удаление", MessageBoxButtons.OKCancel,
        MessageBoxIcon.Warning, MessageBoxDefaultButton.Button2);
    if (dr != DialogResult.Cancel)
    {
        int delet = dataGridView1.SelectedCells[0].RowIndex;
        int DelInd = dataSet1.Tables[0].Rows[delet].Field<int>("Id");
        dataGridView1.Rows.RemoveAt(delet);
        dataGridView1.Refresh();

        using (MyDBNames.Scheme.ApplicationContext db = new MyDBNames.Scheme.ApplicationContext())
        {

            MyDBNames.Scheme.UserException userException = db.UserExceptions.Find(DelInd);

            //удаляем объект
            db.UserExceptions.Remove(userException);
            db.SaveChanges();

        }
    }
}
```

## Шаг 4. Операции с набором данных

---

При удалении с помощью клавиши **Delete** сетка **dataGridView1** среагирует :

```
private void dataGridView1_UserDeletingRow(object sender,
DataGridViewRowCancelEventArgs e)
{
    DialogResult dr = MessageBox.Show("Удалить запись?",
"Удаление", MessageBoxButtons.OKCancel, MessageBoxIcon.Warning,
MessageBoxDefaultButton.Button2);
    if (dr == DialogResult.Cancel)
    {
        e.Cancel = true;
    }
}
```

## Шаг 4. Операции с набором данных

---

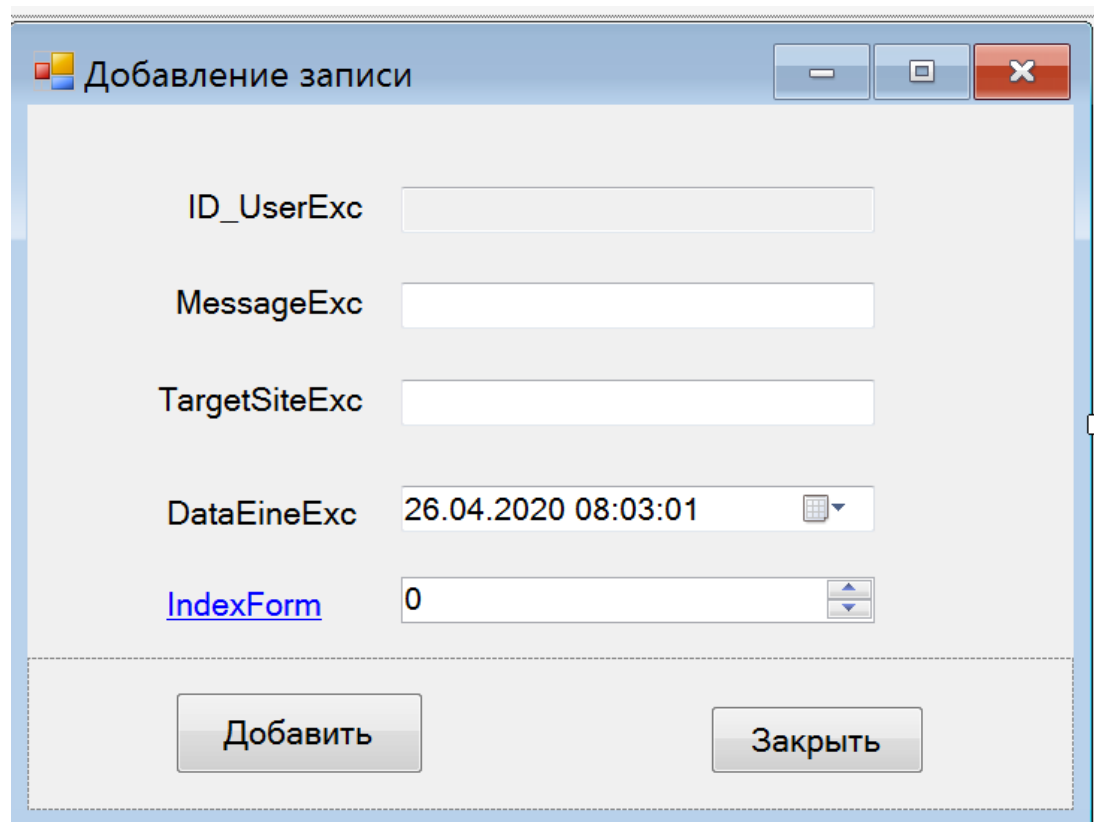
Обработчик события **Load** при загрузке формы с БД:

```
private void Form6_Load(object sender, EventArgs e)
{
    // TODO: данная строка кода позволяет загрузить данные в таблицу
//"dataSet11.UserExceptions". При необходимости она может быть перемещена или удалена.
    this.userExceptionsTableAdapter.Fill(this.dataSet11.UserExceptions);
    // TODO: данная строка кода позволяет загрузить данные в таблицу
//"dB_MyDataSet.UserExceptionsSet".
    // При необходимости она может быть перемещена или удалена.
    //this.userExceptionsSetTableAdapter.Fill(this.dB_MyDataSet.UserExceptionsSet);

    dataGridView1.AllowUserToAddRows = false; //запрещаем пользователю самому
//добавлять строки
}
```

## Шаг 4. Операция добавления данных

Первичный ключ менять нельзя, поэтому будем его только отображать, остальные данные позволим вводить пользователю:



Добавление записи

ID\_UserExc

MessageExc

TargetSiteExc

DataEineExc 26.04.2020 08:03:01

[IndexForm](#) 0

Добавить Заккрыть

## Шаг 4. Операция добавления данных

Кнопка «**Добавить**» на этой форме автоматически генерирует первичный ключ, используя метод `findMaxIdForUserException(main_db.dataSet1.Tables[0])`; реализацию которого рассмотрим далее

ссылка: 1

```
private void button1_Click(object sender, EventArgs e)
{
    Form6 main_db = this.Owner as Form6;

    if (main_db != null)
    {
        //разрешаем пользователю самому добавлять строки
        main_db.dataGridView1.AllowUserToAddRows = true;
        DataRow nRow = main_db.dataSet11.Tables[0].NewRow();
        int maxId = findMaxIdForUserException(main_db.dataSet11.Tables[0]);
        nRow[0] = maxId + 1;
        nRow[1] = textBox_msg.Text;
        nRow[2] = textBox_targetsite.Text;
        nRow[3] = DateTime.Now;
        nRow[3] = dateTimePicker1.Text;
        nRow[4] = numericUpDown1.Value;
```

## Шаг 4. Операция добавления данных

---

Продолжение...

```
main_db.dataSet11.Tables[0].Rows.Add(nRow);
main_db.userExceptionsTableAdapter.Update(main_db.dataSet11.UserExceptions);
main_db.dataSet11.Tables[0].AcceptChanges();
main_db.dataGridView1.Refresh();
textBox_id.Text = (maxId + 2).ToString();
textBox_msg.Text = "";
textBox_targetsite.Text = "";
dateTimePicker1.Text = "";
numericUpDown1.Text = "";
```

```
}
```

```
}
```

## Шаг 4. Операция добавления данных

---

Метод `findMaxIdForUserException` автоматически определяет максимальное значение поля `Id` в таблице, чтобы при добавлении записи первичный ключ никогда не повторялся и был уникальным

```
private static int findMaxIdForUserException(DataTable table)
{
    int maxId = -1;
    for (int i = 0; i < table.Rows.Count; i++)
    {
        int id = table.Rows[i].Field<int>("Id");
        if (id > maxId) maxId = id;
    }

    return maxId;
}
```

## Шаг 4. Операция добавления данных

---

В момент загрузки форма добавления записи первичный ключ увеличивается на единицу (автоинкремент)

```
private void addForm_Load(object sender, EventArgs e)
{
    Form6 main_db = this.Owner as Form6;

    if (main_db != null)
    {
        int newId = findMaxIdForUserException(main_db.dataSet11.Tables[0]) + 1;
        textBox_id.Text = newId.ToString();
    }
}
```



## Шаг 4. Операция добавления данных

---

Кнопка «**Заккрыть**» запрещает пользователю добавлять строки вне формы добавления, которую мы реализовали

```
private void button2_Click(object sender, EventArgs e)
{
    Form6 main_db = this.Owner as Form6;

    if (main_db != null)
    {
        //запрещаем пользователю самому добавлять строки
        main_db.dataGridView1.AllowUserToAddRows = false;
    }
    Close();
}
```

Сохраняем проект и тестируем... 😊