

Рязанский станкостроительный колледж РГРТУ

МДК.01.01 Разработка программных модулей

Тема 6. Основы Entity Framework

**Основы разработки БД в среде Visual Studio
на языке C#»**

(версия 0.8.9)

Рязань 2022

Оглавление

Общие сведения	4
Введение.....	4
Способы взаимодействия с БД.....	5
Общая схема работы с базой данных в Visual Studio C#	5
Проектирование и создание базы данных.....	5
Подключение к БД в среде Visual Studio C#	6
Структура модели ADO.NET EDM.....	12
Применение паттерна Singleton (Одиночка) для упрощения доступа к контексту данных	13
Реализация интерфейса для отображения БД	15
Форма список (таблица).....	15
Алгоритм создания формы списка.....	15
Форма - бланк.....	16
Типовые операции с данными в таблице.....	16
Алгоритм добавления записи в БД.....	16
Алгоритм редактирования записи в БД	18
Алгоритм удаления записи в БД.....	20
Настройка полей таблицы <i>DataGrid</i>	21
Поиск информации.....	23
Фильтрация данных	24
Извлечение данных из таблиц БД	25
Извлечение данных из таблицы <i>DataGrid</i> с привязанным элементом <i>DataTable</i>	25
Извлечение данных из таблицы <i>DataGrid</i> с привязанной таблицей из БД.....	25
Корректировка структуры БД.....	25
Алгоритм изменения модели данных	26
Практическая работа №17	28
Создание SQL запросов	31
Использование LINQ to Entities.....	32

Выборка из базы данных (фильтр)	32
Проекция базы данных	33
Сортировка базы данных	33
Агрегатные операции	33
Использование представлений, функций и процедур для создания запросов	35
Представления	35
Функции	35
Процедуры	36
Выбор подходов для реализации запросов	38
Настройка пути к базе данных	38
Практическая работа № 18	39
Создание формы авторизации	42
Постановка задачи	42
Проектирование БД	42
Алгоритм создания формы авторизации	44
Практическая работа № 19	48
Создание отчетов	49
Экспорт в Word и Excel	49

Общие сведения

Введение

При работе с компьютером всегда возникает необходимость манипулировать большими массивами данных, в таких случаях используют базы данных. Создают базы данных и обрабатывают запросы к ним системы управления базами данных - СУБД. Известно множество СУБД, которые различаются своими возможностями и конкурируют друг с другом: Microsoft Access, Oracle, Microsoft SQL и много других.

Разные СУБД по разному организуют и хранят данные. Например, Access несколько таблиц хранит в одном файле, некоторые СУБД каждую таблицу хранят в отдельном файле. Система типа клиент-сервер Microsoft SQL хранит данные на отдельном компьютере и доступ к ним возможен только с помощью специального языка SQL.

Entity Framework представляет специальную объектно-ориентированную технологию на базе фреймворка .NET для работы с данными. Если традиционные средства позволяют создавать подключения, команды и прочие объекты для взаимодействия с базами данных, то **Entity Framework** представляет собой более высокий уровень абстракции, который позволяет абстрагироваться от самой базы данных и работать с данными независимо от типа хранилища. Если на физическом уровне мы оперируем таблицами, индексами, первичными и внешними ключами, но на концептуальном уровне, который нам предлагает **Entity Framework**, мы уже работаем с объектами.

Центральной концепцией **Entity Framework** является понятие *сущности или entity*. Сущность представляет набор данных, ассоциированных с определенным объектом. Поэтому данная технология предполагает работу не с таблицами, а с объектами и их наборами.

Любая сущность, как и любой объект из реального мира, обладает рядом свойств. Например, если сущность описывает человека, то мы можем выделить такие свойства, как имя, фамилия, рост, возраст, вес. Свойства необязательно представляют простые данные типа int, но и могут представлять комплексные структуры данных. И у каждой сущности может быть одно или несколько свойств, которые будут отличать эту сущность от других и будут уникально определять эту сущность. Подобные свойства называют **ключами**.

При этом сущности могут быть связаны ассоциативной связью один-ко-многим, один-ко-одному и многие-ко-многим, подобно тому, как в реальной базе данных происходит связь через внешние ключи.

Другим ключевым понятием является **Entity Data Model**. Эта модель сопоставляет классы сущностей с реальными таблицами в БД.

Entity Data Model состоит из трех уровней: концептуального, уровень хранилища и уровень сопоставления (маппинга).

На концептуальном уровне происходит определение **классов сущностей**, используемых в приложении.

Уровень хранилища определяет таблицы, столбцы, отношения между таблицами и типы данных, с которыми сопоставляется используемая база данных.

Уровень сопоставления (маппинга) служит посредником между предыдущими двумя, определяя сопоставление между свойствами класса сущности и столбцами таблиц.

Таким образом, мы можем через классы, определенные в приложении, взаимодействовать с таблицами из базы данных.

Также отличительной чертой *Entity Framework* является использование запросов *LINQ* для выборки данных из БД. С помощью *LINQ* мы можем не только извлекать определенные строки, хранящие объекты, из БД, но и получать объекты, связанные различными ассоциативными связями.

Способы взаимодействия с БД

Entity Framework предполагает три возможных способа взаимодействия с базой данных:

- *Database first*: сначала строится БД, по которой создается набор классов, которые отражают модель конкретной базы данных;
- *Model first*: сначала создает модель базы данных, по которой затем создается реальная БД данных на сервере и набор классов;
- *Code first*: создается набор классов отражающих модель данных, которые будут храниться в БД, а затем по этой модели генерируется БД.

Далее за основу изучения *Entity Framework* будем использовать подход *Database first*.

Общая схема работы с базой данных в Visual Studio C#

В этом разделе мы рассмотрим по шагам общие принципы работы с базами данных с использованием среды программирования Visual Studio и технологии доступа к базам данных *Entity Framework*. За основу изучения *Entity Framework* будем использовать подход *Database first*.

Проектирование и создание базы данных

Первоначально изучаем предметную область и проектируем базу данных, т.е. определяем поля и их характеристики, выбираем ключевое поле, определяем индексы полей, создаем связи между таблицами, определяем основные функции которые потребуются от базы данных и т.д.. В данном случае мы будем рассматривать однотабличные базы данных.

В качестве СУБД можно выбрать любую базу данных. На дальнейшую разработку приложения выбор базы данных практически не влияет.

Для своей работы мы выберем широко распространенную СУБД Microsoft SQL Server. Выбор этой СУБД обусловлен тем, что она изучается на предмете "Базы данных", т.е. мы готовы ее использовать без дополнительной подготовки.

Итак, на этом этапе проектируем базу данных согласно предметной области технического задания, подробно этот процесс не рассматривает т.к. вы должны уже владеть такими навыками.

В СУБД Microsoft SQL Server создаем структуру базы данных, т.е. определяем поля их базовые характеристики, такие как название, тип поля. Определяем ключевые поля, задаем индексы полей, создаем связи между таблицами и т.д. Также можно заполнить несколько записей для использования их в приложении, для отладки функций этой программы.

В итоге мы получаем БД в СУБД Microsoft SQL Server, который содержит одну или несколько таблиц описывающих заданную предметную область.

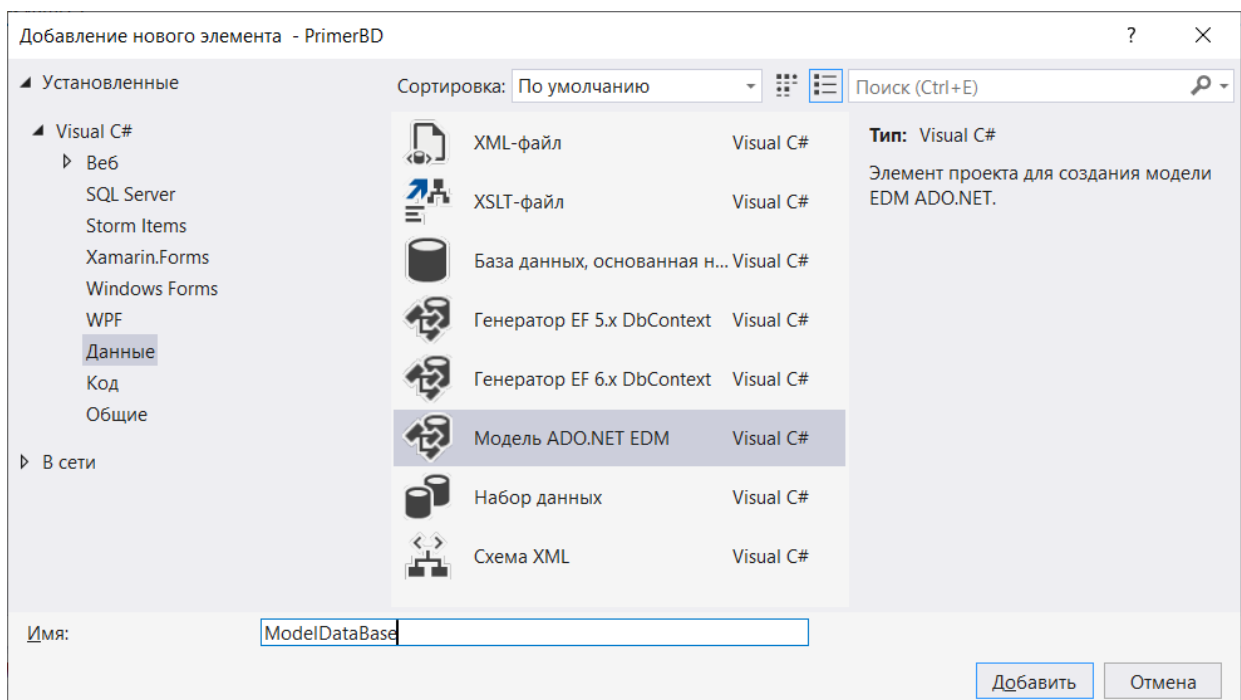
Спроектируем простую таблицу - адресный справочник см. рисунок. Также сразу можно задать основные настройки полей: размер поля, индексацию полей. Можно заполнить несколько записей для дальнейшего использования в отладочных целях.

Abonent			
	Имя столбца	Тип данных	Разрешить значения NULL
	Id	int	<input type="checkbox"/>
	Fio	nvarchar(50)	<input type="checkbox"/>
	Gender	nvarchar(7)	<input type="checkbox"/>
	Age	date	<input checked="" type="checkbox"/>
	Inn	nvarchar(12)	<input checked="" type="checkbox"/>
	Phone	nvarchar(10)	<input checked="" type="checkbox"/>
	Photo	image	<input checked="" type="checkbox"/>
			<input type="checkbox"/>

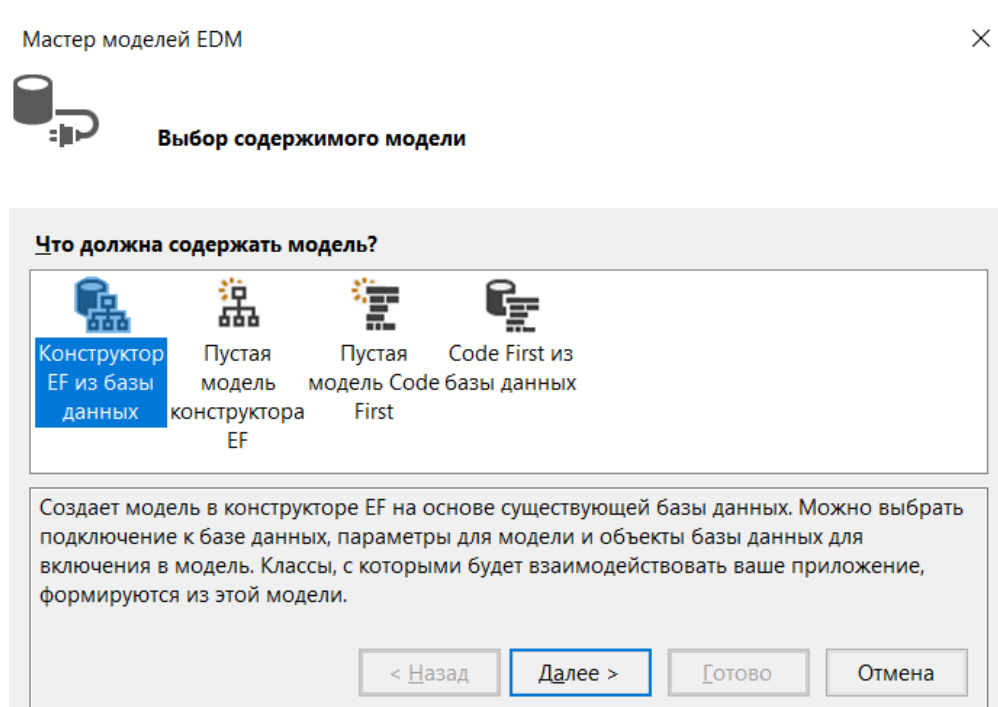
Замечание: При внесении изменений в уже готовую базу данных может стоять запрет на пересоздание таблицы, которое необходимо для внесения изменений. Отключить запрет можно командой *Сервис – Параметры – Конструкторы* и снять галочку с опции *Запретить сохранение изменений, требующих повторного создания таблицы*.

Подключение к БД в среде Visual Studio C#

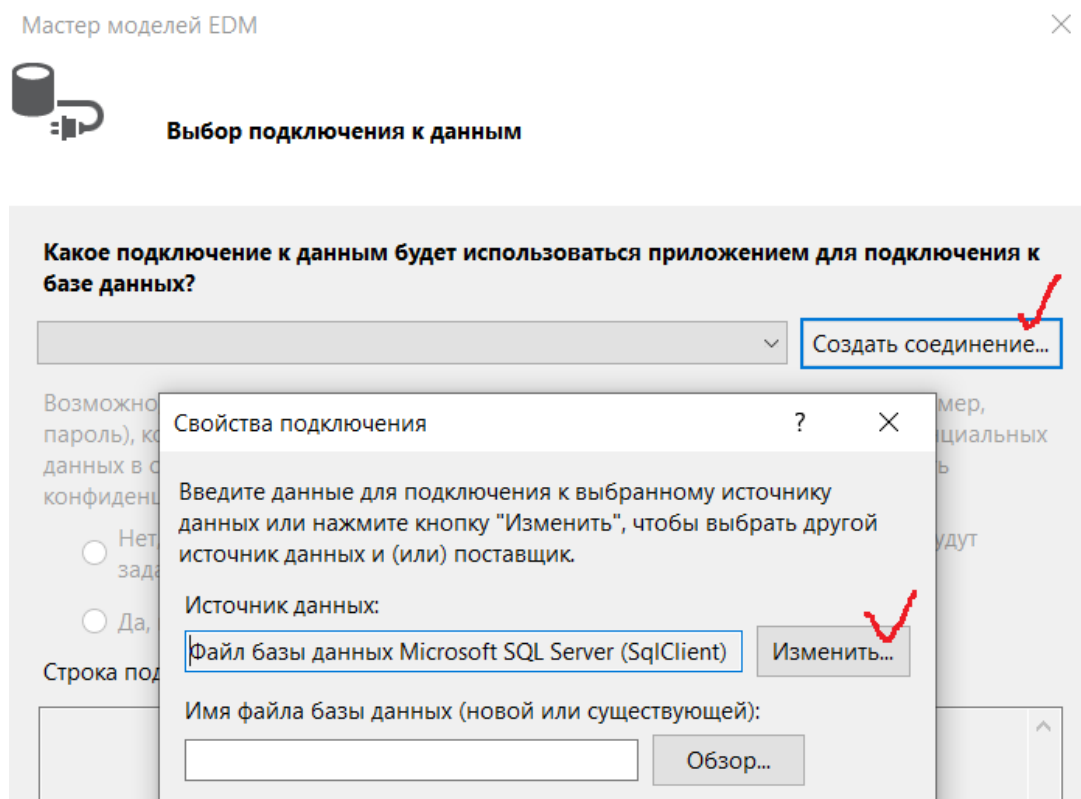
1. Выбрать команду *Проект – Добавить новый элемент*. В разделе *Данные*, выбрать элемент *Модель ADO.NET EDM*. Дайте название модели и нажмите кнопку *Добавить*.



2. Откроется окно **Мастер моделей EDM**, в котором выбрать **Конструктор EF из базы данных**.



3. Выбрать подключение к базе данных. Для нового подключения нажать кнопку **Создать соединение**, выбрать **Источник данных** в нашем случае **Microsoft SQL Server**.



4. Настроить свойства подключения. При использовании локального сервера SQL указать:

- Имя сервера *localhost\sqlexpress*;
- Параметры авторизации на сервере *Проверка подлинности Windows*;
- Имя базы данных, в нашем случае Abonents.

Свойства подключения

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:
localhost\sqlexpress Обновить

Вход на сервер

Проверка подлинности: Проверка подлинности SQL Server

Имя пользователя: sa

Пароль:

☐ Сохранить пароль

Подключение к базе данных

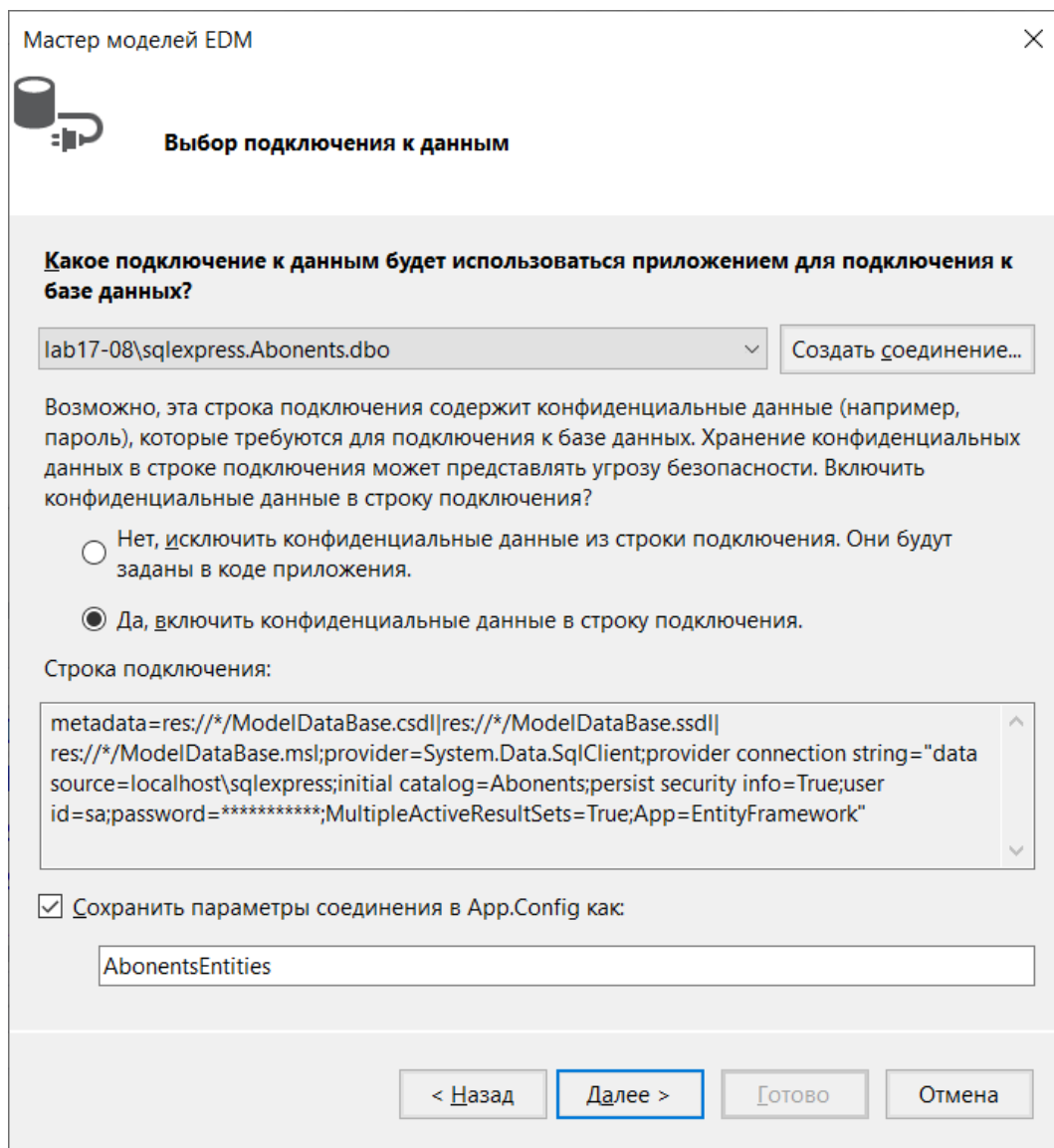
☒ Выберите или введите имя базы данных:

- Abonents
- master
- model
- msdb
- tempdb
- Колледж
- Корпорация добра

Дополнительно...

Проверить подключение OK Отмена

5. Подтвердить параметры подключения. Нажать кнопку *Далее*.



Мастер моделей EDM

Выбор подключения к данным

Какое подключение к данным будет использоваться приложением для подключения к базе данных?

lab17-08\sqlexpress.Abonents.dbo Создать соединение...

Возможно, эта строка подключения содержит конфиденциальные данные (например, пароль), которые требуются для подключения к базе данных. Хранение конфиденциальных данных в строке подключения может представлять угрозу безопасности. Включить конфиденциальные данные в строку подключения?

☐ Нет, исключить конфиденциальные данные из строки подключения. Они будут заданы в коде приложения.

☒ Да, включить конфиденциальные данные в строку подключения.

Строка подключения:

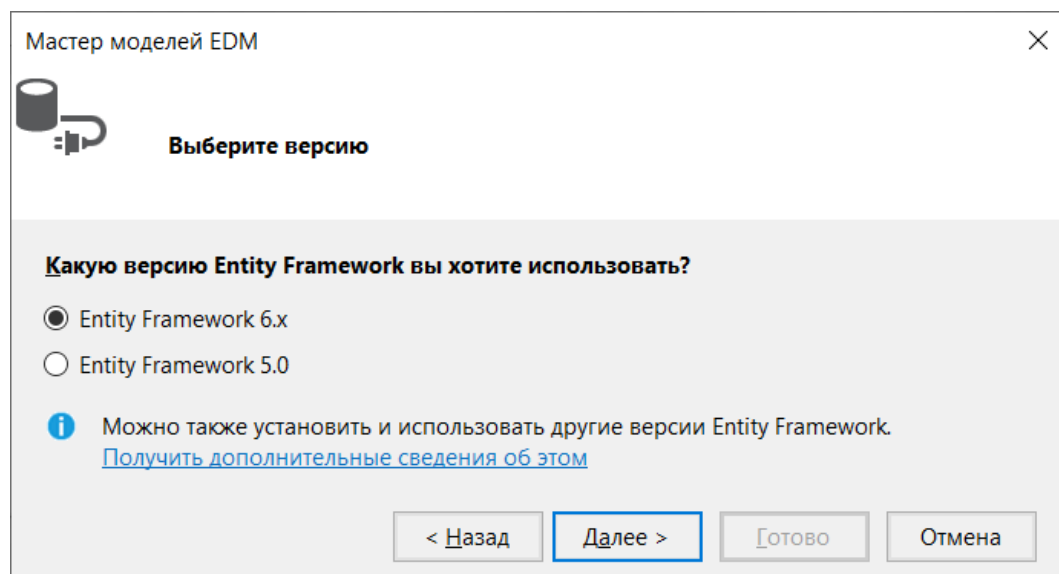
```
metadata=res://*/ModelDataBase.csdl|res://*/ModelDataBase.ssdl|
res://*/ModelDataBase.msl;provider=System.Data.SqlClient;provider connection string="data
source=localhost\sqlexpress;initial catalog=Abonents;persist security info=True;user
id=sa;password=*****;MultipleActiveResultSets=True;App=EntityFramework"
```

☒ Сохранить параметры соединения в App.Config как:

AbonentsEntities

< Назад Далее > Готово Отмена

6. Выбрать версию Entity Framework, по умолчанию 6.x.



Мастер моделей EDM

Выберите версию

Какую версию Entity Framework вы хотите использовать?

☒ Entity Framework 6.x

☐ Entity Framework 5.0

i Можно также установить и использовать другие версии Entity Framework.
[Получить дополнительные сведения об этом](#)

< Назад Далее > Готово Отмена

7. Выбрать объекты базы данных, к которым нужно получить доступ. В нашем примере это таблица *Абонент*. Установить опцию **Формировать имена объектов во множественном или единственном числе**. Нажать кнопку *Готово*.

Мастер моделей EDM

Выберите параметры и объекты базы данных

Какие объекты базы данных нужно включить в модель?

- ☒ Таблицы
 - ☒ dbo
 - ☒ Abonent
 - ☒ sysdiagrams
- ☐ Представления
- ☐ Хранимые процедуры и функции

☒ Формировать имена объектов во множественном или единственном числе

☒ Включить столбцы внешних ключей в модель

☒ Импортировать выбранные хранимые процедуры и функции в модель сущностей

Пространство имен модели:

AbonentsModel

< Назад Далее > Готово Отмена

8. При создании моделей сущностей и набора классов сущностей появляется запросы на подтверждение выполнения шаблонов. Подтвердить, нажать кнопку **ОК**.

Предупреждение системы безопасности

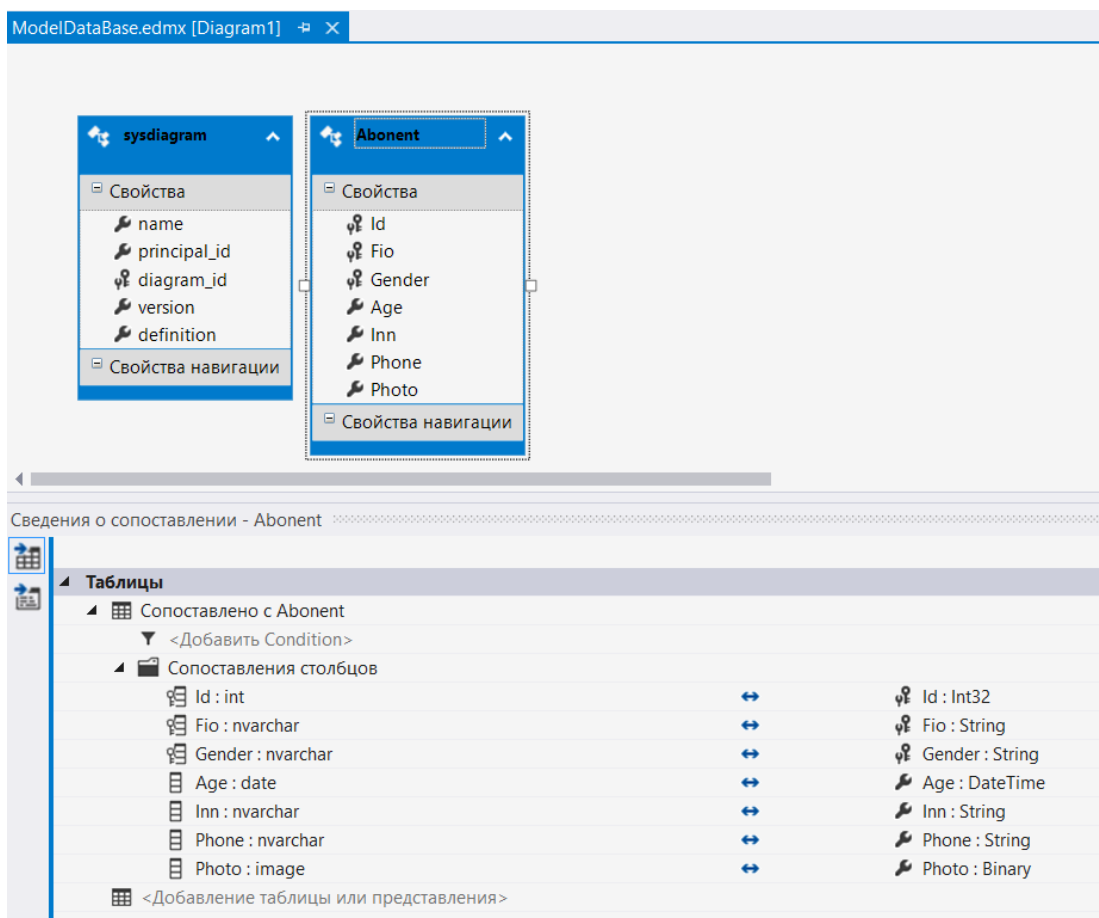
Выполнение этого текстового шаблона потенциально может повредить ваш компьютер. Не запускайте его, если получили его из ненадежного источника.

Нажмите кнопку "ОК", чтобы выполнить шаблон.
Нажмите кнопку "Отмена", чтобы остановить процесс.

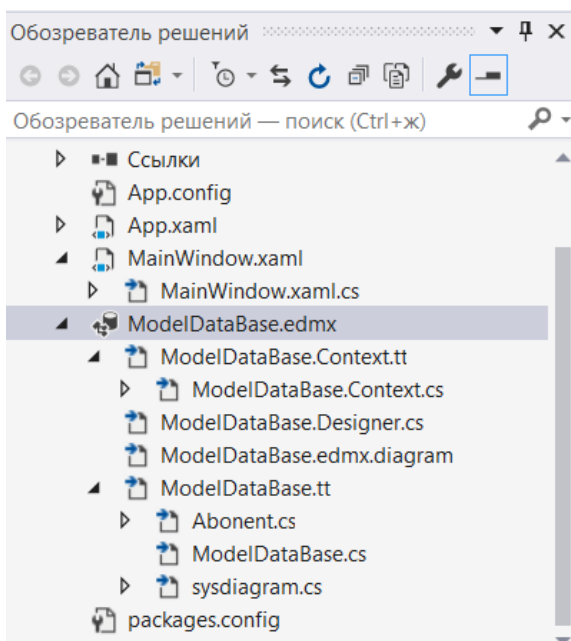
☐ Больше не выводить это сообщение

ОК Отмена

9. После генерации модели, она отобразилась в окне Visual Studio. В модели отображаются все сущности БД и их связи. Для выбранных таблиц отображаются сведения о сопоставления типа данных в СУБД Microsoft SQL Server и классе сущности.

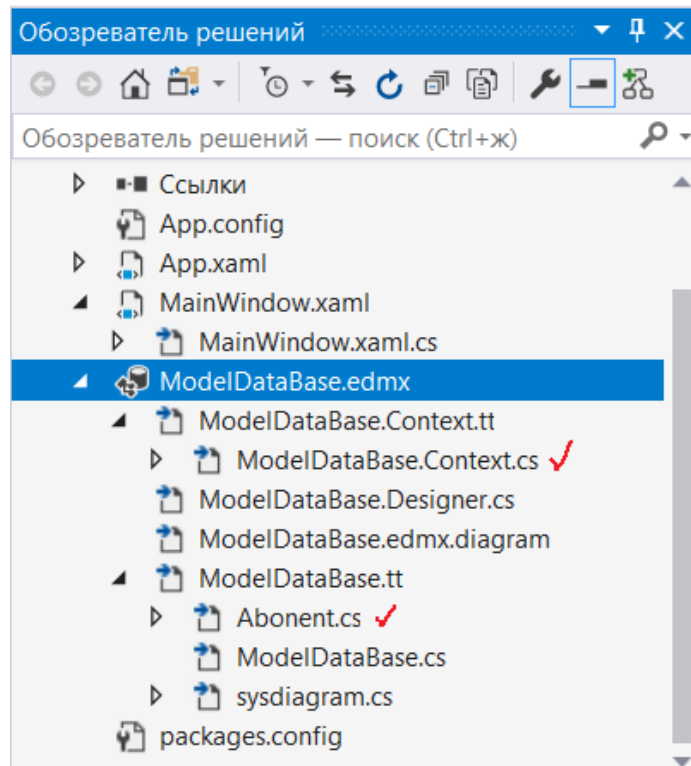


10. В окне **Обозреватель решений** появится элемент **ModelDataBase.edmx**, который содержит модель БД. Двойной щелчок на этом элементе открывает модель данных, которая позволяет управлять сущностями на уровне модели.



Структура модели ADO.NET EDM

Как было сказано выше, в окне *Обозреватель решений* находится элемент *ModelDataBase.edmx*, который содержит модель БД. Двойной щелчок на этом элементе открывает модель данных, которая позволяет управлять сущностями на уровне модели.



Модель данных содержит классы сущностей, которые ассоциированы с таблицами в БД. В данном случае, это раздел *ModelDataBase.tt* и класс *Abonents.cs*.

```
public partial class Abonent
{
    Ссылка: 0
    public int Id { get; set; }
    Ссылка: 0
    public string Fio { get; set; }
    Ссылка: 0
    public string Gender { get; set; }
    Ссылка: 0
    public Nullable<System.DateTime> Age { get; set; }
    Ссылка: 0
    public string Inn { get; set; }
    Ссылка: 0
    public string Phone { get; set; }
    Ссылка: 0
    public byte[] Photo { get; set; }
}
```

Модель данных содержит класс контекста *ModelDataBase.Context.cs*, производный от *DbContext* и набор данных *DbSet*, через который мы сможем взаимодействовать с таблицами из БД.

```

namespace PrimerBD
{
    using System;
    using System.Data.Entity;
    using System.Data.Entity.Infrastructure;

    Ссылка: 13
    public partial class AbonentsEntities : DbContext
    {
        Ссылка: 1
        public AbonentsEntities()
            : base("name=AbonentsEntities")
        {
        }

        Ссылка: 0
        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            throw new UnintentionalCodeFirstException();
        }

        Ссылка: 32
        public virtual DbSet<Abonent> Abonents { get; set; }
    }
}

```

Применение паттерна Singleton (Одиночка) для упрощения доступа к контексту данных

Для работы данными используется класс контекста данных, расположенный в файле *ModelDataBase.Context.cs*. Соответственно, чтобы использовать этот класс надо создать объект класса.

```

//Создаем набор данных
DatabaseEntities db = new DatabaseEntities();

```

Когда приложение имеет несколько форм, то при применении такого подхода на каждой форме создаются новые контексты данных, через которые идет взаимодействие с БД. Соответственно изменения, произведенные на одной из форм, не видны на других формах и необходимо перезагружать информацию из БД.

Чтобы этого не делать можно использовать паттерн Singleton (Одиночка) и создать единую ссылку на контекст данных, который будет использоваться на всех формах.

Для применения этого способа выполните следующие действия:

1. Создайте новый программный элемент *ModelExtension* с помощью команды **Проект – Добавить класс**.
2. Замените содержимое элемента *ModelExtension* классом контекста.
3. Удалите тело класса контекста и замените следующим содержимым, которое создает статическую ссылку на контекст данных и позволяет с помощью метода получить единый контекст данных в любом месте программы.

```
namespace PrimerBD
{
    using System;
    using System.Data.Entity;
    using System.Data.Entity.Infrastructure;

    Ссылка: 13
    public partial class AbonentsEntities : DbContext
    {
        //Статичную ссылку на контент
        private static AbonentsEntities context;
        //Получение контекста
        Ссылка: 4
        public static AbonentsEntities GetContext()
        {
            if (context == null) context = new AbonentsEntities();
            return context;
        }
    }
}
```

Обратите внимание, что в данном случае описание класса *AbonentsEntities* производится в двух местах, это элемент *ModelDataBase.Context.cs* и *ModelExtension.cs*. Такая возможность доступна при использовании модификатора класса *partial*.

Таким образом элемент *ModelDataBase.Context.cs* содержит основное описание класса контекста и создаётся или обновляется автоматически при генерации модели из базы данных. Элемент *ModelExtension.cs* создается вручную и содержит дополнительные возможности класса контекста.

Такой подход расширения модели данных будет далее применяться неоднократно.

Реализация интерфейса для отображения БД

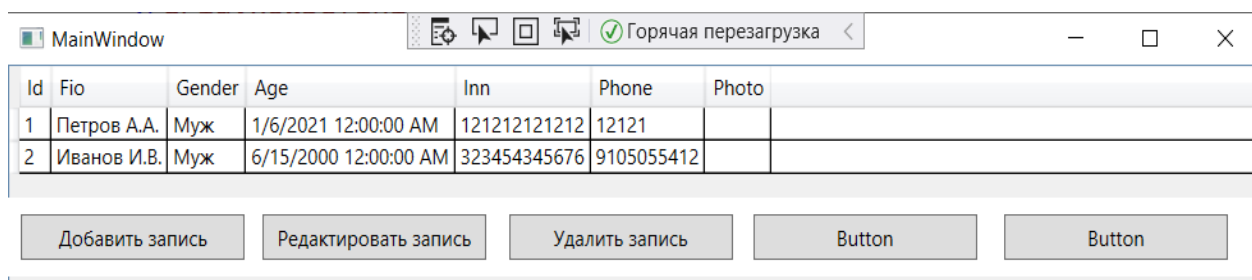
При разработке БД одной из важных задач является отображение информации из БД. Правильным подходом является продуманное отображение информации, т.е. главный критерий это размещение информации в удобном виде для работы с ней пользователем этой программы и решение поставленных перед ним задач.

Рассмотрим наиболее типовые подходы при реализации интерфейса.

Форма список (таблица)

Форма список отображает информацию в виде таблицы (списка). В таком виде обычно отображаются таблицы – справочники, полная или обобщённая информация из таблицы.

В нашем примере – это таблица Абонент.



Алгоритм создания формы списка

1. На форме размещаем элемент таблицу (*DataGrid*).

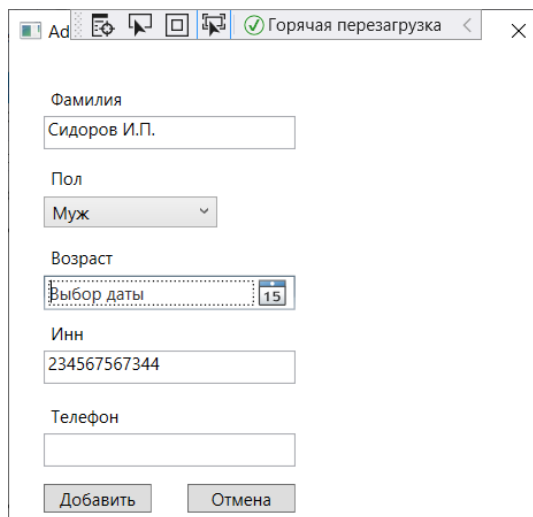
```
<DataGrid x:Name="DataGrid1" IsReadOnly="True"/>
```

2. Объявляем глобальную переменную для получения доступа к контексту данных и получаем контекст данных. При загрузке формы с использованием контекста данных загружаем таблицу из БД. Привязываем или загружаем в *DataGrid* таблицу с данными с отслеживанием изменений контекста данных или без отслеживания изменений контекста данных.

```
//Получаем доступ к контексту данных
AbonentsEntities db = AbonentsEntities.GetContext();
ссылка: 1
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    //Загружаем таблицу из БД
    db.Abonents.Load();
    //Загружаем таблицу в DataGrid без отслеживания изменения контекста
    //DataGrid1.ItemsSource = db.Abonents.ToList();
    //Загружаем таблицу в DataGrid с отслеживанием изменения контекста
    DataGrid1.ItemsSource = db.Abonents.Local.ToBindingList();
}
```

Форма - бланк

Эта форма похожа на анкету. На форме, как правило, располагаются поля одной записи. Далее приведен пример формы для добавления записи в таблицу. Пример создания этой формы будет рассмотрен далее.



The screenshot shows a Windows-style dialog box titled "Горячая перезагрузка" (Hot Restart). It contains several input fields: "Фамилия" (Surname) with the value "Сидоров И.П.", "Пол" (Gender) with a dropdown menu set to "Муж" (Male), "Возраст" (Age) with a date picker set to "15", "Инд" (INN) with the value "234567567344", and "Телефон" (Phone) which is empty. At the bottom are two buttons: "Добавить" (Add) and "Отмена" (Cancel).

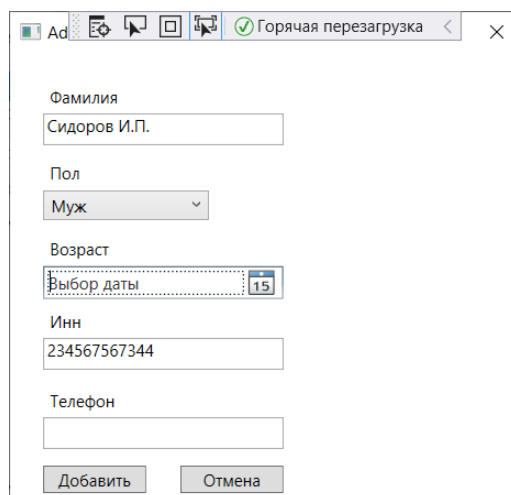
Типовые операции с данными в таблице

Над данными в таблице обычно выполняются такие операции как добавления записи, изменение записи, удаление записи. Как правило, при добавлении или изменении записи открывается дополнительная форма, где добавляемая или изменяемая запись представлена в виде формы – бланка. Ввод или редактирование записи происходит на этой дополнительной форме. Так же часто при работе с текущей записью происходит контроль данных по заданным критериям и в БД попадает только та информация, которая соответствует этим критериям.

Алгоритм добавления записи в БД

1. На основную форму добавим кнопку *Добавить запись*. Кнопка *Добавить запись* будет открывать форму – бланк содержащие элементы для добавления новой записи.

2. Создаем форму *Добавить запись* см. пример. Эта форма будет содержать обычные TextBox, ComboBox для поля пол, DatePicker для поля дата рождения.



This is an identical screenshot to the one above, showing the "Горячая перезагрузка" (Hot Restart) dialog box with the same fields and values: "Фамилия" (Сидоров И.П.), "Пол" (Муж), "Возраст" (15), "Инд" (234567567344), and "Телефон" (empty). The buttons "Добавить" and "Отмена" are also present.

3. На основной форме пишем код кнопки *Добавить запись*. Кнопка *Добавить запись* будет открывать форму *Добавить запись*.

```
private void Button_Add(object sender, RoutedEventArgs e)
{
    //Открываем форму Добавить
    AddRecord f = new AddRecord();
    f.ShowDialog();
    DataGrid1.Focus();
}
```

4. На форме *Добавить запись* объявляем глобальную переменную для получения доступа к контексту данных и получаем контекст данных. Создаем сущность таблицы.

```
//Получаем доступ к контексту данных
AbonentsEntities db = AbonentsEntities.GetContext();
//Создаем элемент таблицы
Abonent p1 = new Abonent();
```

4. На форме *Добавить запись* пишем код кнопки *Добавить*. Добавлять информацию в БД будем с помощью запроса на добавление.

Учтем еще требования задания, что:

- поля Фамилия, Имя, Пол, ИНН должны быть заполнены;
- поле Пол = Муж или Жен;
- поле Дата рождения – заполнено, можно проверить например на совершеннолетие;
- поле ИНН 12 знаков и только цифры.

```
private void Btn_Click_Add(object sender, RoutedEventArgs e)
{
    //Пример проверки всех полей на заполненность
    //if (TextFio.Text.Length == 0 || TextInn.Text.Length == 0)
    //{ MessageBox.Show("Заполните все поля"); return; }

    //Пример проверки каждого поля отдельно
    StringBuilder errors = new StringBuilder();
    if (TextFio.Text.Length == 0) errors.AppendLine("Введите фамилию");
    if (CBGender.Text != "Муж" && CBGender.Text != "Жен")
        errors.AppendLine("Введите пол Муж/Жен");
    if (TextInn.Text.Length != 12 ||
        double.TryParse(TextInn.Text, out double x) == false)
        errors.AppendLine("Неправильный ИНН");
    if (DatePicker1.Text.Length == 0) errors.AppendLine("Введите дату");
    /*
    // Проверка что абоненту более 18 лет
    DateTime dt = DateTime.Now; //Текущая дата
    DateTime dp = Convert.ToDateTime(DatePicker1.SelectedDate); //Дата рождения
    int yt = dt.Year; //Получение года
    int yp = dp.Year;
    if (yt - yp < 18)
    {
        errors.AppendLine("Введите правильно дату рождения");
    }
    */
}
```

```
if (errors.Length > 0)
{
    MessageBox.Show(errors.ToString());
    return;
}
//Создаем элемент таблицы
//Abonent p1 = new Abonent();
//Заполняем этот элемент
p1.Fio = TextFio.Text;
p1.Gender = CBGender.Text;
p1.Age = DatePicker1.SelectedDate;
p1.Inn = TextInn.Text;
p1.Phone = TextPhone.Text;

try
{
    //Добавляем в БД
    db.Abonents.Add(p1);
    //Сохраняем изменения
    db.SaveChanges();
    //MessageBox.Show("Информация сохранена!");
    this.Close();
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message.ToString());
}
}
```

Алгоритм редактирования записи в БД

1. На основную форму добавим кнопку **Изменить запись**. Кнопка **Изменить запись** будет открывать форму – бланк содержащие элементы для изменения текущей записи.

2. Создаем форму **Изменить запись** см. предыдущий пример. Эта форма будет содержать обычные TextBox, ComboBox для поля пол, DateTimePicker для поля дата рождения.

Так как форма **Изменить запись** имеет такой же вид как и форма **Добавить запись**, то можно в окне **Обозреватель решений** с помощью операций **Копировать** и **Вставить** сделать копию формы **Добавить запись**.

При создании копии:

- измените название формы в обозревателе решений;
- измените название класса в разметке XAML;
- измените название класса и конструктора в программном коде окна.

3. Создайте глобальный статический класс для передачи данных между формами. В этом классе опишем элемент Id для передачи кода текущей записи см. пример.

```
namespace PrimerBD
{
    Ссылка: 2
    public static class Data {
        public static int Id; //Код записи
    }
}
```

4. На основной форме пишем код кнопки **Изменить запись**. Кнопка **Изменить запись** будет открывать форму **Изменить запись**.

Чтобы изменять текущую запись запоминаем ее код.

Также т.к. при изменении записи контекст автоматически не обновляется, в таблице на основной форме изменения не отобразятся. Чтобы увидеть обновленную запись нужно вручную обновить таблицу.

Также таблица при нажатии кнопки Изменить теряет фокус и соответственно теряется выделение текущей записи, возвращаем фокус таблице см. код кнопки **Изменить запись**.

```
private void Button_Edit(object sender, RoutedEventArgs e)
{
    int indexRow = DataGrid1.SelectedIndex;
    if (indexRow != -1)
    {
        //Получаем ключ текущей записи
        Abonent row = (Abonent)DataGrid1.Items[indexRow];
        Data.Id = row.Id;
        //Открываем форму Редактировать
        EditRecord f = new EditRecord();
        f.ShowDialog();
        //Обновляем таблицу
        DataGrid1.Items.Refresh();
        DataGrid1.Focus();
    }
}
```

5. На форме **Изменить запись** при загрузке формы отображаем содержимое текущей записи.

```
//Получаем доступ к контексту данных
AbonentsEntities db = AbonentsEntities.GetContext();
Abonent p1; //Элемент для работы с записью

private void Window_Loaded(object sender, RoutedEventArgs e)
{
    //Получаем запись по коду
    p1 = db.Abonents.Find(Data.Id);
    //Отображаем запись
    TextFio.Text = p1.Fio;
    CBGender.Text = p1.Gender;
    DatePicker1.SelectedDate = p1.Age;
    TextInn.Text = p1.Inn;
    TextPhone.Text = p1.Phone;
}
```

6. На форме **Изменить запись** пишем код кнопки **Изменить**. Учтем заданные ограничения полей.

```
private void Btn_Click_Edit(object sender, RoutedEventArgs e)
{
    //Пример проверки всех полей на заполненность
    //if (TextFio.Text.Length == 0 || TextInn.Text.Length == 0)
    //{ MessageBox.Show("Заполните все поля"); return; }

    //Пример проверки каждого поля отдельно
    StringBuilder errors = new StringBuilder();
    if (TextFio.Text.Length == 0) errors.AppendLine("Введите фамилию");
    if (CBGender.Text != "Муж" && CBBGender.Text != "Жен")
        errors.AppendLine("Введите пол Муж/Жен");
    if (TextInn.Text.Length != 12 ||
        double.TryParse(TextInn.Text, out double x) == false)
        errors.AppendLine("Неправильный ИНН");
    if (DatePicker1.Text.Length == 0) errors.AppendLine("Введите дату");
    /*
    // Проверка что абоненту более 18 лет
    DateTime dt = DateTime.Now; //Текущая дата
    DateTime dp = Convert.ToDateTime(DatePicker1.SelectedDate); //Дата рождения
    int yt = dt.Year; //Получение года
    int yp = dp.Year;
    if (yt - yp < 18)
    {
        errors.AppendLine("Введите правильно дату рождения");
    }
    */
    if (errors.Length > 0)
    {
        MessageBox.Show(errors.ToString());
        return;
    }
    //Заполняем этот элемент
    p1.Fio = TextFio.Text;
    p1.Gender = CBBGender.Text;
    p1.Age = DatePicker1.SelectedDate;
    p1.Inn = TextInn.Text;
    p1.Phone = TextPhone.Text;

    try
    {
        db.SaveChanges();
        //MessageBox.Show("Информация сохранена!");
        this.Close();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message.ToString());
    }
}
```

Алгоритм удаления записи в БД

1. На основную форму добавим кнопку **Удалить запись**. Кнопка **Удалить запись** будет удалять текущую запись с подтверждением.

2. На основной форме пишем код кнопки **Удалить запись**.

Вначале используем элемент **MessageBox** для вывода окна подтверждения удаления. При подтверждении удаления, удаляем текущую запись.

```
private void Button_Delete(object sender, RoutedEventArgs e)
{
    MessageBoxResult result;
    result = MessageBox.Show("Удалить запись?", "Удаление записи",
        MessageBoxButton.YesNo, MessageBoxImage.Warning);
    if (result == MessageBoxResult.Yes)
    {
        try
        {
            //Получаем текущую запись
            //Abonent row = (Abonent)DataGrid1.Items[indexRow];
            Abonent row = (Abonent)DataGrid1.SelectedItems[0];
            //Abonent row = (Abonent)DataGrid1.CurrentCell.Item;
            //Удаляем запись
            db.Abonents.Remove(row);
            db.SaveChanges();
        }
        catch (ArgumentOutOfRangeException)
        {
            ..
            MessageBox.Show("Выберите запись");
            ..
        }
    }
}
```

Настройка полей таблицы **DataGrid**

Если вы обратили внимание, что в элементе **DataGrid** информация выводится в упрощенном виде: размеры столбцов по размеру данных, названия столбцов по названию полей, справа всегда есть 1 пустой столбец, выравнивание по левому краю. Происходит это потому, что используется динамическая привязка данных к таблице и отображение информации происходит стандартным образом см. пример ниже.

Id	Fio	Gender	Age	Inn	Phone	Photo
1	Петров А.А.	Муж	1/6/2021 12:00:00 AM	121212121212	12121	
2	Иванов И.В.	Муж	6/15/2000 12:00:00 AM	323454345676	9105055412	

Чтобы вывести информацию в более удобном виде нужно провести форматирование таблицы **DataGrid**. Для форматирования таблицы нужно в нее добавить столбцы, которые можно добавить вручную в коде **XAML** или через контекстное меню командой **Добавить столбец**. При добавлении столбца указываем его тип **DataGridTextColumn** для простых

столбцов, *DataGridTemplateColumn* если хотим настроить параметры столбца более подробно и т.д..

Настраиваем основные параметры столбца и привязываем его с помощью атрибута *Binding* к полям таблицы в базе данных, см. пример ниже:

```
<DataGrid x:Name="DataGrid1" IsReadOnly="True" AutoGenerateColumns="False" Grid.Row="1">
    <DataGrid.Columns>
        <DataGridTextColumn Width="15*" Header="Код" Binding="{Binding Id}"/>
        <DataGridTextColumn Width="100*" Header="ФИО" Binding="{Binding Fio}"/>
        <DataGridTextColumn Width="35*" Header="Пол" Binding="{Binding Gender}"/>
        <DataGridTextColumn Width="55*" Header="Дата рождения" Binding="{Binding Age}"/>
        <DataGridTextColumn Width="50*" Header="ИНН" Binding="{Binding Inn}"/>
        <DataGridTextColumn Width="50*" Header="Телефон" Binding="{Binding Phone}"/>
    </DataGrid.Columns>
</DataGrid>
```

Код	ФИО	Пол	Дата рождения	ИНН	Телефон
1	Петров	Жен	2/14/1990 12:00:00 AM	454547845445	
2	Иванов	Жен	2/8/1979 12:00:00 AM	121245454545	

Обратите внимание! Для корректного отображение информации установите свойства *IsReadOnly="True"* *AutoGenerateColumns="False"*.

Для более точной настройки столбцов используем *DataGridTemplateColumn*. Пример настройки приведен ниже:

```
<DataGridTemplateColumn Header="Код2" Width="35*">
    <DataGridTemplateColumn.CellTemplate>
        <DataTemplate>
            <StackPanel Orientation="Horizontal" HorizontalAlignment="Right">
                <TextBlock Text="{Binding Id, StringFormat=F2}"/>
                <TextBlock Text=" руб"/>
            </StackPanel>
        </DataTemplate>
    </DataGridTemplateColumn.CellTemplate>
</DataGridTemplateColumn>
<DataGridTemplateColumn Header="ФИО2" Width="50*">
    <DataGridTemplateColumn.CellTemplate>
        <DataTemplate>
            <TextBlock Text="{Binding Fio}" HorizontalAlignment="Center"/>
        </DataTemplate>
    </DataGridTemplateColumn.CellTemplate>
</DataGridTemplateColumn>
<DataGridTemplateColumn Header="Дата2" Width="50*">
    <DataGridTemplateColumn.CellTemplate>
        <DataTemplate>
            <TextBlock Text="{Binding Age, StringFormat=d}"
                HorizontalAlignment="Right"/>
        </DataTemplate>
    </DataGridTemplateColumn.CellTemplate>
</DataGridTemplateColumn>
```

Код2	ФИО2	Дата2
1.00 руб	Петров	2/14/1990
2.00 руб	Иванов	2/8/1979

Рассмотрим основные форматы используемые при форматировании значений с использованием атрибута **StringFormat**.

Описатель формата	Имя	Описание	Примеры
"C" или "c"	Валюта	Результат: значение валюты. Поддерживается все числовые типы.	123.456 ("C", en-US) - > \$123.46 -123.456 ("C3", en-US) - > (\$123.456)
"E" или "e"	Экспоненциальный (научный)	Результат: экспоненциальная нотация. Поддерживается все числовые типы.	1052.0329112756 ("E", en-US) -> 1.052033E+003 -1052.0329112756 ("e2", en-US) -> -1.05e+003
"F" или "f"	С фиксированной запятой	Результат: цифры целой и дробной частей с необязательным отрицательным знаком. Поддерживается все числовые типы.	1234.567 ("F", en-US) -> 1234.57 1234.56 ("F4", en-US) -> -1234.5600
"N" или "n"	Число	Результат: цифры целой и дробной частей, разделители групп и разделитель целой и дробной частей с необязательным отрицательным знаком. Поддерживается все числовые типы.	1234.567 ("N", ru-RU) -> 1 234,57 -1234.56 ("N3", ru-RU) - > -1 234,560
"d"	Короткий шаблон даты		2009-06-15T13:45:30 -> 6/15/2009 (en-US)
"D"	Полный шаблон даты.		2009-06-15T13:45:30 -> 15 июня 2009 г. (ru-RU)
"t"	Короткий шаблон времени.		2009-06-15T13:45:30 -> 1:45 PM (en-US)
"T"	Полный шаблон времени.		2009-06-15T13:45:30 -> 1:45:30 PM (en-US)

Поиск информации

Поиск информации в БД задача довольно непростая и может осуществляться разными способами, которые зависят от того какой конечный вариант нужно получить.

Рассмотрим один из простых вариантов поиска через таблицу **DataGrid**, в которой в цикле проходим нужный столбец и ищем заданное значение. Далее выделяем строку с найденным значением, см. пример интерфейса и кода программы:

Поиск по ФИО

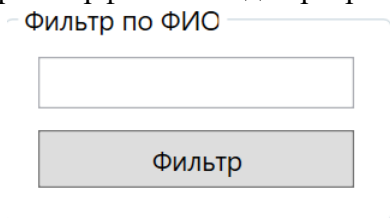
Поиск


```
private void ButtonFind_Click(object sender, RoutedEventArgs e)
{
    for (int i = 0; i < DataGrid1.Items.Count; i++)
    {
        //Получаем строку таблицы
        var row = (Abonent)DataGrid1.Items[i];
        string findContent = row.Fio;
        try
        {
            if (findContent != null && findContent.Contains(txtFind.Text))
            {
                object item = DataGrid1.Items[i];
                DataGrid1.SelectedItem = item; //Выделяем элемент
                DataGrid1.ScrollIntoView(item); //Скролим к нему окно
                DataGrid1.Focus();
                break;
            }
        }
        catch { }
    }
}
```

Фильтрация данных

Фильтрация данных - это выбор данных по заданному критерию (условию).

Одним из способов установить фильтр – это таблицу загрузить в коллекцию *List* и используя метод *Where* сформировать новую коллекцию с отображенными значениями, см. пример интерфейса и кода программы:



```
List<Abonent> _abonent; //Описываем коллекцию
```

ссылка: 1

```
private void ButtonFiltered_Click(object sender, RoutedEventArgs e)
{
    //Загружаем в коллекцию таблицу
    _abonent = db.Abonents.ToList();
    //Формируем новую таблицу по фильтру
    var filtered = _abonent.Where(_abonent => _abonent.Fio == txtFiltered.Text);
    //var filtered = _abonent.Where(_abonent => _abonent.Fio.Contains(txtFiltered.Text));
    //var filtered = _abonent.Where(_abonent => _abonent.Fio.StartsWith(txtFiltered.Text));
    //Отображаем полученную таблицу
    DataGrid1.ItemsSource = filtered;
}
```

Обратите внимание, что для задания критериев поиска используется лямда-выражение. Первым задается объект, в котором производится выборка и через знак *=>* условие отбора.

Извлечение данных из таблиц БД

Извлечение данных из таблицы DataGrid с привязанным элементом DataTable

```
//Определяем номер столбца
int indexColumn = DataGrid1.CurrentCell.Column.DisplayIndex;
//Определяем номер строки
int indexRow = DataGrid1.SelectedIndex;
// Получаем текущую строку
DataRowView row = (DataRowView)DataGrid1.Items[indexRow];
//DataRowView row = (DataRowView)DataGrid1.SelectedItems[0];
//DataRowView row = (DataRowView)DataGrid1.CurrentCell.Item;
// Обращаемся в выбранной строке в указанный столбец
int Value = Convert.ToInt32(row[indexColumn].ToString());
```

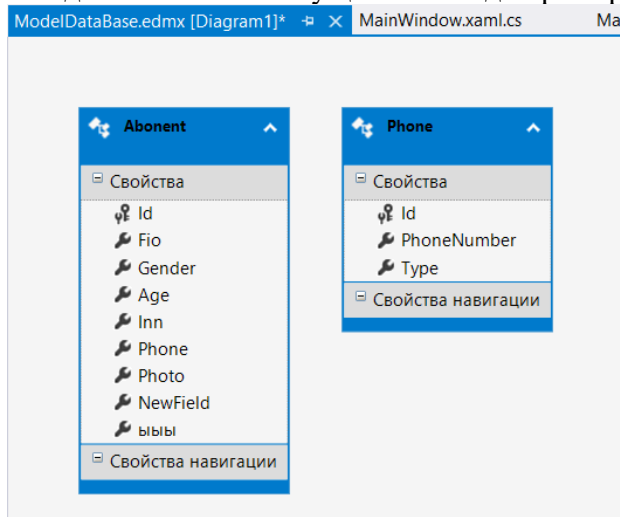
Извлечение данных из таблицы DataGrid с привязанной таблицей из БД

```
//Определяем номер строки
int indexRow = DataGrid1.SelectedIndex;
//Получаем текущую строку
Abonent row = (Abonent)DataGrid1.SelectedValue;
//Abonent row = (Abonent)DataGrid1.Items[indexRow];
//Abonent row = (Abonent)DataGrid1.SelectedItems[0];
//Обращаемся в выбранной строке в указанный столбец
String value = row.Fio;
```

Корректировка структуры БД

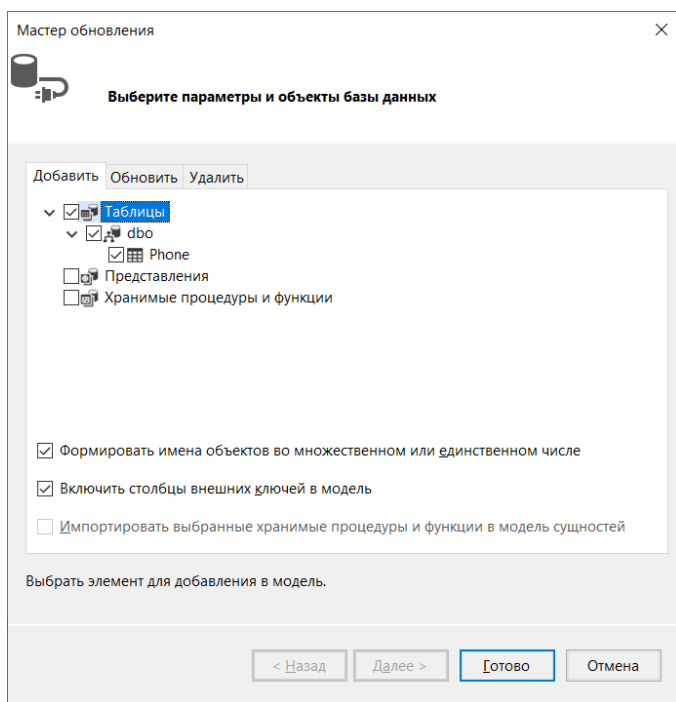
При создании программы обрабатывающей БД очень часто возникает задача изменении структуры таблицы в БД, добавление новой таблицы и т.д.

Соответственно если в БД были внесены изменения, то нужно изменить модель данных в программе *ModelDataBase.edmx*, чтобы программа правильно работала с таблицами БД. Например, добавили новое поле, чтобы его увидеть в программе, поле должно быть добавлено в модель данных и соответственно в класс сущности. Изменили тип данных в поле, тип данных должен быть изменен в модели данных и в классе сущности. Добавили новую таблицу, таблица должна быть добавлена в модель данных и для нее создан новый класс сущности и т.д. Пример модели данных см. ниже:



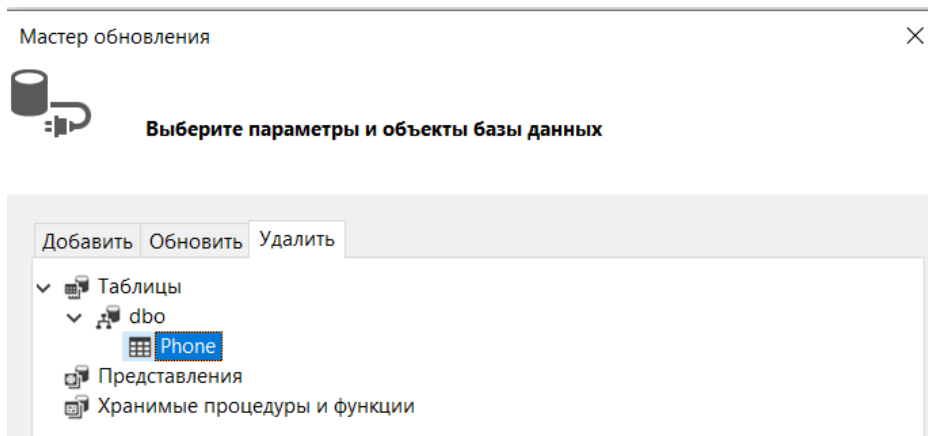
Алгоритм изменения модели данных

1. Открыть модель данных *ModelDataBase.edmx*, вызвать контекстное меню, выбрать команду **Обновить модель данных из базы данных**, откроется мастер обновления.
2. В окне мастера доступны 3 режима работы: *Добавить, Обновить, Удалить*.
3. При добавлении необходимо выбрать таблицу, которую добавляем в модель из БД, нажать кнопку **Готово**. В примере ниже добавляем новую таблицу Phone.

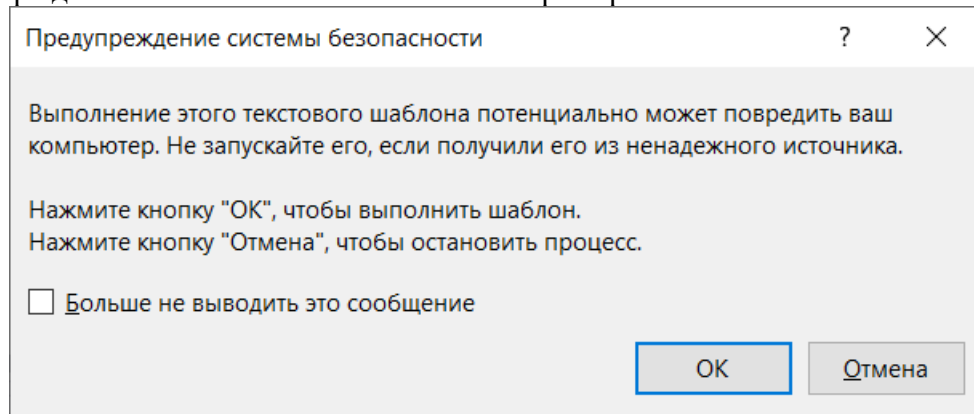


4. Обновление происходит автоматически. Во вкладке **Обновить** отображаются существующие таблицы, которые будут обновлены, если в них были внесены изменения. Обновлению подлежит только добавление новых столбцов из БД в модель данных. Если вы из БД убрали столбец, то из модели данных удалить его нужно вручную. Если в БД изменили тип столбца, то в свойствах поля тип данных нужно изменить вручную.

5. При удалении необходимо выбрать таблицу, которую удаляем, нажать кнопку **Готово**. Удаляются из модели данных те таблицы, которые были удалены из БД. Если таблица из модели данных не удалась, удалите ее вручную. В примере ниже удаляем таблицу Phone.



5. При внесении изменений в модель данных при сохранении запускаются два шаблона, которые вносят изменения в соответствующие программные элементы. Подтверждаем выполнение этих шаблонов. Пример окна см. ниже:



Замечание: при изменении модели данных класс *ModelDataBase.Context.cs* принимает свою первоначальную структуру. А как вы должны помнить, в него мы добавили поле и метод для получения статической ссылки на контекст данных. Эту ссылку нужно будет заново писать в данный элемент, поэтому заранее сохраните часть кода в файле для более быстрого восстановления содержимого этого класса.

Пример изменения класса *ModelDataBase.Context.cs* см. ниже.

```
public partial class AbonentsEntities : DbContext
{
    ссылка: 1
    public AbonentsEntities()
    : base("name=AbonentsEntities") { }
    Ссылка: 0
    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        throw new UnintentionalCodeFirstException();
    }
    Ссылка: 0
    public virtual DbSet<sysdiagram> sysdiagrams { get; set; }
    Ссылка: 3
    public virtual DbSet<Abonent> Abonents { get; set; }
    //Добавляем статичную ссылку на контекст
    private static AbonentsEntities context;
    //Добавляем метод получения ссылки на контекст
    Ссылка: 2
    public static AbonentsEntities GetContext()
    {
        if (context == null)
            context = new AbonentsEntities();
        return context;
    }
}
```

Практическая работа №17

Создание приложения с БД.

1. Разработать интерфейс для доступа и управления однотабличной базой данных.
2. Создать меню.
3. Реализовать кнопки для функций Добавить, Редактировать, Удалить.
4. Реализовать функции Добавить, Изменить с помощью окна – бланка.
5. Реализовать функцию поиск и замена информации.
6. Реализовать фильтр.
7. Использовать исключения.

Варианты заданий

Вариант 1.

Учет рейтинга теннисистов за 5 лет. Каждая запись содержит поля: фамилия, имя, отчество спортсмена, пол, год рождения, фамилия, имя, отчество тренера, названия стран и пять полей с рейтингом.

Вариант 2.

Сведения о рейсах Аэрофлота. База данных должна содержать следующую информацию: номер рейса, пункт назначения, время вылета, время прибытия, количество свободных мест, тип самолета и его вместимость.

Вариант 3.

Сведения об ассортименте обуви в магазине. База данных должна содержать следующую информацию: артикул, наименование обуви, количество пар, стоимость одной пары, имеющиеся размеры, название фабрики и срок поставки обуви в магазин.

Вариант 4.

Сведения о нападающих команд “Спартак” и “Динамо”. База данных должна содержать следующую информацию: фамилию, имя, отчество, название команды, дату приема в команду, число заброшенных шайб, количество голевых передач, штрафное время и количество сыгранных матчей.

Вариант 5.

Сведения о выборе дисциплины студентом. База данных должна содержать следующую информацию: фамилию, имя, отчество студента, номер зачетной книжки и сведения о том, живет ли студент в общежитии, индекс группы, а также пять дисциплин (1 – желает изучать, 0 – не желает).

Вариант 6.

Журнал регистрации расходов в бухгалтерии. База данных должна содержать следующую информацию: номер пункта, дату перечисления, название организации-получателя, ее адрес и сведения о том, является ли организация коммерческой, а также вид затрат перечисления и общую сумму перечисления.

Вариант 7.

Учет оптовых продаж. База данных должна содержать следующую информацию: наименование товара, цену единицы товара и дату его поступления, номер партии, размер партии, название фирмы-покупателя, размер проданной партии, цену единицы товара и дату продажи.

Вариант 8.

Учет лекарств в аптеке. База данных должна содержать следующую информацию: наименование лекарства, стоимость одной единицы, количество единиц, дату изготовления, срок годности, а также название фабрики, где производится данное лекарство, ее адрес.

Вариант 9.

Сведения о ветеранах спорта. Ассоциация ветеранов спорта проводит Всероссийские соревнования ветеранов. Для организации соревнований составляются списки участников, которые используются для размещения спортсменов в гостиницах. Для каждого спортсмена указывается гостиница, номер комнаты и количество мест в комнате. Для нужд самой ассоциации ветеранов спорта необходимо хранить информацию следующего вида: фамилию, имя, отчество спортсмена, возрастную группу, название города и вид спорта.

Вариант 10.

Данные для простой складской системы. База данных должна содержать следующую информацию: уникальный номер поставщика, фамилию, имя, отчество поставщика, название города местонахождения поставщика, а также детали, ее название, цвет, вес и название города хранения деталей этого типа.

Вариант 11.

Сведения об участниках конкурса бальных танцев. База данных должна содержать следующую информацию: фамилию, имя, отчество участника, город, фамилию тренера, оценки за каждый танец.

Вариант 12.

Сведения об успеваемости студентов. База данных должна содержать следующую информацию: фамилию, имя, отчество студента, номер группы, в которой обучается студент, название учебной дисциплины, номер задания, коэффициент сложности, оценку данного студента по данной дисциплине за данное задание от 0 до 1 (как доля сделанной работы).

Вариант 13.

Сведения о месячной зарплате рабочих. База данных должна содержать следующую информацию: фамилию, имя, отчество рабочего, название цеха, в котором он работает, дату поступления на работу. По заработной плате необходимо хранить информацию о ее размере, стаже работника, его разряде и должности.

Вариант 14.

Учет изделий, собранных в цехе за неделю. База данных должна содержать следующую информацию: фамилию, имя, отчество сборщика, количество изготовленных изделий за каждый день недели раздельно, название цеха, а также тип изделия и его стоимость.

Вариант 15.

Учет изделий категорий А, В, С, собранных рабочим цеха за месяц. База данных должна содержать следующую информацию: фамилию, имя, отчество рабочего, название цеха, количество изделий по категориям, количество рабочих в цехе и фамилию начальника цеха.

Вариант 16.

Сведения об абонентах АТС. База данных должна содержать следующую информацию: фамилию, имя, отчество владельца телефона, год установки телефона, номер телефона, тип установки телефона (спаренный или нет), льготу (процентную скидку при оплате).

Вариант 17.

Сведения об ассортименте игрушек в магазине. База данных должна содержать следующую информацию: название игрушки, ее цену, количество, возрастную категорию

детей, для которых она предназначена, а также название фабрики и города, где изготовлена игрушка.

Вариант 18.

Результаты сессии на первом курсе кафедры ВТ. База данных должна содержать следующую информацию: индекс группы, фамилию, имя, отчество студента, пол студента, семейное положение и оценки по пяти экзаменам.

Создание SQL запросов

В большинстве случаев разработчики смогут создать эффективные запросы с помощью методов и операторов *LINQ*. Также часть запросов можно и нужно реализовывать на серверной части информационной системы с помощью представлений, функций и процедур.

Однако в *Entity Framework* можно использовать также прямое выполнение sql-запросов.

Для осуществления прямых sql-запросов к базе данных можно воспользоваться свойством *Database*, которое имеется у класса контекста данных. Данное свойство позволяет получать информацию о базе данных, подключении и осуществлять запросы к БД.

Непосредственно для создания запроса нам надо использовать метод *SqlQuery*, который принимает в качестве параметра sql-выражение.

Для примера возьмем базу данных, созданную как пример в данной лекции, которая описывается следующей моделью:

```
public partial class Abonent
{
    Ссылка: 1
    public int Id { get; set; }
    Ссылка: 5
    public string Fio { get; set; }
    Ссылка: 3
    public string Gender { get; set; }
    Ссылка: 3
    public Nullable<System.DateTime> Age { get; set; }
    Ссылка: 3
    public string Inn { get; set; }
    Ссылка: 3
    public string Phone { get; set; }
    Ссылка: 0
    public byte[] Photo { get; set; }
}
```

Создадим запрос на извлечение БД:

```
var gen = db.Database.SqlQuery<Abonent>("SELECT * FROM Abonent WHERE Gender = 'Муж' ");
DataGrid1.ItemsSource = gen.ToList();
```

Другая версия метода *SqlQuery* позволяет создавать параметрические запросы. Для задания параметра используется класс *SqlParameter* из пространства имен *System.Data.SqlClient*, который позволяет задать параметр, который затем передается в запрос sql.

Например, продемонстрированный выше запрос сделаем параметрическим:

```
//Задаем параметр
SqlParameter param1 = new SqlParameter();
param1.ParameterName = "@gender";
param1.Value = "Муж"; //Можно задать из TextBox
//Выполняем и отображаем запрос
var gen = db.Database.SqlQuery<Abonent>("SELECT * FROM Abonent WHERE Gender = @gender", param1);
DataGrid1.ItemsSource = gen.ToList();
```

Метод *SqlQuery* осуществляет выборку из БД, но кроме выборки нам, возможно, придется удалять, обновлять уже существующие или вставлять новые записи. Для этой цели применяется метод *ExecuteSqlCommand*, который возвращает количество затронутых командой строк:

```
// вставка
int numberIns = db.Database.ExecuteSqlCommand("INSERT INTO Abonent (Fio, Gender) VALUES ('Петров', 'Муж')");
// обновление
int numberUpd = db.Database.ExecuteSqlCommand("UPDATE Abonent SET Fio='Козлов' WHERE Id=3");
// удаление
int numberDel = db.Database.ExecuteSqlCommand("DELETE FROM Abonent WHERE Id=3");
```

Использование LINQ to Entities

Хотя при работе с базой данных мы оперируем запросами *LINQ*, но база данных понимает только запросы на языке *SQL*. Поэтому между *LINQ to Entities* и базой данных есть проводники, который позволяет им взаимодействовать. Однако разработчику не надо напрямую взаимодействовать с этими объектами, фреймворк все сделает за него. Задача же разработчика сводится в основном к написанию запросов к базе данных с помощью *LINQ*.

Выборка из базы данных (фильтр)

Для выборки применяется оператор *Select* или метод *Where*. Выберем из таблицы абонент все записи с фамилией на “И”:

```
//Применяем оператор для выборки
var fioA1 = from p in db.Abonents
            where p.Fio.StartsWith("И")
            && p.Fio.EndsWith("И")
            select p;
DataGridView1.ItemsSource = fioA1.ToList();

//Обратимся к элементам полученной таблицы
foreach (var p in fioA1)
{
    MessageBox.Show(p.Fio);
}

//Применяем метод для выборки
var fioA2 = db.Abonents.Where(p => p.Fio.StartsWith("И"));
fioA2 = db.Abonents.Where(p => p.Fio.StartsWith("И")).Where(p => p.Fio.EndsWith("И"));
//Отображаем полученную таблицу
DataGridView1.ItemsSource = fioA2.ToList();
```

Для сравнения могут применяться стандартные операции сравнения, а также функции применимые для полей с различным типом данных.

Оператор, функция сравнения	Описание
==, >=, <=, >, <	Стандартные операции сравнения применимые к большинства типам данных: int, double, string, DateTime
Bool StartsWith(string value)	Определяет, совпадает ли начало данной строки с указанной строкой.
Bool EndWith(string value)	Определяет, совпадает ли конец данной строки с указанной строкой.
Bool Contains(string value)	Определяет, совпадает ли данная строка значение указанной подстроки.

В качестве альтернативы мы можем использовать методы Linq *First()/FirstOrDefault()* или *Last()/LastOrDefault()*. Они получают первый элемент

выборки, который соответствует определенному условию. Использование метода **FirstOrDefault()** является более гибким, так как если выборка пуста, то он вернет значение **null**. А метод **First()** в той же ситуации выбросит ошибку.

Проекция базы данных

Проекция заключается в том, что осуществляется выборка не всей строки таблицы, а только ее определенных полей.

```
//Применяем оператор выборки для проекции
var fioA3 = from p in db.Abonents
            where p.Fio.StartsWith("И")
            select p.Fio;
DataGrid1.ItemsSource = fioA3.ToList();

//Применяем оператор выборки для проекции
var fioA4 = from p in db.Abonents
            where p.Fio.StartsWith("И")
            select new {
                Fio = p.Fio,
                Gender = p.Gender
            };
DataGrid1.ItemsSource = fioA4.ToList();

//Применяем метод проекции 1 поля
var fioA5 = db.Abonents.Select(p => p.Fio);
```

Сортировка базы данных

Для упорядочивания полученных из бд данных по возрастанию служит метод **OrderBy** или оператор **orderby**. **Descending** применяется для сортировки по убыванию.

```
//Применяем операторы сортировки при выборке
var fioA6 = from p in db.Abonents
            orderby p.Fio
            select p;
fioA6 = from p in db.Abonents
        orderby p.Fio, p.Age descending
        select p;
DataGrid1.ItemsSource = fioA5.ToList();

//Применяем методы для сортировки
var fioA7 = db.Abonents.OrderBy(p => p.Fio);
var fioA8 = db.Abonents.OrderByDescending(p => p.Fio);
```

Агрегатные операции

Linq to Entities поддерживает обращение к встроенным функциям SQL через специальные методы **Count**, **Sum** и т.д.

Количество элементов в выборке

```
//Определяем кол-во
var fioA9 = from p in db.Abonents
            select new {Count = db.Abonents.Count() };
int Count = db.Abonents.Count();

//Определяем кол-во по группам
//g.Key - это поле группировки
var fioA10 = from p in db.Abonents
              group p by p.Gender into g
              select new { Gender = g.Key, Count = g.Count() };
DataGrid1.ItemsSource = fioA10.ToList();
```

Среднее значение

```
//Определяем среднее в столбце
var fioA11 = from p in db.Abonents
              select new { Average = db.Abonents.Average(g => g.Id) };
double Average = db.Abonents.Average(p => p.Id);

//Определяем среднее по группам
//g.Key - это поле группировки
var fioA12 = from p in db.Abonents
              group p by p.Gender into g
              select new { Gender = g.Key, Average = g.Average(p => p.Id) };
DataGrid1.ItemsSource = fioA12.ToList();
```

Максимальное, минимальное значение Max/Min

```
//Определяем Максимальное в столбце
var fioA14 = from p in db.Abonents
              select new { Average = db.Abonents.Max(g => g.Id) };
int max = db.Abonents.Max(p => p.Id);

//Определяем максимальное по группам
//g.Key - это поле группировки
var fioA15 = from p in db.Abonents
              group p by p.Gender into g
              select new { Gender = g.Key, Max = g.Max(p => p.Id) };
DataGrid1.ItemsSource = fioA12.ToList();
```

Сумма значений

```
//Определяем сумму в столбце
var fioA16 = from p in db.Abonents
              select new { Average = db.Abonents.Sum(g => g.Id) };
int sum = db.Abonents.Where(p => p.Gender.Contains("Муж")).Sum(p => p.Id);

//Определяем сумму по группам
//g.Key - это поле группировки
var fioA17 = from p in db.Abonents
              where p.Gender.Contains("и")
              group p by p.Gender into g
              select new { Gender = g.Key, Max = g.Sum(p => p.Id) };
DataGrid1.ItemsSource = fioA17.ToList();
```

Использование представлений, функций и процедур для создания запросов

Так как в качестве БД используется SQL Server, то, по сути, мы создаем клиент – серверное приложение. Основными задачами такого приложения является минимизация передачи данных между клиентом и сервером, поэтому основная обработка информации выносится на серверную часть, а в клиенте отображается только готовый вариант.

При реализации такой концепции очень активно используются представления, функции и процедуры, которые обрабатывают информация на серверной части, а клиент получает только готовый вариант.

Представления

Представления как вы уже знаете представляет виртуальную таблицу, которая актуализируется при обращении к ней.

Работа с ней осуществляется как с обычной таблицей:

1. Создаем в БД представление с нужными нам данными.
2. Импортируем представление в модель данных, как это осуществляется рассмотрено ранее.
3. Осуществляем работу с представлением как с обычной таблицей.

Функции

Entity Framework позволяют работать с функциями возвращающими табличное значение. Функция позволяет получить набор данных в табличном виде, с которым можно работать далее, как с обычной таблицей.

Рассмотрим пример получения списка абонентов заданного пола:

1. Создаем в БД функцию с заданными характеристиками.

```
ALTER FUNCTION [dbo].[SpisokGender]
(
    @Gender nvarchar(3)
)
RETURNS TABLE
AS
RETURN
(
    SELECT Id, Fio, Gender, Age, Inn, Phone
    FROM Abonent
    WHERE (Gender LIKE @Gender)
)
```

2. Импортируем функцию в модель данных, как это осуществляется рассмотрено ранее. При импорте автоматически создается класс, ассоциированный со структурой табличных данных.

```
public partial class SpisokGender_Result
{
    Ссылка: 0
    public int Id { get; set; }
    Ссылка: 0
    public string Fio { get; set; }
    Ссылка: 0
    public string Gender { get; set; }
    Ссылка: 0
    public Nullable<System.DateTime> Age { get; set; }
    Ссылка: 0
    public string Inn { get; set; }
    Ссылка: 0
    public string Phone { get; set; }
}
```

А также в файл описания контекста данных добавляется описание этой функции.

```
[DbFunction("AbonentsEntities", "SpisokGender")]
Ссылка: 1
public virtual IQueryable<SpisokGender_Result> SpisokGender(string gender)
{
    var genderParameter = gender != null ?
        new ObjectParameter("Gender", gender) :
        new ObjectParameter("Gender", typeof(string));

    return ((IObjectContextAdapter)this).ObjectContext.CreateQuery<SpisokGender_Result>
        ("[AbonentsEntities].[SpisokGender](@Gender)", genderParameter);
}
```

3. С помощью функции получаем данные в виде таблицы и далее работаем как с обычной таблицей.

```
var spisok = db.SpisokGender("Муж");
dataGrid2.ItemsSource = spisok;
```

Процедуры

Entity Framework позволяют работать с процедурами. Процедуры позволяют получить набор данных в табличном виде, с которым можно работать далее, как с обычной таблицей.

Рассмотрим пример получения списка абонентов заданного пола:

1. Создаем в БД процедуру с заданными характеристиками.

```
ALTER PROCEDURE [dbo].[SpisokAbonentovPoGemder]
    @Gender nvarchar(3)
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for procedure here
    SELECT Id, Fio, Gender, Age, Inn, Phone
    FROM Abonent
    WHERE (Gender LIKE @Gender)
    ORDER BY Fio
END
```

2. Импортируем процедуру в модель данных, как это осуществляется рассмотрено ранее. При импорте автоматически создается класс, ассоциированный со структурой табличных данных.

```
public partial class SpisokAbonentovPoGemder_Result
{
    Ссылка: 0
    public int Id { get; set; }
    Ссылка: 0
    public string Fio { get; set; }
    Ссылка: 0
    public string Gender { get; set; }
    Ссылка: 0
    public Nullable<System.DateTime> Age { get; set; }
    Ссылка: 0
    public string Inn { get; set; }
    Ссылка: 0
    public string Phone { get; set; }
}
```

А также в файл описания контекста данных добавляется описание этой процедуры.

```
public virtual ObjectResult<SpisokAbonentovPoGemder_Result> SpisokAbonentovPoGemder(string gender)
{
    var genderParameter = gender != null ?
        new ObjectParameter("Gender", gender) :
        new ObjectParameter("Gender", typeof(string));

    return ((IObjectContextAdapter)this).ObjectContext.ExecuteFunction<SpisokAbonentovPoGemder_Result>
        ("SpisokAbonentovPoGemder", genderParameter);
}
```

3. С помощью процедуры получаем данные в виде таблицы и далее работаем как с обычной таблицей.

```
var spisok = db.SpisokAbonentovPoGemder("Муж");
dataGridView2.ItemsSource = spisok;
```

Рассмотрим запрос на обновление ИНН у заданного абонента:

1. Создаем в БД процедуру с заданными характеристиками.

```
ALTER PROCEDURE [dbo].[ChangeInn]
@Fio nvarchar(50), @Inn nvarchar(12), @text nvarchar(max) OUTPUT
AS
BEGIN
    SET NOCOUNT ON;
    -- Insert statements for procedure here
    IF LEN(@Inn) <> 12 or ISNUMERIC(@Inn) = 0
        SET @text = 'Введите ИНН'
    ELSE BEGIN
        UPDATE Abonent set Inn = @Inn
        WHERE Fio = @Fio
        SET @text = 'Обновлено ' + CAST(@@ROWCOUNT as nvarchar(max)) + ' строк'
    END
END
```

2. Импортируем процедуру в модель данных, как это осуществляется рассмотрено ранее. При импорте в файл описания контекста данных добавляется описание этой процедуры.

```
public virtual int ChangeInn(string fio, string inn, ObjectParameter text)
{
    var fioParameter = fio != null ?
        new ObjectParameter("Fio", fio) :
        new ObjectParameter("Fio", typeof(string));

    var innParameter = inn != null ?
        new ObjectParameter("Inn", inn) :
        new ObjectParameter("Inn", typeof(string));

    return ((IObjectContextAdapter)this).ObjectContext.ExecuteFunction("ChangeInn",
        fioParameter, innParameter, text);
}
```

Обратите внимание, параметр *«text»* в программе возвращается не в виде строки как в процедуре БД, а как элемент типа *ObjectParameter*, что далее нужно учитывать при использовании данной функции.

3. С помощью процедуры обновляем информацию.

```
//Создаем элемент для параметра "text", в конструкторе указываем название
//параметра в процедуре БД и его ассоциированный тип
ObjectParameter param = new ObjectParameter("text", typeof(string));
//Используем процедуру
db.ChangeInn("Петров Иван Сергеевич", "125444443", param);
//Перезагружаем значения сущностей
db.ChangeTracker.Entries().ToList().ForEach(p => p.Reload());
DataGrid1.Items.Refresh();
//Используем полученный выходной параметр
MessageBox.Show("param.Value");
```

Выбор подходов для реализации запросов

Как мы видим для реализации запросов есть много подходов, соответственно возникает вопрос какой из подходов лучше. Тут нет однозначного ответа, главный критерий минимизация передачи данных между клиентов и сервером.

Например, у нас уже есть загруженная информация в виде какой-либо таблицы и нам нужно в данной таблице отобразить часть информации по какому-либо критерию, тут однозначно лучше *LINQ*. В этом случае передача данных не осуществляется, а мы работаем на уровне клиента с уже существующей информацией.

Например, из БД нужно собрать информацию из нескольких таблиц в более удобный вид для дальнейшего использования и отображения, тут лучше создать представление и далее клиенте работать уже с представлением. Основная работа по сборке информации ложится на серверную часть.

Представления как вы знаете не позволяют делать выборки данных по заданным критериям, поэтому если из полученного представления нужно отображать не всю информацию, а часть по какому-либо критерию, нужно создать процедуру с параметрами на базе представления и получать нужную информацию по заданным параметрам.

Настройка пути к базе данных

При создании подключения к БД, в свойства подключения записываются параметры подключения к БД: имя SQL сервера, логин, пароль. При переносе БД на другой сервер эти параметры могут отличаться, чтобы программа могла работать эти параметры нужно изменить.

Для изменения параметров подключения в окне *Обозреватель решений* откройте элемент *App.Config* и задайте новые параметры подключения к БД в разделе *connectionStrings*.

```
<connectionStrings>
|   <add name="AbonentsEntities" connectionString="metadata=res://*/ModelDataBase.csdl
|       |res://*/ModelDataBase.ssdl|res://*/ModelDataBase.msl; provider=System.Data.SqlClient;
|       provider connection string='data source=localhost\sqlexpress;initial catalog=Abonents;
|       user id=sa;password="На горшке сидел король";MultipleActiveResultSets=True;
|       App=EntityFramework'" providerName="System.Data.EntityClient" /></connectionStrings>
```

Практическая работа № 18

Создание запросов к БД

1. Разработать интерфейс для доступа и управления однотабличной базой данных.
2. Создать меню.
3. Использовать кнопки для функций Добавить, Изменить, Просмотр, Удалить.
4. Реализовать функции добавить, изменить, просмотр с помощью окна – бланка.
5. Реализовать функцию удалить текущую запись.
6. SQL Запросы к базе данных, выбор параметров запроса:
 - a. Запрос на выборку LINQ (2-5 штуки)
 - b. Запрос на обновление (1-2 штуки)
 - c. Запрос на удаление (1-2 штуки)
7. Исключения
8. Заполненная БД 15 -20 записей

Вариант 1.

Учет изделий, собранных в цехе за неделю. База данных должна содержать следующую информацию: фамилию, имя, отчество сборщика, количество изготовленных изделий за каждый день недели раздельно, название цеха, а также тип изделия и его стоимость.

Вариант 2.

Учет изделий категорий А, В, С, собранных рабочим цеха за месяц. База данных должна содержать следующую информацию: фамилию, имя, отчество рабочего, название цеха, количество изделий по категориям, количество рабочих в цехе и фамилию начальника цеха.

Вариант 3.

Сведения о выборе дисциплины студентом. База данных должна содержать следующую информацию: фамилию, имя, отчество студента, номер зачетной книжки и сведения о том, живет ли студент в общежитии, индекс группы, а также пять дисциплин (1 – желает изучать, 0 – не желает).

Вариант 4.

Журнал регистрации расходов в бухгалтерии. База данных должна содержать следующую информацию: номер пункта, дату перечисления, название организации-получателя, ее адрес и сведения о том, является ли организация коммерческой, а также вид затрат перечисления и общую сумму перечисления.

Вариант 5.

Учет оптовых продаж. База данных должна содержать следующую информацию: наименование товара, цену единицы товара и дату его поступления, номер партии, размер партии, названии фирмы-покупателя, размер проданной партии, цену единицы товара и дату продажи.

Вариант 6.

Учет лекарств в аптеке. База данных должна содержать следующую информацию: наименование лекарства, стоимость одной единицы, количество единиц, дату

изготовления, срок годности, а также название фабрики, где производится данное лекарство, ее адрес.

Вариант 7.

Учет рейтинга теннисистов за 5 лет. Каждая запись содержит поля: фамилия, имя, отчество спортсмена, пол, год рождения, фамилия, имя, отчество тренера, названия стран и пять полей с рейтингом.

Вариант 8.

Сведения о рейсах Аэрофлота. База данных должна содержать следующую информацию: номер рейса, пункт назначения, время вылета, время прибытия, количество свободных мест, тип самолета и его вместимость.

Вариант 9.

Сведения об ассортименте обуви в магазине. База данных должна содержать следующую информацию: артикул, наименование обуви, количество пар, стоимость одной пары, имеющиеся размеры, название фабрики и срок поставки обуви в магазин.

Вариант 10.

Сведения о нападающих команд “Спартак” и “Динамо”. База данных должна содержать следующую информацию: фамилию, имя, отчество, название команды, дату приема в команду, число заброшенных шайб, количество голевых передач, штрафное время и количество сыгранных матчей.

Вариант 11.

Сведения о ветеранах спорта. Ассоциация ветеранов спорта проводит Всероссийские соревнования ветеранов. Для организации соревнований составляются списки участников, которые используются для размещения спортсменов в гостиницах. Для каждого спортсмена указывается гостиница, номер комнаты и количество мест в комнате. Для нужд самой ассоциации ветеранов спорта необходимо хранить информацию следующего вида: фамилию, имя, отчество спортсмена, возрастную группу, название города и вид спорта.

Вариант 12.

Сведения об абонентах АТС. База данных должна содержать следующую информацию: фамилию, имя, отчество владельца телефона, год установки телефона, номер телефона, тип установки телефона (спаренный или нет), льготу (процентную скидку при оплате).

Вариант 13.

Сведения об ассортименте игрушек в магазине. База данных должна содержать следующую информацию: название игрушки, ее цену, количество, возрастную категорию детей, для которых она предназначена, а также название фабрики и города, где изготовлена игрушка.

Вариант 14.

Результаты сессии на первом курсе кафедры ВТ. База данных должна содержать следующую информацию: индекс группы, фамилию, имя, отчество студента, пол студента, семейное положение и оценки по пяти экзаменам.

Вариант 15.

Данные для простой складской системы. База данных должна содержать следующую информацию: уникальный номер поставщика, фамилию, имя, отчество поставщика, название города местонахождения поставщика, а также детали, ее название, цвет, вес и название города хранения деталей этого типа.

Вариант 16.

Сведения об участниках конкурса бальных танцев. База данных должна содержать следующую информацию: фамилию, имя, отчество участника, город, фамилию тренера, оценки за каждый танец.

Вариант 17.

Сведения об успеваемости студентов. База данных должна содержать следующую информацию: фамилию, имя, отчество студента, номер группы, в которой обучается студент, название учебной дисциплины, номер задания, коэффициент сложности, оценку данного студента по данной дисциплине за данное задание от 0 до 1 (как доля сделанной работы).

Вариант 18.

Сведения о месячной зарплате рабочих. База данных должна содержать следующую информацию: фамилию, имя, отчество рабочего, название цеха, в котором он работает, дату поступления на работу. По заработной плате необходимо хранить информацию о ее размере, стаже работника, его разряде и должности.

Создание формы авторизации

Постановка задачи

Рассмотрим последовательность действий и код программы для создания формы авторизации по следующим критериям.

При запуске приложения окно входа – первое, что видит пользователь. На ней пользователю предлагается ввести свой логин и пароль или есть возможность перейти на главное окно в роли гостя.

Только после удачной авторизации пользователь получает доступ к остальным модулям системы:

- авторизованный клиент может просмотреть информация в главном окне;
- менеджер может просмотреть информация в главном окне;
- администратор может добавлять/редактировать/удалять информацию.

Реализуйте необходимые интерфейсы для всех пользователей системы. После входа в любую учетную запись должна быть реализована возможность выхода на главный экран – окно входа. При переходе в любую учетную запись в интерфейсе (правый верхний угол) должны отображаться ФИО пользователя.

После первой попытки неуспешной авторизации система выдает сообщение о неуспешной авторизации, а затем помимо ввода логина и пароля просит ввести captcha, состоящую из 4 символов (цифры и буквы латинского алфавита) и графического шума.

CAPTCHA - должна содержать минимум 4 символа (буква или цифра), которые выведены не в одной линии. Символы должны быть либо перечеркнуты, либо наложены друг на друга.

После попытки неудачной авторизации с вводом captcha, система блокирует возможность входа на 10 секунд.

Проектирование БД

Хранить информацию о пользователях будем в БД. Соответственно создадим две таблицы о пользователях и их ролях см. рисунок 1.

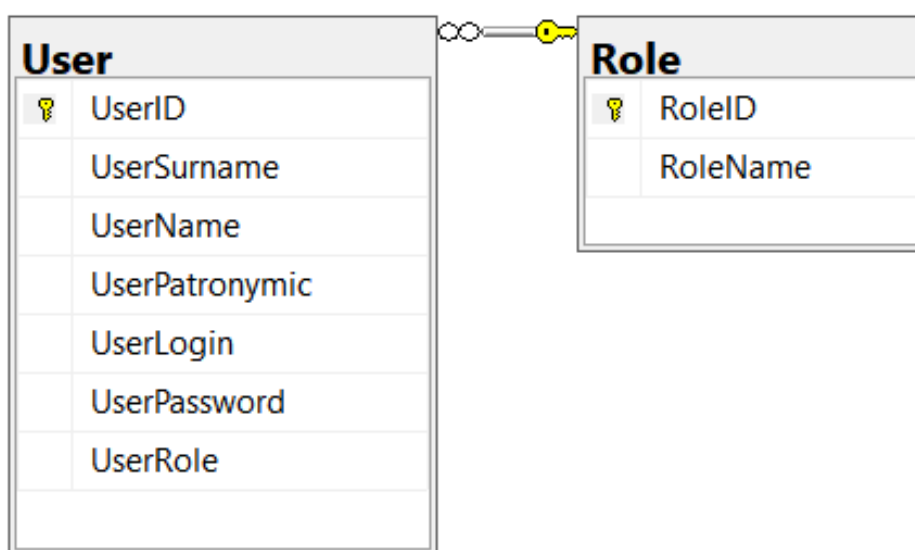


Рисунок 1 – БД авторизация

Рассмотрим класс сущности пользователи для разработанной БД. Кроме свойств, которые ассоциированы с полями таблицы БД добавилось еще одно навигационное свойство **Role**, которое ссылается на связанную запись, в сущности, **Role**. Использовать это навигационное свойство можно для обращения к связанной записи, что и будет сделано далее для получения названия роли пользователя.

```
public partial class User
{
    Ссылка: 0
    public int UserID { get; set; }
    Ссылка: 1
    public string UserSurname { get; set; }
    Ссылка: 1
    public string UserName { get; set; }
    Ссылка: 1
    public string UserPatronymic { get; set; }
    Ссылка: 1
    public string UserLogin { get; set; }
    Ссылка: 1
    public string UserPassword { get; set; }
    Ссылка: 0
    public int UserRole { get; set; }

    Ссылка: 1
    public virtual Role Role { get; set; }
}
```

Алгоритм создания формы авторизации

1. Создаем форму авторизации см. рисунок.

```

        Title="Авторизация" FontSize="20" ResizeMode="NoResize" SizeToContent="Height"
        Width="300" Activated="Window_Activated" >
<StackPanel x:Name="stackPanel">
    <Label>Пользователь</Label>
    <TextBox x:Name="tbLogin"></TextBox>
    <Label>Пароль</Label>
    <PasswordBox x:Name="tbPas" PasswordChar="*"></PasswordBox>
    <Grid x:Name="grid" Visibility="Collapsed">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"></RowDefinition>
            <RowDefinition ></RowDefinition>
        </Grid.RowDefinitions>
        <TextBlock x:Name="txtCaprcha" Background="Gray" FontSize="28" Grid.Row="0"
            TextAlignment="Center"></TextBlock>
        <Line x:Name="line" Grid.Row="0" Stroke="Black"></Line>
        <TextBox x:Name="tbCaptcha" Grid.Row="1"></TextBox>
    </Grid>
    <Button Content="Войти" IsDefault="True" Click="btnEnter_Click"></Button>
    <Button Content="Отмена" IsCancel="True" Click="btnEsc_Click"/>
    <Button Content="Войти как гость" Click="btnGuest_Click"></Button>
</StackPanel>

```

2. Создаем глобальный статический класс для передачи данных между формами. В этом классе опишем элемент *Доступ* для передачи уровня доступа текущего пользователя, элемент *Логин* для отслеживания авторизации, элементы *Фамилия* и *Имя* для отображения в заголовке окна см. пример.

```

public static class Data
{
    //Признак авторизации
    public static bool Login = false;
    public static string Familia;
    public static string Name;
    public static string Otchestvo;
    //Права доступа
    public static string Right;
}

```

3. В главном окне при инициализации основной формы, до ее показа открываем форму авторизации.

```
private void Window_Initialized(object sender, EventArgs e)
{
    //Открываем окно авторизации
    Authorization authorization = new Authorization();
    authorization.ShowDialog();
    //При отказе от авторизации выходим из программы
    if (Data.Login == false) Close();
    //Устанавливаем права доступа
    if (Data.Right == "Администратор") ;
    else
    {
        //Можно запретить какие либо действия
        //btnDelete.IsEnabled = false;
    }
    //Выводим информацию о пользователе в заголовок окна
    mainWindow1.Title = Data.Familia + " " +
        Data.Name + " " + Data.Otchestvo + " - "+Data.Right;
```

4. На форме *Авторизация* описываем глобальные переменные.

```
//Таймер
DispatcherTimer _timer;
//Счетчик попыток входа
int _countLogin = 1;
TradeEntities _db = TradeEntities.GetContext();
```

5. На форме *Авторизация* описываем метод формирования капчи.

```
void GetCaptcha()
{
    string masChar = "QWERTYUIOPLKJHGFDSAZXCVBNMmbvcxzasdfghjk" +
        "lpoiuytrewq1234567890";
    string captcha = "";
    Random rnd = new Random();
    for (int i = 1; i <= 6; i++)
    {
        captcha = captcha + masChar[rnd.Next(0, masChar.Length)];
    }
    grid.Visibility = Visibility.Visible;
    txtCaprcha.Text = captcha;
    tbCaptcha.Text = null;
    txtCaprcha.LayoutTransform = new RotateTransform(rnd.Next(-15, 15));
    line.X1 = 10;
    line.Y1 = rnd.Next(10, 40);
    line.X2 = 280;
    line.Y2 = rnd.Next(10, 40);
}
```

6. На форме *Авторизация* пишем код события загрузка формы.

```
private void Window_Activated(object sender, EventArgs e)
{
    tbLogin.Focus();
    Data.Login = false;
    _timer = new DispatcherTimer();
    _timer.Interval = new TimeSpan(0, 0, 10);
    _timer.Tick += new EventHandler(timer_Tick);
}
```

7. На форме *Авторизация* пишем код события таймера.

```
private void timer_Tick(object sender, EventArgs e)
{
    stackPanel.IsEnabled = true;
}
```

8. На форме *Авторизация* пишем код кнопки *Войти*.

```
private void btnEnter_Click(object sender, RoutedEventArgs e)
{
    //Ищем запись с заданным логином и паролем через LINQ
    var user = from p in _db.User
                where p.UserLogin==tbLogin.Text &&
                       p.UserPassword==tbPas.Password
                select p;
    //Если запись найдена
    if (user.Count() == 1 && txtCaprcha.Text == tbCaptcha.Text)
    {
        //Запоминаем информацию
        Data.Login = true;
        Data.Familia = user.First().UserSurname;
        Data.Name = user.First().UserName;
        Data.Otchestvo = user.First().UserPatronymic;
        Data.Right = user.First().Role.RoleName;
        Close();
    }
    else
    {
        if (user.Count() == 1)
        {
            MessageBox.Show("Капча неверна! Повторите ввод.");
        }
        else
        {
            MessageBox.Show("Логин, пароль неверны! Повторите ввод.");
        }
    }
}
```

```
        GetCaptcha();

        if (_countLogin >= 2)
        {
            stackPanel.IsEnabled = false;
            _timer.Start();
        }
        _countLogin++;
        tbLogin.Focus();
    };
}
```

9. На форме *Авторизация* пишем код кнопки *Отмена*.

```
private void btnEsc_Click(object sender, RoutedEventArgs e)
{
    Close();
}
```

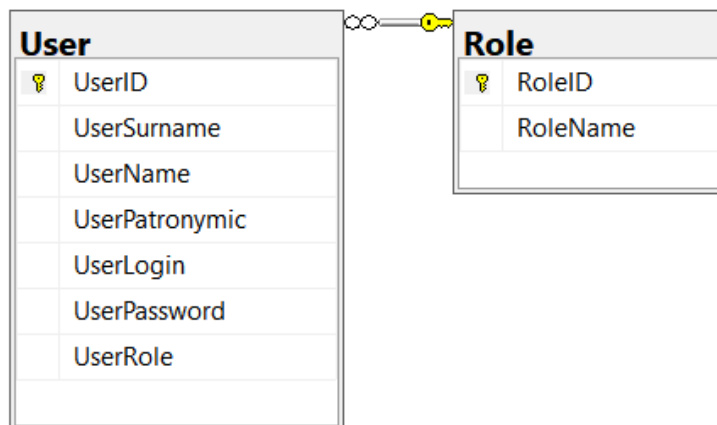
10. На форме *Авторизация* пишем код кнопки *Войти как гость*.

```
private void btnGuest_Click(object sender, RoutedEventArgs e)
{
    Data.Login = true;
    Data.Familia = "Гость";
    Data.Name = "";
    Data.Otchestvo = "";
    Data.Right = "Клиент";
    Close();
}
```

Практическая работа № 19

Создание формы авторизации

1. Открыть программу созданную в предыдущей практической работе.
2. Добавить таблицу Авторизация в БД.



3. Создать форму авторизации.

Создание отчетов

Экспорт в Word и Excel