ГОСУДАРСТВЕННОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Программное обеспечение информационных технологий»

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Методические рекомендации к лабораторным работам для студентов направлений подготовки 09.03.01 «Информатика и вычислительная техника», 09.03.04 «Программная инженерия» дневной формы обучения

Часть 2

Могилев 2018

УДК 621.01
ББК 36.4
О 87

Рекомендовано к изданию
учебно-методическим отделом
Белорусско-Российского университета

Одобрено кафедрой ПОИТ «20» мая 2018 г., протокол № 9

Составитель канд. техн. наук, доц. Н. Н. Горбатенко

Рецензент канд. техн. наук, доц. И. В. Лесковец

Методические рекомендации разработаны на основе рабочей программы по дисциплине «Объектно-ориентированное программирование» для студентов направлений подготовки 09.03.01 «Информатика и вычислительная техника», 09.03.04 «Программная инженерия» и предназначены для использования при проведении лабораторных работ.

Учебно-методическое издание

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Ответственный за выпуск К. В. Овсянников

Технический редактор С. Н. Красовская

Компьютерная верстка Н. П. Полевничая

Подписано в печать . Формат 60×84/16. Бумага офсетная. Гарнитура Таймс.
Печать трафаретная. Усл. печ. л. . Уч.-изд. л. . Тираж 31 экз. Заказ №

Издатель и полиграфическое исполнение:
Государственное учреждение высшего профессионального образования
«Белорусско-Российский университет».
Свидетельство о государственной регистрации издателя,
изготовителя, распространителя печатных изданий
№ 1/156 от 24.01.2014.
Пр. Мира, 43, 212000, Могилев.

Содержание

1 Лабораторная работа № 1. Обработка исключений

Цель работы: изучение способов обнаружения ошибок времени выполнения с помощью исключений; изучение основных принципов обработки исключений в языке C#; знакомство со структурным управлением исключениями.

1.1 Необходимые теоретические сведения

Обработка исключительных ситуаций [2, с. 89–95].

1.2 Индивидуальные задания

1 Описать класс, реализующий десятичный счетчик, который может увеличивать или уменьшать свое значение на единицу в заданном диапазоне. Предусмотреть инициализацию счетчика значениями по умолчанию и произвольными значениями. Счетчик имеет два метода: увеличения и уменьшения, – и свойство, позволяющее получить его текущее состояние. При выходе за границы диапазона выбрасываются исключения. Написать программу, демонстрирующую все разработанные элементы класса.

2 Описать класс, реализующий шестнадцатеричный счетчик, который может увеличивать или уменьшать свое значение на единицу в заданном диапазоне. Предусмотреть инициализацию счетчика значениями по умолчанию и произвольными значениями. Счетчик имеет два метода: увеличения и уменьшения; и свойство, позволяющее получить его текущее состояние. При выходе за границы диапазона выбрасываются исключения. Написать программу, демонстрирующую все разработанные элементы класса.

3 Описать класс, представляющий треугольник. Предусмотреть методы для создания объектов, перемещения на плоскости, изменения размеров и вращения на заданный угол. Описать свойства для получения состояния объекта. При невозможности построения треугольника выбрасывается исключение. Написать программу, демонстрирующую все разработанные элементы класса.

Контрольные вопросы

1 Что такое «исключительная ситуация (исключение)»?

2 Для чего предназначен механизм исключений в языке `C#`?

3 Какие операторы используются для обработки исключений?

4 Для чего используется ключевое слово `try`?

5 Проиллюстрируйте обработку исключительной ситуации фрагментом программы на языке `C#`.

6 Перечислите основные системные исключения.

7 Требуется ли соблюдение соответствия типа перехватываемого исключения типу исключения в обработчике `catch`?

8 Как реализуется перехват всех исключений?

9 Что происходит, если исключение не обработано?

10 Как выполняется оператор `catch` без параметров?
11 Объясните назначение свойства `StackTrace` класса `System.Exception`.
12 Может ли быть в программе блок `try` без блока `catch`?
13 Куда передается управление после обработки исключительной ситуации?

2 Лабораторная работа № 2. Создание простых приложений WPF

Цель работы: приобретение навыков разработки приложений на платформе `MS Windows Presentation Foundation (WPF)` с использованием языка разметки `XAML`.

2.1 Необходимые теоретические сведения

1 Создание простейшего `WPF`-приложения. Компоновка элементов управления с помощью сетки `Grid` [4, с. 434–438].
2 Использование эффектов анимации [4, с. 439–441].
3 Выбор компоновки интерфейса [5, с. 329–330].
4 Добавление строки меню [5, с. 330–331].
5 Добавление панели инструментов [5, с. 332–333].
6 Использование стандартных и нестандартных команд [5, с. 333–336].
7 Перевод команд из неактивного состояния в активное и обратно [5, с. 333–336].
8 Реагирование на события [5, с. 338–339].

2.2 Индивидуальные задания

Общая часть задания: написать `Windows`-приложение, заголовок главного окна которого содержит ФИО, группу и номер варианта. В программе должна быть предусмотрена обработка исключений, возникающих из-за ошибочного ввода пользователя.

1 Создать меню с командами `Input`, `Calc` и `Exit`.

При выборе команды `Input` открывается диалоговое окно, содержащее:

– три поля типа `TextBox` для ввода длин трех сторон треугольника;
– группу из двух флажков (Периметр и Площадь) типа `CheckBox`;
– кнопку типа `Button`.

Обеспечить возможность:

– ввода длин трех сторон треугольника;
– выбора режима с помощью флажков: подсчет периметра и/или площади треугольника.

При выборе команды `Calc` открывается диалоговое окно с результатами. При выборе команды `Exit` приложение завершается.

2 Создать меню с командами `Size`, `Color`, `Paint`, `Quit`.

Команда `Paint` недоступна. При выборе команды `Quit` приложение завершается. При выборе команды `Size` открывается диалоговое окно, содержащее:

– два поля типа `TextBox` для ввода длин сторон прямоугольника;
– группу из трех флажков (`Red`, `Green`, `Blue`) типа `CheckBox`;
– кнопку типа `Button`.

Обеспечить возможность:

– ввода длин сторон прямоугольника в пикселах в поля ввода;
– выбора его цвета с помощью флажков.

После задания параметров команда `Paint` становится доступной.

При выборе команды `Paint` в главном окне приложения выводится прямоугольник заданного размера и сочетания цветов или выдается сообщение, если введенные размеры превышают размер окна.

3 Создать меню с командами `Input`, `Work`, `Exit`.

При выборе команды `Exit` приложение завершает работу. При выборе команды `Input` открывается диалоговое окно, содержащее:

– три поля ввода типа `TextBox` с метками `Radius`, `Height`, `Density`;
– группу из двух флажков (`Volume`, `Mass`) типа `CheckBox`;
– кнопку типа `Button`.

Обеспечить возможность:

– ввода радиуса, высоты и плотности конуса;
– выбора режима с помощью флажков: подсчет объема и/или массы конуса.

При выборе команды `Work` открывается окно сообщений с результатами.

Контрольные вопросы

1 Каково назначение платформы `WPF`?
2 Для чего предназначен язык `XAML`?
3 Каковы основные пространства имен, используемые в документах `XAML`?
4 Что понимают под присоединёнными свойствами в `XAML` и как они записываются в `XAML`-документе?
5 Как в коде `XAML` событие для элемента привязывается к обработчику?
6 Каким образом панель `StackPanel` размещает вложенные элементы?
7 С помощью каких свойств осуществляется привязка элемента к определенной строке и столбцу панели `Grid`?
8 Внутри каких элементов `XAML` может быть размещен элемент `Page`?
9 Как в элементе `Hyperlink` задается адрес для перехода на заданную страницу?

3 Лабораторная работа № 3. Создание графических приложений WPF

Цель работы: приобретение навыков выполнения привязки данных к элементам управления, а также форматирования данных с помощью шаблонов в приложениях WPF.

3.1 Необходимые теоретические сведения

1 Использование команд [3, с. 1183–1185].
2 Привязка данных [3, с. 1185–1185].
3 Привязка с помощью XAML [3, с. 1187–1190].
4 Привязка к простым объектам [3, с. 1190–1194].
5 Привязка к спискам [3, с. 1196–1199].
6 Множественная привязка [3, с. 1199–1201].
7 Элемент управления DataGrid [3, с. 1221–1232].
8 Шаблоны [3, с. 1154–1163].

3.2 Индивидуальные задания

Разработать графическое приложение WPF, обеспечивающее выполнение указанных операций над матрицами.

1 Даны квадратные матрицы A и B порядка n. Получить матрицу $C = A^T - B^T$ (где A^T и B^T – транспонированные матрицы).

2 Даны квадратные матрицы A и B порядка n. Получить произведение $A \cdot B^T$.

3 Даны квадратные матрицы A и B порядка n. Получить матрицу $A^2 + B^2$.

Контрольные вопросы

1 Что понимают под привязкой данных WPF?
2 Из каких основных компонентов состоит привязка данных?
3 Какие выделяют виды привязки по направлению потока данных?
4 Какие свойства в элементах управления служат для привязки к ним источника данных?
5 Что называют шаблоном данных WPF?
6 Какие свойства элементов управления служат для задания шаблона данных?

4 Лабораторная работа № 4. Создание библиотек с использованием сборок .NET

Цель работы: получение навыков разработки многомодульных программ с использованием сборок .NET.

4.1 Необходимые теоретические сведения

1 Основные сведения о сборках [3, с. 549–550].
2 Структура сборки. Просмотр содержимого сборок [3, с. 550–552].
3 Создание модулей и сборок [3, с. 553–554].
4 Атрибуты сборок [3, с. 554–556].
5 Создание и загрузка сборок динамическим образом [3, с. 556–558].
6 Разделяемые сборки [3, с. 562–568].
7 Контроль версий сборок [3, с. 573–578].

4.2 Индивидуальные задания

1 Разработать динамическую библиотеку для работы с графами.
2 Разработать динамическую библиотеку для работы с комплексными числами.
3 Разработать динамическую библиотеку для работы с матрицами.

Контрольные вопросы

1 Что понимается в .NET под термином «сборка»?
2 Перечислите составные части сборки.
3 Объясните, как можно просмотреть содержимое сборок.
4 Перечислите этапы создания сборки.
5 Что такое атрибуты сборок?
6 Как создать и загрузить сборку динамическим образом?
7 Что понимается под термином «разделяемая сборка»?
8 Каким требованиям должны удовлетворять разделяемые сборки?
9 Перечислите этапы создания и использования разделяемой сборки?

5 Лабораторная работа № 5. Программирование с использованием LINQ to SQL

Цель работы: получение навыков разработки приложений с использованием LINQ to SQL.

5.1 Необходимые теоретические сведения

Полное описание доступа к базам данных с помощью LINQ to SQL изложено в файле LINQ to SQL.doc.

В приведенном ниже примере вначале создается база данных городов, содержащая два поля: название города и численность его населения. Затем организуется доступ к этой базе данных с помощью LINQ to SQL и создается запрос на извлечение коллекции городов, численность населения которых превышает миллион жителей.

Вначале запускаем Visual Studio 2010 и выбираем проект шаблона Windows Forms Application, укажем имя Name – LinqToSqlГорода. Далее, попав в конструктор формы, из панели элементов Toolbox (Панель элементов) перетаскиваем элемент управления для отображения и редактирования табличных данных DataGridView, на этот элемент в конечном итоге будет попадать результат запроса.

Теперь создадим базу данных SQL Server. Для этого в меню Project (Проект) выберем команду Add New Item (Добавить новый элемент). В появившемся окне выберем элемент База данных, основанная на службах, а в поле Name укажем имя базы данных Города.mdf. Далее в окне мастера настройки источника данных зададим тип модели базы данных – Набор данных. Затем согласимся на сохранение строки подключения в файле конфигурации приложения. Теперь после щелчка на кнопке Готово будет создан пустой набор данных. Этот набор данных Города.mdf теперь будет виден в окне Solution Explorer (Обозреватель решений).

Чтобы заполнить этот набор данных, дважды щелкнем мышью по значку Города.mdf; таким образом, мы попадаем в окно Server Explorer/Database Explorer (Обозреватель серверов/Обозреватель баз данных). Здесь в контекстном меню узла Таблицы выберем команду Добавить новую таблицу. В результате мы попадаем уже в другое окно – dbo.Table1, где зададим имена двух столбцов: Город и Население (рисунок 5.1).

При сохранении (<Ctrl>+<S>) пустой таблицы появится запрос на выбор имени для таблицы, здесь мы зададим имя Города.

Теперь будем заполнять сформированную таблицу. Для этого в Обозревателе серверов щелкнем правой кнопкой мыши на узле Города (имя нашей таблицы) и в появившемся контекстном меню выберем команду Показать таблицу данных. Теперь в окне Города мы имеем возможность заполнять нашу таблицу (рисунок 5.2).

На этом этапе задача создания базы данных и заполнения в ней таблицы городов выполнена. Приступаем к организации запроса к таблице городов.

Как уже указывалось ранее, LINQ-запрос можно построить через набор данных DataSet, а можно LINQ-запрос организовать с помощью классов LINQ to SQL. Эти классы сопоставляются с таблицами и представлениями базы данных и называются классами сущностей DataContext. Класс сущности сопоставляется с записью, а отдельные свойства класса сущности сопоставляются с отдельными столбцами, образующими запись. Сказанное, вероятно, звучит запутанно, но практически сводится к перетаскиванию мышью созданной нами таблицы Города из окна Server Explorer/Database Explorer (Обозреватель серверов/Обозреватель баз данных) на так называемый Object Relational Designer (реляционный конструктор объектов). В результате получим класс сущностей именно для нашей таблицы Города, наследованный от базового класса DataContext, и в тексте нашей программы уже легко сможем строить LINQ-запросы, обращаясь к объекту класса сущностей.

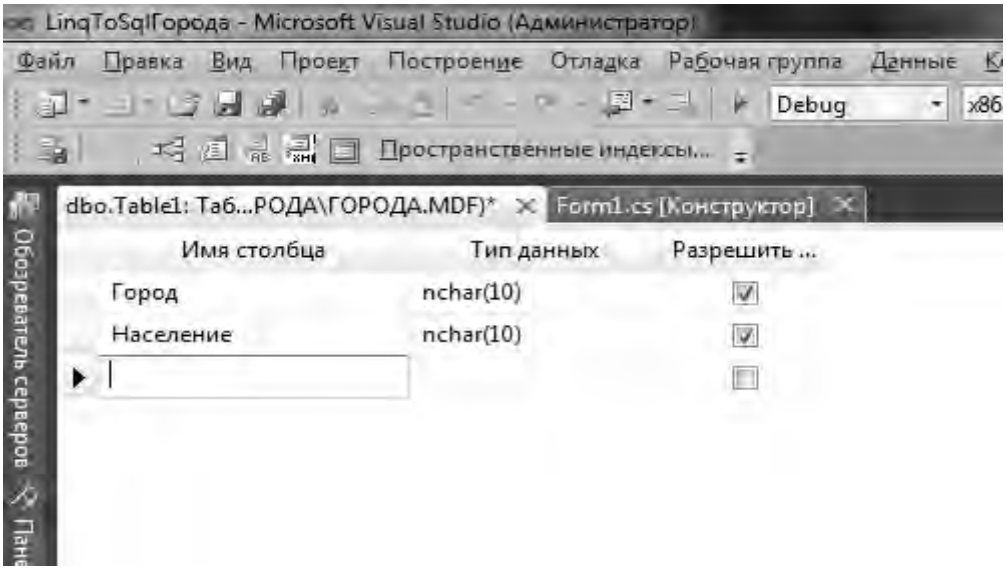

Рисунок 5.1 – Заказ полей таблицы в базе данных

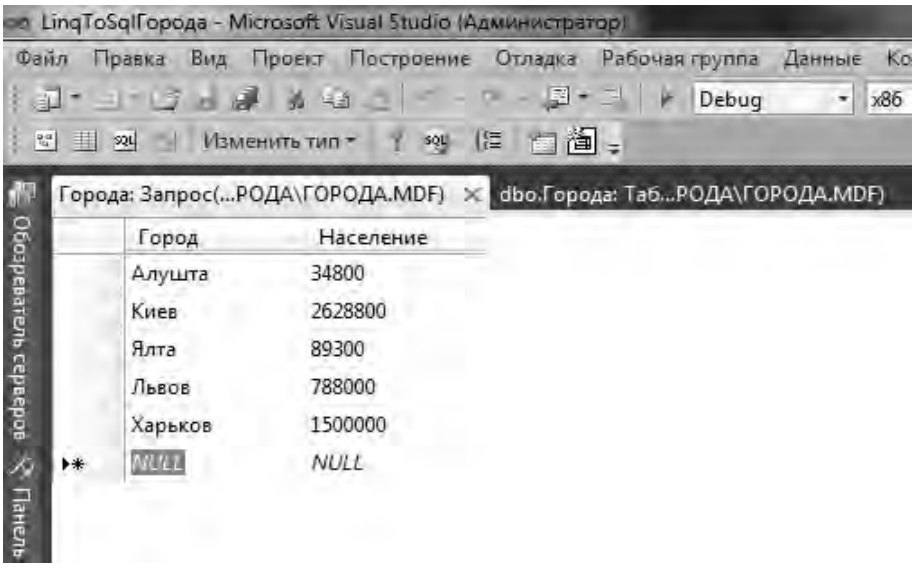

Рисунок 5.2 – Заполнение таблицы городов в базе данных

Чтобы получить в нашем проекте реляционный конструктор объектов, в меню Project выберем команду Add New Item (Добавить новый элемент), а в появившемся одноименном окне – шаблон (элемент) LINQ to SQL Classes (классы LINQ to SQL). В поле Name укажем имя файла Сущности.dbml и щелкнем на кнопке Add. Внешний вид реляционного конструктора объектов можно увидеть на рисунке 5.3.

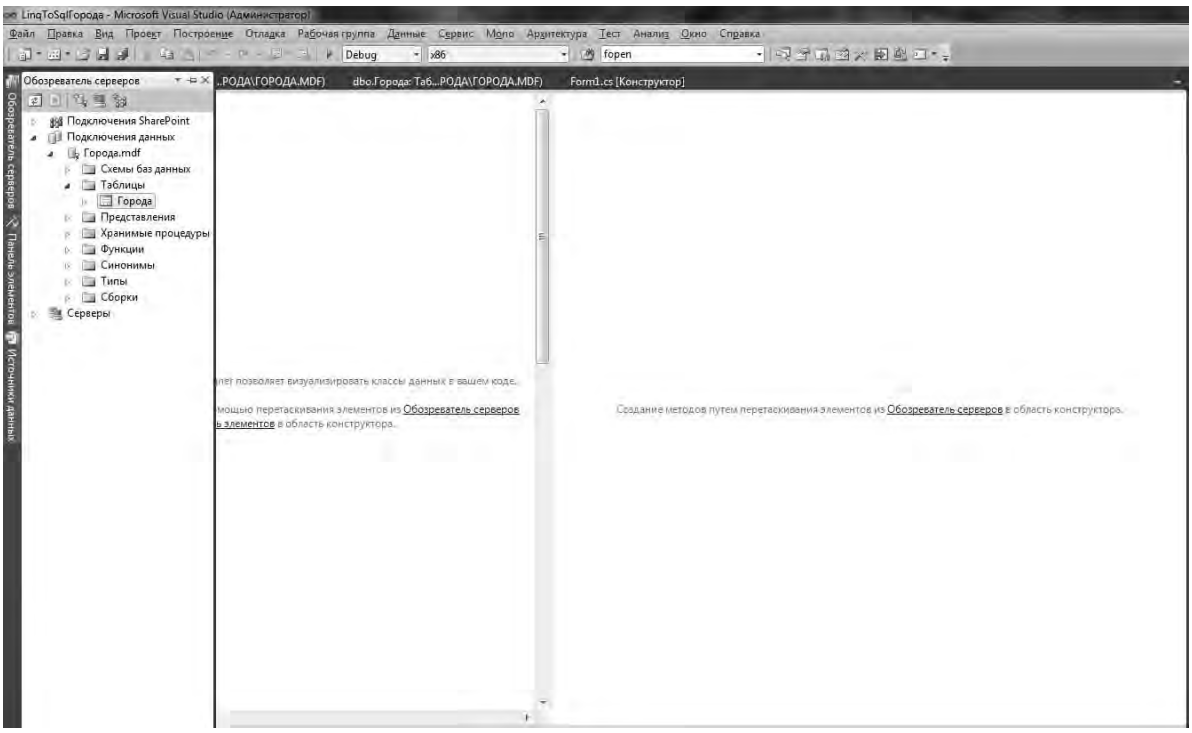

Рисунок 5.3 – Добавление в проект реляционного конструктора объектов

Теперь, просто перетаскиваем мышью таблицу Города из окна Server Explorer/Database Explorer (Обозреватель серверов/Обозреватель баз данных) на реляционный конструктор объектов. Реляционный конструктор объектов создает классы и применяет специфические для LINQ to SQL атрибуты, чтобы иметь функциональные возможности LINQ to SQL (возможности передачи данных и редактирования, какие имеются у DataContext). А нам остается всего лишь на вкладке программного кода ввести текст, представленный в листинге 5.1.

Листинг 5.1 Организация LINQ-запроса к базе данных.

```
// Данное Windows-приложение состоит из экранной формы и элемента управления
// DataGridView. В программе организован LINQ-запрос к базе данных городов с
// помощью базового класса сущностей DataContext. Для этого в данную программу
// добавлен (Project | Add New Item) элемент (шаблон) "Классы LINQ to SQL",
// Name=Сущности.dbml. После связывания таблицы "Города" из базы данных
// с базовым классом сущностей (путем перетаскивания мышью таблицы
// из окна Server Explorer/Database Explorer
// в окно конструктора Object Relational Designer) автоматически был класс
// СущностиDataContext, производный (наследованный) от базоваго класса
```

```
// DataContext. Используя этот класс в данной программе организован
// LINQ-запрос к базе данных на получение коллекции (списка) городов,
// численность населения в которых превышает миллион жителей. Результат запроса
// выведен на элемент управления DataGridView.
using System;
using System.Linq;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace LinqToSqlГорода
{
    public partial class Form1 : Form
    {
        // С помощью объекта базового класса DataContext будем иметь
        // доступ к таблице базы данных:
        private СущностиDataContext БД = new СущностиDataContext();
        //private ГородаDataSet
        public Form1()
        {
            InitializeComponent();
            // В результате запроса получаем коллекцию записей из
            // таблицы базы данных, удовлетворяющей условию where:
            var ГородаМлн = from города in БД.Города
                    where Convert.ToInt32(города.Население) >
                                        1000000
                    select города;
            // или select new { города.Город, города.Население };
            // Результат запроса выводим на элемент управления
            // DataGridView, отображающий табличные данные:
            dataGridView1.DataSource = ГородаМлн;
        }
    }
}
```

Фрагмент работы программы показан на рисунке 5.4. Убедиться в работоспособности программы можно, открыв решение `LinqToSqlГорода.sln` папки `LinqToSqlГорода`.

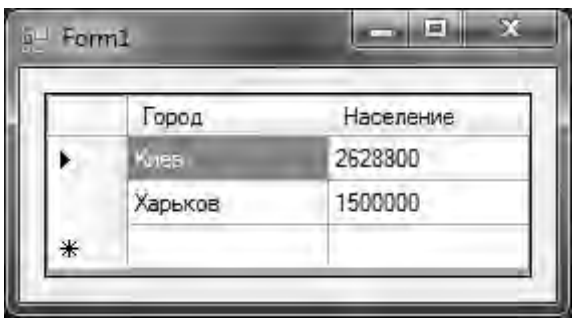

Рисунок 5.4 – Запрос к базе данных на города-миллионеры

5.2 Индивидуальные задания

Задание 1

Разработать схему базы данных для хранения информации:

– кинофильм – название, год выпуска, жанр, режиссёр;

– спектакль – название, автор, жанр;

– актеры – фамилия, имя, отчество, год рождения, амплуа.

Добавить в таблицы необходимые поля для связи с учетом того, что каждый актер может быть задействован в нескольких спектаклях и кинофильмах.

Разработать обзор для отображения данных: актер, название спектакля или кинофильма.

Задание 2

Разработать схему базы данных для хранения данных:

– товар – название, стоимость, количество (в штуках);

– склад – название, общая площадь, количество автопогрузчиков, год постройки;

– производитель – название фирмы, руководитель.

Добавить в таблицы необходимые поля для связи с учетом того, что каждый товар может производиться различными фирмами, каждая фирма может производить различные товары. На складе могут храниться несколько различных товаров, но отдельный товар может храниться только на одном складе.

Разработать представление для отображения данных: товар (наименование, количество), фирма-производитель (наименование), склад (наименование).

Задание 3

Разработать схему базы данных для хранения данных:

– корабль – название, число пассажиров, пароходство, год постройки, тип (речной, морской);

– река – название, длина, максимальная ширина, число притоков;

– море – название, площадь.

Добавить в таблицы необходимые поля для связи с учетом того, что каждый корабль может эксплуатироваться только на реке или только на море. На реке и на море могут эксплуатироваться несколько кораблей.

Разработать представление для отображения данных: корабль (наименование, количество пассажиров, год постройки); название реки/моря.

Контрольные вопросы

1 Что понимают под объектно-реляционным соответствием классов приложения и таблиц базы данных?

2 Что такое двухзвенная технология работы с базой данных?

3 Из каких этапов состоит процесс разработки приложения работы с базой данных на основе `LINQ SQL`?

4 Для чего служит файл .dbml?

5 Каковы основные свойства классов LINQ SQL?

6 Как создается источник данных на основе LINQ SQL?

7 С помощью какого приема обеспечивается визуализация данных источника на экранной форме?

8 Каковы два основных подхода к отображению данных из таблиц базы данных?

9 С помощью какого метода обеспечивается сохранение изменений в базе данных?

10 Как можно осуществить привязку элементов отображения данных с запросом к базе данных?

11 В каком случае может потребоваться переопределение операций вставки, удаления и внесения изменений в таблицы базы данных?

12 Как обеспечивается соответствие полей набора данных и параметров хранимых процедур при выполнении операций вставки, удаления и снесения изменений?

6 Лабораторная работа № 6. Программирование с использованием ADO.NET

Цель работы: ознакомление с основой разработки Windows-приложений на языке C# для работы с базами данных с использованием СУБД MS SQL Server.

6.1 Необходимые теоретические сведения

В этом разделе обсуждаются основы работы с базами данных на платформе .NET. Если на домашнем компьютере нет никакой базы данных, следует установить SQL Server Express Edition или MySql Community Edition, дистрибутивы для которых можно найти в Интернете. Обе базы данных имеют компоненты, позволяющие создавать и редактировать базы данных непосредственно из Visual Studio.

Во всех примерах методических рекомендаций используется база данных TestDB, содержащая единственную таблицу под названием Books. В первом разделе обсуждается создание базы данных на сервере SQL Server.

Создание базы данных в Visual Studio.

Вначале установите сервер базы данных, например, SQL Server (также можно использовать Express Edition). Хотя SQL Server автоматически интегрируется с Visual Studio, не исключено, что придется загрузить дополнительные компоненты, чтобы базы данных от сторонних производителей могли работать в среде Visual Studio. В этом разделе предполагается, что используется SQL Server.

1 Откройте Server Explorer (в меню View (Вид) или комбинацией клавиш <Ctrl>+<W>, <L>).

2 Сделайте щелчок правой кнопкой мыши по Data Connections (Соединения с данными) и выберите Create New SQL Server Database (Создать базу данных в SQL Server).

3 Выберите экземпляр сервера в раскрывающемся списке и укажите метод аутентификации (в соответствии с настройками сервера).

4 Введите имя, например, TestDB и щелкните по кнопке ОК.

5 Разверните окно созданной базы данных.

6 Сделайте щелчок правой кнопкой мыши по Tables и выберите пункт Add New Table (Добавить новую таблицу).

7 Сконфигурируйте таблицу примерно так, как показано на рисунке 6.1.

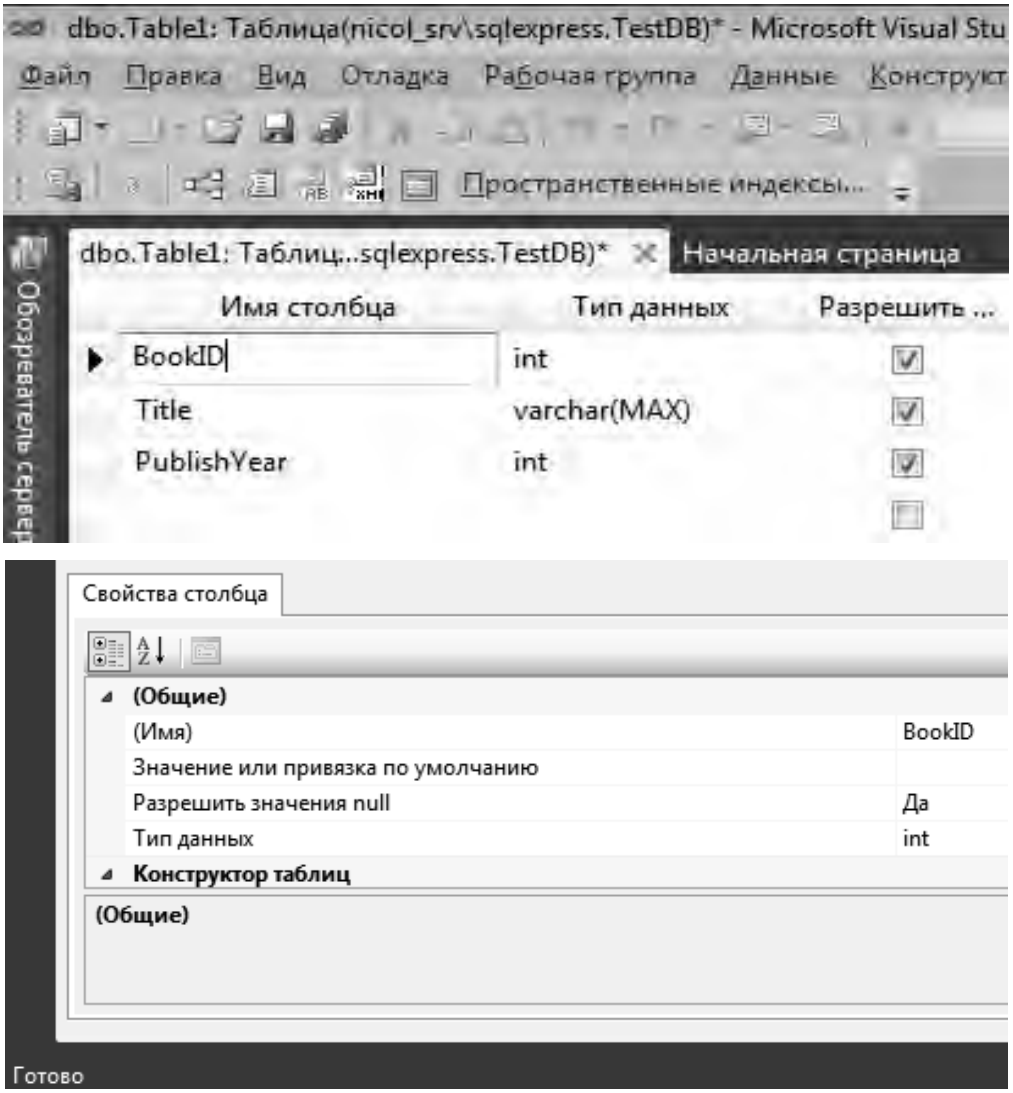

Рисунок 6.1 – Конфигурация таблицы в тестовой базе данных

8 Закройте окно и дайте таблице имя Books.

9 Чтобы заполнить таблицу данными, сделайте на ней щелчок правой кнопкой мыши и выберите Show Table Data (Показать данные таблицы). Значения, которые добавлены таблицу, показаны на рисунке 6.2.

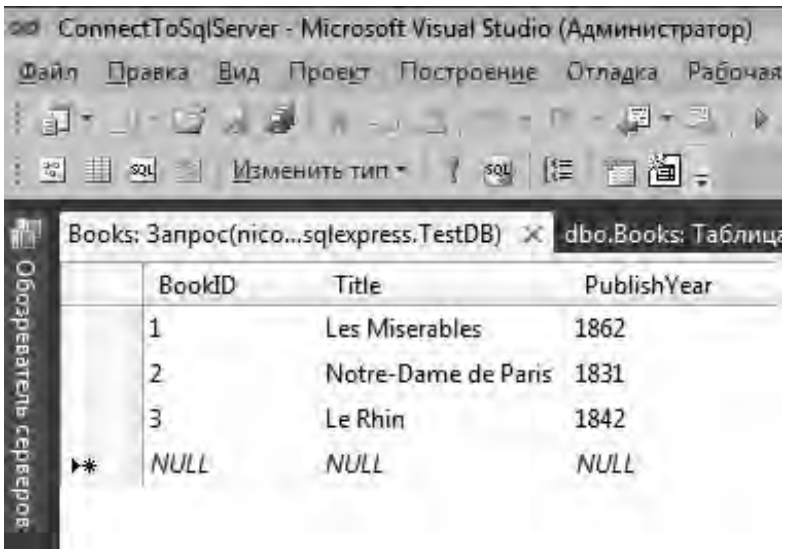

Рисунок 6.2 – Данные в таблице

Соединение с базой и чтение данных.

Работа с большинством баз данных сводится к следующим шагам:

– создать соединение с помощью строки соединения;
– выдать команду в соответствии с синтаксисом SQL;
– выполнить команду через соединение и, возможно, получить данные.

Реализацию данной задачи см. проект `ConnectToSqlServer`.

Пример соединения с SQL Server.

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

namespace ConnectToSqlServer
{
   class Program
   {
      static void Main(string[] args)
      {
          try
          {
              string connectionString = GetConnectionString();
              using (SqlConnection conn =
                          new SqlConnection(connectionString))
              {
                  conn.Open();
                  //Не забудьте передать команде объект-соединение
                  using (SqlCommand cmd = new SqlCommand(
                    "SELECT * FROM Books", conn)
               using (SqlDataReader reader = cmd.ExecuteReader())
```

```
                {
                    // Класс SqlDataReader читает ряды и базы данных
                    // по одному, по мере того, как их запрашиваем
                    while (reader.Read())
                    {
                        Console.WriteLine("{0}\t{1}\t{2}",
                            reader.GetInt32(0),
                            reader.GetString(1),
                            reader.GetInt32(2));
                    }
                }
            }
        }
        catch (SqlException ex)
        {
            Console.Write(ex);
        }
        Console.ReadKey();
    }
    // Хранить строки соединения в исходном коде не нужно.
    // но здесь это сделано в иллюстративных целях
    static string GetConnectionString()
    {
        // Укажите источник данных, соответствующий
        // локальной копии SQL Server на вашем компьютере
        return @"Data source=NICOL_SRV\SQLEXPRESS;
            Initial Catalog=TestDB; Integrated Security=SSPI";
    }
  }
}
```

Этот код выводит следующие результаты:

```
Les Miserables 1862
Notre-Dame de Paris 1831
Le Rhin 1842
```

Соединение с MySQL.

Фирма Sun поставляет .NET-классы для работы с базами данных MySQL. Вы можете выбрать любой из двух вариантов установки:

– воспользоваться программой-установщиком, которая поместит библиотеку MySql.Data.dll в глобальный кэш сборки (GAC, Global Assembly Cache), чтобы приложения MySql могли обращаться к ней;

– поместить MySql.Data.dll в свой проект и обращаться к ней непосредственно из файловой системы.

В коде, сопровождающем этот раздел, выбрана вторая возможность. После добавления ссылки на компонент код оказывается практически идентичным коду, работающему с SQL Server.

Реализацию данной задачи см. проект ConnectToMySQL.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using MySql.Data.MySqlClient;

namespace ConnectToMySQL
{
   class Program
   {
      static void Main(string[] args)
      {
         try
         {
            string connectionString = GetConnectionString();
            // Практически код тот же самый,
            // но везде появился префикс “My”
            using (MySqlConnection conn =
                   new MySqlConnection(connectionString))
            {
               conn.Open();
               using (MySqlCommand cmd = new MySqlCommand(
                      "SELECT * FROM Books", conn))
               using (MySqlDataReader reader = cmd.ExecuteReader())
                    {
                       while (reader.Read())
                       {
                          Console.WriteLine("{0}\t{1}\t{2}",
                                            reader.GetInt32(0),
                                            reader.GetString(1),
                                            reader.GetInt32(2));
                       }
                    }
                }
            }
            catch (MySqlException ex)
            {
                Console.Write(ex);
            }
            Console.ReadKey();
         }
         // Хранить строки соединения в исходном коде не надо,
         // но здесь это сделано в иллюстративных целях
         static string GetConnectionString()
         {
            return @"Data source=localhost;Initial Catalog=TestDB;
                 user=ben;password=password;";
         }
    }
}
```

Добавление и удаление данных в таблицу.
Реализацию данных задач смотри проект `InsertAndDeleteData`.

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

namespace InsertData
{
  class Program
  {
    static void Main(string[] args)
    {
      try
      {
        string connectionString = GetConnectionString();
        using (SqlConnection conn = new
               SqlConnection(connectionString))
        {
          conn.Open();
//insert
         using (SqlCommand cmd = new SqlCommand(
        "INSERT INTO Books (BookID, Title, PublishYear) VALUES (
                          @BookID,@Title, @PublishYear)", conn))
         {
            cmd.Parameters.Add(new SqlParameter("@BookID", "4"));
            cmd.Parameters.Add(new SqlParameter("@Title", "Test
                              Book"));
            cmd.Parameters.Add(new SqlParameter("@PublishYear",
                             "2010"));
            // Здесь используется метод ExecuteNonQuery(), потому
            // что мы не выдаем запрос на чтение строк во время
            // вставки
            int rowsAffected = cmd.ExecuteNonQuery();
            Console.WriteLine("{0} rows affected by insert",
                              rowsAffected);
         }
//display
          using (SqlCommand cmd = new SqlCommand("SELECT * FROM
                 Books", conn))
          using (SqlDataReader reader = cmd.ExecuteReader())
          {
            while (reader.Read())
            {
              Console.WriteLine("{0}\t{1}\t{2}", reader.GetInt32(0),
                           reader.GetString(1), reader.GetInt32(2));
            }
          }
//delete
```

```
            using (SqlCommand cmd = new SqlCommand(
                 "DELETE FROM Books WHERE Title LIKE '%Test%'", conn))
            {
              int rowsAffected = cmd.ExecuteNonQuery();
              Console.WriteLine("{0} rows affect by delete",
                                rowsAffected);
            }
      //display
            using (SqlCommand cmd = new SqlCommand(
                                       "SELECT * FROM Books", conn))
            using (SqlDataReader reader = cmd.ExecuteReader())
            {
              while (reader.Read())
              {
                Console.WriteLine("{0}\t{1}\t{2}",
                                  reader.GetInt32(0),
                                  reader.GetString(1),
                                  reader.GetInt32(2));
              }
            }
          }
        }
        catch (SqlException ex)
        {
          Console.Write(ex);
        }
        Console.ReadKey();
      }
        // Хранить строки соединения в исходном коде не нужно.
        // но здесь это сделано в иллюстративных целях
      private static string GetConnectionString()
      {
         return @"Data source=NICOL_SRV\SQLEXPRESS;
         Initial Catalog=TestDB;Integrated Security=SSPI";
      }
    }
}
```

Связывание данных с элементом управления в приложении Windows Forms.

В следующем коде предполагается, что форма содержит единственный элемент управления `DataGridview`. Полный текст программы, включая опущенный здесь код для реализации графического пользовательского интерфейса, можно увидеть в проекте `DataBindingWinForms` из кода, который сопровождает эти методические указания.

Реализацию данной задачи см. проект `DataBindingWinForms`.

```
public partial class Form1 : Form
{
   DataSet _dataSet;
   public Form1 ()
   {
```

```
            InitializeComponent();
            _dataSet = CreateDataSet();
            dataGridView.DataSource = _dataSet.Tables["Books"];
        }
        protected override void OnFormClosing(FormClosingEventArgs e)
        {
            _dataSet.Dispose();
        }
        private DataSet CreateDataSet()
        {
            string connectionString = @"Data source=BEN-DESKTOP\SQLEXPRESS;
               Initial Catalog=TestDB;Integrated Security=SSPI";
            using (SqlConnection conn = new
                   SqlConnection(connectionString))
            {
               conn.Open();
               using (SqlCommand cmd =
                  new SqlCommand("SELECT * FROM Books", conn))
               using (SqlDataAdapter adapter = new SqlDataAdapter())
               {
                  // Объекты-адаптеры умеют взаимодействовать
                  // с конкретными серверами баз данных
                  adapter.TableMappings.Add("Table", "Books");
                  adapter.SelectCommand = cmd;
                  DataSet dataSet = new DataSet("Books");
                  // Поместить все строки в набор данных
                  adapter.Fill(dataSet);
                  return dataSet;
               }
            }
        }
    }
```


На рисунке 6.3 изображен элемент управления `DataGridview`, связанный с нашей базой данных. Он является удобным средством вывода информации из базы данных прямо на экран.

Рисунок 6.3 – Элемент управления `DataGridview`

6.2 Индивидуальные задания

Разработать программное средство с использованием `Microsoft ADO.NET`, реализующее работу с одной из следующих баз данных.

1 Создать базу данных, которая должна обеспечить возможность просмотра книг по отделам (например, художественный, искусство, научный и т. д.); разделам, которые входят в состав каждого отдела (например, в отдел <художественная литература> входят разделы: русская литература, советская литература и т. д.); издательствам и авторам.

2 В базе данных «Банк» должна храниться информация о вкладах населения как в рублях, так и в валюте. Каждый вклад имеет свой срок хранения и начальный взнос по разным вкладам различен. Также банк может предоставлять различные кредиты – в рублях и в валюте. Пример структуры базы данных: таблица Клиенты, содержащая информацию о клиентах (поля: КодКлиента, Фамилия, Имя, Отчество, Паспорт, Гражданство, Индекс, Страна, Город, Адрес); таблица Вклады, содержащую информацию о рублевых и валютных вкладах (поля: КодВклада, КодКлиента, НомерСчета, ТипСчета, ВидВклада, СуммаВклада, ДатаНачала, ДатаЗавершения) и таблица Кредиты (поля: КодКредита, КодКлиента, ВидКредита, СуммаКредита, ТипВалюты, ДатаВыдачи, ДатаВозврата).

3 База данных должна содержать информацию о предприятиях, готовых предоставить рабочее место по специальности или имеющих вакансии, полную информацию о безработном, его предпочтения, а также дату.

Контрольные вопросы

1 Что такое `ADO.NET`?
2 Что такое провайдер в `ADO.NET`?
3 Как происходит взаимодействие клиента и провайдера в `ADO.NET`?
4 Что является источником данных?
5 Объясните назначение класса `SqlConnection` и его членов.
6 В чем преимущество использования экземпляра класса `SqlConnection` и секции `using`?
7 Объясните основное назначение класса `SqlCommand`.
8 Объясните назначение класса `SqlDataReader`.

7 Лабораторная работа № 7. Программирование с использованием ADO.NET Entity Framework

Цель работы: ознакомление с основой разработки Windows-приложений на языке C# для работы с базами данных на основе технологии ADO.NET Entity Framework.

7.1 Необходимые теоретические сведения

ADO.NET Entity Framework (EF) – объектно-ориентированная технология доступа к данным, является object-relational mapping (ORM) решением для .NET Framework от Microsoft. Предоставляет возможность взаимодействия с объектами как посредством LINQ в виде LINQ to Entities, так и с использованием Entity SQL.

В данной лабораторной работе будет использоваться LocalDB – специальный режим выполнения MS SQL Express. При соединении программы с БД необходимая инфраструктура SQL Server создается и запускается автоматически, что позволяет приложению использовать базу данных без выполнения сложной настройки, занимающей много времени. Средства разработчика позволяют использовать Компонент SQL Server Database Engine для написания и проверки кода Transact-SQL без необходимости управления полным экземпляром сервера SQL Server. Таким образом, можно сказать, что LocalDB используется на этапе разработки ПО, в то время как при эксплуатации используются полнофункциональные СУБД.

Создание файла БД.

LocalDB работает с локальными базами данных в формате mdf. Mdf файл может быть создан и задан вручную или автоматически. Ручное создание более удобно и позволяет получить больший контроль над данными. Добавьте в проект файл базы данных, основанной на службах, как это показано на рисунке 7.1.

Создание модели Entity Framework.

Entity Framework довольно гибок и позволяет разработчику работать в трёх режимах:

1) Code First – на основе существующих классов создаются классы для доступа к данным;

2) Model First – в редакторе строится схема БД, на основе которой генерируются классы для доступа к данным и скрипт для создания БД;

3) Database First – на основе существующей базы данных генерируется код доступа к данным.

При выполнении данной лабораторной работы логично использование как Code First, так и Model First. Ниже будет показан процесс использования подхода CodeFirst, как более универсального и простого для понимания.

За дополнительной информацией можно обратиться по адресу:

https://metanit.com/sharp/entityframework/1.2.php.

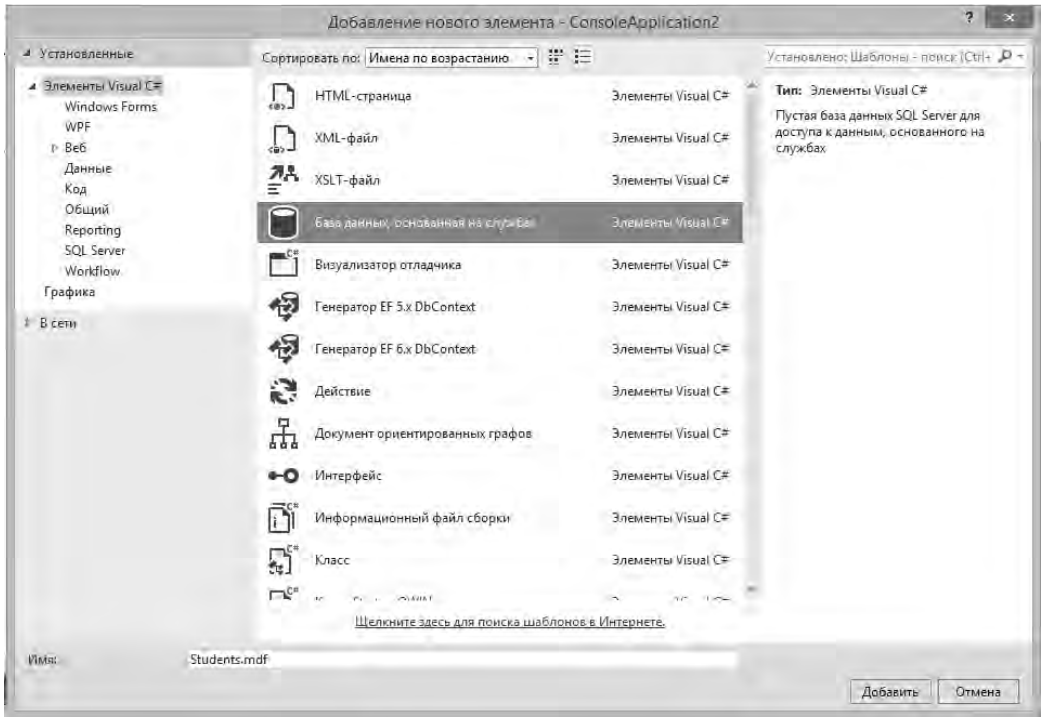

Рисунок 7.1

В нашем случае уже имеется класс, описывающий данные. Изменим его согласно примеру (варианту задания на лабораторную работу) и добавим свойство `Id` для создания первичного ключа в базе данных.

```
class Student
{
    ссылок 0
    public int Id { get; set; }
    ссылок 0
    public string Name { get; set; }
    ссылок 0
    public int Age { get; set; }
}
```

По умолчанию при генерации базы данных `Entity Framework` в качестве первичных ключей будет рассматривать свойства с именами `Id` или `[Имя_класса]Id` (то есть `StudentId`).

Чтобы добавить библиотеку для `Entity Framework` необходимо нажать на проект правой кнопкой мыши и выбрать в контекстном меню `Manage NuGet Packages…` . В появившемся окне выбрать пакет и установить его (рисунок 7.2).

Рисунок 7.2

После установки добавьте в проект новый класс контекста данных `StudentContext`:

```
using System.Data.Entity;

namespace ConsoleApplication2
{
    ссылок 3
    class StudentContext:DbContext
    {
        ссылка 1
        public StudentContext():base("DbConnection")
        { }
        ссылок 3
        public DbSet<Student> Students { get; set; }
    }
}
```

Установите подключение к базе данных в файле конфигурации приложения `App.config`:

```
 <connectionStrings>
   <add name="DBConnection" connectionString="Data Source=(LocalDB)\v11.0;AttachDB
providerName="System.Data.SqlClient"/>
 </connectionStrings>
```

Где название подключения указывается в атрибуте `name=”DBConnection”`, т. к. в конструкторе класса `StudentContext` передается в качестве названия именно эта строка.

Настройку строки подключения задает атрибут `connectionString`. Местоположение базы данных указывается в ключевом слове `AttachDBFilename`.

На этом этапе база полностью готова для работы.

Работа с EF.

Работа с записями через `EF` происходит как с объектами в обычных контейнерах. В первую очередь с помощью оператора using следует создать новый контейнер. Вся следующая работа будет проходить через этот контейнер. В нём содержатся коллекции объектов и различные сервисные объекты и методы. На рисунке 7.3 показан код добавления нового объекта в коллекцию.

```
using (StudentContext db = new StudentContext())
{
    Student stud = new Student { Name = "Sam", Age = 26 };
    db.Students.Add(stud);
    db.SaveChanges();
    Console.WriteLine("Объекты успешно сохранены");
}
```

Рисунок 7.3

Обратите внимание на метод `SaveChanges()`. Именно этот вызов приводит к сохранению данных на сервере.

Получение объектов из БД показано на рисунке 7.4.

```
    // получаем объекты из бд и выводим на консоль
using (StudentContext db = new StudentContext())
{
    var students = db.Students;
    Console.WriteLine("Список объектов:");
    foreach (Student u in students)
    {
        Console.WriteLine("{0}.{1} - {2}", u.Id, u.Name, u.Age);
    }
}
```

Рисунок 7.4

Код удаления записи показан на рисунке 7.5.

```
using(StudentContext db=new StudentContext())
{
    // Выбор и удаление записи по ключевому полю
    var student = db.Students.Find(2);
    if (student!=null)
    {
        db.Students.Remove(student);
    }

    // Выбор и удаление записи по значению свойства
    student = db.Students.FirstOrDefault(o => o.Name == "Tom");
    if (student != null)
    {
        db.Students.Remove(student);
    }
    db.SaveChanges();
}
```

Рисунок 7.5

7.2 Индивидуальные задания

Выполнить задание из лабораторной работы № 6 с использованием технологии `Entity Framework`.

Контрольные вопросы

1 С помощью какой технологии возможен переход от отношений к объектам? Как она реализована?

2 Какие подходы к разработке клиент-серверных приложений существуют в `Entity Framework`? В каких случаях какой из них используется?

3 Модель `EDM`. Как в ней описаны сущности?

4 Объясните структуру сгенерированных сущностных классов.

5 Что генерируется помимо сущностных классов?

8 Лабораторная работа № 8. Программирование с использованием ASP.NET Entity Framework

Цель работы: получить практические навыки разработки программ с использованием технологий `ASP.NET` и `Entity Framework`.

8.1 Постановка задачи

Разработать `Web`-приложение для просмотра и редактирования небольшой базы данных (БД) согласно варианту задания, приведенного в файле `Задания.doc`.

8.2 Этапы выполнения работы

Список шагов выполнения работы приведен ниже. Содержание шагов определено в соответствующих разделах данных методических указаний:

– ознакомление с требованиями к приложению;
– определение архитектуры приложения;
– проектирование и создание базы данных приложения;
– разработка компонентов модели данных приложения;
– определение архитектуры пользовательского интерфейса приложения;
– разработка системных тестов для приложения;
– проектирование отдельных форм приложения;
– разработка модульных тестов для отдельных форм приложения;
– разработка отдельных форм и интеграция приложения.
– перечисленные шаги могут выполняться и в другой последовательности. Схема зависимостей между отдельными шагами изображена на рисунке 8.1. Шаги, не зависящие друг от друга, можно выполнять в произвольном порядке и поручать их различным студентам, не организовывая специальным образом взаимодействие между ними.

Ознакомление с требованиями к приложению.

В рамках данного шага следует ознакомиться с требованиями к разрабатываемому приложению и прояснить все вопросы, связанные с его функциями.

При ознакомлении с требованиями студенты изучают приводимые в Приложении 1 (см. файл `Apendix.doc`) требования и разрешают все возникающие недоумения и вопросы при помощи обсуждения, а также ответов преподавателя, если вопрос четко сформулирован.

Описание требований выдается каждому участвующему в работе студенту в таком виде, чтобы он мог делать на нем индивидуальные пометки. Например, так, как оно оформлено в файле `Apendix.doc`, приложении 1.

По окончании данного шага студенты должны уметь давать четкие ответы на все вопросы, касающиеся требований к приложению.

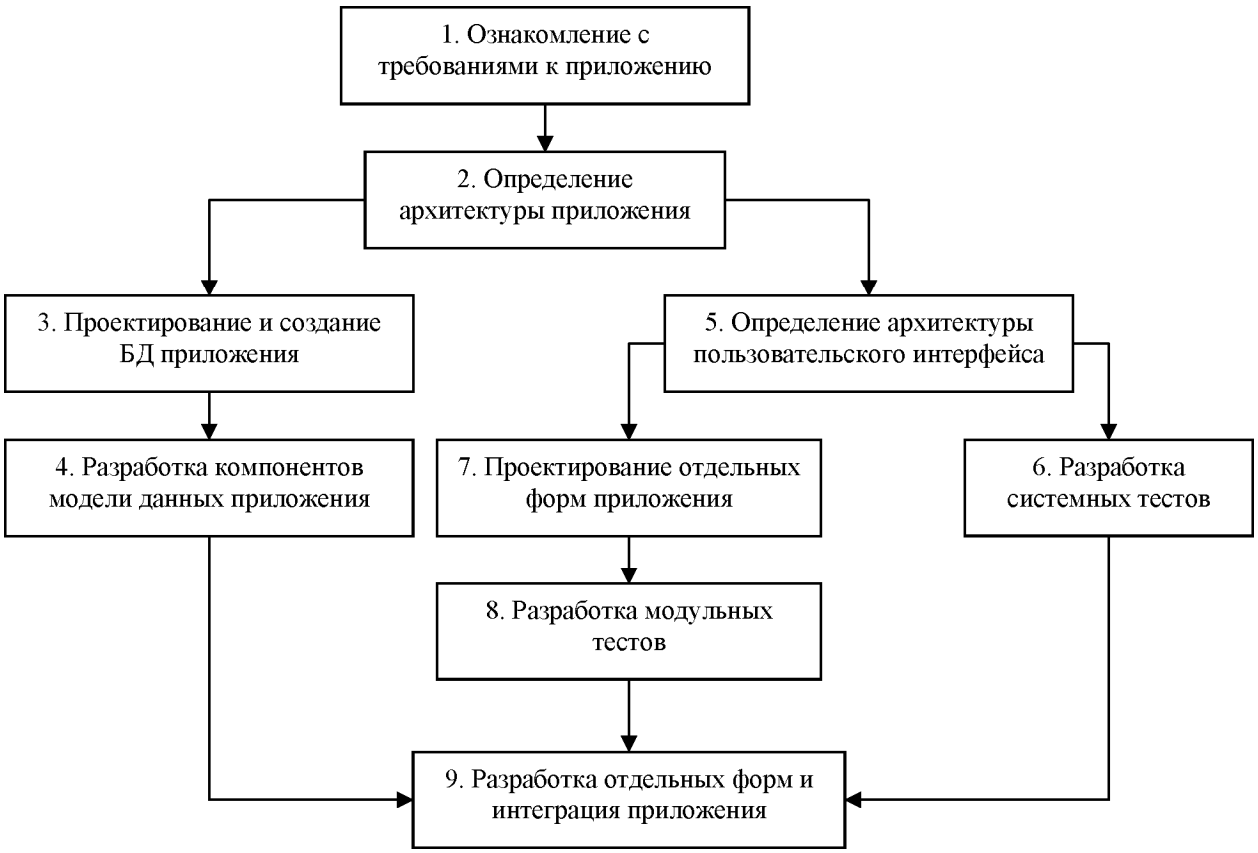

Рисунок 8.1 – Схема зависимости шагов выполнения лабораторной работы

Примеры вопросов по требованиям также даны в файле `Apendix.doc`, приложении 1.

Определение архитектуры приложения.

В рамках этого шага должна быть определена общая архитектура приложения. Архитектура определяется на основании требований в виде набора компонентов, которые, взаимодействуя друг с другом, обеспечивают реализацию всех функций приложения и выполнение нефункциональных требований к нему.

Основные элементы этой архитектуры прямо указаны в требованиях.

Студенты при определении архитектуры должны явно определить ее основные компоненты и связи между ними.

Если лабораторная работа выполняется группой студентов, на этом этапе определяется распределение работ между ними. Рекомендуется поручать одну работу двум студентам с разделением ролей («программист» и «разработчик тестов и документации») для выработки навыков эффективного общения во время разработки.

По окончании данного шага должна быть определена архитектура разрабатываемого приложения – набор основных компонентов, возможные связи и способы взаимодействия между компонентами, интерфейсы компонентов, возможные методы реализации компонентов или использование уже существующих. Дополнительно должна быть определена схема обеспечения защиты данных приложения.

Варианты архитектуры приложения, интерфейсов его основных компонентов и схемы обеспечения защиты представлены в Приложении 2 (см. файл Apendix.doc).

Проектирование и создание БД приложения.

В рамках этого шага на основе сформулированных требований проектируется и создается БД приложения. Созданная БД наполняется тестовыми данными, необходимыми для проверки работоспособности как самой базы, так и приложения в целом.

В результате этого шага должна быть полностью определена схема БД приложения – набор таблиц, их полей, типы полей, связи между таблицами, первичные ключи и альтернативные ключи таблиц. БД приложения, соответствующая этой схеме, должна быть создана в рамках используемой СУБД. Кроме того, она должна быть наполнена тестовыми данными.

Варианты схемы БД приложения приведены в Приложении 3 (см. файл Apendix.doc).

Разработка компонентов модели данных приложения.

На этом шаге разрабатываются компоненты модели данных приложения, используемые как внутреннее представление его данных, хранимых в БД. Кроме того, определяется связь модели данных приложения с базой данных. Такая связь должна обеспечивать синхронизацию данных объектов приложения и таблиц БД, поддержку транзакций при работе с объектами и пр. В рамках технологии ADO.NET все эти функции реализуются автоматически.

Модель данных приложения, построенная на основе ADO.NET, представляет собой набор классов, реализующих запросы к БД приложения.

В результате выполнения этого шага должен быть написан код классов модели данных приложения на языке C#.

Разрабатываемый на этом шаге код компонентов модели данных зависит от их интерфейса, принятого на шаге определения архитектуры приложения.

Вариант кода компонентов модели данных приложения приведен в файле Apendix.doc, приложении 4.

Определение архитектуры пользовательского интерфейса приложения.

На этом шаге определяется набор основных форм приложения и схема навигации между ними.

При наличии требований по защите доступа к различным элементам приложения, определяется политика защиты и методы защиты отдельных форм и потоков данных, в тои числе, методы аутентификации и авторизации пользователей, используемые протоколы аутентификации и пр.

От данного приложения требуется предоставлять всем пользователям возможность просматривать данные о книгах, а выделенным пользователям – возможность редактирования этих данных.

В результате выполнения этого шага должны быть определены набор

форм приложения, функции каждой из форм, схема навигации между формами, а также защищенные области приложения (группы форм с функциями, к которым должны иметь доступ только пользователи в рамках определенных ролей) и способы реализации защиты доступа к каждой из таких областей.

Решения, касающиеся защищенных областей и способов их защиты, зависят от общей схемы защиты, принятой на шаге определения архитектуры приложения.

Вариант архитектуры пользовательского интерфейса разрабатываемого приложения приведен в Приложении 5 (см. файл `Apendix.doc`).

Разработка системных тестов.

На данном шаге должен быть определен набор сценариев работы с приложением и выполняемые по их ходу проверки, которые совместно обеспечивают проверку всех основных функций приложения.

В результате этого шага должен быть составлен список системных тестов – сценариев работы с приложением, в ходе которых проверяются все его основные функции и совместная работа всех форм.

Вариант описания набора системных тестов представлен в Приложении 6 (см. файл `Apendix.doc`).

Проектирование отдельных форм приложения.

На этом шаге определяются информация и набор функций, предоставляемых каждой формой приложения, и набор элементов управления, используемых для выполнения этих функций и навигации между формами.

При проектировании отдельной формы учитываются аспекты удобства использования, относящиеся к реализуемым ею функциям.

В результате этого шага должны быть получены точные описания или проекты всех форм приложения в визуальном редакторе. Для каждой формы должны быть определены ее набор элементов пользовательского интерфейса и информация, представленная в каждом информационном (не только управляющем) элементе.

Вариант проекта форм представлен в Приложении 7 (см. файл `Apendix.doc`).

Разработка модульных тестов.

На этом этапе определяется набор тестов для каждой из форм.

В результате этого шага для каждой из форм приложения должен быть составлен список модульных тестов – сценариев работы с данной формой и ее ближайшими соседями, в ходе которых проверяются все функции и возможности, предоставленные этой формой.

Вариант описания набора модульных тестов представлен в Приложении 8 (см. файл `Apendix.doc`).

Разработка отдельных форм приложения и их интеграция.

На этом шаге выполняется разработка форм, кода элементов управления, конфигурационных файлов приложения и пр. Здесь также осуществляется интеграция и отладка всех элементов приложения.

По окончании этого шага должны быть получены следующие результаты:

– должен быть разработан код всех форм приложения, как страниц ASP.NET, так и поддерживающий их код на C#;

– должен быть доработан конфигурационный файл приложения – в нем должны быть отражены принятые решения по защите доступа к отдельным формам. При помещении форм из разных областей защиты в различные директории приложения должны быть разработаны отдельные конфигурационные файлы для всех таких директорий;

– разработанный код должен пройти отладку и модульное тестирование. Необходимые для этого модульные тесты были разработаны на предыдущем шаге;

– приложение в целом должно пройти системное тестирование, используемые в котором тесты также были разработаны ранее.

Варианты кода форм приложения и конфигурационных файлов представлены в Приложении 9 (см. файл Apendix.doc).

Контрольные вопросы

1 Для чего предназначен пакет Entity Framework?
2 Как осуществляется привязка к данным в Web-приложениях?
3 Что такое ASP.NET?
4 Как реализуется форма для аутентификации?
5 Для чего нужен объект Session?

9 Лабораторная работа № 9. Программирование с использованием ASP.NET MVC

Цель работы: получить практические навыки разработки приложений с использованием технологии ASP.NET MVC.

9.1 Постановка задачи

Разработать приложение с использованием шаблона ASP.NET MVC для базы данных для СУБД Microsoft SQL Server, реализованной согласно варианту задания, приведенного в файле Задания.doc.

9.2 Использование шаблона ASP.NET MVC

Существует довольно много платформ, работающих над ASP.NET. Корпорация Microsoft поставляет вместе с Visual Studio 2010 платформу под названием ASP.NET MVC, где MVC расшифровывается как Model-View-Controller («модель-представление-контроллер»).

Построим простое приложение, выводящее информацию о книгах из базы данных.

Создание приложения.

Создаем в Visual Studio 2010, используя SQL Server базу данных, которая имеет следующую структуру (рисунок 9.1).

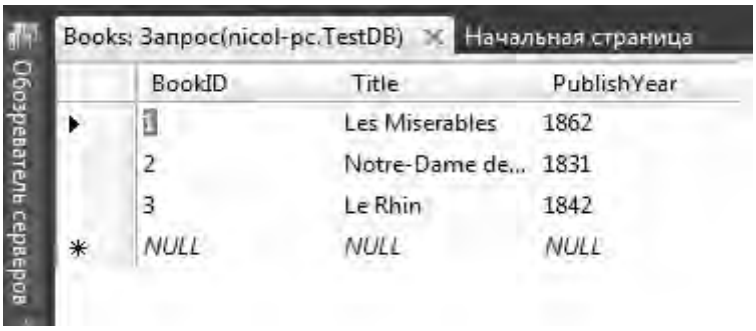

Рисунок 9.1 – Структура базы данных TestDB

В среде Visual Studio 2010 создайте новый проект под именем MVCBooksApp и в качестве его типа укажите ASP.NET MVC 2 Web Application (Веб-приложение ASP.NET MVC 2).

На вопрос, желаете ли вы создать тестирующий проект, ответьте No (Нет). Появится новый проект с некоторым количеством уже сгенерированных папок и файлов, включая папки для компонентов каждого из трех типов: контроллеров, моделей и представлений.

Создание модели.

В качестве модели выберем Entity Framework. Выполните щелчок правой кнопкой мыши на папке Models в окне Solution Explorer (Обозреватель решений) щелкнуть Add (Добавить) → New Element (Создать элемент) (рисунок 9.2).

Откроется окно шаблонов Visual Studio 2010, в котором выбираем Модель ADO.NET EDM и задаем имя Book.edmx (рисунок 9.3).

Откроется окно мастера Entity Data Model Wizard (Мастер модели EDM) (рисунок 9.4), в котором вам следует выбрать Generate from Database (Создать из базы данных) и щелкнуть по Next (Далее).

Откроется окно «Выбор подключения к данным» (рисунок 9.5). Выберите созданную ранее базу данных TestDB, введите BookDBEntities и щелкните по Next (Далее).

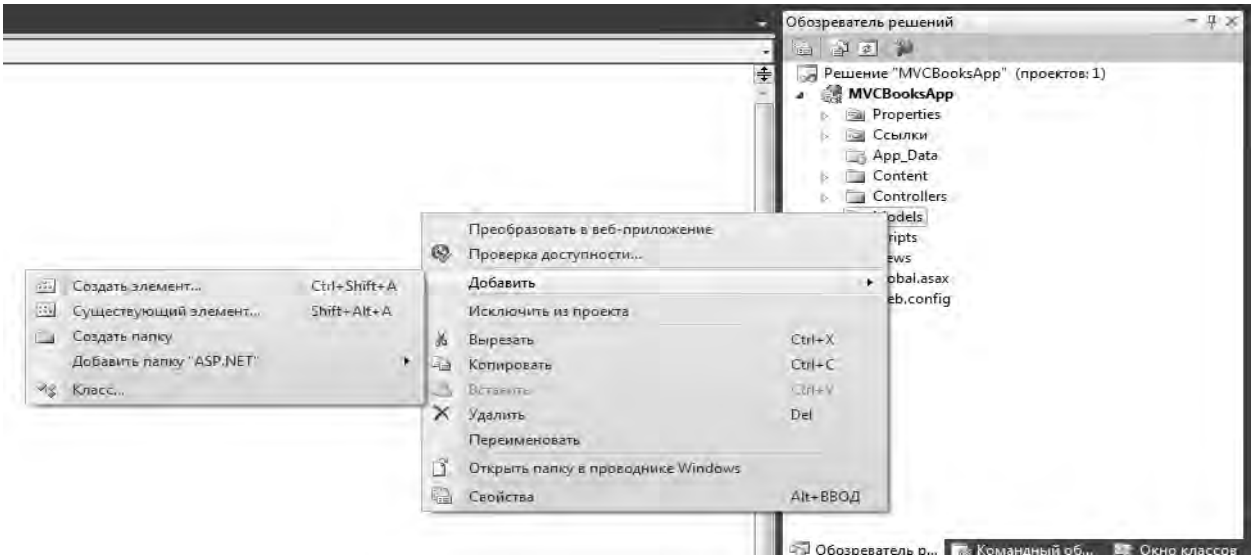

Рисунок 9.2 – Добавление элемента

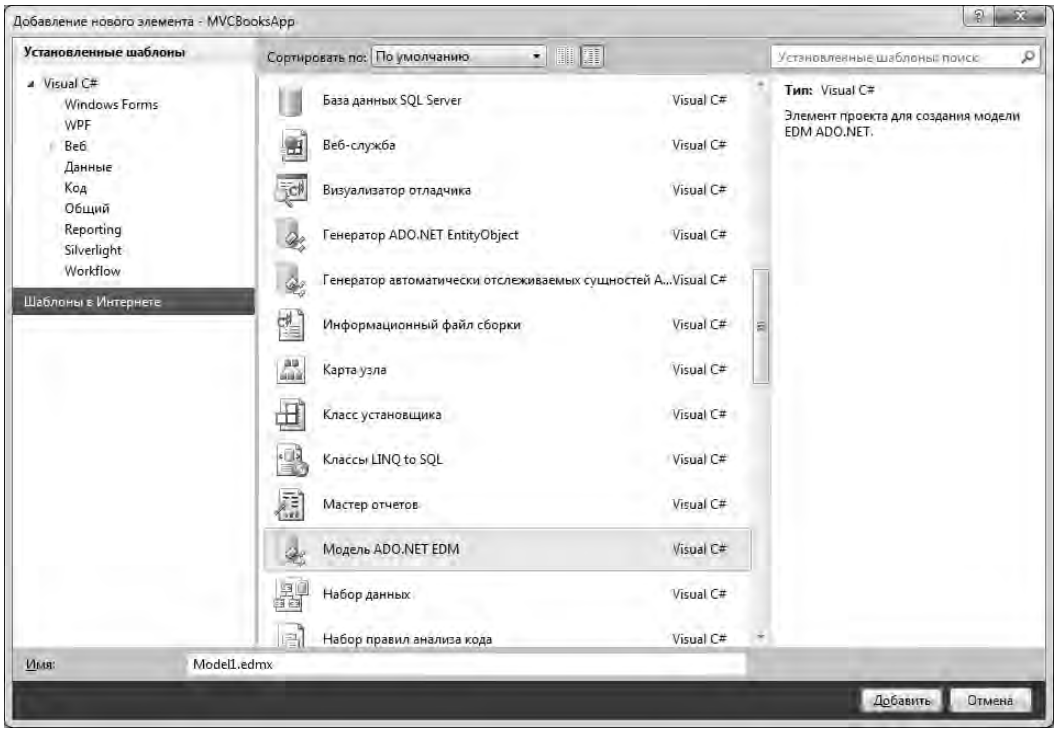

Рисунок 9.3 – Выбор модели `Entity Framework`

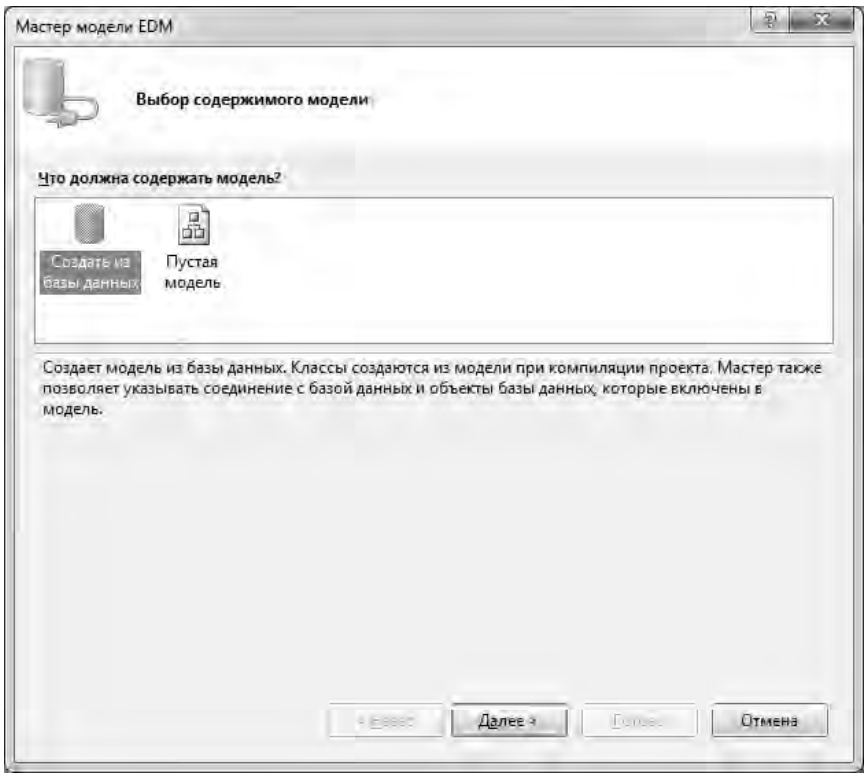

Рисунок 9.4 – Выбор базы данных

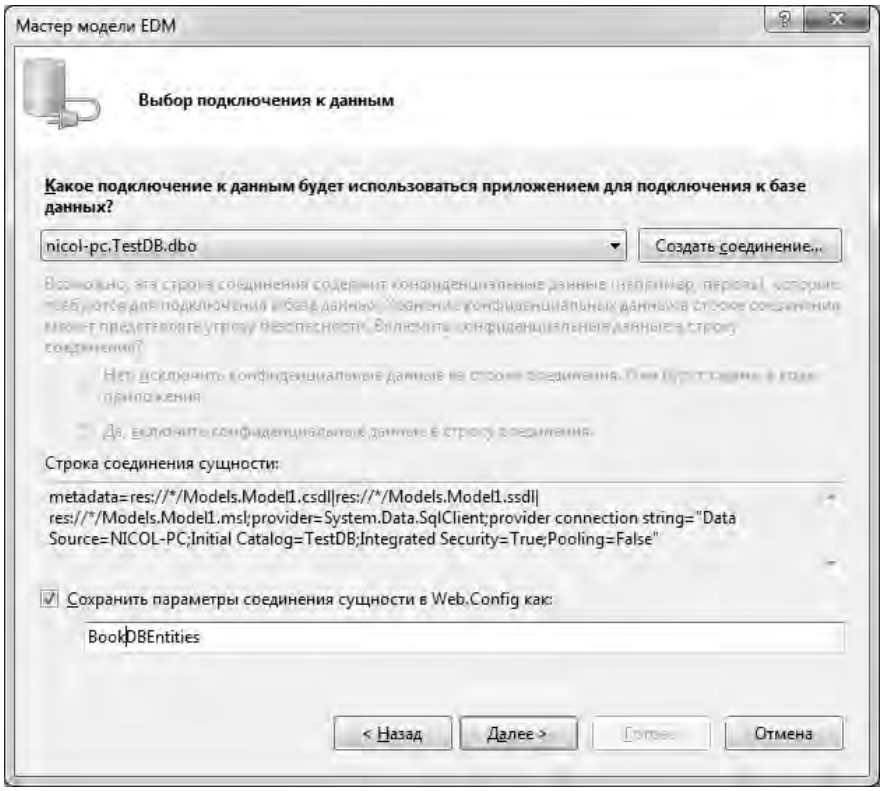

Рисунок 9.5 – Выбор подключения к данным

Откроется окно «Выбор объектов базы» данных (рисунок 9.6). Разверните дерево `Tables` (Таблицы) и установить флажок у таблицы `Books`.

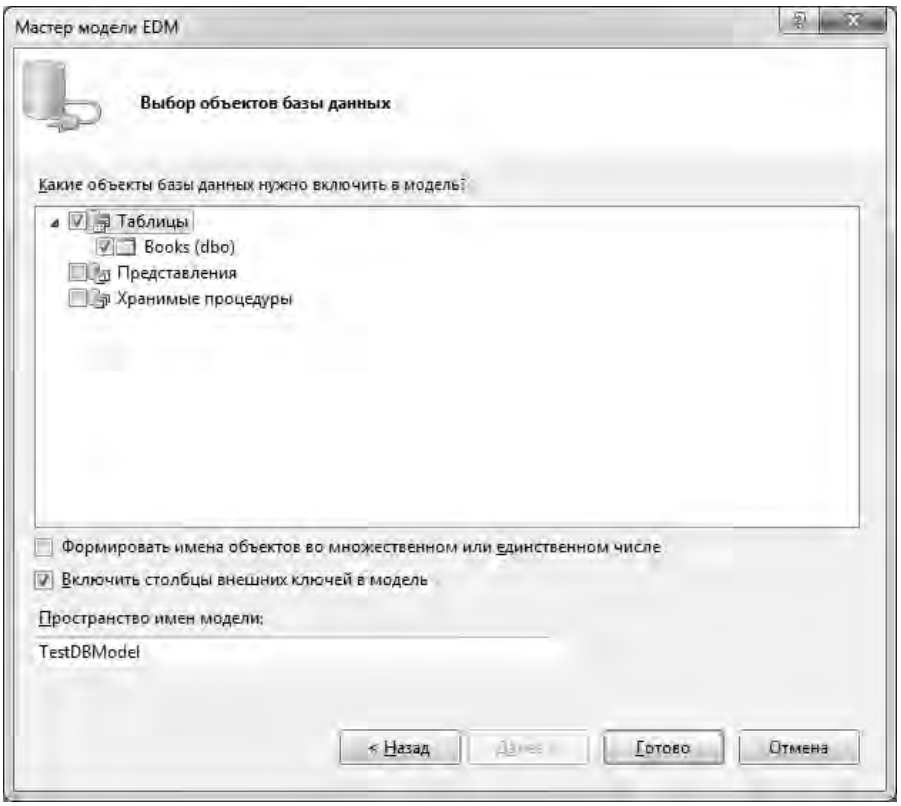

Рисунок 9.6 – Выбор таблицы базы данных

Далее щелкнуть кнопку `Finish` (Готово). Будет создана объектная модель, являющаяся отображением данных из базы.

Создание контроллера.

В отличие от традиционного для `ASP.NET` подхода, при котором каждый `URL`-адрес отображается на соответствующий файл страницы в вашем исходном коде, платформа `MVC` отображает `URL`-адреса на контроллеры и методы. Например, адрес `http://localhost: 1000/Home/Index` отображается на контроллер по имени `HomeController`, который должен находиться в папке `Controllers` вашего проекта в файле `HomeController.cs`. (Мастер создания проекта должен был сгенерировать этот файл.)

Кроме того, часть `"Index"` этого `URL`-адреса отображается на открытый метод `Index()`, возвращающий перечисление `ActionResult`.

Модифицируем код `HomeController` так, чтобы он выглядел следующим образом.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
namespace MVCBooksApp.Controllers
{
[HandleError]
  public class HomeController : Controller
```

```
  {
    MVCBooksApp.Models.BooksDBEntities _db =
      new Models.BooksDBEntities();
    public ActionResuit Index()
    {
      // Возвратить представление по имени Index
      return View(_db.Books.ToList());
    }
  }
}
```

Теперь можно строить проект.

Создание представителя.

На платформе ASP.NET MVC многое происходит автоматически, и это облегчает программисту вывод данных на веб-страницу. В частности, путь к представлению определяется именами контроллера и метода. Если файл Views\Home\lndex.aspx уже существует, удалите его. Сейчас создадим его заново.

1 Выполните щелчок правой кнопкой мыши по методу Index() в контроллере HomeController и выберите Add View (Добавить представление).

2 Дайте представлению имя Index.

3 Установите флажок Create a Strongly-Typed View (Создать строго типизированное представление).

4 В раскрывающемся списке View data class (Класс данных представления) выберите MVCBooksApp.Models.Books.

5 В раскрывающемся списке View content (Содержимое представления) выберите List (Список).

6 Щелкните по Add (Добавить).

Теперь у вас имеется приложение, которое выводит список всех книг из базы данных, как показано на рисунке 9.7.

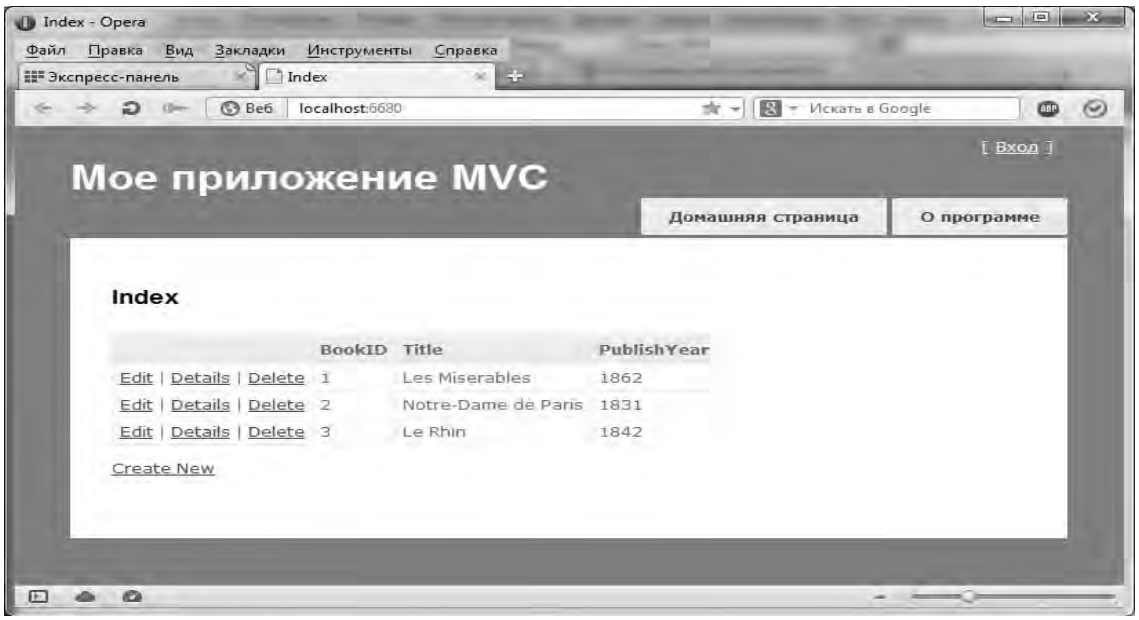

Рисунок 9.7 – Список книг из базы данных

Создание новых записей.

Версия индексной страницы, принимаемая по умолчанию, содержит пункт меню `Create New` (Создать новый компонент). Вам будет совсем нетрудно придать ей необходимую функциональность.

Для этого мы должны создать новое действие в элементе `HomeController`. Допишем в файл `HomeController.cs` два новых метода.

```
// Возвратить представление, создаваемое по умолчанию
public ActionResult Create()
{
  return View();
}

// Вызывается, когда пользователь отправляет на сервер
// информацию о новой книге
[AcceptVerbs(HttpVerbs.Post)]
public ActionResult Create(
  [Bind(Exclude="Id")]
   MVCBooksApp.Models.Books newBook)
{
  if (!ModelState.IsValid)
    return View();
  _db.AddToBooks(newBook);
  _db.SaveChanges();

  // Возвращаемся в Index
  return RedirectToAction("Index");
}
```

Чтобы создать представление, выполните те же действия, что и при создании представления `Index`, но теперь в раскрывающемся списке `View content` (Содержимое представления) выберите `Create` (Создать).

Далее вы должны удалить код поля для идентификационного номера (если таковой имеется) из сгенерированного `HTML`-файла `Create.aspx`, поскольку в нашей базе данных соответствующее поле генерируется автоматически:

```
<p>
<label for="ID">ID:</label>
<%= Html.TextBox("ID") %>
<%= Html .ValidationMessage ("ID",    %>
</p>
```

Законченная страница должна выглядеть примерно так, как показано на рисунке 9.8. Заметим, что она автоматически обладает функциональностью для базовой проверки допустимости данных.

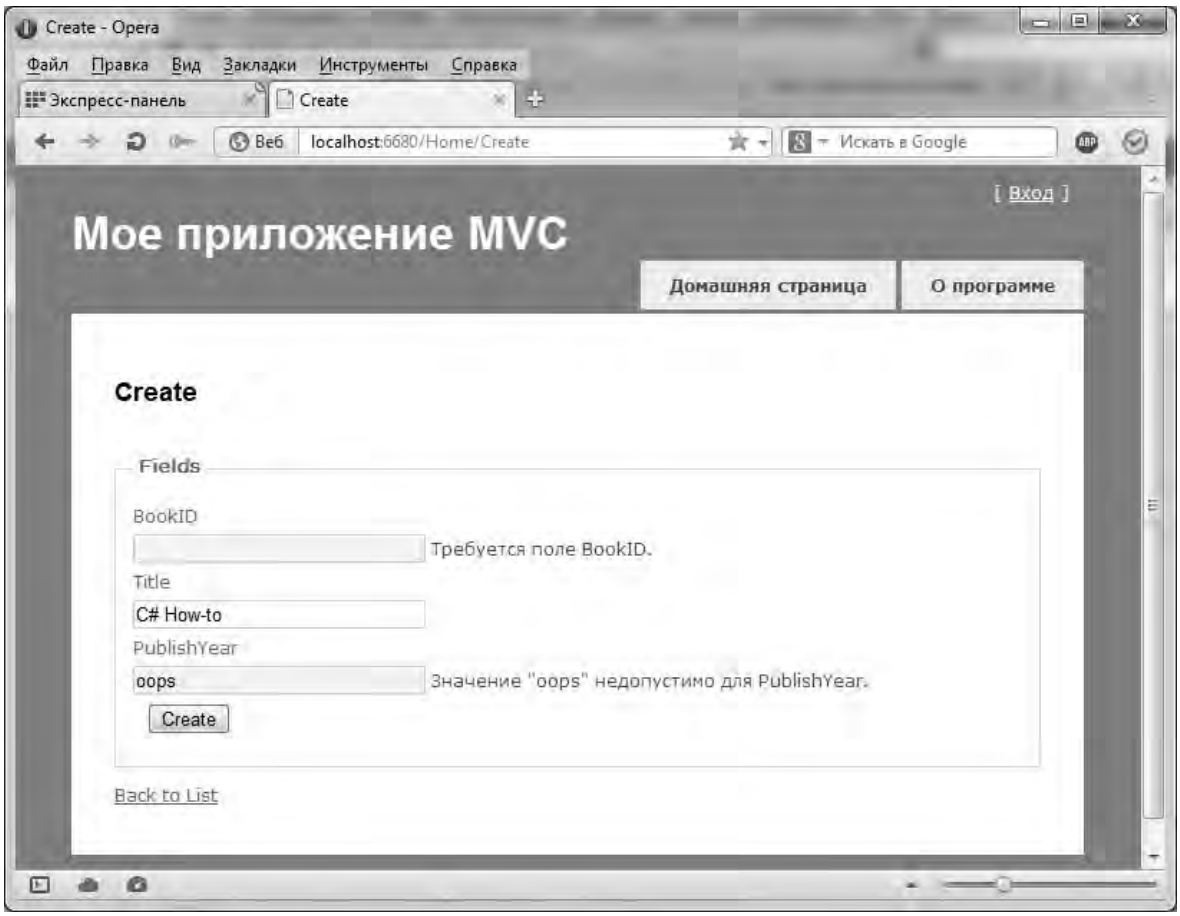

Рисунок 9.8 – Внешний вид MVC приложения

Редактирование записей.

Для редактирования имеющихся записей мы опять вынуждены добавить в контроллер пару методов. Одному мы будем передавать идентификационный номер записи, чтобы было понятно, какая запись редактируется, а другой будет принимать запрос POST и возвращать обновленную запись в базу данных.

Добавьте в файл HomeController.cs такие методы:

```
public ActionResuit Edit(int id)
{
   // Выражение LINQ
   Models.Book book = (from b in _db.Books
               where b.ID == id
               select b).First();
   return View(book;
}

[AcceptVerbs(HttpVerbs.Post)]
public ActionResuit Edit(Models.Book editBook)
{
  // Получить запись о книге из базы данных
  Models.Book book = (from b in _db.Books
              where b.ID == editBook.ID
              select b).First();
```

```
    if (!ModelState.IsValid)
      return View(book);

    // Вернуть измененную запись в базу данных
    _db.ApplyCurrentValues(book.EntityKey.EntitySetName, editBook);
    _db.SaveChanges();
    return RedirectToAction("Index");
  }
```

Для создания элемента `Edit View` (Редактировать представление) выберите `Edit` (Редактировать) в раскрывающемся списке `View content` (Содержимое представления). Вам и на этот раз придется удалить код поля идентификационного номера из сгенерированного `HTML`-файла.

Контрольные вопросы

1 Охарактеризуйте активную модель `MVC`.
2 Приведите достоинства и недостатки пассивной модели `MVC`.
3 Каким образом формируется веб-адрес?
4 Каким образом в контроллере проводится обращения к представлению?
5 Охарактеризуйте класс `HtmlHelper`.
6 Что такое роутер?
7 Опишите требования к пространству имен проекта и сборок.
8 Охарактеризуйте директории проекта.
9 Для чего предназначен двигатель `Razor`?
10 Создайте `ASP`-приложение, которое позволит выполнить `CRUD`-операции для произвольной предметной области.

Список литературы

1 **Шилд, Г.** C# 4.0: полное руководство : пер. с англ. / Г. Шилд. – Москва : И. Д. Вильямс, 2011. – 1056 с.

2 **Павловская, Т. А.** C#. Программирование на языке высокого уровня : учебник для вузов / Т. А. Павловская. – Санкт-Петербург : Питер, 2014. – 432 с.

3 C# 5.0 и платформа .NET 4.5 для профессионалов : пер. с англ. / К. Нейгел [и др.]. – Москва : И. Д. Вильямс, 2014. – 1440 с.

4 **Зиборов, В. В.** Visual C# 2012 на примерах / В. В. Зиборов. – Санкт-Петербург : БХВ-Петербург, 2013. – 480 с.

5 **Ватсон, Б.** C#4.0 на примерах / Б. Ватсон. – Санкт-Петербург : БХВ-Петербург, 2011. – 608 с.

6 Объектно-ориентированное программирование и проектирование : методические рекомендации к лабораторным работам для студентов специальности 1-53 01 02 «Автоматизированные системы обработки информации» дневной формы обучения / Сост. Н. К. Борисов. – Могилёв : Белорус.-Рос. ун-т, 2016. – Ч. 1. – 46 с.