Министерство образования и науки РФ

Белгородский государственный технологический университет им. В. Г. Шухова

Кафедра программного обеспечения вычислительной техники  
и автоматизированных систем

Курсовой проект

по дисциплине «Базы данных»  
  
тема: «Разработка информационной системы  
„Библиотека вуза“ с использованием клиент-серверной технологии в среде Visual Studio»

**Автор работы:**  
студент группы ПВ-31  
Адаменко И. И.

**Руководитель проекта:**  
ст. преподаватель  
Гарибов А. И.

Белгород  
2015

Оглавление

[Цель выполнения курсового проекта 3](#_Toc410817000)

[Анализ предметной области 4](#_Toc410817001)

[Описание 4](#_Toc410817002)

[Основные задачи 4](#_Toc410817003)

[Проектирование информационной системы 5](#_Toc410817004)

[Определение основных сущностей 5](#_Toc410817005)

[Выявление связей между сущностями 5](#_Toc410817006)

[Структура базы данных 6](#_Toc410817007)

[Проектирование программного обеспечения 7](#_Toc410817008)

[Разработка на стороне сервера 15](#_Toc410817009)

[SQL-запросы для создания базы данных 15](#_Toc410817010)

[Представления 15](#_Toc410817011)

[Триггеры 15](#_Toc410817012)

[Хранимые процедуры 16](#_Toc410817013)

[Разработка клиентского приложения 17](#_Toc410817014)

[Подключение к базе данных 17](#_Toc410817015)

[Структура клиента 17](#_Toc410817016)

[Пример связывания данных 19](#_Toc410817017)

[Отчёты 20](#_Toc410817018)

[Реализация экспорта и импорта содержимого БД 23](#_Toc410817019)

[Заключение 25](#_Toc410817020)

[Список литературы 26](#_Toc410817021)

[Приложение. SQL-запросы для создания БД 27](#_Toc410817022)

[Приложение. Скриншоты программы 29](#_Toc410817023)

# Цель выполнения курсового проекта

Целями данного курсового проекта являются: закрепление знаний и навыков, полученных в рамках курса «Базы данных» и получение практического опыта проектирования и реализации приложений по технологии «клиент—сервер».

# Анализ предметной области

## Описание

Библиотека высшего учебного заведения занимается хранением и выдачей книг и периодических изданий. Для этого необходимо хранить большие объёмы информации об авторах, книгах, экземплярах и читателях, потому во всех вузах нашей страны используются электронные базы данных, с которыми библиотекарь «общается» посредством специального ПО, так называемого «клиента».

В этом курсовом проекте описывается упрощённая модель вузовской базы данных, которая, однако, позволяет хранить и оперировать самой необходимой информацией.

## Основные задачи

Клиент должен решать следующие задачи:

1. Добавление, изменение и удаление информации о книгах и их экземплярах, жанрах, авторах, читателях и разделах библиотеки.
2. «Выдача» экземпляра книги читателю.
3. Вывод только доступных книг (т. е. тех, которые можно выдать в данный момент читателю).
4. Автоматическое изменение количества доступных экземпляров книг, при добавлении, удалении или изменении статуса оных.
5. Предоставление отчёта о книгах и читателях библиотеки.
6. Импорт и экспорт базы данных библиотеки в XML.

# Проектирование информационной системы

## Определение основных сущностей

Первый шаг в проектировании информационной системы — это создание набора сущностей, информация о которых будет храниться в базе данных, и которой будет оперировать клиент.

Определим сущности, о которых необходимо хранить информацию:

* Книга — содержит информацию о книге;
* Экземпляр — содержит информацию об отдельном экземпляре книги;
* Автор — содержит информацию об авторе книг;
* Предметная область (жанр) — содержит информацию о предметной области, к которой относятся книги;
* Зал — содержит информацию о зале, в котором находятся книги определённой предметной области.
* Читатель — содержит информацию о студенте, который может брать экземпляры книг в определённых залах библиотеки.

## Выявление связей между сущностями

Связи будем строить исходя из истинности следующих высказываний:

* Одна книга может быть написана одним или несколькими авторами;
* Один автора может написать одну или несколько книг;
* Каждая книга может храниться в одном или более экземплярах;
* Каждая книга посвящена одной предметной области;
* Каждая предметная область содержится только в одном зале;
* В одном зале могут храниться книги нескольких предметных областей;
* В любой момент времени один экземпляр книги может быть или в библиотеки, или у одного читателя;
* Каждый читатель может быть записан в одном или нескольких залах и может одновременно брать один или несколько экземпляров книг.

На основе этих данных можно составить ER-диаграмму («сущность—связь»), по которой затем можно будет спроектировать базу данных (рис. 1).

Остаётся определить атрибуты, которые характеризуют экземпляры каждой сущности. В нашей базе данных будет храниться следующая информация:

* Книга: идентификационный номер, название, издательство, год издания, количество свободных экземпляров, количество страниц, ISBN и язык изложения материала книги;
* Автор: идентификационный номер, фамилия, имя, отчество и дата рождения;
* Экземпляр книги: идентификационный номер, дата выдачи, дата возврата и текущий статус (свободна или нет);
* Читатель: идентификационный номер, фамилия, имя, отчество, дата рождения, номер телефона, серия и номер паспорта, группа.
* Зал: идентификационный номер, название, местоположение.
* Предметная область: идентификационный номер, название.



Рисунок 1. ER-диаграмма описания библиотеки

## Структура базы данных

На основе полученной инфологической модели предметной области «Библиотека вуза» можно спроектировать структуру базы данных (рис. 2). На этой схеме структуры связь поля одной таблицы и поля ID другой таблицы означает, что вторая таблица главная, а первая — подчинённая. Жирным начертанием обозначены поля, являющиеся первичными ключами. Таблицы AuthorBook и SectionReader служат для определения связей «многие-ко-многим».

Все поля в базе данных являются обязательными, кроме:

* Author: MiddleName, Birth;
* Instance: ReaderID, Issue, Return;
* Reader: MiddleName, Birth, Passport.

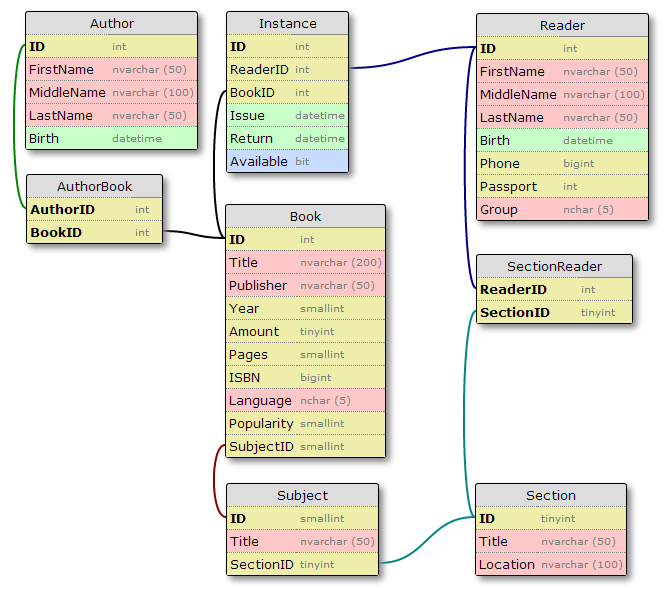


Рисунок 2. Структура базы данных «Библиотека вуза»

## Проектирование программного обеспечения

Следующим шагом в разработке будет описание классов, характеризующих таблицы базы данных. Для разработки клиентского приложения использовался язык C#.

Первым делом был разработан класс **DB**, который осуществляет открытие и закрытие подключения к базе данных, выполняет запросы на вставку, удаление и обновление информации, а также вызов хранимых процедур. Методы класса разработаны максимально гибко, так, чтобы с их помощью можно было выполнить практически любой запрос. Список методов:

* **Open** — открывает подключение к базе данных;
* **Close** — закрывает подключение к базе данных;
* **Select** — создаёт и выполняет SELECT-запрос для выбора данных из базы данных;
* **Insert** — создаёт и выполняет INSERT-запрос для вставки данных в базу данных;
* **Update** — создаёт и выполняет UPDATE-запрос для обновления данных в базе данных;
* **Delete** — создаёт и выполняет DELETE-запрос для удаления данных из базы данных;
* **Execute** — отправляет в базу данных запрос на выполнение в оной хранимой процедуры.

Код этих методов в составе всего класса DB можно увидеть на фрагменте кода 1.

|  |  |
| --- | --- |
| 001 | **class** DB |
| 002 | { |
| 003 | **SqlConnection** connector; |
| 004 | **SqlCommand** commander; |
| 005 |  |
| 006 | **public** DB() |
| 007 | { |
| 008 | **string** connString = **ConfigurationManager** |
| 009 | .ConnectionStrings[*"Program.Properties.Setting"* +   *".libraryConnectionString"*] |
| 010 | *//.ConnectionStrings["Program.Properties.Settings" +   // ".universityConnectionString"]* |
| 011 | .ConnectionString; |
| 012 | connector = **new** **SqlConnection**(connString); |
| 013 |  |
| 014 | commander = **new** **SqlCommand**(); |
| 015 | commander.Connection = connector; |
| 016 | commander.CommandType = **CommandType**.Text; |
| 017 | } |
| 018 |  |
| 019 | **public** **bool** Open() |
| 020 | { |
| 021 | **try** |
| 022 | { |
| 023 | connector.Open(); |
| 024 | **return** **true**; |
| 025 | } |
| 026 | **catch** (**Exception** e) |
| 027 | { |
| 028 | **MessageBox**.Show(e.Message + " " +   connector.State.ToString()); |
| 029 | **return** **false**; |
| 030 | } |
| 031 | } |
| 032 |  |
| 033 | **public** **bool** Close() |
| 034 | { |
| 035 | **if** (connector != **null** && connector.State ==   **ConnectionState**.Open) |
| 036 | { |
| 037 | connector.Close(); |
| 038 | **return** **true**; |
| 039 | } |
| 040 | **return** **false**; |
| 041 | } |
| 042 |  |
| 043 | **public** **SqlDataReader** Select(**string** table,  **List**<**string**> objects = **null**,  **List**<**string**> where = **null**) |
| 044 | { |
| 045 | **string** selection = objects == **null**  ? *"\*"*  : **String**.Join(*", "*, objects); |
| 046 | **string** conditions = (where == **null** || where.Count == 0 ||  (where.Count == 1 && where[0] == **null**)) |
| 047 | ? *""* |
| 048 | : *" WHERE "* + **String**.Join(*" AND "*, where); |
| 049 |  |
| 050 | commander.CommandText = *"SELECT "* + selection + *" FROM "* +  table + conditions; |
| 051 |  |
| 052 | **return** commander.ExecuteReader(); |
| 053 | } |
| 054 |  |
| 055 | **public** **SqlDataReader** Select(**string** table,  **List**<**string**> objects = **null**,  **string** where = **null**) |
| 056 | { |
| 057 | **return** Select(table, objects, **new** List<**string**>() { where }); |
| 058 | } |
| 059 |  |
| 060 | **public** **int** Insert(**string** table,   **List**<**string**> objects,  **List**<**string**> values, **bool** output = **true**) |
| 061 | { |
| 062 | commander.CommandText = (output ?  *"DECLARE @T TABLE ( ID INT ) "* : *""*) |
| 063 | + *"INSERT INTO "* + table + *" (["* +  **String**.Join(*"], ["*, objects) + *"]) "* |
| 064 | + (output ?  *"OUTPUT INSERTED.ID INTO @T "* : *""*) |
| 065 | + *"VALUES "* + *"("* +  **String**.Join(*", "*, values) + *")"* |
| 066 | + (output ? *"SELECT \* FROM @T "* : *""*); |
| 067 | **try** |
| 068 | { |
| 069 | **if** (output) |
| 070 | { |
| 071 | **int**? id = **Globals** .ToNullableInt32(commander.ExecuteScalar()); |
| 072 | **return** id == **null** ? -1 : **Convert**.ToInt32(id); |
| 073 | } |
| 074 |  |
| 075 | **Globals**.ToNullableInt32(commander.ExecuteNonQuery()); |
| 076 | **return** 0; |
| 077 | } |
| 078 | **catch** |
| 079 | { |
| 080 | **return** -1; |
| 081 | } |
| 082 | } |
| 083 |  |
| 084 | **public** **bool** Update(**string** table,  **List**<**string**> objects,  **List**<**string**> values, **List**<**string**> where) |
| 085 | { |
| 086 | commander.CommandText = *"UPDATE "* + table + *" SET "*; |
| 087 | **var** eqs = **new** **List**<**string**>(); |
| 088 | **for** (**int** i = 0; i < objects.Count; i++) |
| 089 | { |
| 090 | eqs.Add(*'['* + objects[i] + *']'* + *" = "* + values[i]); |
| 091 | } |
| 092 |  |
| 093 | commander.CommandText += **String**.Join(*", "*, eqs) + *" WHERE "*   + **String**.Join(*" AND "*, where); |
| 094 | **try** |
| 095 | { |
| 096 | commander.ExecuteNonQuery(); |
| 097 | **return** **true**; |
| 098 | } |
| 099 | **catch** |
| 100 | { |
| 101 | **return** **false**; |
| 102 | } |
| 103 | } |
| 104 |  |
| 105 | **public** **bool** Delete(**string** table,  **List**<**string**> whereObjects = **null**,   **List**<**string**> whereValues = **null**) |
| 106 | { |
| 107 | commander.CommandText = *"DELETE FROM "* + table; |
| 108 |  |
| 109 | **if** (whereObjects != **null** && whereValues != **null**) |
| 110 | { |
| 111 | **var** eqs = **new** **List**<**string**>(); |
| 112 | **for** (**int** i = 0; i < whereObjects.Count; i++) |
| 113 | { |
| 114 | eqs.Add(whereObjects[i] + *" = "* + whereValues[i]); |
| 115 | } |
| 116 |  |
| 117 | commander.CommandText += *" WHERE "*   + **String**.Join(*" AND "*, eqs); |
| 118 | } |
| 119 |  |
| 120 | **try** |
| 121 | { |
| 122 | commander.ExecuteNonQuery(); |
| 123 | **return** **true**; |
| 124 | } |
| 125 | **catch** |
| 126 | { |
| 127 | **return** **false**; |
| 128 | } |
| 129 | } |
| 130 |  |
| 131 | **public** **bool** Execute(**string** title,  **List**<**string**> objects = **null**,  **List**<**string**> values = **null**) |
| 132 | { |
| 133 | **var** cmd = **new** **SqlCommand**(title, connector); |
| 134 | cmd.CommandType = **CommandType**.StoredProcedure; |
| 135 |  |
| 136 | **if** (objects != **null** && values != **null**) |
| 137 | { |
| 138 | **for** (**int** i = 0; i < objects.Count; i++) |
| 139 | { |
| 140 | cmd.Parameters.Add(**new** **SqlParameter**(objects[i],  values[i])); |
| 141 | } |
| 142 | } |
| 143 |  |
| 144 | **try** |
| 145 | { |
| 146 | commander.ExecuteNonQuery(); |
| 147 | **return** **true**; |
| 148 | } |
| 149 | **catch** |
| 150 | { |
| 151 | **return** **false**; |
| 152 | } |
| 153 | } |
| 154 | } |

*Фрагмент кода 1. Реализация класса DB*

Следом были реализованы классы, представляющие таблицы базы данных[[1]](#footnote-1), и в них методы для получения, обновления и удаления сущностей из этих таблиц. В этих методах использовались описанные выше методы класса DB. Реализация одного из классов во фрагменте кода 2. Как видно, во всех классах для работы с сущностями базы данных реализованы следующие методы:

* **Get** — метод для получения списка объектов данного класса из базы данных. С помощью перегрузок реализовано несколько разных вариантов Get. Например, в классе Book можно получить список книг по автору, по жанру и т. д. Эти методы используют метод Select класса DB.
* **Insert —** метод для вставки объекта класса в базу данных. Этот метод использует метод Insert класса DB.
* **Update —** метод для обновления объекта в базе данных. Этот метод использует метод Update класса DB.

На фрагменте кода 2 не показано, однако, у всех классов ещё есть метод Delete для удаления объекта из базы данных, а также методы для импорта в базу данных из XML.

|  |  |
| --- | --- |
| 001 | **public** **class** **Section** : **Binder** |
| 002 | { |
| 003 | **public** **byte** ID { **get**; **private** **set**; } |
| 004 |  |
| 005 | **private** **string** \_title; |
| 006 | **public** **string** Title |
| 007 | { |
| 008 | **get** { **return** \_title; } |
| 009 | **set** |
| 010 | { |
| 011 | **if** (value == **null** || value.Length == 0) |
| 012 | **throw** **new** **ArgumentException**(*"Название указывать  обязательно."*); |
| 013 |  |
| 014 | **if** (value.Length > 50) |
| 015 | **throw** **new** **ArgumentException**(*"Длина названия не может   быть больше 50 символов."*); |
| 016 |  |
| 017 | \_title = **value**; |
| 018 | } |
| 019 | } |
| 020 |  |
| 021 | **private** **string** \_location; |
| 022 | **public** **string** Location |
| 023 | { |
| 024 | **get** { **return** \_location; } |
| 025 | **set** |
| 026 | { |
| 027 | **if** (value == **null** || value.Length == 0) |
| 028 | **throw** **new** **ArgumentException**(*"Местоположение указывать   обязательно."*); |
| 029 |  |
| 030 | **if** (value.Length > 100) |
| 031 | **throw** **new** **ArgumentException**(*"Длина местоположения не   может быть больше 100 символов."*); |
| 032 |  |
| 033 | \_location = **value**; |
| 034 | } |
| 035 | } |
| 036 |  |
| 037 | **public** **List**<**string**> Objects |
| 038 | { |
| 039 | **get** |
| 040 | { |
| 041 | **return** **new** **List**<**string**>() { *"Title"*, *"Location"* }; |
| 042 | } |
| 043 | } |
| 044 |  |
| 045 | **public** **List**<**string**> Values |
| 046 | { |
| 047 | **get** |
| 048 | { |
| 049 | **return** **new** **List**<**string**>() { *"'"* + Title + *"'"*,  *"'"* + Location + *"'"* }; |
| 050 | } |
| 051 | } |
| 052 |  |
| 053 | **public** **Section**() { } |
| 054 |  |
| 055 | **public** **Section**(**string** title, **string** location) |
| 056 | { |
| 057 | Title = title; |
| 058 | Location = location; |
| 059 | } |
| 060 |  |
| 061 | **public** **static** **List**<**Section**> Get(**byte**? id = **null**) |
| 062 | { |
| 063 | **string** where = (id != **null**)  ? "ID = " + id.ToString()  : **null**; |
| 064 | **var** result = **new** **List**<**Section**>(); |
| 065 |  |
| 066 | **SqlDataReader** reader = **Globals**.db.Select(*"Section"*, **null**,   where); |
| 067 |  |
| 068 | **while** (reader.Read()) |
| 069 | { |
| 070 | **var** s = **new** **Section**(**Convert**.ToString(reader[*"Title"*]),   **Convert**.ToString(reader[*"Location"*])); |
| 071 |  |
| 072 | s.ID = **Convert**.ToByte(reader[*"ID"*]); |
| 073 | result.Add(s); |
| 074 | } |
| 075 |  |
| 076 | reader.Close(); |
| 077 |  |
| 078 | **return** result; |
| 079 | } |
| 080 |  |
| 081 | **public** **static** **Section** Get(**byte** id) |
| 082 | { |
| 083 | **return** Get((**byte**?)id)[0]; |
| 084 | } |
| 085 |  |
| 086 | **public** **bool** Insert() |
| 087 | { |
| 088 | **var** res = **Globals**.db.Insert(*"Section"*, Objects, Values); |
| 089 |  |
| 090 | **if** (res != -1 && res <= **Byte**.MaxValue) |
| 091 | { |
| 092 | ID = **Convert**.ToByte(res); |
| 093 | **return** **true**; |
| 094 | } |
| 095 | **else** |
| 096 | { |
| 097 | **return** **false**; |
| 098 | } |
| 099 | } |
| 100 |  |
| 101 | **public** **bool** Update() |
| 102 | { |
| 103 | **return** **Globals**.db.Update(*"Section"*, Objects, Values,  **new** **List**<**string**>() { *"ID = "* +   ID.ToString() }); |
| 104 | } |
| 105 |  |
| 106 | *// some methods here...* |
| 107 |  |
| 108 | **public** **override** **string** ToString() |
| 109 | { |
| 110 | **return** Title; |
| 111 | } |
| 112 | } |

*Фрагмент кода 2. Реализация класса Subject*

# Разработка на стороне сервера

В этом курсовом проекте в качестве сервера использовался Microsoft SQL Server 2012, соответственно все фрагменты кода, в которых используется код на языке SQL написаны с использованием синтаксических особенностей этого сервера.

## SQL-запросы для создания базы данных

SQL-код запросов для создания базы данных представлен в приложении.

## Представления

В рамках курсового проекта для базы данных было создано одно представление, позволяющее получить список «свободных» книг, то есть тех, которые в данный момент можно выдать читателям. Код представления представлен во фрагменте кода 3.

|  |  |
| --- | --- |
| 01 | **CREATE** **VIEW** FreeBooks |
| 02 | **AS** |
| 03 | **SELECT** B.Title **AS** *"Book title"*, |
| 04 | B.Amount **AS** *"Amount"*, |
| 05 | S.Title **AS** *"Section"* |
| 06 | **FROM** Book B, Subject Sbj, Section S |
| 07 | **WHERE** B.SubjectID = Sbj.ID **AND** Sbj.SectionID = S.ID |

*Фрагмент кода 3. Реализация представления FreeBooks*

## Триггеры

Структура базы данных такова, что у одной книги может быть несколько экземпляров и счётчик количества оных, доступных в данный момент. При добавлении новых экземпляров или удалении старых счётчик необходимо обновлять. Так же и при сдаче и выдаче книги читателю. Чтобы облегчить задачу и программную реализацию клиента, обновления счётчика были реализованы на стороне сервера с помощью триггеров.

Триггер — это хранимая процедура, вызываемая автоматически при возникновении определённого события. Поскольку добавление, удаление и обновление экземпляров — это события, то при их возникновении и вызываются соответствующие триггеры. Их код в целом выполняет одну и ту же функцию — обновляет счётчик, поэтому в качестве примера во фрагменте кода 4 приведён только один триггер, срабатывающий при обновлении экземпляра книги.

|  |  |
| --- | --- |
| 01 | **CREATE** **TRIGGER** UpdateBookAmount |
| 02 | **ON** [Instance] **AFTER** **UPDATE** |
| 03 | **AS** **BEGIN** |
| 04 | **DECLARE** @available\_old bit, |
| 05 | @available\_new bit; |
| 06 | **SET** @available\_old = (**SELECT** Available **FROM** deleted); |
| 07 | **SET** @available\_new = (**SELECT** Available **FROM** inserted); |
| 08 |  |
| 09 | **IF** @available\_old != @available\_new |
| 10 | **BEGIN** |
| 11 | **IF** @available\_new = 1 |
| 12 | **BEGIN** |
| 13 | **UPDATE** Book |
| 14 | **SET** Amount = Amount + 1 |
| 15 | **WHERE** ID in (**SELECT** BookID **FROM** inserted) |
| 16 | **END** |
| 17 | **ELSE** |
| 18 | **BEGIN** |
| 19 | **UPDATE** Book |
| 20 | **SET** Amount = Amount – 1 |
| 21 | **WHERE** ID in (**SELECT** BookID **FROM** inserted) |
| 22 | **END** |
| 23 | **END** |
| 24 | **END**; |

*Фрагмент кода 4. Реализация триггера UpdateBookAmount*

## Хранимые процедуры

Не все задачи можно решить только с помощью триггеров или представлений. Иногда требуется составить более сложные запросы, которые будут поддерживать ввод параметров, относительно которых и будут производиться какие-то действия над базой данных. Для этого в SQL используются хранимые процедуры.

В этом курсовом проекте было создано две хранимые процедуры: для сдачи и выдачи книг. «Выдать книгу» — обновить параметры экземпляра, такие как: Available, ReaderID и Issue. Аналогично и со сдачей книг. Во фрагменте кода 5 представлена одна из созданных хранимых процедур.

|  |  |
| --- | --- |
| 01 | **CREATE** **PROCEDURE** GiveBook |
| 02 | @id\_instance int, |
| 03 | @id\_reader int |
| 04 | **AS** **BEGIN** |
| 05 | **UPDATE** Instance |
| 06 | **SET** Available = 0, |
| 07 | ReaderID = @id\_reader, |
| 08 | Issue = **CONVERT**(date, GETDATE()) |
| 09 | **WHERE** ID = @id\_instance |
| 10 | **END**; |

*Фрагмент кода 5. Реализация хранимой процедуры GiveBook*

# Разработка клиентского приложения

## Подключение к базе данных

Для подключения к базе данных из приложения на базе WPF необходимо составить строку подключения. Она состоит из набора пар «ключевое слово=значение», разделённых точкой с запятой.

Строка подключения должна содержать: сервер СУБД, название БД, логин и пароль пользователя. Поскольку разрабатываемая база данных будет использоваться как минимум на двух ПК (на компьютере разработчика и на компьютере руководителя), то в программе (в параметрах приложения) сохранено две строки:

1. Data Source=.\SQLExpress; Initial Catalog=library; Integrated Security=True
2. Data Source=win2012srv; Initial Catalog=library; User ID=sa; Password=masterkey

По строкам видно, что в первом случае (на компьютере разработчика) используется авторизация с помощью Windows, а во втором (у руководителя) — с помощью логина и пароля.

## Структура клиента

На рисунке 3 приведена структура проекта, созданного в среде Visual Studio 2012. Итоговое приложение состоит из 13 окон, соединённых в единую логическую цепь. Это значит, что окна не появляются все разом, а выдаются пользователю в зависимости от его требований.

В приложении реализованы следующие классы, отвечающие за его логику работы с таблицами в базе данных:

* **Author**;
* **Book**;
* **Instance**;
* **Reader**;
* **Section**;
* **Subject**.

Помимо этого, в приложении также есть следующие классы:

* **Binder** — реализует два метода, упрощающих работу с представлениями данных (окнами и формами): BindView и ChangeView, которые связывают и изменяют данные на форме соответственно.
* **DB** — реализует логику работы с БД в целом; предоставляет методы Insert, Update, Delete и Execute, которые используются далее в других классах (о них было сказано выше);
* **ExtensionClass** — как понятно из названия, класс, используемый для расширения всех остальных; реализует такие методы как: Bind (многократно переопределён для элементов TextBox, ComboBox и пр., используется для связывания данных элементов и объектов данных), Modify, Rename, Hide и **Resize** (последние четыре используются для управления элементом   
  DataGrid при его автоматическом формировании);
* **Globals** — используется в качестве глобального класса, доступного из любой точки приложения, а потому хранит статические объекты со списками элементов БД, а также методы для обновления связываемых объектов данных;
* **SerializeClass** — предоставляет методы Import и Export для, соответственно, импорта и экспорта данных из БД в XML и обратно;
* **ValidationRules** — класс, используемый для реализации методов проверки введённых библиотекарем значений в программу.



Рисунок 3. Структура проекта

### Пример связывания данных

Фрагмент кода 6 описывает форму Section, используемую для создания или изменения информации о залах библиотеки.

|  |  |
| --- | --- |
| 01 | <Window x:Class=*"Program.SectionView"* |
| 02 | xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" |
| 03 | xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" |
| 04 | Title="Зал" Height="190" Width="300"   ResizeMode="CanMinimize" Closing="Window\_Closing"> |
| 05 | <Grid> |
| 06 | <Label Content="Название: " Margin="10,10,0,0"/> |
| 07 | <TextBox x:Name="Title" Margin="93,12,0,0" Width="180"/> |
| 08 | <Label Content="Место: " Margin="10,40,0,0"/> |
| 09 | <TextBox x:Name="Location" Margin="93,42,0,0" Width="180"/> |
| 10 | <Label x:Name="IDLabel" Content="ИД: " Margin="10,80,0,0"/> |
| 11 | <Label x:Name="ID" Margin="42,80,0,0"/> |
| 12 | <Button x:Name="Delete" |
| 13 | Content="Удалить" |
| 14 | Style="{StaticResource ResourceKey=SetLang}" |
| 15 | Height="27" |
| 16 | Width="80" |
| 17 | Margin="193,80,0,0" Click="Delete\_Click" /> |
| 18 | <Button x:Name="Complete" Content="Готово"  Click="Complete\_Click"> |
| 19 | <Button.Style> |
| 20 | <Style BasedOn="{StaticResource ResourceKey=Add}"   TargetType="{x:Type Button}"> |
| 21 | <Setter Property="IsEnabled" Value="False" /> |
| 22 | <Style.Triggers> |
| 23 | <MultiDataTrigger> |
| 24 | <MultiDataTrigger.Conditions> |
| 25 | <Condition  Binding="{Binding ElementName=Title,   Path=(Validation.HasError)}"   Value="False" /> |
| 26 | <Condition  Binding="{Binding ElementName=Location,   Path=(Validation.HasError)}"   Value="False" /> |
| 27 | </MultiDataTrigger.Conditions> |
| 28 | <Setter Property="IsEnabled"   Value="True" /> |
| 29 | </MultiDataTrigger> |
| 30 | </Style.Triggers> |
| 31 | </Style> |
| 32 | </Button.Style> |
| 33 | </Button> |
| 34 | </Grid> |
| 35 | </Window> |

*Фрагмент кода 6. Реализация формы Section*

Поскольку программа написана на WPF, то все формы программируются с помощью языка разметки XAML. В данном случае стоит обратить на следующие моменты:

* 04: Все формы обязательно имеют атрибут Title, что задаёт заголовок формы, а также, если форму нельзя расширить на весь экран (как в этом случае), то установлено соответствующее значение атрибута ResizeMode.
* 14: Все основные стили (FontSize, HorizontalAligment и др.) вынесены в общий файл ресурсов и прописаны для каждого элемента формы (TextBox,  
  Label, ComboBox и др.). Уникальные атрибуты прописываются отдельно, однако, если где-то встречаются дублирования уникальных атрибутов, то для таких элементов создаётся отдельный класс. В данном случае к кнопке с названием Delete применён класс SetLang, который используется ещё примерно десяток раз в других формах.
* 22-30: Почти во всех формах есть нижняя кнопка (её атрибуты так же вынесены в отдельный стиль — Add), которая становится активной только в том случае, когда выполнены все условия. В данной форме условиями являются отсутствия ошибок в полях Title и Location. Ошибками в данном случае являются исключения типа ArgumentException, описанные несколькими страницами выше при разборе класса Section.

## Отчёты

В приложении реализована возможность печати отчётов списков книг или читателей. Для реализации отчётов было использовано несколько подходов и технологий.

Отчёты в XLS-файлы (формат Excel) формируются с помощью копирования всех данных DataGrid и записи их в новый файл. Поскольку формат файла XLS относительно прост, Excel с лёгкостью открывает его и все данные абсолютно читабельны. Пример на рисунке 4 наглядно это показывает.

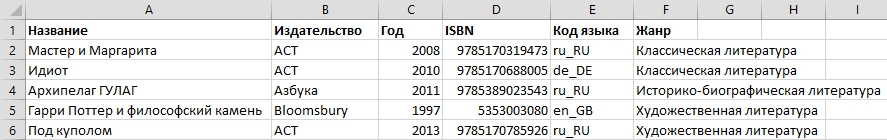


Рисунок 4. Экспортированные данные в XLS-файле

Для копирования информации из DataGrid и записи её в XSL-файл используется прямое копирование с помощью классов ApplicationCommands и Clipboard. Код копирования представлен на фрагменте кода 7.

|  |  |
| --- | --- |
| 01 | *// dg is DataGrid object* |
| 02 | dg.SelectAllCells(); |
| 03 | dg.ClipboardCopyMode = **DataGridClipboardCopyMode**.IncludeHeader; |
| 04 | **ApplicationCommands**.Copy.Execute(null, dg); |
| 05 | **var** res = (**string**)**Clipboard**.GetData(**DataFormats** .CommaSeparatedValue); |
| 06 | **var** result = (**string**)**Clipboard**.GetData(**DataFormats**.UnicodeText); |
| 07 | dg.UnselectAllCells(); |
| 08 |  |
| 09 | **var** file = **new** **StreamWriter**(path, **false**, **Encoding**.Unicode); |
| 10 | file.WriteLine(result.Replace(*','*, *' '*)); |
| 11 | file.Close(); |

*Фрагмент кода 7. Копирование DataGrid в XSL*

Отчёты же в DOC-формате не так просты в построении, поэтому для их создания используется технология **COM (Component Object Model)**. Это технологический стандарт от компании Microsoft, предназначенный для создания программного обеспечения на основе взаимодействующих компонентов, каждый из которых может использоваться во многих программах одновременно. Результат работы экспорта в DOC-файл показан на рисунках 5 и 6.



Рисунок 5. Экспортированные данные в DOC-файле — 1

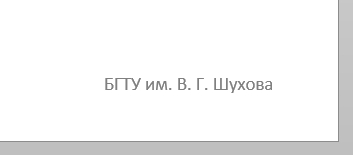


Рисунок 6. Экспортированные данные в DOC-файле — 2

Для работы с MO Word потребовалось подключить дополнительную библиотеку —   
**Microsoft.Office.Interop.Word.dll**, которая поставлялась в пакете **Microsoft Word 15.0 Object Library**. После подключения библиотеки необходимо также подключить её в программе (с помощью ключевого слова using).

Для экспорта в файл необходимо сперва запустить само приложение, с помощью интерфейса Application. После этого создать файл с помощью интерфейса   
\_Document. Далее в распоряжении программиста огромный набор методов и свойств, позволяющих настраивать и наполнять созданный файл по своему усмотрению. Вот лишь некоторые из них:

* **Table** — интерфейс для управления таблицами, имеющий, например, свойство Columns для управления колонками таблицы, и метод set\_Style, для применения стиля.
* **Selection** — интерфейс для добавления, удаления и изменения текста и его формата. Например, имеет такие свойства как:
  + **PageSetup** — настройки страницы (поля, отступы);
  + **Font** — настройки шрифтов (размер, насыщенность, тип, семейство);
  + **ParagraphFormat** — настройки абзацев;
  + **Headers** и **Footers** — верхние и нижние колонтитулы соответственно.

Пример работы с этой библиотекой можно увидеть на фрагменте кода 8.

|  |  |
| --- | --- |
| 01 | Word.**Application** wrdApp; |
| 02 | Word.**\_Document** wrdDoc; |
| 03 | **Object** oMissing = System.Reflection.**Missing**.Value; |
| 04 | **Object** oFalse = **false**; |
| 05 | wrdApp = **new** Word.**Application**(); |
| 06 | Word.Selection wrdSelection; |
| 07 |  |
| 08 | wrdDoc = wrdApp.Documents.Add(**ref** oMissing, **ref** oMissing,  **ref** oMissing, **ref** oMissing); |
| 09 | wrdDoc.Select(); |
| 10 | wrdSelection = wrdApp.Selection; |
| 11 |  |
| 12 | wrdDoc.PageSetup.BottomMargin = wrdDoc.PageSetup.LeftMargin = |
| 13 | wrdDoc.PageSetup.RightMargin = wrdDoc.PageSetup.TopMargin = 50; |
| 14 | wrdDoc.PageSetup.Orientation = **WdOrientation**.wdOrientLandscape; |
| 15 |  |
| 16 | **var** StrToAdd = *"Отчёт научной библиотеки БГТУ им. В. Г. Шухова"*; |
| 17 | wrdSelection.ParagraphFormat.Alignment   = Word.**WdParagraphAlignment**.wdAlignParagraphCenter; |
| 18 | wrdSelection.Font.Size = 22; |
| 19 | wrdSelection.Font.Name = *"PT Serif Caption"*; |
| 20 | wrdSelection.TypeText(StrToAdd); |

*Фрагмент кода 8. Работа с Mirosoft.Office.Interop.Word*

## Реализация экспорта и импорта содержимого БД

Экспорт и импорт данных БД — это очень полезные функции, которые позволяют быстро переносить данные с одного сервера на другой. В качестве документа, который используется для экспорта и импорта был выбран XML-файл. Формат XML очень удобен для описания структурированных данных. Работу с этим форматом поддерживает среда .NET.

После экспорта данных в XML-файл записывается код, пример которого во фрагменте кода 9.

|  |  |
| --- | --- |
| 01 | <?xml version="1.0" encoding="utf-8"?> |
| 02 | <SerializeClass.DataBase   xmlns:i="http://www.w3.org/2001/XMLSchema-instance" z:Id="i1"   xmlns:z="http://schemas.microsoft.com/2003/10/Serialization/"   xmlns="http://schemas.datacontract.org/2004/07/Program"> |
| 03 | <Authors> |
| 04 | <Author> |
| 05 | <Birth>11.11.1821</Birth> |
| 06 | <FirstName>Фёдор</FirstName> |
| 07 | <LastName>Достоевский</LastName> |
| 08 | <MiddleName>Михайлович</MiddleName> |
| 09 | </Author> |
| 10 | <!-- many-many similar lines below --> |

*Фрагмент кода 9. Результат работы экспорта в XML*

Для создания импорта и экспорта был реализован класс **SerializeClass**, который позволяет сохранить данные обо всех переданных ему элементах, учитывая их вложенность. Поскольку важным фактором при импорте данных является проверка на наличие импортируемых элементов в базе, то для этого в классах были реализованы специальные методы **ImportXML**, которые перед тем, как вставить данные в БД проверяют, нет ли их там.

Стоит отметить, что проверять данные по ID элемента ни в коем случае нельзя, поскольку ID — это неконтролируемый параметр, который автоматически увеличивается при вставке данных в базу. Поэтому в разных базах, даже если они хранят одинаковые данные, скорее всего будут разные ID элементов. А значит проверять вставляемые данные следует по какому-то другому, уникальному полю, или же по нескольким полям сразу. Например, для объектов класса Book — это ISBN, для Reader — Phone, потому что у каждой книги свой уникальный ISBN-код, а у каждого читателя свой личный номер телефона, который тоже уникален.

Для записи XML использовался класс XMLWriter, а для чтения — XMLReader. Примеры работы с этими классами видны на фрагменте кода 10.

|  |  |
| --- | --- |
| 01 | **private** **static** **void** Serialize(**Object** obj, **String** path) |
| 02 | { |
| 03 | **using** (**XmlWriter** writer = **XmlWriter**.Create(path)) |
| 04 | { |
| 05 | **try** |
| 06 | { |
| 07 | **DataContractSerializer** serializer =   **new** **DataContractSerializer**(obj.GetType()); |
| 08 | serializer.WriteObject(writer, obj); |
| 09 | } |
| 10 | **catch** |
| 11 | { |
| 12 | **throw** **new** **Exception**(*"Ошибка сериализации!"*); |
| 13 | } |
| 14 | } |
| 15 | } |
| 16 |  |
| 17 | **private** **static** **object** Deserialize(**string** path, **Type** type) |
| 18 | { |
| 19 | **using** (**XmlReader** reader = **XmlReader**.Create(path)) |
| 20 | { |
| 21 | **try** |
| 22 | { |
| 23 | **DataContractSerializer** serializer =  **new** **DataContractSerializer**(type); |
| 24 | **return** serializer.ReadObject(reader); |
| 25 | } |
| 26 | **catch** |
| 27 | { |
| 28 | **throw** **new** **Exception**(*"Ошибка в XML!"*); |
| 29 | } |
| 30 | } |
| 31 | } |

*Фрагмент кода 10. Работа с классами XMLWriter и XMLReader*

# Заключение

В этом курсовом проекте была создана база данных «Библиотека вуза», которая содержит информацию о содержащихся в библиотеке книгах, их экземплярах, авторах, читателях, жанрах и залах, в которых они хранятся.

В ходе работы были выполнены поставленные цели, а именно:

* были закреплены знания и навыки, полученные в рамках курса «Базы данных»;
* был получен практический опыт проектирования и реализации приложений по технологии «клиент—сервер».

Помимо этого, было изучено несколько ранее неизвестных технологий, таких как: WPF, COM (Word), XML, XAML.

В ходе проектирования и реализации базы данных были получены знания об основах работы с СУБД Microsoft SQL, написании программ на языке SQL, получен практический опыт самостоятельного проектирования и реализации баз данных в архитектуре «клиент—сервер» на языке высокого уровня C#.

# Список литературы

1. Стэкер Мэтью А., «Разработка клиентских Windows-приложений на платформе Microsoft .NET Framework: Учебный курс Microsoft» / Пер. с англ. — М.: Издательство «Русская Редакция»; Спб.: Питер, 2008. — 624 с.
2. Тернстрем, Т., «Microsoft SQL Server 2008. Разработка баз данных. Учебный курс Microsoft» / Пер. с англ. — М.: Издательство «Русская Редакция», 2010. — 496 с.
3. Каталог API (Microsoft) и справочных материалов [электронный ресурс]. — msdn.microsoft.com/ru-ru/library/.
4. Википедия — Свободная энциклопедия на русском языке [электронный ресурс] — ru.wikipedia.org.
5. Stack Overflow [электронный ресурс] — stackoverflow.com.
6. CodeProject — For those who code [электронный ресурс] — codeproject.com.

# Приложение. SQL-запросы для создания БД

|  |  |
| --- | --- |
| 01 | **CREATE** **TABLE** [Section] ( |
| 02 | [ID] tinyint **NOT** **NULL** **IDENTITY** (1, 1) **PRIMARY** **KEY**, |
| 03 | [Title] nvarchar(50) **NOT** **NULL** **CHECK** ([Title] <> ''), |
| 04 | [Location] nvarchar(100) **NOT** **NULL** **CHECK** ([Location] <> '') |
| 05 | ) |
| 06 | **GO** |
| 07 |  |
| 08 | **CREATE** **TABLE** [Reader] ( |
| 09 | [ID] int **NOT** **NULL** **IDENTITY** (1, 1) **PRIMARY** **KEY**, |
| 10 | [FirstName] nvarchar(50) **NOT** **NULL** **CHECK** ([FirstName] <> ''), |
| 11 | [MiddleName] nvarchar(100), |
| 12 | [LastName] nvarchar(50) **NOT** **NULL** **CHECK** ([LastName] <> ''), |
| 13 | [Birth] datetime, |
| 14 | [Phone] bigint **NOT** **NULL** **CHECK** ([Phone] > 0), |
| 15 | [Passport] int, |
| 16 | [Group] nchar(5) **NOT** **NULL** **CHECK** ([Group] <> '') |
| 17 | ) |
| 18 | **GO** |
| 19 |  |
| 20 | **CREATE** **TABLE** [SectionReader] ( |
| 21 | [ReaderID] int **NOT** **NULL** **REFERENCES** [Reader] |
| 22 | **ON** **DELETE** **CASCADE** |
| 23 | **ON** **UPDATE** **CASCADE**, |
| 24 | [SectionID] tinyint **NOT** **NULL** **REFERENCES** [Section] |
| 25 | **ON** **DELETE** **CASCADE** |
| 26 | **ON** **UPDATE** **CASCADE**, |
| 27 | **PRIMARY** **KEY** ([SectionID], [ReaderID]) |
| 28 | ) |
| 29 | **GO** |
| 30 |  |
| 31 | **CREATE** **TABLE** [Subject] ( |
| 32 | [ID] smallint **NOT** **NULL** **IDENTITY** (1, 1) **PRIMARY** **KEY**, |
| 33 | [Title] nvarchar(50) **NOT** **NULL** **CHECK** ([Title] <> ''), |
| 34 | [SectionID] tinyint **REFERENCES** [Section] |
| 35 | **ON** **DELETE** **SET** **NULL** |
| 36 | **ON** **UPDATE** **CASCADE** |
| 37 | ) |
| 38 | **GO** |
| 39 |  |
| 40 | **CREATE** **TABLE** [Book] ( |
| 41 | [ID] int **NOT** **NULL** **IDENTITY** (1, 1) **PRIMARY** **KEY**, |
| 42 | [Title] nvarchar(200) **NOT** **NULL** **CHECK** ([Title] <> ''), |
| 43 | [Publisher] nvarchar(50) **NOT** **NULL** **CHECK** ([Publisher] <> ''), |
| 44 | [Year] smallint **NOT** **NULL**, |
| 45 | [Amount] tinyint **NOT** **NULL**, |
| 46 | [Pages] smallint **NOT** **NULL** **CHECK** ([Pages] > 0), |
| 47 | [ISBN] bigint **NOT** **NULL** **CHECK** ([ISBN] > 0), |
| 48 | [Language] nchar(5) **NOT** **NULL** **CHECK** ([Language] <> ''), |
| 49 | [Popularity] smallint **NOT** **NULL** **CHECK** ([Popularity] >= 0), |
| 50 | [SubjectID] smallint **REFERENCES** [Subject] |
| 51 | **ON** **DELETE** **SET** **NULL** |
| 52 | **ON** **UPDATE** **CASCADE** |
| 53 | ) |
| 54 | **GO** |
| 55 |  |
| 56 | **CREATE** **TABLE** [Author] ( |
| 57 | [ID] int **NOT** **NULL** **IDENTITY** (1, 1) **PRIMARY** **KEY**, |
| 58 | [FirstName] nvarchar(50) **NOT** **NULL** **CHECK** ([FirstName] <> ''), |
| 59 | [MiddleName] nvarchar(100), |
| 60 | [LastName] nvarchar(50) **NOT** **NULL** **CHECK** ([LastName] <> ''), |
| 61 | [Birth] datetime |
| 62 | ) |
| 63 | **GO** |
| 64 |  |
| 65 | **CREATE** **TABLE** [AuthorBook] ( |
| 66 | [AuthorID] int **NOT** **NULL** **REFERENCES** [Author] |
| 67 | **ON** **DELETE** **CASCADE** |
| 68 | **ON** **UPDATE** **CASCADE**, |
| 69 | [BookID] int **NOT** **NULL** **REFERENCES** [Book] |
| 70 | **ON** **DELETE** **CASCADE** |
| 71 | **ON** **UPDATE** **CASCADE**, |
| 72 | **PRIMARY** **KEY** ([AuthorID], [BookID]) |
| 73 | ) |
| 74 | **GO** |
| 75 |  |
| 76 | **CREATE** **TABLE** [Instance] ( |
| 77 | [ID] int **NOT** **NULL** **IDENTITY** (1, 1) **PRIMARY** **KEY**, |
| 78 | [ReaderID] int **REFERENCES** [Reader] |
| 79 | **ON** **DELETE** **SET** **NULL** |
| 80 | **ON** **UPDATE** **CASCADE**, |
| 81 | [BookID] int **NOT** **NULL** **REFERENCES** [Book] |
| 82 | **ON** **DELETE** **CASCADE** |
| 83 | **ON** **UPDATE** **CASCADE**, |
| 84 | [Issue] datetime, |
| 85 | [Return] datetime, |
| 86 | [Available] bit **NOT** **NULL**, |
| 87 | ) |
| 88 | **GO** |

# Приложение. Скриншоты программы

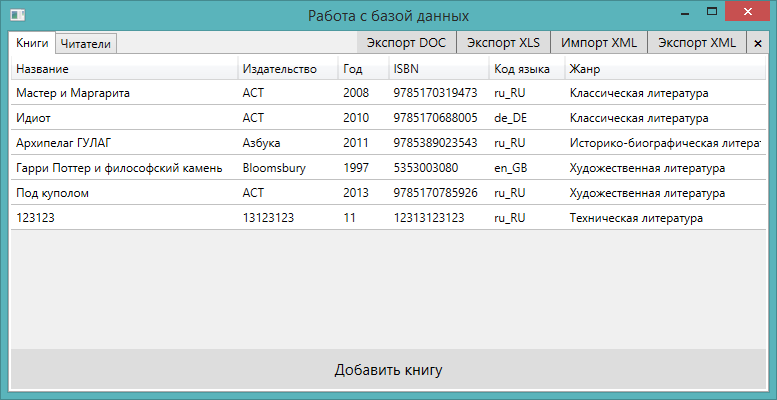


Рисунок 7. Главное окно программы

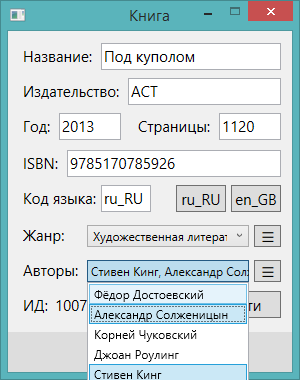


Рисунок 8. Окно редактирования книги

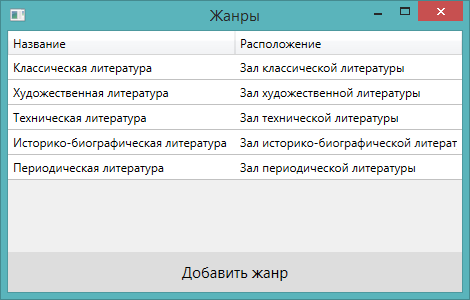


Рисунок 9. Список жанров

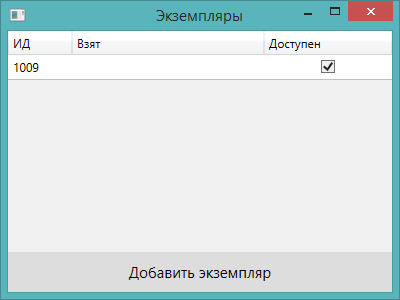


Рисунок 10. Список экземпляров

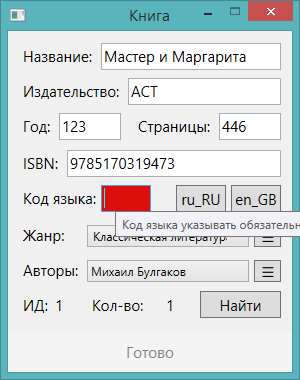


Рисунок 11. Обработка ошибок в полях ввода

1. Об этих классах подробнее написано в разделе «Разработка клиентского приложения. Структура клиента». [↑](#footnote-ref-1)