Министерство транспорта Российской Федерации
Федеральное агентство железнодорожного транспорта

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Дальневосточный государственный университет путей сообщения»

Естественно-научный институт
Кафедра «Вычислительная техника и компьютерная графика»

РАЗРАБОТКА ПРИКЛАДНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ БЮРО ТЕХНИЧЕСКОЙ ИНВЕНТАРИЗАЦИИ

Курсовая работа

КР.09.03.03. БД.08.00-БО921ПРИ–ПЗ

Исполнитель
студент, гр. БО921ПРИ_____ Д.Е. Дуда

Руководитель
старший преподаватель _____ А.А. Холодилов

Хабаровск 2020

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В современном мире происходит активная интеграция информационных технологий в жизнь среднего человека. В связи с этим, спрос на разработку прикладного программного обеспечения растет в геометрической прогрессии.

Реализация значительной части программного обеспечения невозможна без современных способов хранения и доступа к информации. Зачастую это обеспечивается интеграцией и отладкой базы данных.

Данная работа имеет высокую актуальность, поскольку позволяет рассмотреть в деталях процесс проектирования прикладного программного обеспечения с нуля до проекта, который не стыдно добавить в портфолио. Кроме того, благодаря этой работе можно получить бесценный опыт ведения проекта с момента зарождения идеи до презентации.

Основные задачи работы:

– анализ предметной области;

– проектирование базы данных;

– подключение базы данных к IDE;

– разработка программного обеспечения на выбранном языке программирования с интеграцией данных из базы данных проекта.

Главная цель работы – разработка прикладного программного обеспечения для бюро технической инвентаризации.

1 ТЕОРЕТИЧЕСКИЕ ИССЛЕДОВАНИЯ

1.1 Общее определение системы баз данных

Система баз данных — это компьютеризированная система хранения записей, т.е. компьютеризированная система, основное назначение которой — хранить информацию, предоставляя пользователям средства ее извлечения и модификации. К информации может относиться все, что заслуживает внимания отдельного пользователя или организации, использующей систему, иначе говоря, все необходимое для текущей работы данного пользователя или предприятия.

Система баз данных — это, по сути, не что иное, как компьютеризированная система хранения однотипных записей. Саму же базу данных можно рассматривать как подобие электронной картотеки, т.е. хранилище или контейнер для некоторого набора файлов данных, занесенных в компьютер.

На рисунке 1 показана весьма упрощенная схема системы баз данных. Здесь отражено четыре главных компонента системы: данные, аппаратное обеспечение, программное обеспечение и пользователи. Каждый из этих компонентов кратко рассматривается ниже.

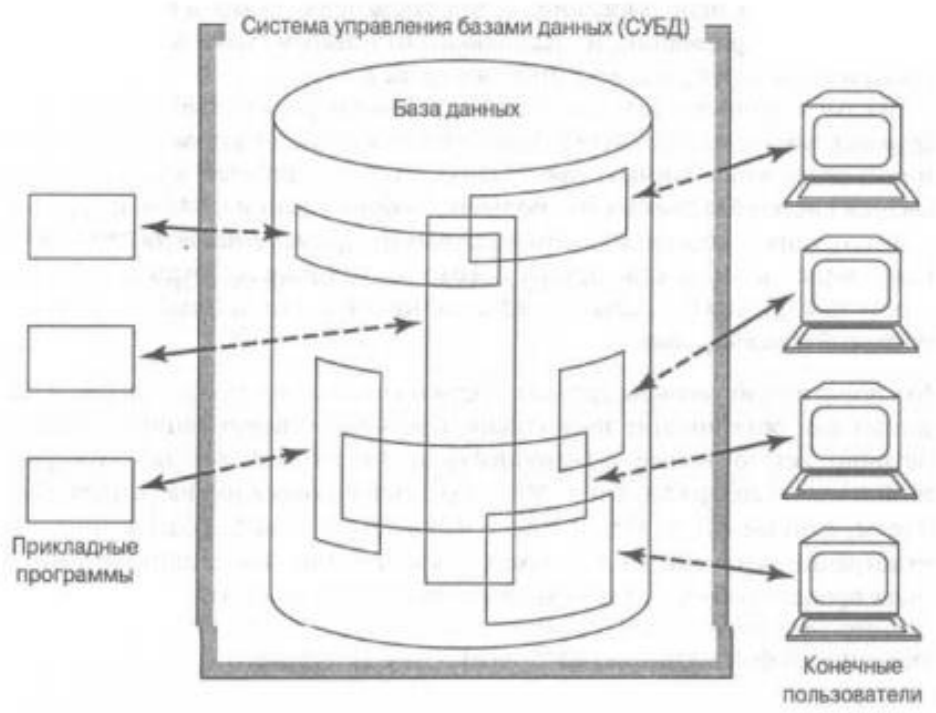

Рисунок 1 – Упрощенная схема системы БД

1.1.1 Данные

Системы с базами данных применяются как на самых малых компьютерах, так и на крупнейших мэйнфреймах. Несомненно, предоставляемые каждой конкретной системой средства в значительной мере зависят от мощности и возможностей базовой машины. В частности, на больших вычислительных машинах применяются в основном многопользовательские системы ("большие системы"), а на малых компьютерах, как правило, — однопользовательские системы ("малые системы").

Однопользовательская система (single-user system) — это система, в которой к базе данных может получить доступ одновременно только один пользователь, а многопользовательская система (multi-user system) — это такая система, в которой к базе данных могут получить доступ сразу несколько пользователей. Как показано на рисунке 1, исходя из соображений общности, мы обычно будем изучать именно второй вид систем, хотя с точки зрения пользователей между этими системами фактически не существует большого различия, поскольку основная цель многопользовательских систем состоит в том, чтобы позволить каждому отдельному пользователю работать с ней так, как если бы он работал с однопользовательской системой. Различия между этими двумя видами систем проявляются в их внутренней структуре и потому практически не видны конечному пользователю.

Обычно для упрощения предполагается, что все данные в системе хранятся в одной базе данных. Мы также будем придерживаться этого предположения, поскольку количество применяемых баз данных несущественно для всех дальнейших рассуждений. Однако на практике, даже при использовании малых систем, могут быть серьезные основания для распределения информации по нескольким отдельным базам данных. В общем случае данные в базе данных (по крайней мере, в больших системах) являются интегрированными и разделяемыми. Интеграция и разделение данных, представляют собой наиболее важные преимущества использования систем баз данных на "большом" оборудовании и, по меньшей мере, один из них— интеграция — является преимуществом их

применения и на "малом" оборудовании. Конечно, есть множество других преимуществ (даже на "малом" оборудовании), но о них речь пойдет ниже. Сначала следует объяснить, что понимается под терминами интегрированный и разделяемый.

Под понятием интеграции данных подразумевается возможность представить базу данных как объединение нескольких отдельных файлов данных, полностью или частично исключающее избыточность хранения информации. Например, база данных может содержать таблицу EMPLOYEE, включающий имена сотрудников, адреса, отделы, данные о зарплате и т.д., и таблицу ENROLLMENT, содержащий сведения о регистрации сотрудников на курсах обучения. Допустим, что для контроля процесса обучения необходимо знать отдел каждого зачисленного на курсы сотрудника. Совершенно очевидно, что нет необходимости включать такую информацию в файл ENROLLMENT, поскольку ее всегда можно получить из файла EMPLOYEE.

Под понятием «разделяемое» данных подразумевается возможность использования несколькими различными пользователями отдельных элементов, хранимых в базе данных. Имеется в виду, что каждый из пользователей сможет получить доступ к одним и тем же данным, возможно, даже одновременно (параллельный доступ). Такое разделение данных, с параллельным или последовательным доступом, частично является следствием того факта, что база данных имеет интегрированную структуру. В примере выше, информация об отделе в файле EMPLOYEE может совместно использоваться сотрудниками отдела кадров и отдела обучения. (Если база данных не является разделяемой, то ее иногда называют личной базой или базой данных специального назначения.)

Одним из следствий упомянутых выше характеристик базы данных (интеграции и разделяемости) является то, что каждый конкретный пользователь обычно имеет дело лишь с небольшой частью всей базы данных, причем обрабатываемые различными пользователями части могут произвольным образом перекрываться. Иначе говоря, каждая база данных воспринимается ее различными пользователями по-разному. Фактически, даже те два пользователя базы

данных, которые работают с одними и теми же частями базы данных, могут иметь значительно отличающиеся представления о них.

1.1.2 Аппаратное обеспечение

К аппаратному обеспечению системы относится следующее:

– тома вторичной (внешней) памяти (обычно это магнитные диски), используемые для хранения информации, а также соответствующие устройства ввода-вывода (дисководы и т.п.), контроллеры устройств, каналы ввода—вывода и т.д.;

– аппаратный процессор (или процессоры) вместе с оперативной (первичной) памятью, предназначенные для поддержки работы программного обеспечения системы баз данных (подробности приведены в следующем подразделе).

Не будем уделять много времени этому разделу. Во-первых, эти вопросы составляют достаточно обширную тему, которую нужно рассматривать отдельно. Во-вторых, проблемы, которые присущи в этой области, не являются специфическими исключительно для систем баз данных. И в-третьих, эти проблемы достаточно подробно освещены в других источниках.

1.1.3 Программное обеспечение

Между собственно физической базой данных (т.е. данными, которые реально хранятся на компьютере) и пользователями системы располагается уровень программного обеспечения, который можно называть по-разному: диспетчер базы данных (database manager), сервер базы данных (database server) или, что более привычно, система управления базами данных, СУБД (DataBase Management System — DBMS). Все запросы пользователей на получение доступа к базе данных обрабатываются СУБД. Все имеющиеся средства добавления файлов (или таблиц), выборки и обновления данных в этих файлах или таблицах также предоставляет СУБД.

Основная задача СУБД — дать пользователю базы данных возможность работать с ней, не вникая во все подробности работы на уровне аппаратного обеспечения. (Пользователь СУБД более отстранен от этих подробностей, чем прикладной программист, применяющий языковую среду программирования.)

Иными словами, СУБД позволяет конечному пользователю рассматривать базу данных как объект более высокого уровня по сравнению с аппаратным обеспечением, а также предоставляет в его распоряжение набор операций, выражаемых в терминах языка высокого уровня (например, набор операций, которые можно выполнять с помощью языка SQL)

Необходимо отметить еще две описанные ниже особенности.

– СУБД— это наиболее важный, но не единственный программный компонент системы. В числе других компонентов можно назвать утилиты, средства разра ботки приложений, средства проектирования, генераторы отчетов и диспетчер транзакций (transaction manager), или диспетчер обработки транзакций (transaction processing monitor — TP monitor);

– Термин СУБД также часто используется в отношении конкретных программных продуктов конкретных изготовителей, например, таких как DB2 Universal Database компании IBM. Иногда в тех случаях, когда конкретная копия подобного продукта устанавливается для работы на определенном компьютере, используется термин экземпляр. Безусловно, необходимо строго различать эти два понятия.

Следует отметить, что термин база данных часто используется даже тогда, когда на самом деле подразумевается СУБД (в одном из уже упомянутых толкований). Вот типичный пример: "База данных изготовителя X превосходит по производительности базу данных изготовителя К в два раза". Такое небрежное обращение с терминами предосудительно; тем не менее, оно очень широко распространено. (Проблема, естественно, заключается в том, что если СУБД называют базой данных, как же тогда называть саму базу данных?).

1.1.4 Пользователи

Пользователей можно разделить на три большие и отчасти перекрывающиеся группы.

Первая группа — прикладные программисты, которые отвечают за написание прикладных программ, использующих базу данных. Для этих целей применимы такие языки, как COBOL, PL/I, C++, Java или какой-нибудь высокоуров-

невый язык четвертого поколения. Прикладные программы получают доступ к базе данных посредством выдачи соответствующего запроса к СУБД (обычно это некоторый оператор SQL).

Подобные программы могут быть простыми пакетными приложениями или же интерактивными приложениями, предназначенными для поддержки работы конечных пользователей (см. следующий абзац). В последнем случае они предоставляют пользователям непосредственный оперативный доступ к базе данных через рабочую станцию, терминал или персональный компьютер. Большинство современных приложений относится именно к этой категории.

Вторая группа — конечные пользователи, которые работают с системой баз данных в интерактивном режиме, как указано в предыдущем абзаце. Конечный пользователь может получать доступ к базе данных, применяя одно из интерактивных приложений, упомянутых выше, или же интерфейс, интегрированный в программное обеспечение самой СУБД. Безусловно, подобный интерфейс также поддерживается интерактивными приложениями, но эти приложения не создаются пользователями-программистами, а являются встроенными в СУБД.

Большинство СУБД включает по крайней мере одно такое встроенное приложение, а именно — процессор языка запросов, позволяющий пользователю в диалоговом режиме вводить запросы к базе данных (их часто иначе называют предложениями или командами), например, с операторами SELECT или INSERT. Язык SQL представляет собой типичный пример языка запросов к базе данных. (Общепринятый термин язык запросов не совсем точно отражает рассматриваемое понятие, поскольку слово запрос подразумевает лишь выборку информации, в то время как с помощью этого языка выполняются также операции обновления, вставки, удаления и др.)

Кроме языка запросов, в большинстве систем дополнительно предоставляются специализированные встроенные интерфейсы, в которых пользователь в явном виде не использует предложения, или команды с такими операторами, как SELECT и INSERT. Работа с базой данных осуществляется за счет выбора пользователем необходимых элементов меню или заполнения требуемых полей

в предоставленных формах. Такие некомандные интерфейсы, основанные на меню и формах, облегчают работу с базами данных для тех, кто не имеет опыта работы с информационными технологиями (или ИТ; часто употребляется также сокращение ИС — информационные системы; эти понятия практически эквивалентны). Командный интерфейс, т.е. язык запросов, напротив, требует некоторого профессионального опыта работы с ИТ (но, безусловно, не такого большого, какой необходим для написания прикладных программ на языке программирования, подобном COBOL).

Однако командный интерфейс более гибок, чем некомандный, к тому же языки запросов обычно включают определенные функции, отсутствующие в интерфейсах, основанных на использовании меню или форм.

Третья группа — администраторы базы данных, или АБД. В связи с тем, что данные (как было отмечено выше) — это одна из главных ценностей предприятия, администратор должен разбираться в них и понимать нужды предприятия по отношению к данным на уровне высшего управляющего звена в руководстве предприятием. Сам администратор данных также должен относиться к этому звену. В его обязанности входит принятие решений о том, какие данные необходимо вносить в базу данных в первую очередь, а также выработка требований по сопровождению и обработке данных после их занесения в базу данных. Примером подобных требований может служить распоряжение о том, кто и при каких обстоятельствах имеет право выполнять конкретные операции над теми или иными данными. Другими словами, администратор данных должен обеспечивать защиту данных.

1.2 Общее определение базы данных

1.2.1 Перманентные данные

Обычно данные в базе данных называют перманентными или постоянно хранимыми (хотя иногда на самом деле они недолго остаются таковыми!). Под словом перманентные (persistent) подразумеваются данные, которые отличаются от других, более изменчивых данных, таких как промежуточные результаты,

входные и выходные данные, управляющие операторы, рабочие очереди, программные управляющие блоки и вообще все данные, временные (transient) по своей сути.

Точнее говоря, можно утверждать, что данные в базе являются перманентными, поскольку после того как они были приняты средствами СУБД для помещения в базу, их последующее удаление возможно лишь при использовании соответствующего явного запроса к базе данных, но не в результате какого-либо побочного эффекта от выполнения некоторой программы.

Подобный взгляд на понятие перманентности позволяет точнее определить термин база данных.

Кристофер Дж. Дейт в своей книге [1] дает следующее определение базы данных:

База данных — это некоторый набор перманентных (постоянно хранимых) данных, используемых прикладными программными системами какого-либо предприятия. Здесь слово предприятие— удобный общий термин для относительно независимой коммерческой, научной, технической или любой другой организации. Организация может состоять всего из одного человека (с небольшой частной базой данных), быть целой корпорацией или другой крупной организацией (с очень большой общей базой данных) либо представлять собой нечто среднее между этими крайними случаями.

Ниже приведено несколько примеров:

– промышленная компания;

– банк;

– больница;

– университет;

– министерство.

Любое предприятие неизбежно использует большое количество данных, связанных с его деятельностью. Это и есть перманентные данные, о которых шла речь в приведенном выше определении. Среди перманентных данных упомянутых предприятий обычно встречаются данные, перечисленные ниже:

– данные о продукции;

– бухгалтерские данные;

– данные о пациентах;

– данные о студентах;

– данные о планируемой деятельности.

Раньше вместо термина перманентные данные использовался термин операционные данные. Старый термин первоначально акцентировал внимание на особом значении оперативных, или производственных, приложений баз данных, т.е. рутинных, часто выполняющихся приложений, предназначенных для поддержки повседневной работы предприятия (например, приложений для поддержки операций зачисления и списания средств на счетах в банковской системе).

Для среды такого рода в последнее время используется термин оперативная обработка транзакций (On-Line Transaction Processing— OLTP). Однако теперь базы данных все чаще применяются и в приложениях другого рода, например, в приложениях поддержки принятия решений (decision support), и термин операционные данные для них уже не подходит. На практике сегодняшние предприятия используют две отдельные базы данных — с операционными данными и с данными для поддержки принятия решений; последнюю обычно называют хранилищем данных (data warehouse).

В хранилищах данных часто содержится агрегированная информация (например, итоговые и средние значения), которая, в свою очередь, периодически (например, раз в сутки или раз в неделю) извлекается из операционной базы данных.

1.2.2 Сущности и связи

Рассмотрим более подробно пример некоторой промышленной компании (допустим, она имеет название KnowWare, Inc.). Обычно подобному предприятию требуется записывать информацию об имеющихся проектах (Projects), используемых в этих проектах деталях (Parts), поставщиках (Suppliers) деталей, складах (Warehouses), на которых хранятся детали, служащих (Employees), ра-

ботающих над проектами и т.д. Проекты, детали, поставщики и т.д. представляют собой основные сущности (entity), о которых компании KnowWare, Inc. необходимо хранить информацию.

В теории баз данных термин сущность обычно используется для обозначения любого различимого объекта, который может быть представлен в базе данных (рисунок 2).

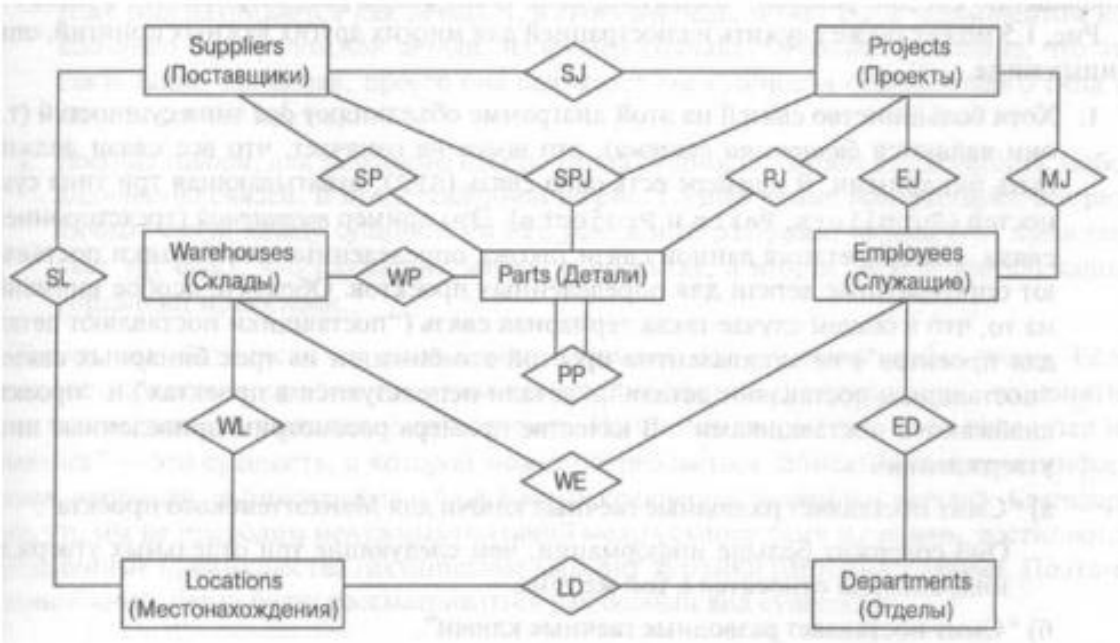

Рисунок 2 – Пример диаграммы "сущность-связь" (ER-диаграммы) для базы данных компании KnowWare, Inc.

Кроме этих основных сущностей (в данном примере это поставщики, детали и т.д.), имеются еще и связи (relationship) между ними, которые объединяют эти основные сущности. На рисунке 2 связи представлены ромбами с соединительными линиями.

Например, между поставщиками и деталями существует связь SP (сокращение от Shipments/Parts): каждый поставщик поставляет определенные детали, и наоборот, каждая деталь поставляется определенными поставщиками. (Точнее, каждый поставщик поставляет определенные виды деталей, и каждый вид деталей поставляется определенными поставщиками.) Аналогично, детали используются в проектах, а для реализации проектов требуются детали (связь PJ —

сокращение от Parts/proJects); детали хранятся на складах, а склады хранят детали (связь WP — сокращение от Warehouses/Parts) и т.д. Обратите внимание, что эти связи — бинарные (или двухсторонние), т.е. их можно прослеживать в любом направлении. В частности, используя связь SP между поставщиками и деталями, можно ответить на следующие вопросы:

– задан поставщик, и требуется определить поставляемые им детали;

– задана деталь, и необходимо найти поставщиков, предоставляющих такую деталь.

Очень важно то, что эта связь (как и другие связи, представленные на рисунке 2) является такой же неотъемлемой составляющей данных предприятия, как и основные сущности. Поэтому связи должны быть представлены в базе данных наравне с основными сущностями предметной области.

Следует отметить, что на рисунке 2 приведен пример информационной структуры, которую принято называть (по очевидным причинам) диаграммой "сущность—связь" (сокращенно ER-диаграммой).

Хотя большинство связей на этой диаграмме объединяют два типа сущностей (т.е. они являются бинарными связями), это вовсе не означает, что все связи должны быть бинарными. В примере есть одна связь (SPJ), охватывающая три типа сущностей (Suppliers, Parts и Projects). Это пример тернарной (трехсторонней) связи. Интерпретация данной связи такова: определенные поставщики поставляют определенные детали для определенных проектов.

Обратите особое внимание на то, что в общем случае такая тернарная связь ("поставщики поставляют детали для проектов") не эквивалентна простой комбинации из трех бинарных связей: "поставщики поставляют детали", "детали используются в проектах" и "проекты снабжаются поставщиками".

Ложной характерной особенностью реляционных баз данных является то, что основные сущности и связи между ними представлены с помощью отношений (relation) или, иными словами, с помощью таблиц. Но следует учитывать, что термин связь (relationship), который используется в текущем разделе, и тер-

мин отношение, применяемый в контексте реляционных баз данных, обозначают разные понятия.

Применяемый в оригинале англоязычный термин statement имеет два разных значения: в данном случае он указывает на утверждение, касающееся некоторого факта (которое в логике принято называть высказыванием), а в другом контексте может служить синонимом термина command (команда).

Вообще говоря, для заданного набора типов сущностей может существовать любое количество связей. В представленной на рисунке 2 диаграмме присутствуют две раз личные связи между сущностями Projects и Employees: первая (EJ) представляет тот факт, что служащие заняты в проектах, а вторая (MJ) — что служащие управляют проектами.

Теперь мы убедились, что связь можно понимать как сущность особого типа. Если сущность определена как "нечто, о чем необходимо хранить информацию", то понятие связи вполне подходит под такое определение. Например, связь "деталь P4 находится на складе W8" — это сущность, о которой может потребоваться записать некоторую информацию, например, зафиксировать в базе данных количество указанных деталей. Благодаря тому, что мы не проводим ненужных различий между сущностями и связями, достигаются определенные преимущества (их описание выходит за рамки настоящей главы).

1.2.3 Свойства

Как было только что отмечено, сущность — это то, о чем необходимо хранить информацию. Отсюда следует, что сущности (а значит, и связи) имеют некоторые свойства (properties), соответствующие тем данным о них, которые необходимо хранить в базе.

Например, поставщики имеют определенное место расположения, детали характеризуются весом, проекты — очередностью выполнения, закрепление служащих за проектами имеет начальную дату и т.д.

В базе данных должны быть представлены именно эти свойства. Например, в базе данных может быть таблица s, представляющая тип сущности "поставщики", а в этой таблице может присутствовать тип поля CITY (город), пред-

ставляющий свойство "место расположения". В общем случае свойства могут быть как простыми, так и сложными, причем настолько простыми или сложными, насколько это потребуется.

Например, свойство "местонахождение поставщика" — относительно простое: оно состоит только из названия города и может быть описано как простая символьная строка. В противоположность этому, сущность "склад" может иметь свойство "схема этажей" с достаточно сложной структурой, включающей архитектурный план здания, дополненный соответствующим текстовым описанием.

Существует и другой, не менее важный, подход к трактовке данных и баз данных. Слово данные (data) происходит от латинского слова "дано" (напрашивается продолжение "требуется доказать"). Отсюда следует, что данные на самом деле являются заданными фактами, из которых можно логически вывести другие факты. (Получение производных фактов из заданных — это именно то, что выполняет СУБД, обслуживая запросы пользователя.)

"Заданный факт", в свою очередь, соответствует тому, что в логике называется истинным высказыванием. Например, высказывание "Поставщик с номером S1 находится в Лондоне" вполне может оказаться истинным. Высказыванием в логике называется такое утверждение, которое может быть недвусмысленно определено как истинное или ложное. Например, "Вильям Шекспир написал Гордость и предубеждение" — это высказывание (как видно, ложное). Отсюда следует, что в действительности база данных — это множество истинных высказываний.

Итак, уже было отмечено, что продукты на основе языка SQL заняли доминирующее положение на рынке. Одной из причин этого является то, что они основаны на использовании формальной теории, называемой реляционной моделью данных, а эта теория, в свою очередь, поддерживает указанную выше интерпретацию данных и баз данных весьма просто (фактически почти тривиально). Конкретнее, реляционная модель характеризуется описанными ниже особенностями.

– данные представлены посредством строк в таблицах, и эти строки могут быть не посредственно интерпретированы как истинные высказывания;

– для обработки строк данных предоставляются операторы, которые напрямую поддерживают процесс логического вывода дополнительных истинных высказываний из существующих высказываний.

Однако реляционная модель — не единственная возможная модель данных. Существуют и другие модели, хотя многие из них отличаются от реляционной модели только тем, что они в определенной степени приспособлены для специальных случаев, а не строго основаны на формальной логике, в отличие от реляционной модели.

Возникает вопрос: что же такое модель данных? Модель данных — это абстрактное, самодостаточное, логическое определение объектов, операторов и прочих элементов, в совокупности составляющих абстрактную машину доступа к данным, с которой взаимодействует пользователь. Упомянутые объекты позволяют моделировать структуру данных, а операторы — поведение данных. Используя это определение, можно эффективно разделить понятия модели данных и ее реализации.

Реализация (implementation) заданной модели данных — это физическое воплощение на реальной машине компонентов абстрактной машины, которые в совокупности составляют эту модель. Короче говоря, модель — это то, о чем пользователи должны знать, а реализация – это то, чего пользователи не должны знать.

Как следует из указанного выше, различие между моделью и ее реализацией в действительности является просто частным случаем знакомого нам различия между логическим и физическим определением данных (хотя и очень важным частным случаем). Однако, как это ни прискорбно, разработчики многих современных систем баз данных (даже систем, которые претендуют на то, чтобы называться реляционными) не проводят такого четкого различия, как требуется.

Действительно, по-видимому, это весьма распространенный недостаток, состоящий в непонимании таких различий и важности их учета. И вследствие этого все чаще наблюдается расхождение между принципами создания баз данных (указывающими, какими должны быть системы баз данных) и практикой их реализации (тем, какими они являются на самом деле).

В завершение этого раздела необходимо отметить, что в действительности термин модель данных используется в литературе в двух различных толкованиях. Первое определение этого термина описано выше. Второе состоит в следующем: модель данных (во втором смысле) представляет собой модель перманентных данных некоторого конкретного предприятия (например, промышленной компании KnowWare, Inc., упоминаемой выше в этом разделе).

Таким образом, различия между этими двумя толкованиями можно охарактеризовать, как описано ниже.

Модель данных в первом значении подобна языку программирования (причем достаточно абстрактному), конструкции которого могут быть использованы для решения широкого круга конкретных задач, но который сам по себе не имеет четкой связи с какой-либо из этих конкретных задач.

Модель данных во втором значении подобна конкретной программе, написанной на таком языке. Иначе говоря, модель данных во втором значении использует средства, предоставляемые некоторой моделью (рассматриваемой в первом значении) и применяет их для решения конкретной проблемы. Ее можно рассматривать как некоторое конкретное приложение некоторой модели в первом значении.

Как уже было сказано выше, системы с поддержкой SQL, или просто системы SQL, в настоящее время стали преобладающими на рынке баз данных, и одной из важных причин подобного состояния дел является именно то, что такие системы основаны на реляционной модели данных. Поэтому не удивительно, что в неформальном общении системы SQL часто называют просто реляционными системами.

Кроме того, подавляющее большинство научных исследований в области баз данных в течение последних 30 лет было посвящено (иногда косвенно) именно этой модели. Фактически, введение реляционной модели в 1969 и 1970 годах было, несомненно, наиболее важным событием во всей истории развития теории баз данных.

Итак, кратко и не совсем точно можно определить, что реляционная система — это система, основанная на описанных ниже принципах.

– данные рассматриваются пользователем как таблицы (и никак иначе);

– пользователю предоставляются операторы (например, для выборки данных), позволяющие генерировать новые таблицы на основании уже существующих.

Причина, по которой такие системы называют реляционными, состоит в том, что английский термин "relation" (отношение), по сути, представляет собой общепринятое математическое название для таблиц.

Поэтому на практике термины отношение и таблица в большинстве случаев можно считать синонимами, по крайней мере, для неформальных целей.

Возможно, следует добавить, что причина, несомненно, заключается не в том, что термин отношение "по существу — просто формальное название" для связи (relationship) в терминах диаграмм "сущность-связь".

На самом деле между реляционными системами и подобными диаграммами существует совсем незначительная прямая зависимость, как отмечено в данном разделе. Как уже упоминалось, в дальнейшем будет дано более точное определение, а пока мы будем использовать приведенное выше. Теперь мы можем различать реляционные и нереляционные Системы по следующим признакам. Как уже отмечалось, пользователь реляционной системы видит данные в виде таблиц и никак иначе. Пользователь нереляционной системы, напротив, видит данные, представленные в других структурах— либо вместо таблиц реляционной системы, либо наряду с ними.

Для работы с этими другими структурами применяются иные операции. В частности, в иерархической системе (например, в СУБД IMS фирмы IBM) дан-

ные представляются пользователю в форме набора древовидных структур (иерархий), а среди операций работы с иерархическими структурами есть операции перемещения (навигации) по иерархическим указателям, позволяющие переходить вверх и вниз по ветвям деревьев.

Реляционные системы, как мы видели, не имеют таких указателей, и это очень важная их отличительная особенность (по крайней мере, в них отсутствуют указатели, видимые для пользователя, иными словами, указатели на уровне модели, но могут быть предусмотрены указатели на уровне физической реализации).

На основании изложенного можно сделать вывод, что системы баз данных могут быть легко распределены по категориям в соответствии со структурами данных и операциями, которые они предоставляют пользователю.

2 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ И ПОСТАНОВКА ЗАДАЧИ

Землевладения всегда являлись неотъемной частью истории человечества. Испокон веков правители государств сражались за любые незанятые клочки Земли, стремясь расширить свое влияние за счет расширения своих владений.

С развитием технологий и промышленности, а также неизбежным ростом социального уровня развития, приоритет сменился с объема землевладения на его качество и обустроенность. Именно поэтому в наше время особенно актуальна современная система учета и описания земельных участков и их состава.

Такая система позволила бы организовать быстрый доступ к любым земельным участкам, на которые был оформлен технический паспорт и иные сопутствующие документы, а также предоставить возможность для сравнения и структуризации земельных участков.

В базе данных должны храниться данные о местоположении земельного участка, его площади, постройках на его территории. Объект базы данных должен быть описан с надлежащим качеством.

Необходимо разработать прикладное программное обеспечение деятельности отдела учета землевладений бюро технической инвентаризации на основе соответствующей базы данных.

В частности, к программному обеспечению применяются следующие требования:

– качественный графический интерфейс;

– возможность добавлять, удалять и изменять записи о землевладениях;

– возможность выгрузки главной таблицы в лист Microsoft Excel;

– функция поиска по записям;

– наличие словарей данных.

3 ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ

Процесс проектирования базы данных есть последовательность переходов от неформального словесного описания информационной структуры предметной области к формализованному описанию объектов предметной области в терминах некоторой модели.

3.1 ER-диаграмма

В первую очередь, необходимо провести концептуальное проектирование БД – создание абстрактной структуры БД с помощью ER-диаграммы (рисунок 3).

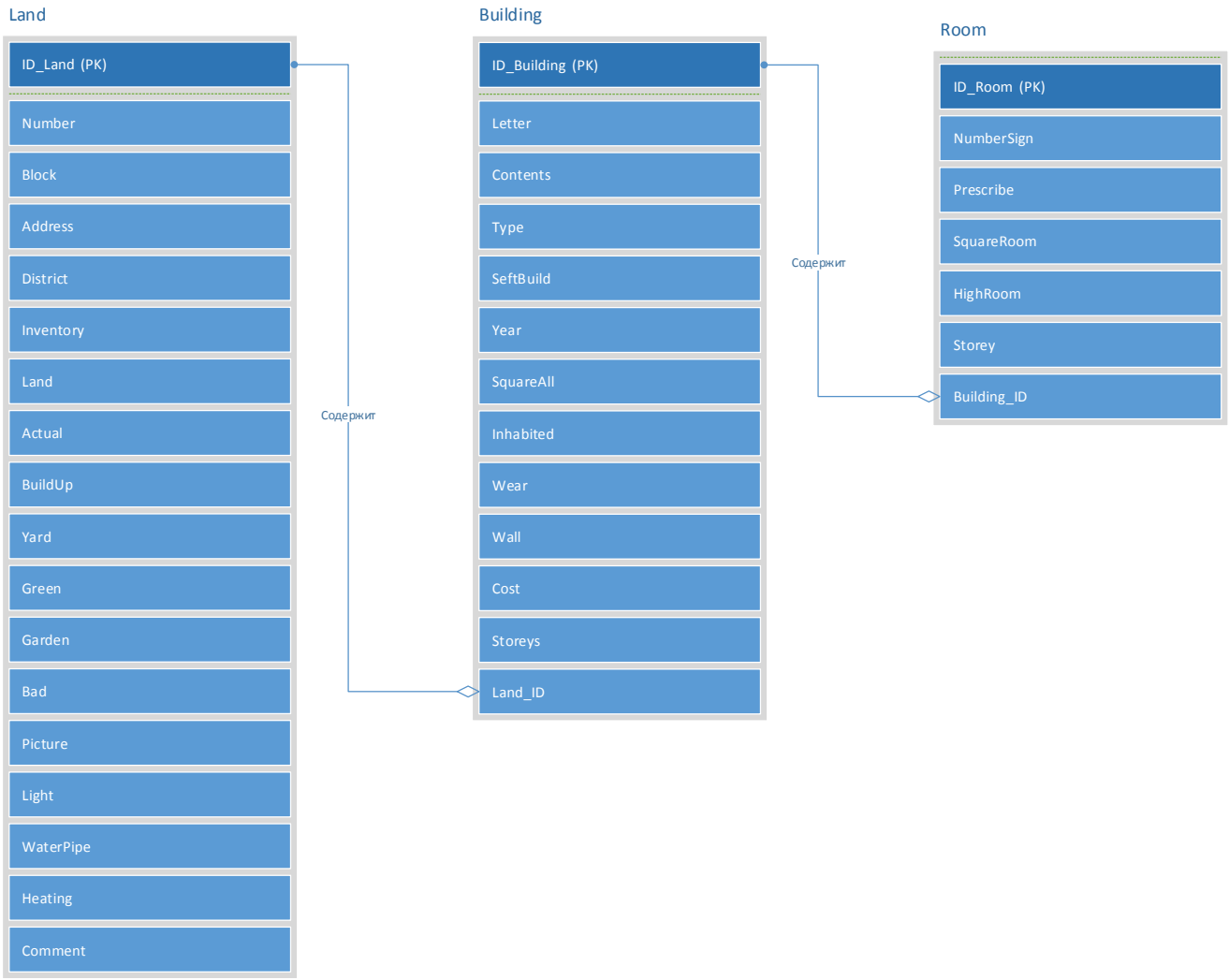

Рисунок 3 – Концептуальное проектирование

Связь между сущностями Land и Building, а также между сущностями Building и Room представляет собой связь один ко многим.

Следующим шагом необходимо провести последовательную нормализацию концептуальной модели к третьей нормальной форме, избавляясь от частичных и транзитивных зависимостей. В результате получена схема, построенная с помощью MySQL Workbench (рисунок 4).

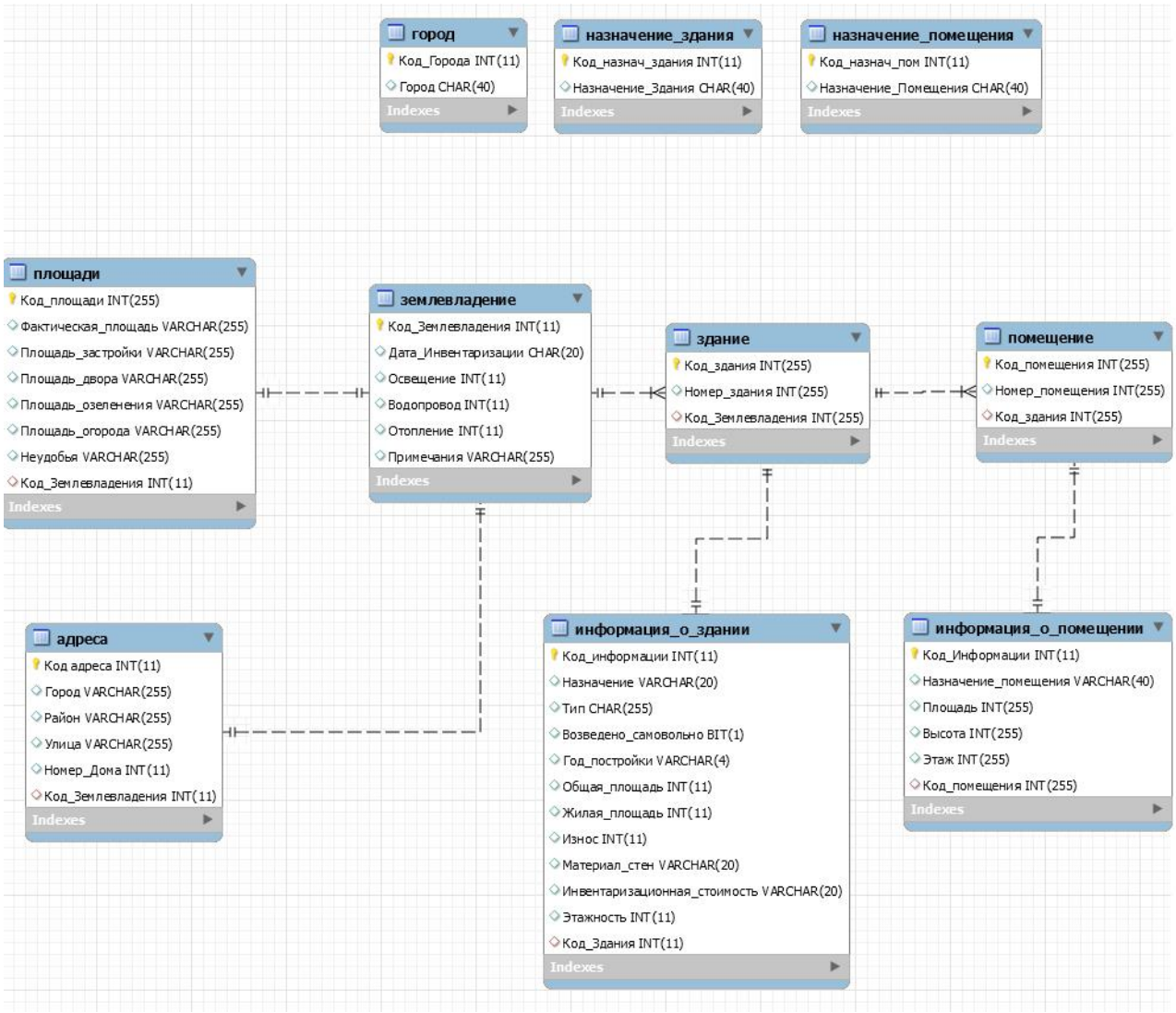

Рисунок 4 – Итоговая ER-диаграмма базы данных

Кроме того, требуется добавить справочные таблицы «Город», «Назначение_здания» и «Назначение_помещения», необходимые для формирования словарей данных. Использование MySQL Workbench позволяет, с помощью функции «Forward Engineer», получить необходимые SQL-запросы для формирования на сервере требуемой базы данных.

3.2 DFD-диаграмма

Данная диаграмма призвана потоки данных в системе, позволяет произвести структурный анализ информационной системы.

Диаграмма на рисунке 5 описывает потоки данных после двух внешних запросов – запроса на удаление и просмотр записи.

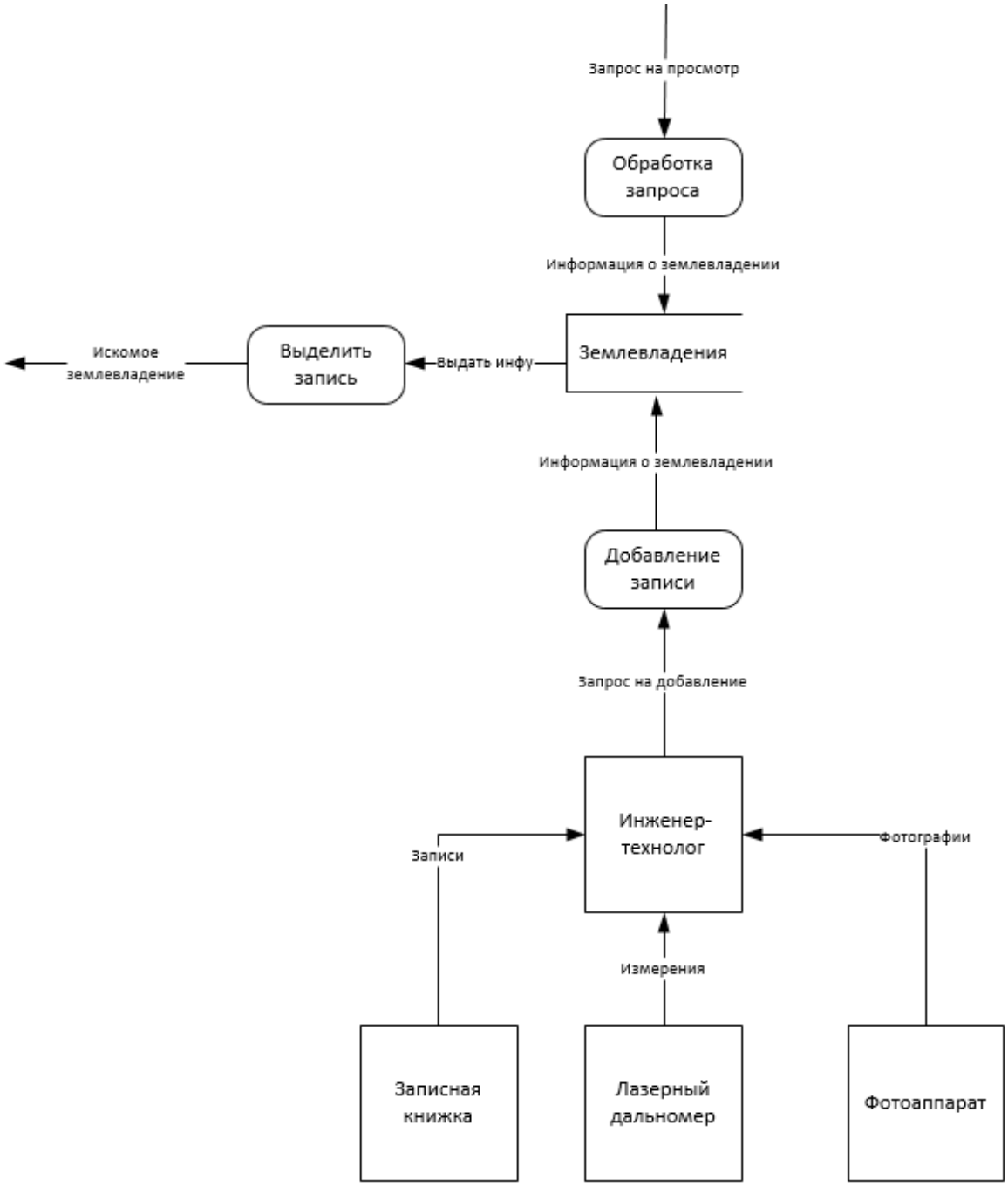

Рисунок 5 – DFD-диаграмма

Для построения диаграммы использованы средства Microsoft Visio.

3.3 EPC-диаграмма

Нотация EPC используется для описания процессов нижнего уровня. Диаграмма EPC представляет собой упорядоченную комбинацию событий и функций в выделенной информационной системе. Диаграмма на рисунке 6 описывает событие просмотра информации о землевладении.

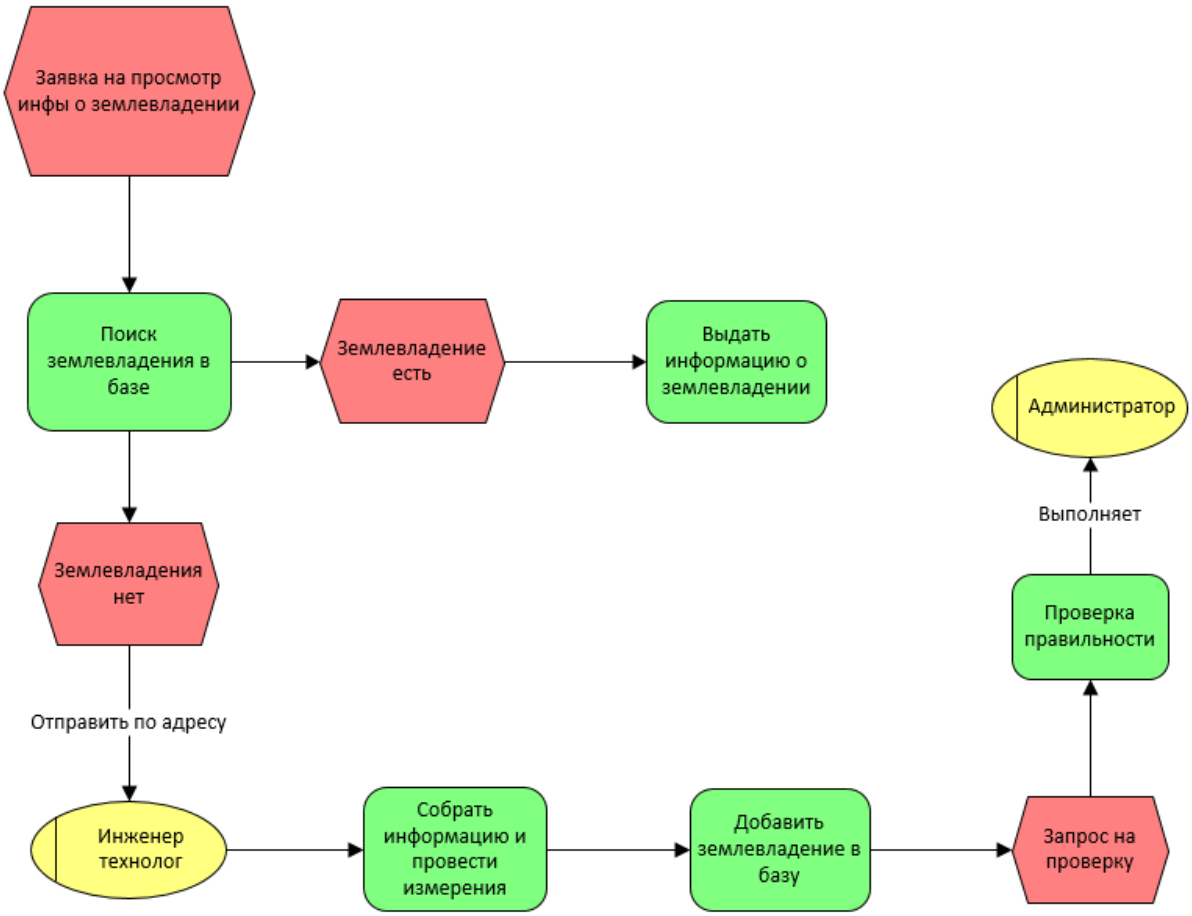

Рисунок 6 – EPC-диаграмма

Для построения диаграммы использованы средства Microsoft Visio.

4 РАЗРАБОТКА ПРИКЛАДНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Разработка окон приложения ведется в два этапа – разметка графического интерфейса с помощью языка разметки XAML (front_end) и формирование кода на языке программирования C# (back_end), обеспечивающего работу окна.

Визуальный стиль приложения основан на контрасте темно-серого, кораллового и белого цветов.

Здесь и далее элемент «TextBox» языка разметки XAML, предназначенный для пользовательского ввода текста, называется текстовым полем.

Дизайн многих элементов языка разметки в данном приложении основан на шаблонах пакета «MaterialDesign», который находится в свободном доступе для пользователей Microsoft Visual Studio. Данный пакет позволяет создавать современный и удобный графический интерфейс с минимальными затратами времени и труда, а также имеет широкий список настроек на любой вкус.

4.1 Разработка окна авторизации – класс Login

4.1.1 Разметка интерфейса

В разметке окна авторизации, как и во многих других окнах приложения, выбран стиль окна без границ, а также запрет на перемещение.

В разметку (рисунок 7) включены:

– изображение иконки приложения, созданной в Adobe Photoshop CC;

– текст, с просьбой выполнить вход;

– два текстовых поля для ввода логина;

– поле «PasswordBox», с сокрытием символов, для ввода пароля;

– кнопка «Отмена»;

– кнопка «Войти»;

– текст, с сообщением о неверном логине или пароле (на рисунке 5 скрыт).

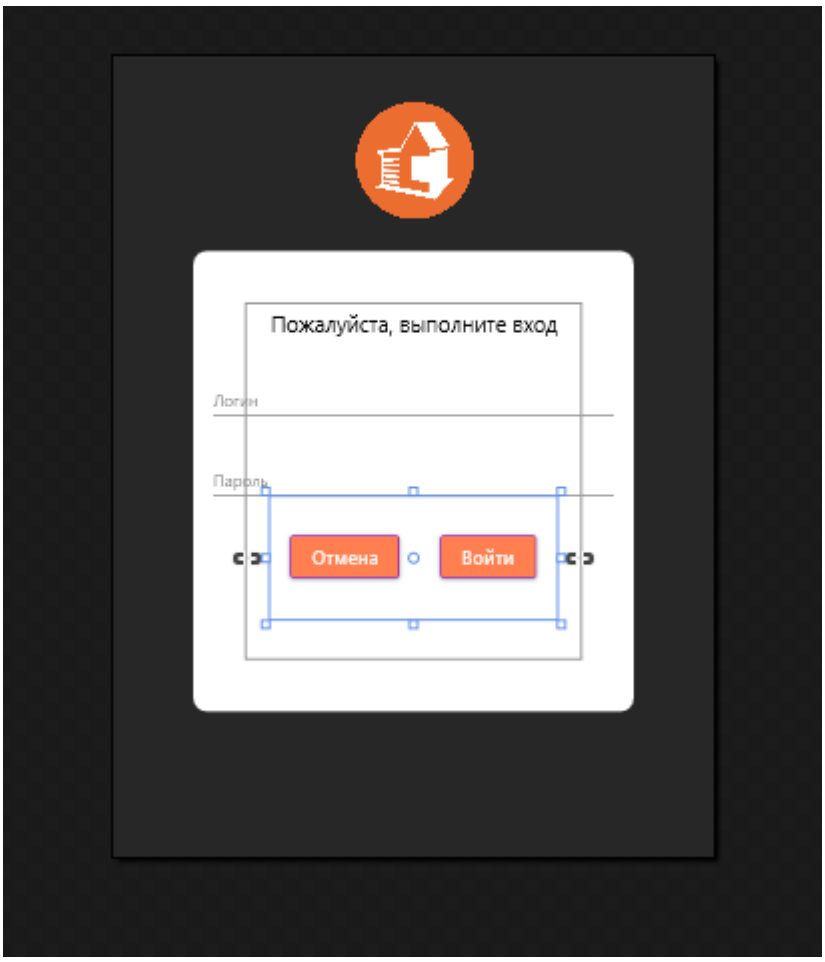

Рисунок 7 – Разметка окна авторизации

Требуется присвоить элементам кнопок и текстовых полей имена и создать обработчики событий нажатия на кнопку.

4.1.2 Выдержки из кода

Основная функция окна авторизации – проверка логина/пароля, которая производится по нажатию кнопки «Войти» (листинг 1).

Листинг 1 – Проверка логина и пароля

```
if(LoginText.Text == "root" && PasswordText.Password == "Dragon2012")
            {
             Invalid_info.Text = "Успешный вход";
             this.Close();
             the_window.Show();
            }
            else
            {
             Invalid_info.Text = "Введен неверный логин/пароль";
            }
```

Как видно, с помощью функции «if» производится простая проверка на совпадение данных с открытием главного окна приложения в случае успеха и вывод сообщения об ошибке в противном случае.

4.1.3 Конечный интерфейс

Конечный интерфейс окна представлен на рисунке 8.

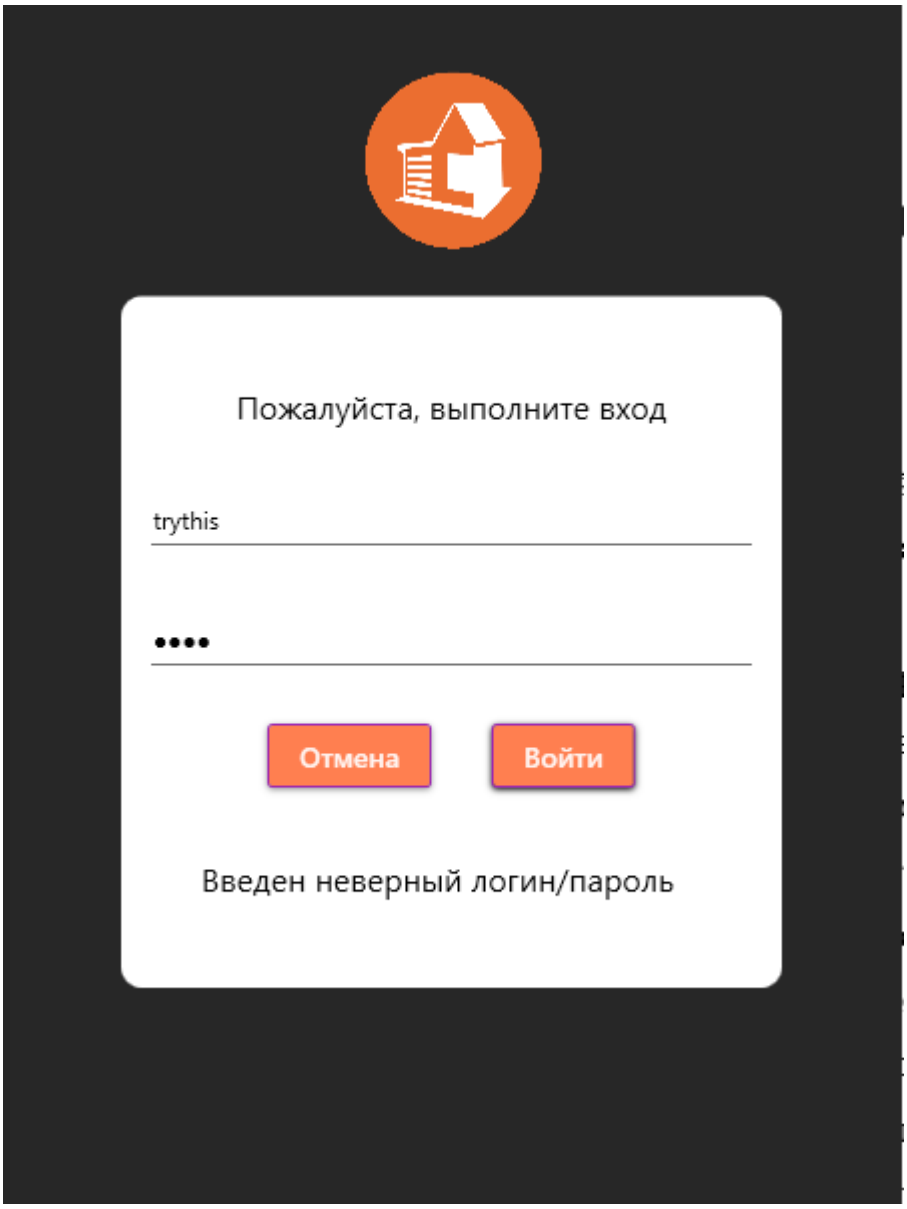

Рисунок 8 – Интерфейс окна авторизации

Как видно, при неправильном вводе логина/пароля выводится сообщение об ошибке авторизации.

4.2 Разработка главного окна приложения – класс MainWindow

4.2.1 Разметка интерфейса

В разметке окна авторизации, как и во многих других окнах приложения, выбран стиль окна без границ, а также запрет на перемещение.

В разметку (рисунок 9) включены

– кастомные кнопки сворачивания и закрытия приложения;

– текстовое поле для ввода информации для поиска;

– меню словарей;

– поле «DataGrid» для вывода информации о землевладениях;

– кнопки для добавления, удаления и изменения землевладений;

– кнопка для экспорта таблицы в документ MS Excel;

– текст с сообщением об ошибке (на рисунке 7 скрыт).

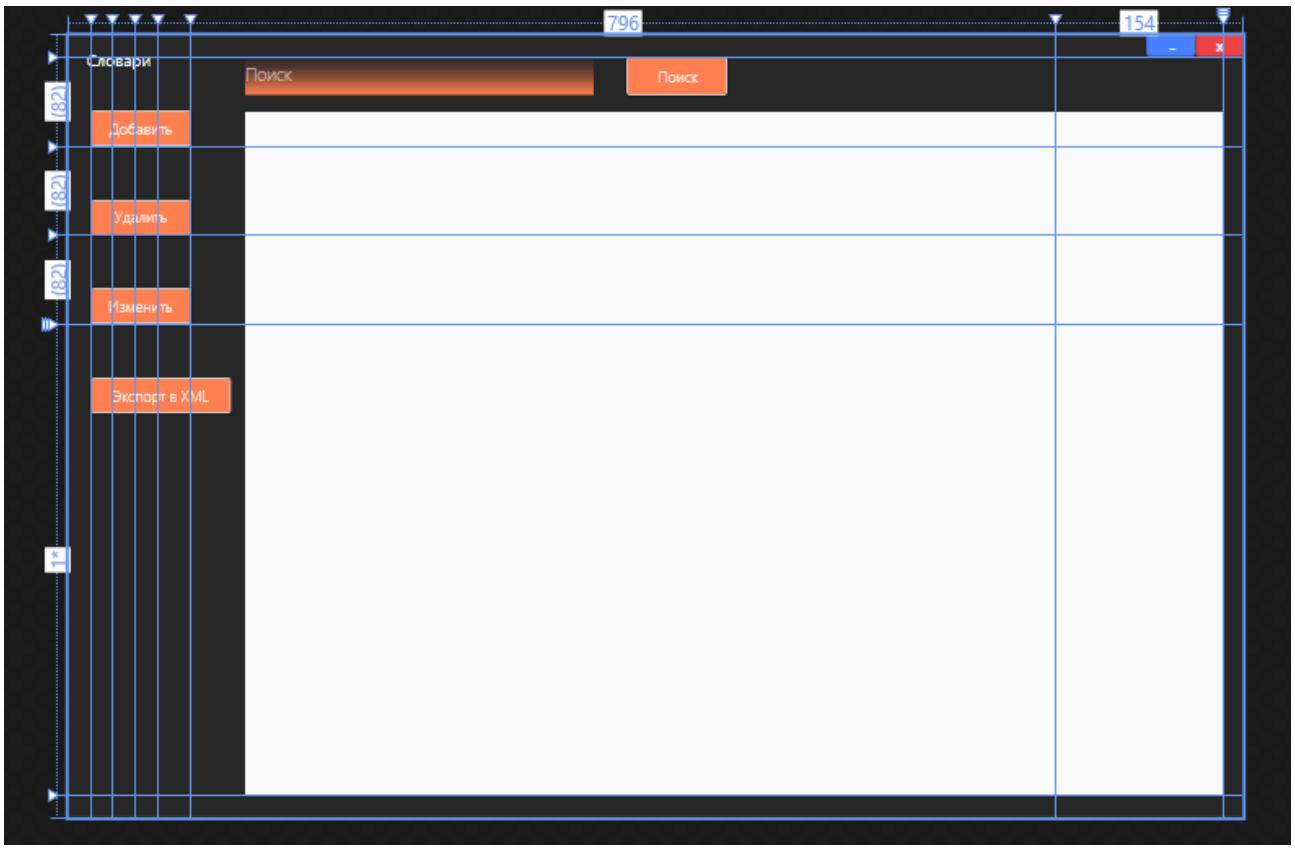

Рисунок 9 – Разметка главного окна

Требуется присвоить элементам кнопок и других полей имена и создать обработчики событий нажатия на кнопку.

4.2.2 Выдержки из кода

Основная функция главного окна – вывод информации о землевладении из базы данных (листинг 2).

Листинг 2 – Вывод информации

```
string connection = "server = localhost; user=root; database=kursach; password=Dragon2012";

            MySqlConnection connector = new MySqlConnection(connection);
```

```
            connector.Open();

             dt = new DataTable();

            string Query = "SELECT землевладение.Код_Землевладения AS '#', " +
                "DATE_FORMAT(Дата_Инвентаризации, '%D,%M,%Y') AS 'Дата инвентаризации', CON-
CAT('г. ', адреса.Город,', район ',адреса.Район, ', ул. ', адреса.Улица, ', д. '," +
                "адреса.Номер_Дома) AS Адрес, Примечания from землевладение RIGHT JOIN адре-
са ON землевладение.Код_Землевладения = адреса.Код_Землевладения;";

            MySqlCommand command = new MySqlCommand(Query, connector);
            dt.Load(command.ExecuteReader());

            TableView.DataContext = dt;
            TableView.ItemsSource = dt.DefaultView;
```

Для реализации архитектуры клиент-сервер требуется задать данные о соединении, создать элемент класса MySqlConnection, открыть соединение, создать таблицу DataTable для загрузки данных, подготовить запрос, отправить его на сервер с помощью открытого соединения и получить результат.

4.2.3 Конечный интерфейс

Конечный интерфейс окна представлен на рисунке 10.

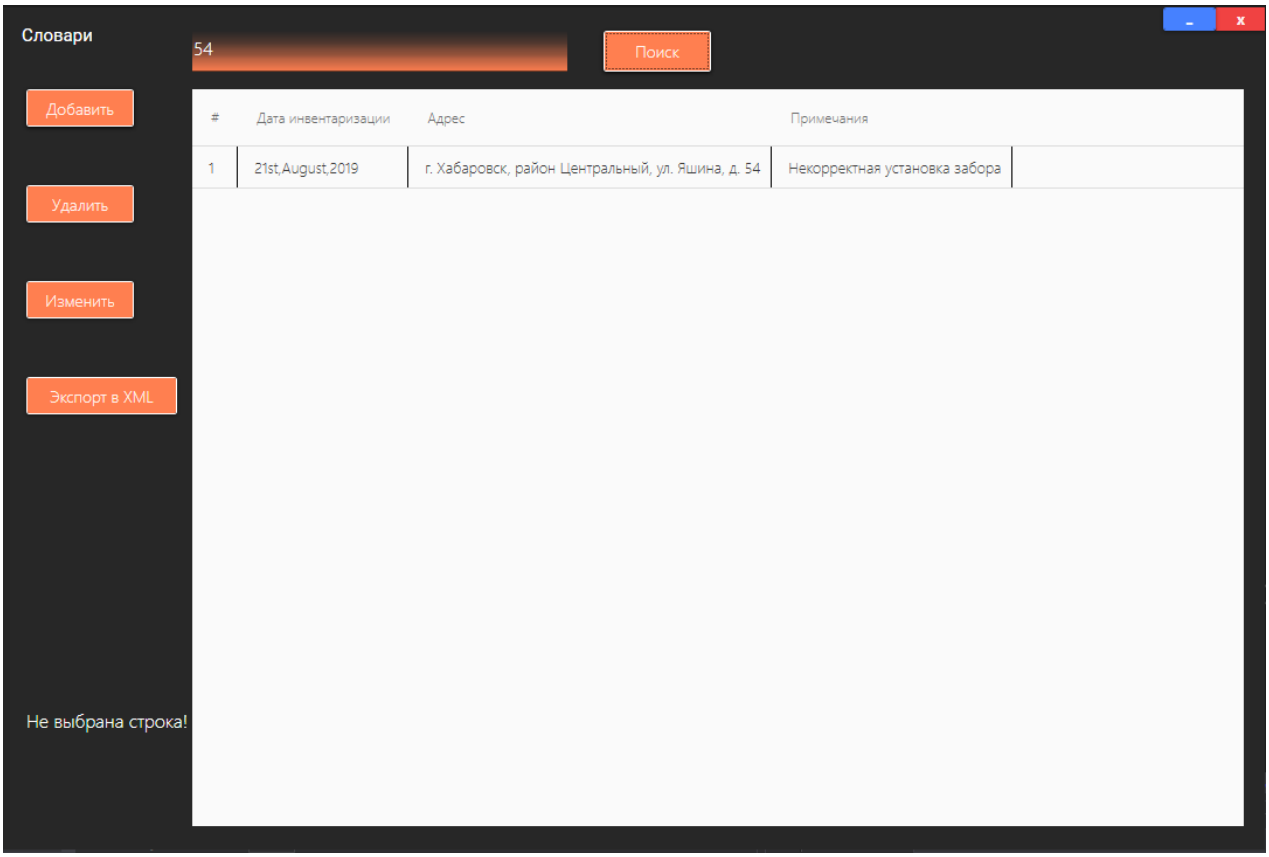

Рисунок 10 – Интерфейс главного окна

Как видно, если не выбрать строку на удаление/изменение, но нажать на соответствующую кнопку, выведется сообщение об ошибке.

4.3 Разработка диалогового окна словарей – класс Dictionary

4.3.1 Разметка интерфейса

В разметку (рисунок 11) включены:

– кнопки для добавления, удаления, изменения и поиска записей;

– поле «DataGrid» для вывода записей;

– кнопки ввода и отмены ввода (по умолчанию скрыты);

– текстовое поле для ввода новой позиции в словаре (по умолчанию скрыто).

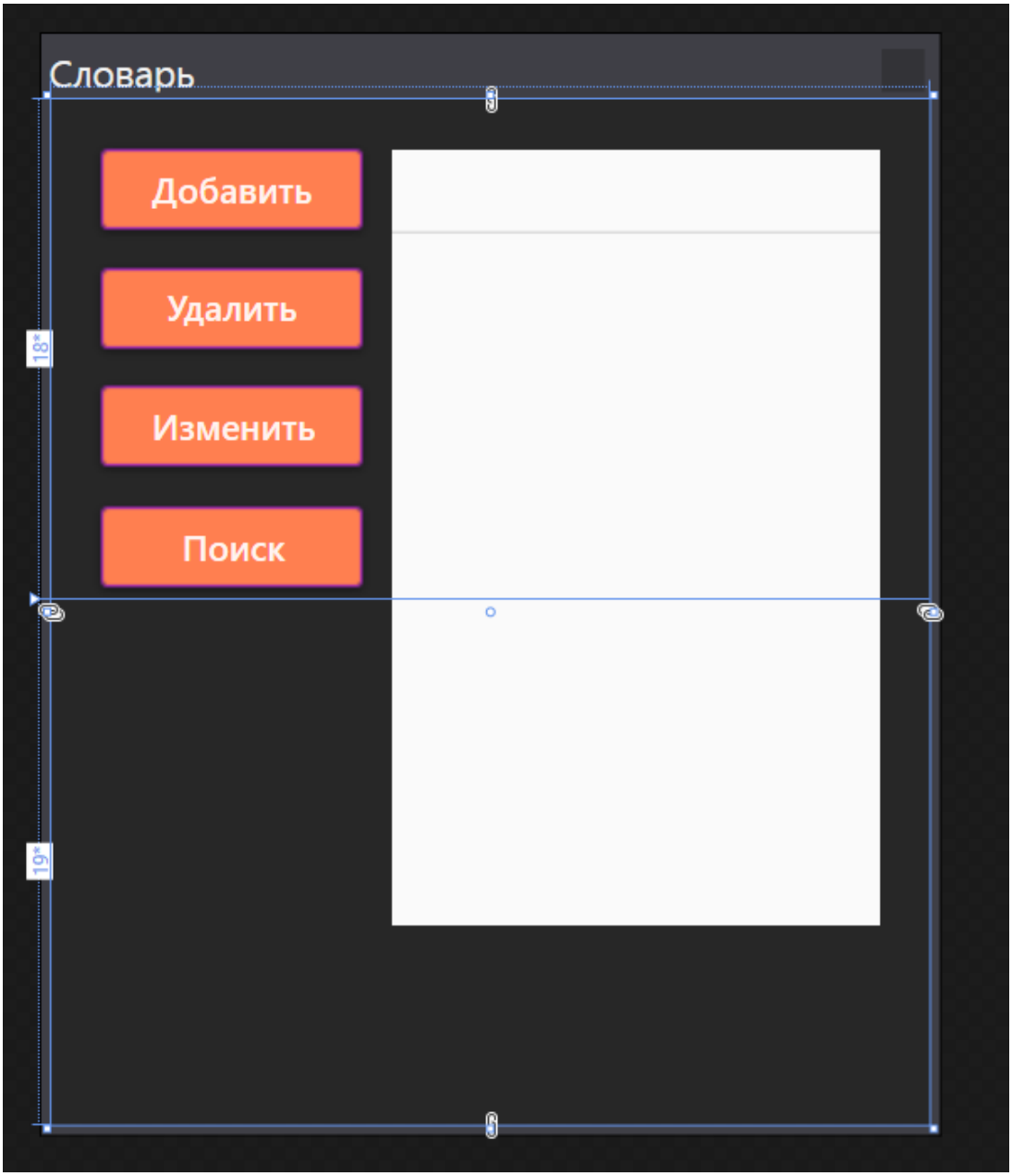

Рисунок 11 – Разметка окна словарей

4.3.2 Выдержки из кода

Окно словарей является универсальным для всех трех словарей приложения (город, назначение помещения и назначение здания), тип словаря передается из класса MainWindow в конструктор класса Dictionary (листинг 3), затем тип словаря сохраняется в приватную переменную DictionaryType, на значении которой основывается – какой тип словарей требуется вывести на экран и с какие SQL-запросы использовать для работы интерфейса (листинг 4).

Листинг 3 – Конструктор класса Dictionary

```
public Dictionary(int _type)
        {
            InitializeComponent();
            DictionaryType = _type;

            if (_type == 1)
            {
                this.Title = "Словарь городов";
            }
            else if (_type == 2)
            {
                this.Title = "Словарь назначений зданий";
            }
            else if (_type == 3)
            {
                this.Title = "Словарь назначений помещений";
            }

            updateTable();
        }
```

Листинг 4 – Пример заполнения таблицы в зависимости от типа словаря

```
switch (DictionaryType)
            {

                case 1:
                    {
                        Query = "SELECT Город FROM город;";
                        break;
                    }
                case 2:
                    {
                        Query = "SELECT Назначение_Здания AS 'Назначение здания' FROM назна-
чение_здания;";
                        break;
                    }
                case 3:
                    {
                        Query = "SELECT Назначение_Помещения AS 'Назначение помещения' FROM
назначение_помещения;";
                        break;
                    }
            }
   MySqlCommand command = new MySqlCommand(Query, connector);
            dt.Load(command.ExecuteReader());
```

4.3.3 Конечный интерфейс

Конечный интерфейс представлен на рисунке 12.

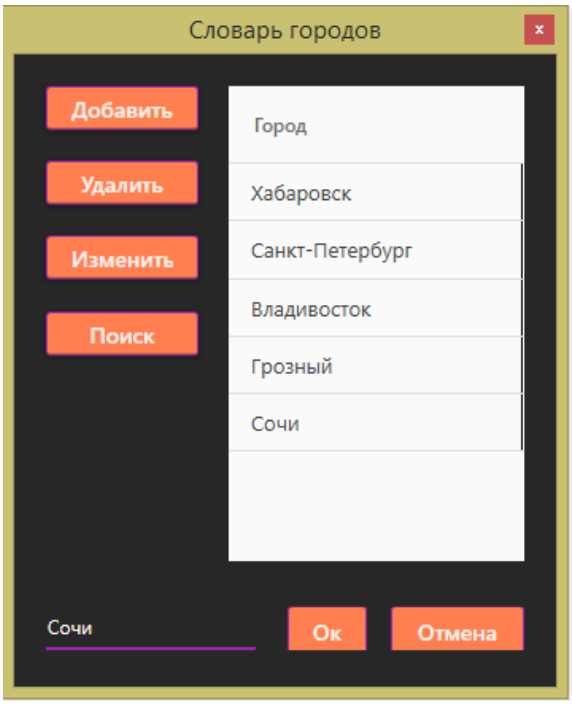

Рисунок 12 – Интерфейс окна словарей

При нажатии кнопки «Ок» произойдет вывод соответствующего города (рисунок 13).

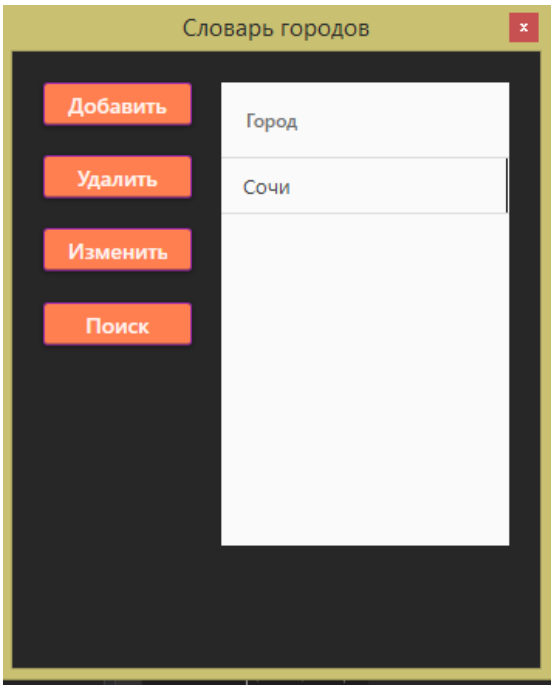

Рисунок 13 – Функция поиска

Аналогичным образом реализованы и другие словари.

4.4 Разработка окна добавления землевладения – класс AddLand

4.4.1 Разметка интерфейса

В разметку (рисунок 14) входят:

кнопки «Отмена» и «Ок»;

чекбоксы для выбора наличия освещения, водопровода и отопления;

элемент DatePicker для выбора даты;

элемент ComboBox для выбора города, с подгрузкой городов из таблицы словарей;

текстовые поля для ввода остальной информации.

Рисунок 14 – Разметка интерфейса

Требуется присвоить элементам кнопок и текстовых полей имена и создать обработчики событий нажатия на кнопку.

4.4.2 Выдержки из кода

Окно добавления землевладения используется для осуществления двух функций – добавления и изменения землевладения, выбор типа окна происходит аналогично с выбором типа словарей для окна словарей.

При этом, при выборе функции изменения требуется заполнить поля соответствующими данными (листинг 5)

Листинг 5 – Заполнение полей

```
if (type == 2) //Изменить
            {
                this.Title = "Изменить землевладение";
            dt = new DataTable();

                Query = "SELECT землевладение.освещение, землевладение.водопровод, землевладение.отопление," +
                    "Дата_Инвентаризации,  адреса.Город, " +
                    " адреса.Район, адреса.Улица, адреса.Номер_Дома, Примечания, Фактическая_Площадь, Площадь_застройки," +
                    "Площадь_двора, Площадь_озеленения, Площадь_огорода, Неудобья from землевладение RIGHT JOIN адреса " +
                    "ON землевладение.Код_Землевладения = " +
                    "адреса.Код_Землевладения RIGHT JOIN площади ON " +
"землевладение.Код_Землевладения = площади.Код_Землевладения WHERE землевладение.Код_Землевладения = " + landIndex + ";";

                command = new MySqlCommand(Query, connector);
                dt.Load(command.ExecuteReader());

                int Light__ = (int)dt.Rows[0][0];
                int Water__ = (int)dt.Rows[0][1];
                int Heat__ = (int)dt.Rows[0][2];

                if (Light__ == 1) LightBox.IsChecked = true;
                if (Water__ == 1) WaterBox.IsChecked = true;
                if (Heat__ == 1) WaterBox.IsChecked = true;

                DateP.Text = (string)dt.Rows[0][3];
                CityBox.Text = (string)dt.Rows[0][4];
                BlockBox.Text = (string)dt.Rows[0][5];
                StreetBox.Text = (string)dt.Rows[0][6];
                HouseBox.Text = dt.Rows[0][7].ToString();
                InfoBox.Text = (string)dt.Rows[0][8];
                FactSq.Text = (string)dt.Rows[0][9];
                BuildSq.Text = (string)dt.Rows[0][10];
                DvorSq.Text = (string)dt.Rows[0][11];
                GreenSq.Text = (string)dt.Rows[0][12];
                FruitSq.Text = (string)dt.Rows[0][13];
                BadSq.Text = (string)dt.Rows[0][14];
            }
```

Таким образом, пользователю не требуется заполнять всю форму добавления землевладения для того, чтобы изменить одно или несколько полей соответствующего землевладения.

4.4.3 Конечный интерфейс

Конечный интерфейс окна добавления представлен на рисунке 15.

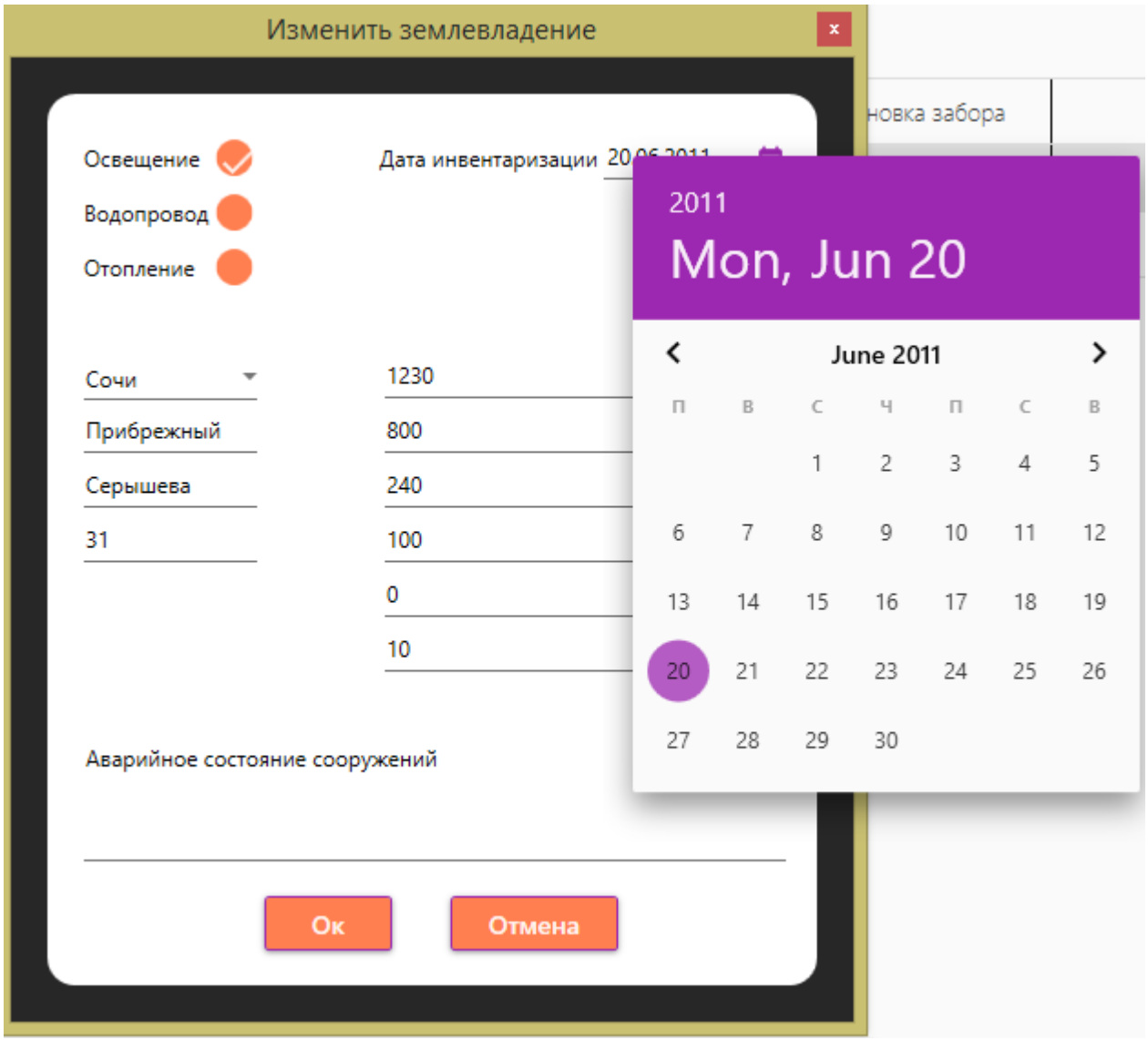

Рисунок 15 – Конечный интерфейс

4.5 Разработка окна информации о землевладении – класс LandInfo

4.5.1 Разметка интерфейса

В разметку (рисунок 16) входят:

– кнопка «Назад», позволяющая вернуться к главному окну;

– чекбоксы, отражающие наличие освещения, водопровода и отопления;

– кнопка «Просмотр информации о зданиях»;

– кнопка «Открыть папку», которая позволяет открыть соответствующую для землевладения папку с фотографиями;

– поле «DataGrid» для вывода информации о площадях;

– кастомные кнопки сворачивания и закрытия приложения;

– текста с подписями для элементов.

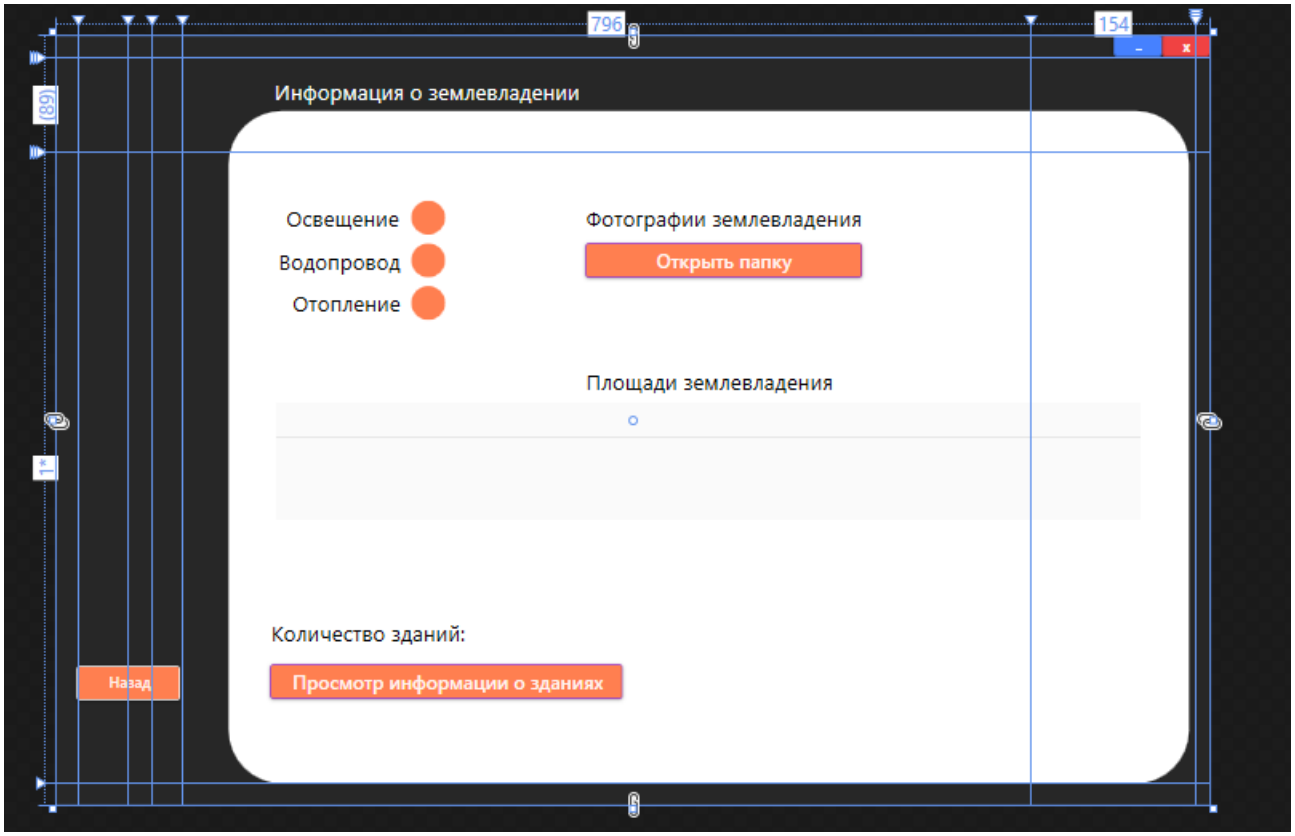

Рисунок 16 – Разметка интерфейса

Требуется присвоить элементам кнопок и текстовых полей имена и создать обработчики событий нажатия на кнопку.

4.5.2 Выдержки из кода

Код данного окна является достаточно тривиальным и не требует представления, рассмотрим, однако, пример заполнения таблицы площадей (листинг 6).

Листинг 6 – Заполнение таблицы площадей

```
Query = "SELECT Фактическая_площадь AS 'Фактическая площадь', Площадь_застройки AS 'Площадь застройки'," +
                "Площадь_двора AS 'Площадь двора', Площадь_озеленения AS 'Площадь озеленения', Площадь_огорода AS 'Площадь огорода'," +
                "Неудобья from площади WHERE Код_Землевладения LIKE " + index + "; ";

            command = new MySqlCommand(Query, connector);
            dt.Load(command.ExecuteReader());

            LandInfoTable.DataContext = dt;
            LandInfoTable.ItemsSource = dt.DefaultView;
```

Как видно из листинга 6, используется простой запрос на выборку, с использованием ключа «index» соответствующего землевладения (ключ передается в конструктор класса).

4.5.3 Конечный интерфейс

Конечный интерфейс окна добавления представлен на рисунке 17.

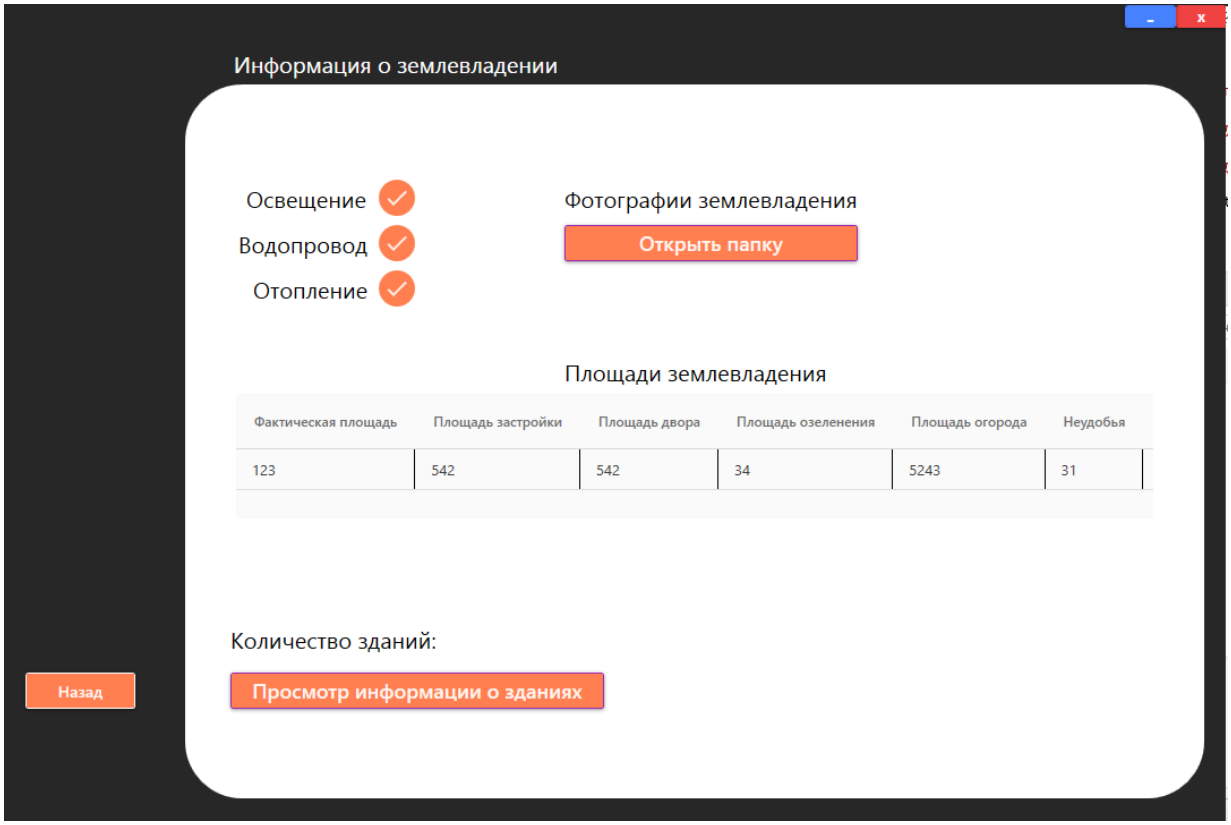

Рисунок 17 – Конечный интерфейс

Таким образом, получено окно информации о землевладении, которое открывается по двойному клику левой кнопки мыши по соответствующей записи о землевладении в главном окне приложения.

4.6 Разработка окна информации о зданиях – класс HouseInfo

4.6.1 Разметка интерфейса

В разметку (рисунок 18) входят:

– кастомные кнопки сворачивания и закрытия приложения;

– кнопки для удаления, изменения и добавления здания;

– кнопка «Назад», позволяющая вернуться в окно информации о землевладении;

– поле «DataGrid» для вывода информации о зданиях.

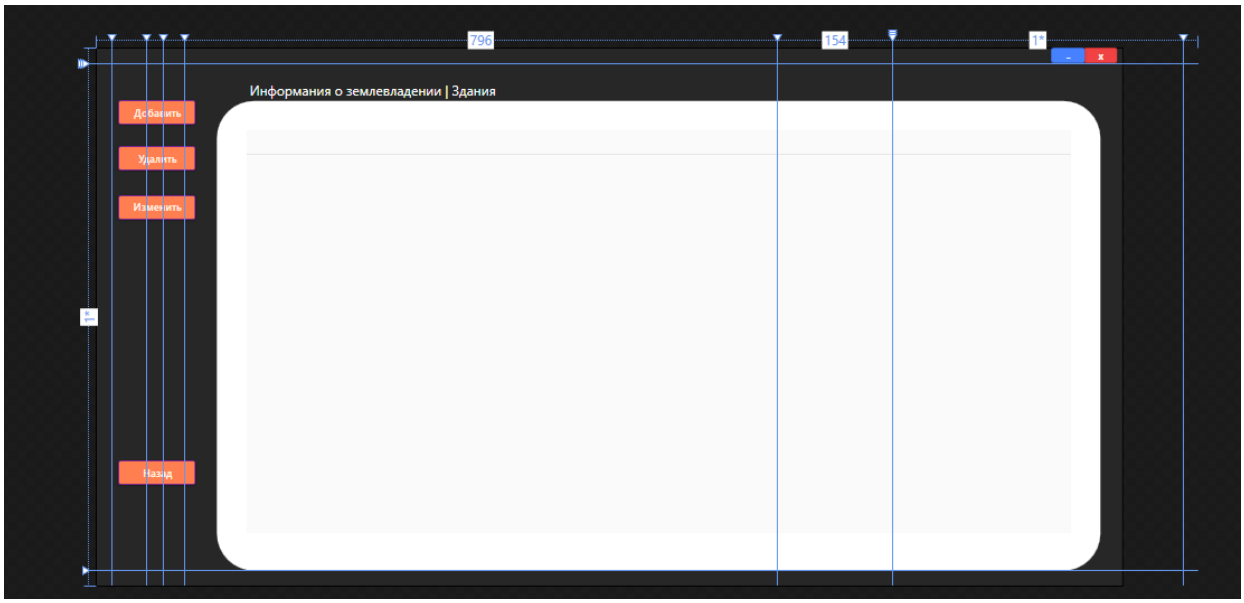

Рисунок 18 – Разметка интерфейса

Требуется присвоить элементам кнопок имена и создать обработчики событий нажатия на кнопку.

4.6.2 Выдержки из кода

Основная функция окна – вывод информации о зданиях, принадлежащих соответствующему землевладению (листинг 7)

Листинг 7 – Заполнение таблицы зданий

```
public void updateTable()
        {
            Error.Text = "";

            connector = new MySqlConnection(connection);

            connector.Open();

            dt = new DataTable();

            Query = "SELECT Номер_Здания AS '#', Назначение, Тип, Возведено_самовольно AS
'Самострой', " +
               "Год_постройки AS 'Год постройки', Общая_площадь AS 'Общая площадь', Жи-
лая_площадь AS 'Жилая площадь'," +
               "Износ AS 'Износ, %', Материал_стен AS 'Материал стен', информа-
ция_о_здании.Инвентаризационная_стоимость AS 'Инв-ая стоимость' ," +
               "Этажность from здание RIGHT JOIN информация_о_здании " +
               "ON здание.Код_Здания = информация_о_здании.Код_Здания WHERE зда-
ние.Код_Землевладения = " + Land_Index + " ;";

            command = new MySqlCommand(Query, connector);
            dt.Load(command.ExecuteReader());

            HouseInfoTable.DataContext = dt;
            HouseInfoTable.ItemsSource = dt.DefaultView;
```

```
connector.Close();
}
```

Код соответствующего землевладения передается в конструктор класса.

4.6.3 Конечный интерфейс

Конечный интерфейс окна добавления представлен на рисунке 19.

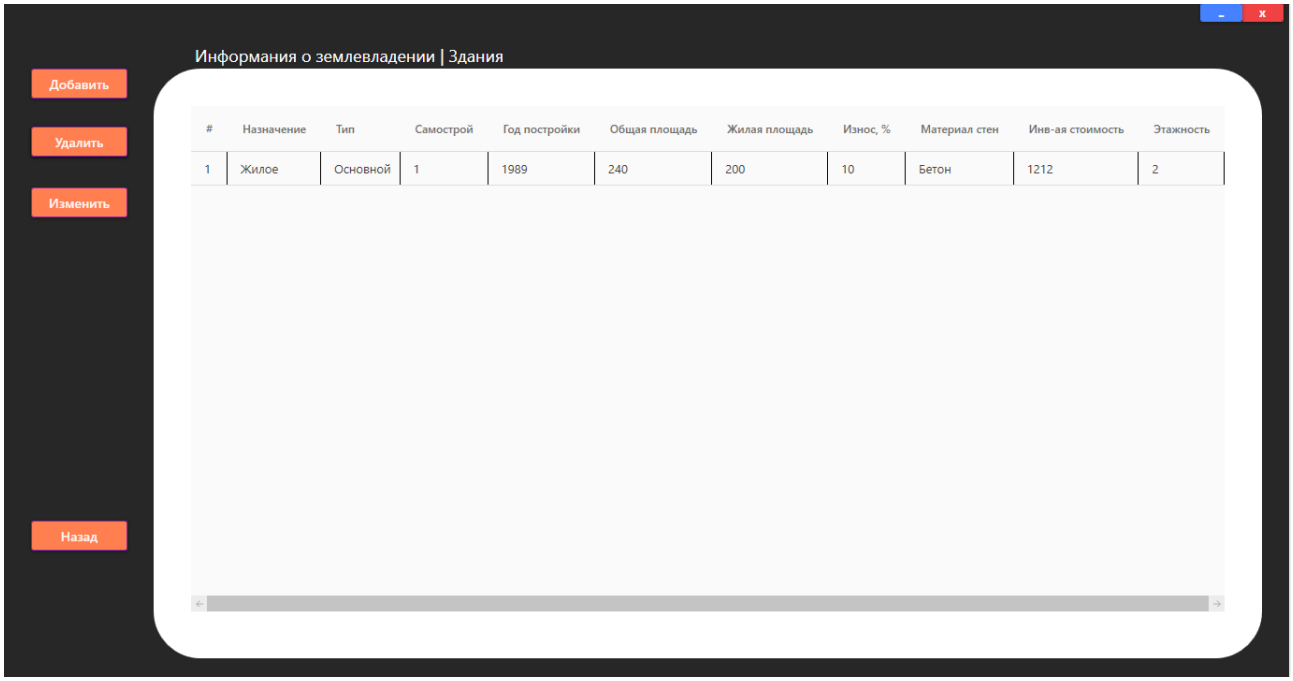

Рисунок 19 – Конечный интерфейс

Таким образом, создано полноценное окно информации о зданиях, принадлежащих соответствующему землевладению.

4.7 Разработка окна добавления здания – класс AddHouse

4.7.1 Разметка интерфейса

В разметку (рисунок 20) входят:

– чекбокс для отражения типа возведения здания;

– поля для выбора назначения сооружения и его типа, с загрузкой данных из таблицы словаря и статичного массива в коде приложения соответственно;

– кнопки «Ок» и «Отмена»;

– текстовые поля для ввода остальной информации.

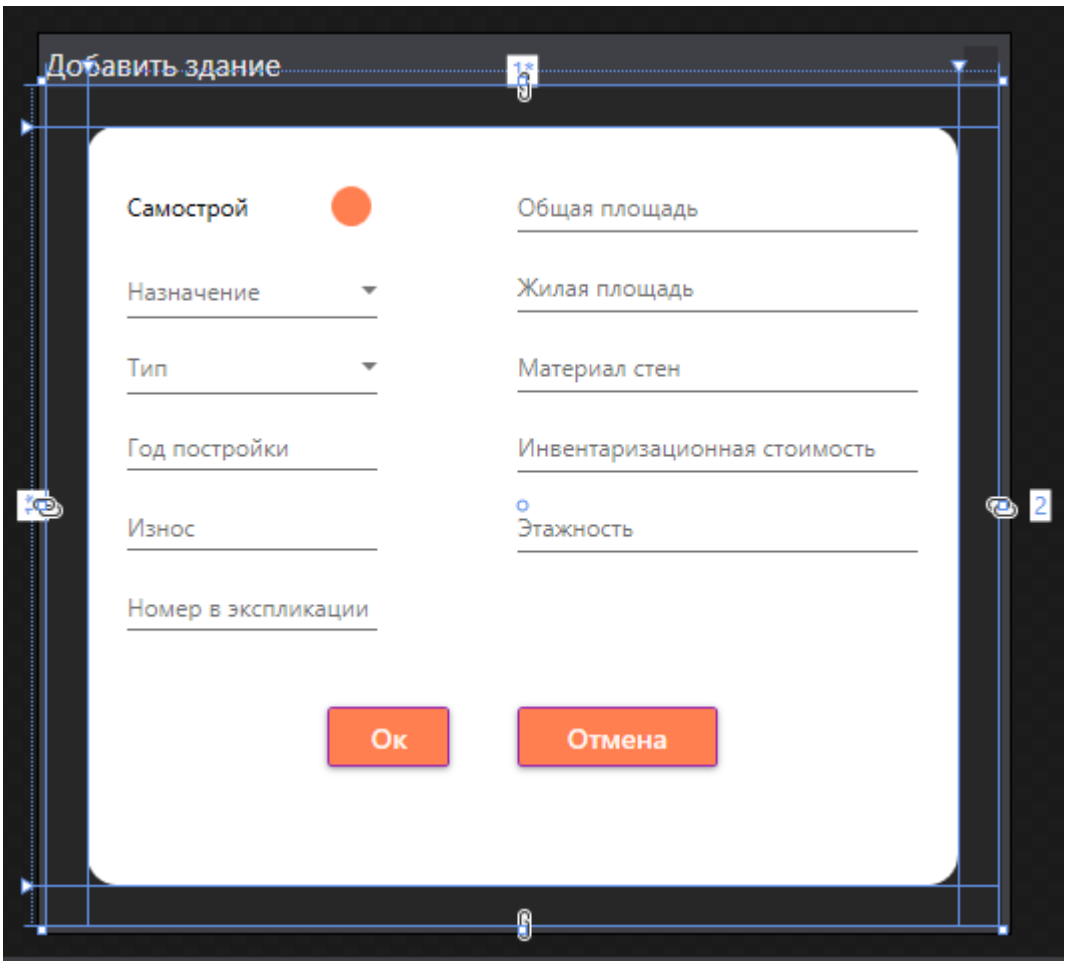

Рисунок 20 – Разметка интерфейса

Требуется присвоить элементам кнопок и текстовых полей имена и создать обработчики событий нажатия на кнопку.

4.7.2 Выдержки из кода

Окно добавления здания используется для осуществления двух функций – добавления и изменения здания, функционал окна аналогичен соответствующему функционалу окна добавления землевладения.

Рассмотрим код для добавления записей (листинг 8)

Листинг 8 – Добавление записей

```
if (type == 1) //Добавить
            {


                string Query = "INSERT INTO здание (Номер_Здания,Код_Землевладения) VALUES "
+
                    "(" + Number + " ," + Land_Index + ");";
                MySqlCommand command = new MySqlCommand(Query, connector);
                command.ExecuteScalar();

                Query = "SELECT Код_Здания FROM здание ORDER BY Код_Здания DESC;";
                command = new MySqlCommand(Query, connector);
```

```
                House_Index = (int)command.ExecuteScalar();

                Query = "INSERT INTO информация_о_здании (Назначение, Тип, Возведе-
но_самовольно, Год_Постройки, Общая_площадь, Жилая_площадь," +
                    "Износ, Материал_стен, Инвентаризацион-
ная_стоимость,Этажность,Код_здания) VALUES " +
                    "(" + Purpose + ", " + Type + ", " + SelfBuild + ", " + Year + ", " +
TotalSquare + ", " + LifeSquare + "" +
                    ", " + Wear + ", " + WallsMaterial + ", " + Price + ", " + Levels + ", "
+ House_Index + ");";
                command = new MySqlCommand(Query, connector);
                command.ExecuteScalar();
            }
```

Таким образом, производится добавление записи в таблицу «Здание» и в дочернюю для нее таблицу «Информация о здании».

4.7.3 Конечный интерфейс

Конечный интерфейс окна добавления представлен на рисунке 21.

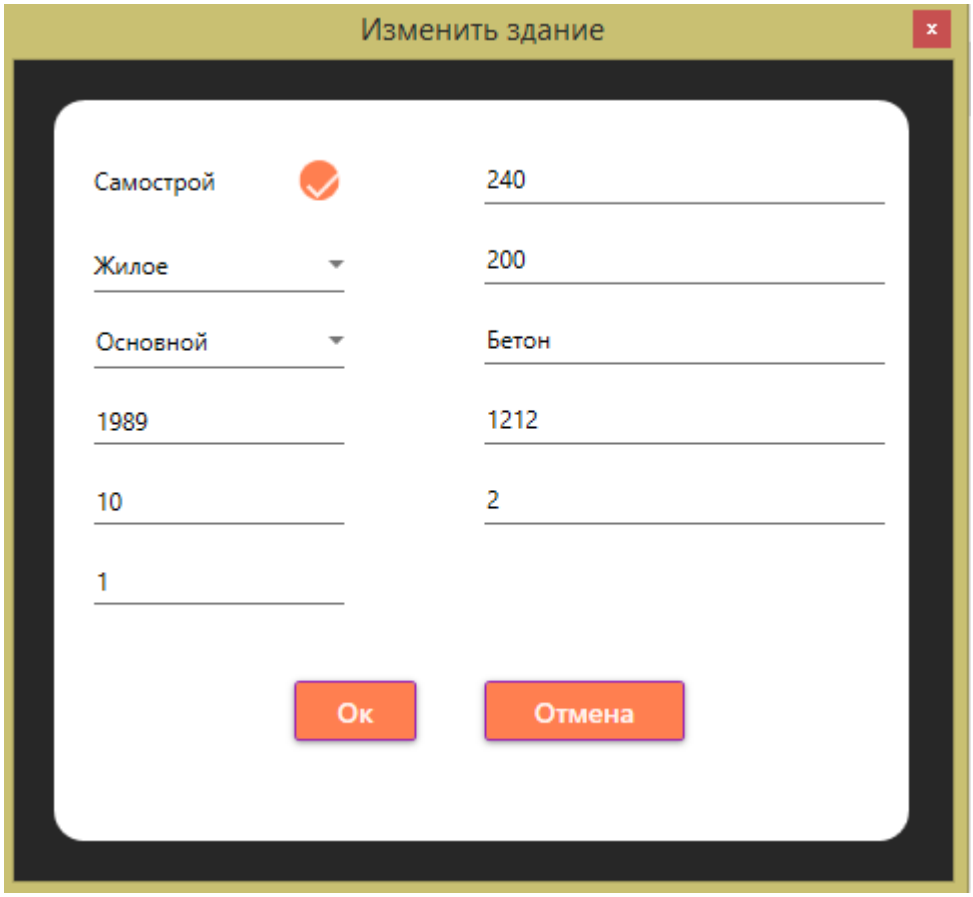

Рисунок 21 – Интерфейс окна

Таким образом, создано окно для добавления/изменения здания.

4.8 Разработка окна информации о помещениях – класс RoomInfo

4.8.1 Разметка интерфейса

В разметку (рисунок 22) входят:

– кастомные кнопки сворачивания и закрытия приложения;

– кнопки для добавления, удаления и изменения помещений;

– кнопка «Назад», позволяющая вернуться в окно информации о здании;

– поле «DataGrid» для вывода информации о площадях.

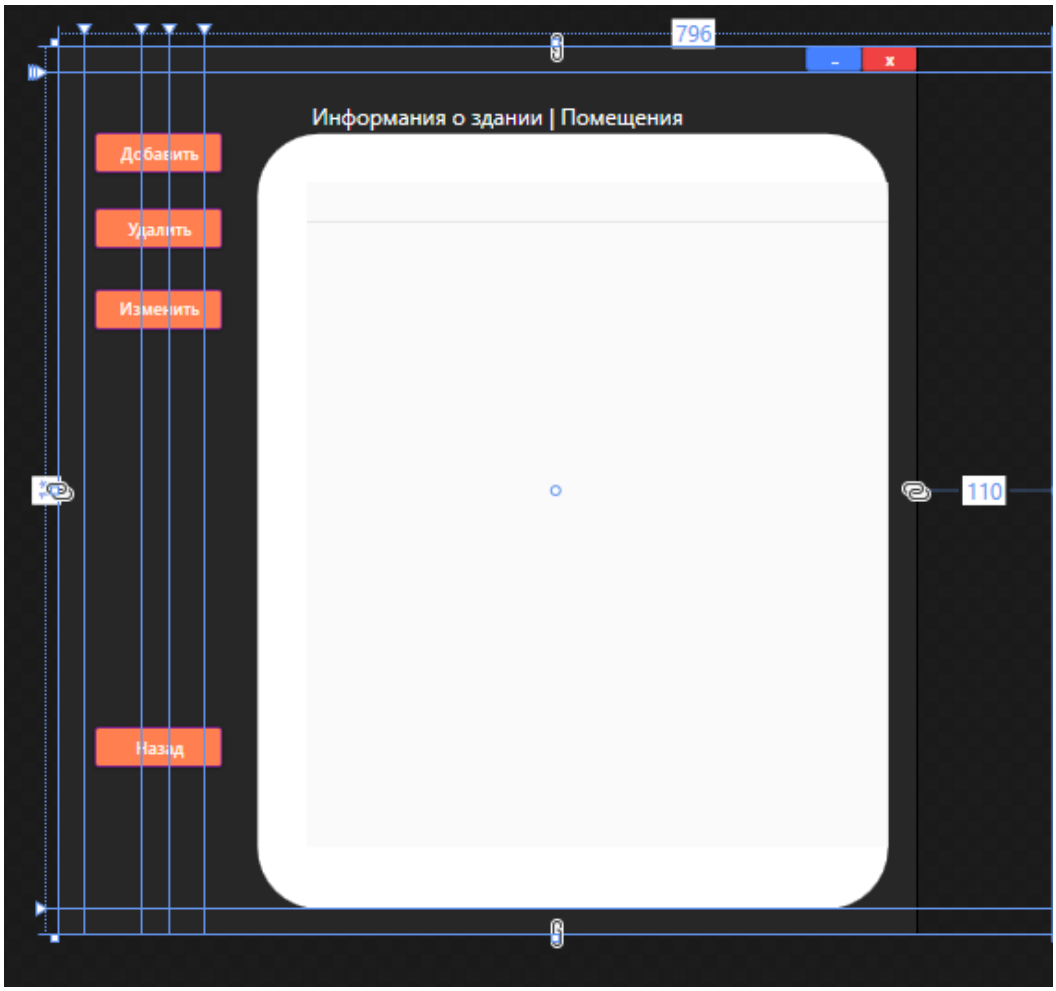

Рисунок 22 – Разметка интерфейса

Требуется присвоить элементам кнопок и имена и создать обработчики событий нажатия на кнопку.

4.8.2 Выдержки из кода

Рассмотрим код изменения информации о здании (листинг 9)

```
private void EditBtn_Click(object sender, RoutedEventArgs e)
        {
            if (RoomInfoTable.SelectedIndex == -1)
            {
                Error.Text = "Не выбрана строка!";
                return;
            }
            Error.Text = "";
            int RowIndex = RoomInfoTable.SelectedIndex;
            int Room_Number = (int)dt.Rows[RowIndex][0];
```

```
            Query = "SELECT Код_помещения FROM помещение WHERE Номер_Помещения =" +
Room_Number + " AND Код_Здания= " + House_Index + ";";
            connector = new MySqlConnection(connection);

            connector.Open();
            command = new MySqlCommand(Query, connector);
            int Room_index = (int)command.ExecuteScalar();


            AddRoom newHouse = new AddRoom(this, 2, House_Index, Room_index);
            newHouse.Show();
            connector.Close();
            }
```

Как видно из листинга 9, в функции существует проверка на выбор строки, которая не позволит выполнить SQL-запрос с некорректным значением кода здания.

4.8.3 Конечный интерфейс

Конечный интерфейс окна представлен на рисунке 23.

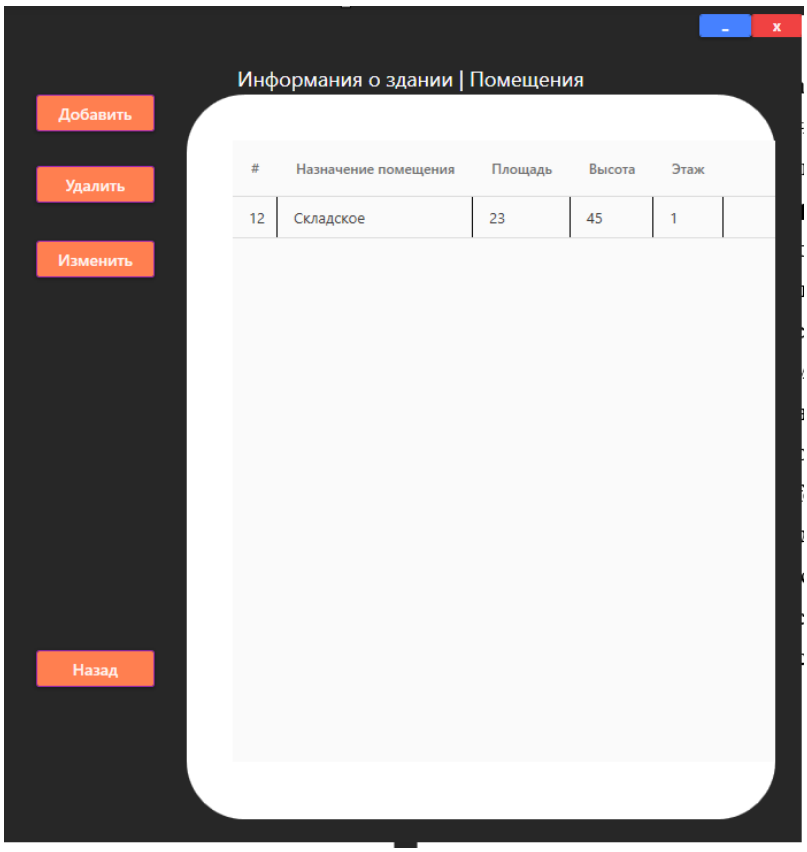

Рисунок 23 – Конечный интерфейс

Таким образом, получено окно информации о помещениях, которое открывается по двойному клику левой кнопки мыши по соответствующей записи о здании в окне информации о землевладениях.

4.9 Разработка окна добавления помещения – класс AddRoom

4.9.1 Разметка интерфейса

В разметку (рисунок 24) входят:

– поле для выбора назначения помещения с загрузкой данных из таблицы словаря;

– кнопки «Ок» и «Отмена»;

– текстовые поля для ввода остальной информации.

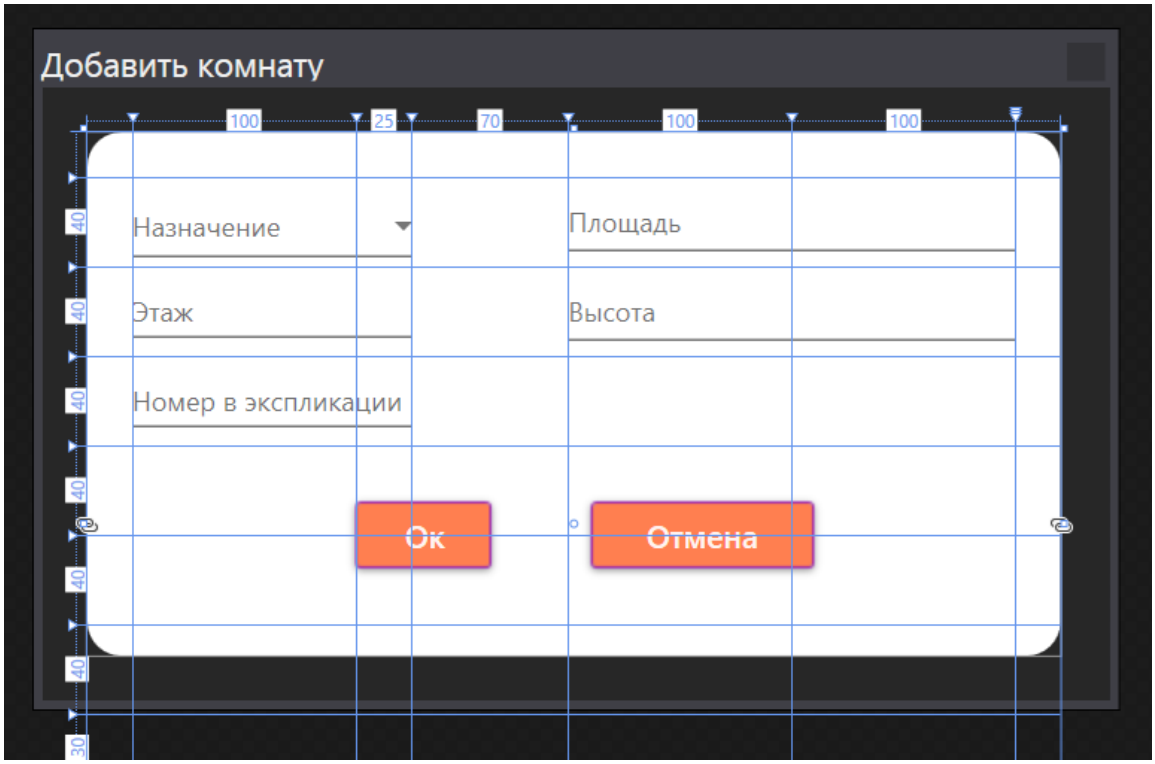

Рисунок 24 – Разметка интерфейса

Требуется присвоить элементам кнопок и имена и создать обработчики событий нажатия на кнопку.

4.9.2 Выдержки из кода

Окно добавления помещения используется для осуществления двух функций – добавления и изменения помещения, функционал окна аналогичен соответствующему функционалу окна добавления землевладения и здания.

Рассмотрим код изменения информации о помещении (листинг 10).

```
else if (type == 2) //Изменить
            {

                string Query = "UPDATE помещение SET Номер_Помещения=" + Number + " WHERE
(Код_Помещения=" + Room_Index + ");";
```

```
                MySqlCommand command = new MySqlCommand(Query, connector);
                command.ExecuteScalar();

                Query = "UPDATE информация_о_помещении SET Назначение_помещения=" + Purpose
+ ", Площадь= " + Square + ", Высота=" + Height + ", " +
                    "Этаж=" + Level + " WHERE (Код_Помещения=" + Room_Index + ");";
                command = new MySqlCommand(Query, connector);
                command.ExecuteScalar();

                }
```

Таким образом, происходит обновление записей в таблице «Помещение» и дочерней для нее таблицу «информация_о_помещении».

4.9.3 Конечный интерфейс

Конечный интерфейс окна представлен на рисунке 23.

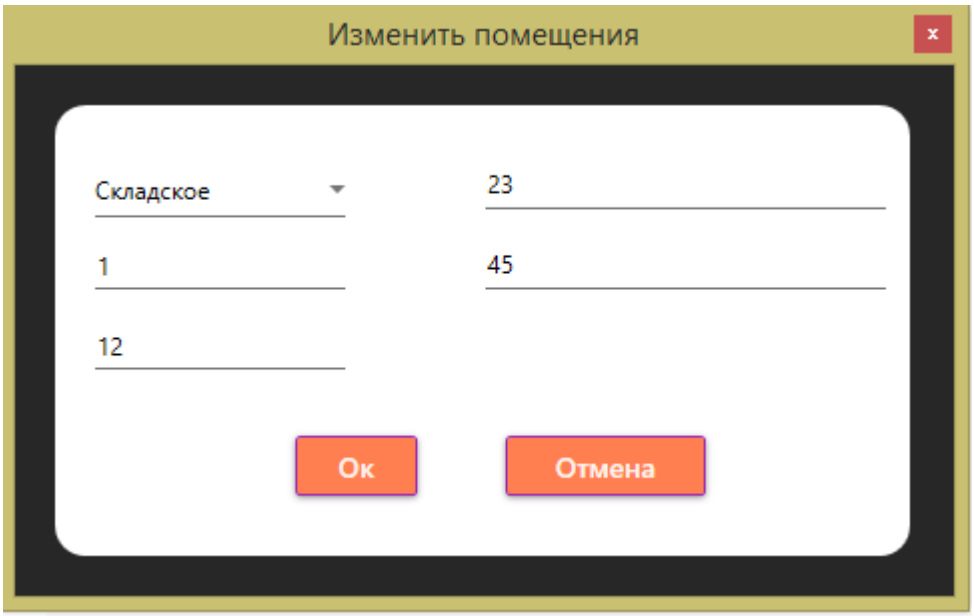

Рисунок 25 – Конечный интерфейс

Таким образом, добавлено заключительное окно приложения, осуществляющее добавления и изменение помещения.

ЗАКЛЮЧЕНИЕ

В данной работе рассматривается разработка прикладного программного обеспечения для бюро технической инвентаризации. В частности, описаны процессы анализа предметной области, проектирования базы данных и разработки прикладного программного обеспечения с интеграцией данных из базы данных проекта в среде разработки Microsoft Visual Studio.

В результате работы получено комфортное и современное прикладное программное обеспечение, обеспечивающее стабильную работу с базой данных проекта.

Благодаря современному программному обеспечению, обеспечивающему удобное и быстрое создание интерфейса и с помощью языка разметки XAML и языка программирования C#, удалось в кратчайшие сроки получить качественное программное обеспечение, с красивым пользовательским интерфейсом и высокой скоростью работы.

Пояснительная записка к работе выполнена в соответствии с требованиями методического пособия [2].

Высокая актуальность работы подтверждена [Приложение Б] низким процентом заимствований сайтом [7].

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Дейт, К. Дж. Введение в системы баз данных [Текст] /К. Дж. Дейт, Москва: Пер. с англ. – Издательский дом "Вильяме, 2005 – 1328 с.
2. Гопкало, В.Н. Выпускная квалификационная работа, / В.Н. Гопкало, О.А. Графский, Хабаровск: Изд-во ДВГУПС, 2014. – 44 с
3. Проверка на заимствования –Антиплагиат [Электронный ресурс]. – Режим доступа: https://www.antiplagiat.ru/, свободный. – Загл. с экрана. – (16.06.20).

ПРИЛОЖЕНИЕ А

(рекомендуемое)

КОД РЕШЕНИЯ

//Mainwindow.xaml.cs

```
using System.Data;
using System.Windows;
using System.Windows.Input;
using ClosedXML.Excel;

using MySql.Data.MySqlClient;
using Kursach.View;

namespace Kursach
{
    /// <summary>
    /// Логика взаимодействия для MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public string[] combo_Names { get; set; }
        private DataTable dt;
        public MainWindow()
        {
            InitializeComponent();

            combo_Names = new string[] { "Словарь городов", "Назначения зданий", "Назначение
помещений" };
            DataContext = this;

            LoadLoginWindow(this);
            updateTable();


        }

        public void updateTable()
        {
            Error.Text = "";
            string connection = "server = localhost; user=root; database=kursach; pass-
word=Dragon2012";

            MySqlConnection connector = new MySqlConnection(connection);

            connector.Open();

             dt = new DataTable();

            string Query = "SELECT землевладение.Код_Землевладения AS '#', " +
                "DATE_FORMAT(Дата_Инвентаризации, '%D,%M,%Y') AS 'Дата инвентаризации', CON-
CAT('г. ', адреса.Город,', район '," +
                " адреса.Район, ', ул. ', адреса.Улица, ', д. '," +
                "адреса.Номер_Дома) AS Адрес, Примечания from землевладение RIGHT JOIN адре-
са " +
                "ON землевладение.Код_Землевладения = " +
                "адреса.Код_Землевладения;";

            MySqlCommand command = new MySqlCommand(Query, connector);
            dt.Load(command.ExecuteReader());
```

```
            TableView.DataContext = dt;
            TableView.ItemsSource = dt.DefaultView;


            connector.Close();
        }

        public void LoadLoginWindow(MainWindow window)
        {
            this.Hide();
            Login login = new Login(window);
            login.Show();
        }

        private void CloseButton_Click(object sender, RoutedEventArgs e)
        {
             this.Close();
        }

        private void HideButton_Click(object sender, RoutedEventArgs e)
        {
            this.WindowState = WindowState.Minimized;
        }

        private void TableView_MouseDoubleClick(object sender, MouseButtonEventArgs e)
        {
            System.Data.DataRowView selectedRow = (Sys-
tem.Data.DataRowView)TableView.SelectedItems[0];
            int landId = (int)selectedRow.Row[0];

            LandInfo l_info = new LandInfo(this,landId);
            this.Hide();
            l_info.Show();
            Error.Text = "";
        }

        private void AddButton_Click(object sender, RoutedEventArgs e)
        {

            AddLand add = new AddLand(this,1,0);
            add.Show();

        }

        private void CityS_Click(object sender, RoutedEventArgs e)
        {
            Dictionary CityDictionary = new Dictionary(1);
            CityDictionary.Show();
        }

        private void HouseS_Click(object sender, RoutedEventArgs e)
        {
            Dictionary HouseDictionary = new Dictionary(2);
            HouseDictionary.Show();
        }

        private void RoomS_Click(object sender, RoutedEventArgs e)
        {
            Dictionary RoomDictionary = new Dictionary(3);
            RoomDictionary.Show();
        }

        private void UpdateButton_Click(object sender, RoutedEventArgs e)
        {
```

```
            if (TableView.SelectedIndex == -1)
            {
                Error.Text = "Не выбрана строка!";

                return;
            }
            Error.Text = "";


            System.Data.DataRowView selectedRow = (Sys-
tem.Data.DataRowView)TableView.SelectedItems[0];
            int landId = (int)selectedRow.Row[0];

            AddLand edit = new AddLand(this, 2, landId);
            edit.Show();

        }

        private void DeleteButton_Click(object sender, RoutedEventArgs e)
        {
            if (TableView.SelectedIndex == -1)
            {
                Error.Text = "Не выбрана строка!";
                return;
            }
            Error.Text = "";

            System.Data.DataRowView selectedRow = (Sys-
tem.Data.DataRowView)TableView.SelectedItems[0];
            int landId = (int)selectedRow.Row[0];

            string connection = "server = localhost; user=root; database=kursach; pass-
word=Dragon2012";

            MySqlConnection connector = new MySqlConnection(connection);
            connector.Open();
            string Query = "DELETE FROM адреса WHERE Код_Землевладения = " + landId + ";";
            MySqlCommand command = new MySqlCommand(Query, connector);
            command.ExecuteScalar();

            Query = "DELETE FROM площади WHERE Код_Землевладения = " + landId + ";";
            command = new MySqlCommand(Query, connector);
            command.ExecuteScalar();

            Query = "DELETE FROM здание WHERE Код_Землевладения = " + landId + ";";
            command = new MySqlCommand(Query, connector);
            command.ExecuteScalar();

            Query = "DELETE FROM землевладение WHERE Код_Землевладения = " + landId + ";";
            command = new MySqlCommand(Query, connector);
            command.ExecuteScalar();

            connector.Close();

            updateTable();
        }

        private void SearchButton_Click(object sender, RoutedEventArgs e)
        {
            if (SearchBox.Text != "")
            {
                string find = SearchBox.Text;

                find = string.Concat("'%", find, "%'");
```

```
                string connection = "server = localhost; user=root; database=kursach; pass-
word=Dragon2012";

                MySqlConnection connector = new MySqlConnection(connection);
                connector.Open();
                string Query = "SELECT землевладение.Код_Землевладения AS '#', " +
                "DATE_FORMAT(Дата_Инвентаризации, '%D,%M,%Y') AS 'Дата инвентаризации', CON-
CAT('г. ', адреса.Город,', район '," +
                " адреса.Район, ', ул. ', адреса.Улица, ', д. '," +
                "адреса.Номер_Дома) AS Адрес, Примечания from землевладение RIGHT JOIN адре-
са " +
                "ON землевладение.Код_Землевладения = " +
                "адреса.Код_Землевладения WHERE (Дата_Инвентаризации LIKE " + find + ") OR
(Город LIKE " + find + ") OR (Район LIKE " + find + ")" +
                "OR (Улица LIKE " + find + ") OR (Номер_Дома LIKE " + find + ") OR (Примеча-
ния LIKE " + find + ");";
                MySqlCommand command = new MySqlCommand(Query, connector);
                dt = new DataTable();

                dt.Load(command.ExecuteReader());

                TableView.DataContext = dt;
                TableView.ItemsSource = dt.DefaultView;

                connector.Close();
            }
            else updateTable();
        }

        private void ExportButton_Click(object sender, RoutedEventArgs e)
        {
            XLWorkbook wb = new XLWorkbook();
            wb.Worksheets.Add(dt, "Таблица землевладений");
            wb.Worksheet(1).Column(1).AdjustToContents();
            wb.Worksheet(1).Column(2).AdjustToContents();
            wb.Worksheet(1).Column(3).AdjustToContents();
            wb.Worksheet(1).Column(4).AdjustToContents();


            wb.SaveAs(@"C:\Users\Дмитрий\Desktop\Отчет.xlsx");
        }
    }
}
```

//Dictionary.xaml.cs

```
using System.Data;
using System.Windows;

using MySql.Data.MySqlClient;

namespace Kursach.View
{
    /// <summary>
    /// Логика взаимодействия для Dictionary.xaml
    /// </summary>
    public partial class Dictionary : Window
    {
        private int Ok_Type = 0;
       private int DictionaryType;
       public string QueryB;

        public Dictionary(int _type)
        {
```

```
            InitializeComponent();
            DictionaryType = _type;

            if (_type == 1)
            {
                this.Title = "Словарь городов";
            }
            else if (_type == 2)
            {
                this.Title = "Словарь назначений зданий";
            }
            else if (_type == 3)
            {
                this.Title = "Словарь назначений помещений";
            }

            updateTable();
        }

        public void updateTable(string Query, bool update)
        {
            string connection = "server = localhost; user=root; database=kursach; pass-
word=Dragon2012";

            MySqlConnection connector = new MySqlConnection(connection);
            DataTable dt = new DataTable();

                connector.Open();
                MySqlCommand command = new MySqlCommand(Query, connector);

            if (update)
            {
                command = new MySqlCommand(Query, connector);
                command.ExecuteScalar();

                switch (DictionaryType)
                {

                    case 1:
                        {
                            Query = "SELECT Город FROM город;";
                            break;
                        }
                    case 2:
                        {
                            Query = "SELECT Назначение_Здания AS 'Назначение здания' FROM
назначение_здания;";
                            break;
                        }
                    case 3:
                        {
                            Query = "SELECT Назначение_Помещения AS 'Назначение помещения'
FROM назначение_помещения;";
                            break;
                        }
                }
                command = new MySqlCommand(Query, connector);
            }

            dt.Load(command.ExecuteReader());

            Table.DataContext = dt;
            Table.ItemsSource = dt.DefaultView;
```

```
            connector.Close();
        }

        public void updateTable()
        {
            string connection = "server = localhost; user=root; database=kursach; pass-
word=Dragon2012";

            MySqlConnection connector = new MySqlConnection(connection);
            DataTable dt = new DataTable();

            connector.Open();

            string Query = "";

            switch (DictionaryType)
            {

                case 1:
                    {
                        Query = "SELECT Город FROM город;";
                        break;
                    }
                case 2:
                    {
                        Query = "SELECT Назначение_Здания AS 'Назначение здания' FROM назна-
чение_здания;";
                        break;
                    }
                case 3:
                    {
                        Query = "SELECT Назначение_Помещения AS 'Назначение помещения' FROM
назначение_помещения;";
                        break;
                    }
            }

                    MySqlCommand command = new MySqlCommand(Query, connector);
            dt.Load(command.ExecuteReader());

            Table.DataContext = dt;
            Table.ItemsSource = dt.DefaultView;

            connector.Close();
        }


        private void AddButton_Click(object sender, RoutedEventArgs e)
        {
           Error.Text = "";
            EnterText.Visibility = Visibility.Visible;
            OkButton.Visibility = Visibility.Visible;
            CancelButton.Visibility = Visibility.Visible;
            Ok_Type = 1;
        }

        private void CancelButton_Click(object sender, RoutedEventArgs e)
        {
            EnterText.Visibility = Visibility.Hidden;
            OkButton.Visibility = Visibility.Hidden;
            CancelButton.Visibility = Visibility.Hidden;

            EnterText.Clear();
```

```
        }

        private void DeleteButton_Click(object sender, RoutedEventArgs e)
        {
            if (Table.SelectedIndex == -1)
            {
                Error.IsEnabled = true;
                Error.Text = "Ошибка! Строка не выбрана!";
                return;
            }

            Error.Text = "";


            System.Data.DataRowView selectedRow = (Sys-
tem.Data.DataRowView)Table.SelectedItems[0];
            string oldValue = selectedRow.Row[0].ToString();
            oldValue = string.Concat("'", oldValue, "'");
            string Query = "";
            switch (DictionaryType)
            {

                case 1:
                    {
                        Query = "DELETE FROM город WHERE Город = " + oldValue + ";";
                        break;
                    }
                case 2:
                    {
                        Query = "DELETE FROM назначение_здания WHERE Назначение_Здания = " +
oldValue + ";";
                        break;
                    }
                case 3:
                    {
                        Query = "DELETE FROM назначение_помещения WHERE Назначение_Помещения
= " + oldValue + ";";
                        break;
                    }
            }


            updateTable(Query, true);
        }

        private void EditButton_Click(object sender, RoutedEventArgs e)
        {
             if (Table.SelectedIndex == -1)
                {
                Error.IsEnabled = true;
                Error.Text = "Ошибка! Строка не выбрана!";
                    return;
                }

                Error.Text = "";

            EnterText.Visibility = Visibility.Visible;
            OkButton.Visibility = Visibility.Visible;
            CancelButton.Visibility = Visibility.Visible;

            Ok_Type = 2;
        }

        private void OkButton_Click(object sender, RoutedEventArgs e)
```

```
        {
            EnterText.Visibility = Visibility.Hidden;
            OkButton.Visibility = Visibility.Hidden;
            CancelButton.Visibility = Visibility.Hidden;
            string newValue = string.Concat("'", EnterText.Text, "'");
            string Query = "";
            if (Ok_Type == 1) //Добавить
            {
                switch (DictionaryType)
                {

                    case 1:
                        {
                            Query = "INSERT INTO город (Город) VALUES(" + newValue + ");";
                            break;
                        }
                    case 2:
                        {
                            Query = "INSERT INTO назначение_здания (Назначение_Здания) VAL-
UES(" + newValue + ");";
                            break;
                        }
                    case 3:
                        {
                            Query = "INSERT INTO назначение_помещения (Назначение_Помещения)
VALUES(" + newValue + ");";
                            break;
                        }
                }
               updateTable(Query, true);
            }
            else if (Ok_Type == 2) // Изменить
            {

                System.Data.DataRowView selectedRow = (Sys-
tem.Data.DataRowView)Table.SelectedItems[0];
                string oldValue = selectedRow.Row[0].ToString();

                oldValue = string.Concat("'", oldValue, "'");

                switch (DictionaryType)
                {

                    case 1:
                        {
                            Query = "UPDATE город SET Город = " + newValue + " WHERE Город =
" + oldValue + ";";
                            QueryB = "UPDATE адреса SET Город = " + newValue + " WHERE Город
= " + oldValue + ";";
                            break;
                        }
                    case 2:
                        {
                            Query = "UPDATE назначение_здания SET Назначение_Здания = " +
newValue + " WHERE Назначение_Здания = " + oldValue + ";";
                            break;
                        }
                    case 3:
                        {
                            Query = "UPDATE назначение_помещения SET Назначение_Помещения =
" + newValue + " WHERE Назначение_Помещения = " + oldValue + ";";
                            break;
                        }
                }
```

```
                updateTable(Query, true);


            }
            else // Найти
            {
                switch (DictionaryType)
                {

                    case 1:
                        {
                            Query = "SELECT Город FROM город WHERE Город = " + newValue +
";";
                            break;
                        }
                    case 2:
                        {
                            Query = "SELECT Назначение_Здания FROM назначение_здания WHERE
Назначение_Здания = " + newValue + ";";
                            break;
                        }
                    case 3:
                        {
                            Query = "SELECT Назначение_Помещения FROM назначение_помещения
WHERE Назначение_Помещения = " + newValue + ";";
                            break;
                        }
                }


                updateTable(Query, false);
            }

            EnterText.Clear();

        }

        private void SearchButton_Click(object sender, RoutedEventArgs e)
        {
            Error.Text = "";
            EnterText.Visibility = Visibility.Visible;
            OkButton.Visibility = Visibility.Visible;
            CancelButton.Visibility = Visibility.Visible;

            Ok_Type = 3;
        }
    }
}
```

//LandInfo.xaml.cs

```
using System.Data;
using System.Windows;


using MySql.Data.MySqlClient;
using System.Diagnostics;

namespace Kursach.View
{
    /// <summary>
    /// Логика взаимодействия для LandInfo.xaml
    /// </summary>
    ///
```

```
    public partial class LandInfo : Window
    {
        private MainWindow back_window;
        private int index;


        public LandInfo(MainWindow _back_window, int _index)
        {
            InitializeComponent();

            back_window = _back_window;
            index = _index;

            string connection = "server = localhost; user=root; database=kursach; pass-
word=Dragon2012";

            MySqlConnection connector = new MySqlConnection(connection);

            connector.Open();

            DataTable dt = new DataTable();

            string Query = "SELECT освещение FROM землевладение WHERE Код_Землевладения LIKE
" + index + "; ";
            MySqlCommand command = new MySqlCommand(Query, connector);
            string test_for_light = command.ExecuteScalar().ToString();

            if (test_for_light == "1")
            {
                LightBox.IsChecked = true;
            }

            Query = "SELECT водопровод FROM землевладение WHERE Код_Землевладения LIKE " +
index + "; ";
            command = new MySqlCommand(Query, connector);
            string test_for_water = command.ExecuteScalar().ToString();

            if (test_for_water == "1")
            {
                WaterBox.IsChecked = true;
            }

            Query = "SELECT отопление FROM землевладение WHERE Код_Землевладения LIKE " +
index + "; ";
            command = new MySqlCommand(Query, connector);
            string test_for_heat = command.ExecuteScalar().ToString();



            if (test_for_heat == "1")
            {
                HeatBox.IsChecked = true;
            }

            Query = "SELECT Фактическая_площадь AS 'Фактическая площадь', Площадь_застройки
AS 'Площадь застройки'," +
                "Площадь_двора AS 'Площадь двора', Площадь_озеленения AS 'Площадь озелене-
ния', Площадь_огорода AS 'Площадь огорода'," +
                "Неудобья from площади WHERE Код_Землевладения LIKE " + index + "; ";

            command = new MySqlCommand(Query, connector);
            dt.Load(command.ExecuteReader());
```

```
            LandInfoTable.DataContext = dt;
            LandInfoTable.ItemsSource = dt.DefaultView;

            connector.Close();


        }

        private void CloseButton1_Click(object sender, RoutedEventArgs e)
        {
            this.Close();
            back_window.Close();
        }

        private void HideButton1_Click(object sender, RoutedEventArgs e)
        {
            this.WindowState = WindowState.Minimized;
        }



        private void BackButton_Click(object sender, RoutedEventArgs e)
        {
            this.Close();
            back_window.Show();
        }

        private void OpenFolder_Click(object sender, RoutedEventArgs e)
        {
            var master_folder = @"E:\Visual Studio projects\Kursach\Photos\";
            var Land_Folder = index.ToString();
            var path = System.IO.Path.Combine(master_folder, Land_Folder);

            Process.Start(path);
        }

        private void HousesInfo_Click(object sender, RoutedEventArgs e)
        {
            HouseInfo info = new HouseInfo(this, index);
            this.Hide();
            info.Show();
        }
        public void CloseBoth()
        {
            back_window.Close();
            this.Close();
        }
    }
}
```

//Login.xaml.cs

```
using System.Windows;

namespace Kursach
{
    /// <summary>
    /// Логика взаимодействия для Login.xaml
    /// </summary>

    public partial class Login : Window
    {
        private MainWindow the_window;

        public Login(MainWindow window)
```

```
        {
            InitializeComponent();
            the_window = window;

        }

        private void EnterButton_Click(object sender, RoutedEventArgs e)
        {
            if(LoginText.Text == "root" && PasswordText.Password == "Dragon2012")
            {
                Invalid_info.Text = "Успешный вход";
                this.Close();
                the_window.Show();
            }
            else
            {
                Invalid_info.Text = "Введен неверный логин/пароль";
            }


        }

        private void CancelButton_Click(object sender, RoutedEventArgs e)
        {
            Close();
            the_window.Close();
        }
    }
}
```

//AddLand.xaml.cs

```
using System.Data;
using System.Collections.Generic;
using System.Windows;

using MySql.Data.MySqlClient;
using System.IO;

namespace Kursach
{
    /// <summary>
    /// Логика взаимодействия для AddLand.xaml
    /// </summary>
    public partial class AddLand : Window
    {
        private int type;
        private MainWindow the_window;
        private string landIndex;


        public AddLand(MainWindow win, int _type, int land_index)
        {
            InitializeComponent();
            landIndex = string.Concat("'", land_index.ToString(), "'");

            type = _type;
            the_window = win;

            string connection = "server = localhost; user=root; database=kursach; pass-
word=Dragon2012";

            MySqlConnection connector = new MySqlConnection(connection);
            DataTable dt = new DataTable();
            string Query = "SELECT Город from город;";
```

```
            connector.Open();
            MySqlCommand command = new MySqlCommand(Query, connector);
            dt.Load(command.ExecuteReader());

            List<string> Cities = new List<string>();

            for (int i = 0; i < dt.Rows.Count; i++)
            {
                string s = dt.Rows[i][0].ToString();
                Cities.Add(s);
            }

            CityBox.DataContext = Cities;
            CityBox.ItemsSource = Cities;

            if (type == 2) //Изменить
            {
                this.Title = "Изменить землевладение";
            dt = new DataTable();

                Query = "SELECT землевладение.освещение, землевладение.водопровод, землевла-
дение.отопление," +
                    "Дата_Инвентаризации,  адреса.Город, " +
                    " адреса.Район, адреса.Улица, адреса.Номер_Дома, Примечания, Фактиче-
ская_Площадь, Площадь_застройки," +
                    "Площадь_двора, Площадь_озеленения, Площадь_огорода, Неудобья from земле-
владение RIGHT JOIN адреса " +
                    "ON землевладение.Код_Землевладения = " +
                    "адреса.Код_Землевладения RIGHT JOIN площади ON землевладе-
ние.Код_Землевладения = площади.Код_Землевладения WHERE землевладение.Код_Землевладения = "
+ landIndex + ";";

                command = new MySqlCommand(Query, connector);
                dt.Load(command.ExecuteReader());

                int Light__ = (int)dt.Rows[0][0];
                int Water__ = (int)dt.Rows[0][1];
                int Heat__ = (int)dt.Rows[0][2];

                if (Light__ == 1) LightBox.IsChecked = true;
                if (Water__ == 1) WaterBox.IsChecked = true;
                if (Heat__ == 1) WaterBox.IsChecked = true;

                DateP.Text = (string)dt.Rows[0][3];
                CityBox.Text = (string)dt.Rows[0][4];
                BlockBox.Text = (string)dt.Rows[0][5];
                StreetBox.Text = (string)dt.Rows[0][6];
                HouseBox.Text = dt.Rows[0][7].ToString();
                InfoBox.Text = (string)dt.Rows[0][8];
                FactSq.Text = (string)dt.Rows[0][9];
                BuildSq.Text = (string)dt.Rows[0][10];
                DvorSq.Text = (string)dt.Rows[0][11];
                GreenSq.Text = (string)dt.Rows[0][12];
                FruitSq.Text = (string)dt.Rows[0][13];
                BadSq.Text = (string)dt.Rows[0][14];
            }

            connector.Close();
        }

        private void CancelButton_Click(object sender, RoutedEventArgs e)
        {
            this.Close();
        }
```

```
        private void OkButton_Click(object sender, RoutedEventArgs e)
        {
            string connection = "server = localhost; user=root; database=kursach; pass-
word=Dragon2012";

            MySqlConnection connector = new MySqlConnection(connection);
            connector.Open();

            int Light = 0;
            int Heat = 0;
            int Water = 0;

            if (LightBox.IsChecked == true) Light = 1;
            if (HeatBox.IsChecked == true) Heat = 1;
            if (WaterBox.IsChecked == true) Water = 1;

            if (CityBox.Text == "" || BlockBox.Text == "" || StreetBox.Text == "" || House-
Box.Text == "" ||
                DateP.Text == "" || FactSq.Text == "" || BuildSq.Text == "" || DvorSq.Text
== "" || GreenSq.Text == "" ||
                FruitSq.Text == "" || BadSq.Text == "")
            {

                test.Text = "Ошибка! Пустое поле ввода!";
                return;
            }

                string City = SQLFixer(CityBox.Text);
                string Block = SQLFixer(BlockBox.Text);
                string Street = SQLFixer(StreetBox.Text);
                string HouseN = SQLFixer(HouseBox.Text);
                string Date = SQLFixer(DateP.Text);

                string FactSquare = SQLFixer(FactSq.Text);
                string BuildSquare = SQLFixer(BuildSq.Text);
                string DvorSquare = SQLFixer(DvorSq.Text);
                string GreenSquare = SQLFixer(GreenSq.Text);
                string OgorodSquare = SQLFixer(FruitSq.Text);
                string BadSquare = SQLFixer(BadSq.Text);

                string InfoSquare = SQLFixer(InfoBox.Text);



            if (type == 1)
            {
                string Query = "INSERT INTO землевладение (Да-
та_Инвентаризации,Освещение,Водопровод,Отопление,Примечания) VALUES (" + Date + " ," + Light
+ "," + Water + ", " + Heat + "," +
                    "" + InfoSquare + ");";
                MySqlCommand command = new MySqlCommand(Query, connector);
                command.ExecuteScalar();

                Query = "SELECT Код_Землевладения FROM землевладение ORDER BY
Код_Землевладения DESC;";
                command = new MySqlCommand(Query, connector);
                int LandIndex = (int)command.ExecuteScalar();

                Query = "INSERT INTO адреса (Город, Район, Улица, Номер_дома,
Код_Землевладения) VALUES (" + City + ", " + Block + ", " + Street + ", " + HouseN + ", " +
LandIndex + ");";
                command = new MySqlCommand(Query, connector);
                command.ExecuteScalar();
```

```
                Query = "INSERT INTO площади (Фактическая_площадь, Площадь_застройки, Пло-
щадь_двора, Площадь_озеленения, Площадь_огорода, Неудобья, Код_Землевладения) " +
                    "VALUES (" + FactSquare + ", " + BuildSquare + ", " + DvorSquare + ", "
+ GreenSquare + ", " + OgorodSquare + ", " + BadSquare + ", " + LandIndex + ");";
                command = new MySqlCommand(Query, connector);
                command.ExecuteScalar();

                string folderpath = @"E:\Visual Studio projects\Kursach\Photos\";
                string foldername = LandIndex.ToString();
                string path = System.IO.Path.Combine(folderpath, foldername);

                Directory.CreateDirectory(path);
            }
            else if (type == 2)
            {


                string Query = "UPDATE землевладение SET Дата_Инвентаризации=" + Date +
",Освещение=" + Light + ",Водопровод=" + Water + "," +
                    " Отопление=" + Heat + ",Примечания=" + InfoSquare + " WHERE
(Код_Землевладения=" + landIndex + ");";
                MySqlCommand command = new MySqlCommand(Query, connector);
                command.ExecuteScalar();

                Query = "UPDATE адреса SET Город=" + City + ", Район= " + Block + ", Улица="
+ Street + ", Номер_дома=" + HouseN + " WHERE (Код_Землевладения=" + landIndex + ");";
                command = new MySqlCommand(Query, connector);
                command.ExecuteScalar();

                Query = "UPDATE площади SET Фактическая_площадь="+ FactSquare +", Пло-
щадь_застройки=" + BuildSquare + ", Площадь_двора=" + DvorSquare + ", Площадь_озеленения=" +
GreenSquare + ", " +
                    "Площадь_огорода=" + OgorodSquare + ", Неудобья=" + BadSquare + "  WHERE
(Код_Землевладения=" + landIndex + ");";

                command = new MySqlCommand(Query, connector);
                command.ExecuteScalar();
            }
            this.Close();
            the_window.updateTable();
        }

      private string SQLFixer(string a)
        {
            a = string.Concat("'", a, "'");

            return a;
        }
    }
}
```

//HouseInfo.xaml.cs

```
using System.Data;
using System.Windows;
using System.Windows.Input;

using MySql.Data.MySqlClient;

namespace Kursach.View
{
    /// <summary>
    /// Логика взаимодействия для HouseInfo.xaml
    /// </summary>
```

```
    public partial class HouseInfo : Window

    {
        private LandInfo back_window;
        private int Land_Index;
        public DataTable dt;
        public string connection = "server = localhost; user=root; database=kursach; pass-
word=Dragon2012";
        public MySqlConnection connector;
        public string Query;
        public MySqlCommand command;

        public HouseInfo(LandInfo w, int ind)
        {
            InitializeComponent();
            back_window = w;
            Land_Index = ind;

            updateTable();
        }

        public void updateTable()
        {
            Error.Text = "";

            connector = new MySqlConnection(connection);

            connector.Open();

            dt = new DataTable();

            Query = "SELECT Номер_Здания AS '#', Назначение, Тип, Возведено_самовольно AS
'Самострой', " +
                "Год_постройки AS 'Год постройки', Общая_площадь AS 'Общая площадь', Жи-
лая_площадь AS 'Жилая площадь'," +
                "Износ AS 'Износ, %', Материал_стен AS 'Материал стен', информа-
ция_о_здании.Инвентаризационная_стоимость AS 'Инв-ая стоимость' ," +
                "Этажность from здание RIGHT JOIN информация_о_здании " +
                "ON здание.Код_Здания = информация_о_здании.Код_Здания WHERE зда-
ние.Код_Землевладения = " + Land_Index + " ;";

            command = new MySqlCommand(Query, connector);
            dt.Load(command.ExecuteReader());

            HouseInfoTable.DataContext = dt;
            HouseInfoTable.ItemsSource = dt.DefaultView;

            connector.Close();
        }

        private void CloseButton2_Click(object sender, RoutedEventArgs e)
        {
            this.Close();
            back_window.CloseBoth();
        }

        public void CloseAll()
        {
            this.Close();
                back_window.CloseBoth();
        }

        private void HideButton2_Click(object sender, RoutedEventArgs e)
        {
```

```
            this.WindowState = WindowState.Minimized;
        }

        private void EditBtn_Click(object sender, RoutedEventArgs e)
        {
            if (HouseInfoTable.SelectedIndex == -1)
            {
                Error.Text = "Не выбрана строка!";
                return;
            }
            Error.Text = "";
           int RowIndex = HouseInfoTable.SelectedIndex;
            int House_Number = (int)dt.Rows[RowIndex][0];



            Query = "SELECT Код_здания FROM здание WHERE Номер_Здания ="+ House_Number +"
AND Код_Землевладения= "+ Land_Index +";";
            connector = new MySqlConnection(connection);

            connector.Open();
            command = new MySqlCommand(Query, connector);
            int House_index = (int)command.ExecuteScalar();


            AddHouse newHouse = new AddHouse(this, 2, Land_Index, House_index);
            newHouse.Show();
            connector.Close();
        }

        private void DeleteBtn_Click(object sender, RoutedEventArgs e)
        {
            if (HouseInfoTable.SelectedIndex == -1)
            {
                Error.Text = "Не выбрана строка!";
                return;
            }
            Error.Text = "";
           int RowIndex = HouseInfoTable.SelectedIndex;
            int House_Number = (int)dt.Rows[RowIndex][0];

            Query = "SELECT Код_здания FROM здание WHERE Номер_Здания =" + House_Number + "
AND Код_Землевладения= " + Land_Index + ";";
            connector = new MySqlConnection(connection);

            connector.Open();
            command = new MySqlCommand(Query, connector);
            int House_index = (int)command.ExecuteScalar();

            Query = "DELETE FROM информация_о_здании WHERE информация_о_здании.Код_Здания =
" + House_index + ";";
            command = new MySqlCommand(Query, connector);
            command.ExecuteScalar();

            Query = "DELETE FROM помещение WHERE помещение.Код_Здания = " + House_index +
";";
            command = new MySqlCommand(Query, connector);
            command.ExecuteScalar();

            Query = "DELETE FROM здание WHERE здание.Код_Здания = " + House_index + ";";
            command = new MySqlCommand(Query, connector);
            command.ExecuteScalar();

            updateTable();
```

```
        }

        private void AddBtn_Click(object sender, RoutedEventArgs e)
        {
            AddHouse newHouse = new AddHouse(this,1,Land_Index,0);
            newHouse.Show();
        }

        private void BackBtn_Click(object sender, RoutedEventArgs e)
        {
            this.Close();
            back_window.Show();
        }

        private void HouseInfoTable_MouseDoubleClick(object sender, MouseButtonEventArgs e)
        {
            int RowIndex = HouseInfoTable.SelectedIndex;
            int houseIndex = (int)dt.Rows[RowIndex][0];
            RoomInfo room = new RoomInfo(this,houseIndex);
            this.Hide();
            room.Show();
            Error.Text = "";
        }
    }
}
```

//AddHouse.xaml.cs

```
using System.Data;
using System.Collections.Generic;
using System.Windows;

using MySql.Data.MySqlClient;
using Kursach.View;

namespace Kursach
{
    /// <summary>
    /// Логика взаимодействия для AddHouse.xaml
    /// </summary>

    public partial class AddHouse : Window
    {
        public string[] House_types { get; set; }
       public string connection = "server = localhost; user=root; database=kursach; pass-
word=Dragon2012";
        private int type;
        private int House_Index;
        private int Land_Index;
        private HouseInfo the_window;


        public AddHouse(HouseInfo win, int _type, int land_index, int house_index)
        {
            InitializeComponent();
            the_window = win;
            type = _type;
            Land_Index = land_index;
            House_Index = house_index;

            House_types = new string[] { "Основной", "Вспомогательный" };
            TypeBox.DataContext = House_types;
            TypeBox.ItemsSource = House_types;

            MySqlConnection connector = new MySqlConnection(connection);
```

```
            DataTable dt = new DataTable();
            string Query = "SELECT Назначение_Здания from назначение_здания;";
            connector.Open();
            MySqlCommand command = new MySqlCommand(Query, connector);
            dt.Load(command.ExecuteReader());

            List<string> Purposes = new List<string>();

            for (int i = 0; i < dt.Rows.Count; i++)
            {
                string s = dt.Rows[i][0].ToString();
                Purposes.Add(s);
            }

            PurposeBox.DataContext = Purposes;
            PurposeBox.ItemsSource = Purposes;
            //////////////////////////


            if (type == 2)
            {
                this.Title = "Изменить здание";
                dt = new DataTable();

                Query = "SELECT Возведено_самовольно, Назначение, Тип," +
                   "Год_постройки, Износ, Общая_площадь, Жилая_площадь, Материал_стен, Ин-
вентаризационная_стоимость," +
                   "Этажность, Номер_здания FROM информация_о_здании RIGHT JOIN здание ON
здание.Код_здания = информация_о_здании.Код_здания" +
                   " WHERE информация_о_здании.Код_Здания = " + House_Index + ";";

                command = new MySqlCommand(Query, connector);
                dt.Load(command.ExecuteReader());

                string SelfBuild__ = dt.Rows[0][0].ToString();


                if (SelfBuild__ == "1") SelfBuildBox.IsChecked = true;

                PurposeBox.Text = (string)dt.Rows[0][1];
                TypeBox.Text = (string)dt.Rows[0][2];
                YearBox.Text = dt.Rows[0][3].ToString();
                BrokenBox.Text = dt.Rows[0][4].ToString();
                TotalSquareBox.Text = dt.Rows[0][5].ToString();
                LifeSquareBox.Text = dt.Rows[0][6].ToString();
                WallsBox.Text = (string)dt.Rows[0][7];
                PriceBox.Text = dt.Rows[0][8].ToString();
                LevelsBox.Text = dt.Rows[0][9].ToString();
                NumberBox.Text = dt.Rows[0][10].ToString();

            }

            connector.Close();
        }

        private void CancelButton_Click(object sender, RoutedEventArgs e)
        {
            this.Close();
        }

        private void OkButton_Click(object sender, RoutedEventArgs e)
        {
            string connection = "server = localhost; user=root; database=kursach; pass-
word=Dragon2012";
```

```
            MySqlConnection connector = new MySqlConnection(connection);
            connector.Open();

            int SelfBuild = 0;

            if (SelfBuildBox.IsChecked == true) SelfBuild = 1;

            if (PurposeBox.Text == "" || TypeBox.Text == "" || YearBox.Text == "" || Bro-
kenBox.Text == "" ||
                TotalSquareBox.Text == "" || LifeSquareBox.Text == "" || WallsBox.Text == ""
|| PriceBox.Text == "" || LevelsBox.Text == "" || NumberBox.Text == "")
            {

                test.Text = "Ошибка! Пустое поле ввода!";
                return;
            }

            string Purpose = SQLFixer(PurposeBox.Text);
            string Type = SQLFixer(TypeBox.Text);
            string Year = SQLFixer(YearBox.Text);
            string Wear = SQLFixer(BrokenBox.Text);
            string TotalSquare = SQLFixer(TotalSquareBox.Text);

            string LifeSquare = SQLFixer(LifeSquareBox.Text);
            string WallsMaterial = SQLFixer(WallsBox.Text);
            string Price = SQLFixer(PriceBox.Text);
            string Levels = SQLFixer(LevelsBox.Text);
            string Number = SQLFixer(NumberBox.Text);

            if (type == 1) //Добавить
            {


                string Query = "INSERT INTO здание (Номер_Здания,Код_Землевладения) VALUES "
+
                    "(" + Number + " ," + Land_Index + ");";
                MySqlCommand command = new MySqlCommand(Query, connector);
                command.ExecuteScalar();

                Query = "SELECT Код_Здания FROM здание ORDER BY Код_Здания DESC;";
                command = new MySqlCommand(Query, connector);
                House_Index = (int)command.ExecuteScalar();

                Query = "INSERT INTO информация_о_здании (Назначение, Тип, Возведе-
но_самовольно, Год_Постройки, Общая_площадь, Жилая_площадь," +
                    "Износ, Материал_стен, Инвентаризацион-
ная_стоимость,Этажность,Код_здания) VALUES " +
                    "(" + Purpose + ", " + Type + ", " + SelfBuild + ", " + Year + ", " +
TotalSquare + ", " + LifeSquare + "" +
                    ", " + Wear + ", " + WallsMaterial + ", " + Price + ", " + Levels + ", "
+ House_Index + ");";
                command = new MySqlCommand(Query, connector);
                command.ExecuteScalar();
            }
            else if (type == 2)
            {


                string Query = "UPDATE здание SET Номер_Здания=" + Number + " WHERE
(Код_Здания=" + House_Index + ");";
                MySqlCommand command = new MySqlCommand(Query, connector);
                command.ExecuteScalar();
```

```
                Query = "UPDATE информация_о_здании SET Назначение=" + Purpose + ", Тип= " +
Type + ", Возведено_самовольно=" + SelfBuild + ", " +
                    "Год_Постройки=" + Year + ", Общая_площадь = " + TotalSquare +
",Жилая_Площадь=" + LifeSquare + ",Износ=" + Wear + "" +
                    ",Материал_стен=" + WallsMaterial + ",Инвентаризационная_стоимость=" +
Price + ", Этажность="+ Levels +"" +
                    " WHERE (Код_Здания=" + House_Index + ");";
                command = new MySqlCommand(Query, connector);
                command.ExecuteScalar();

            }
            this.Close();
            the_window.updateTable();
        }

        private string SQLFixer(string a)
        {
            a = string.Concat("'", a, "'");

            return a;
        }
    }
    }
```

//RoomInfo.xaml.cs

```
using System.Data;
using System.Windows;

using MySql.Data.MySqlClient;
using Kursach.View;

namespace Kursach
{
    /// <summary>
    /// Логика взаимодействия для Window1.xaml
    /// </summary>
    public partial class RoomInfo : Window
    {
        private HouseInfo back_window;
        private int House_Index;
        public DataTable dt;
        public string connection = "server = localhost; user=root; database=kursach; pass-
word=Dragon2012";
        public MySqlConnection connector;
        public string Query;
        public MySqlCommand command;
        public RoomInfo(HouseInfo win, int ind)
        {
            back_window = win;
            House_Index = ind;

            InitializeComponent();
            updateTable();
        }

        public void updateTable()
        {
            Error.Text = "";

            connector = new MySqlConnection(connection);

            connector.Open();

            dt = new DataTable();
```

```
            Query = "SELECT Номер_Помещения AS '#', Назначение_помещения AS 'Назначение по-
мещения', Площадь, Высота, " +
                "Этаж FROM помещение RIGHT JOIN информация_о_помещении " +
                "ON помещение.Код_Помещения = информация_о_помещении.Код_помещения WHERE по-
мещение.Код_Здания = " + House_Index + " ;";

            command = new MySqlCommand(Query, connector);
            dt.Load(command.ExecuteReader());

            RoomInfoTable.DataContext = dt;
            RoomInfoTable.ItemsSource = dt.DefaultView;

            connector.Close();
        }
        private void BackBtn_Click(object sender, RoutedEventArgs e)
        {
            this.Close();
            back_window.Show();
        }

        private void CloseButton_Click(object sender, RoutedEventArgs e)
        {
            this.Close();
            back_window.CloseAll();
        }

        private void HideButton_Click(object sender, RoutedEventArgs e)
        {
            this.WindowState = WindowState.Minimized;
        }

        private void AddBtn_Click(object sender, RoutedEventArgs e)
        {
            AddRoom newHouse = new AddRoom(this, 1, House_Index, 0);
            newHouse.Show();
        }

        private void DeleteBtn_Click(object sender, RoutedEventArgs e)
        {
            if (RoomInfoTable.SelectedIndex == -1)
            {
                Error.Text = "Не выбрана строка!";
                return;
            }
            Error.Text = "";
            int RowIndex = RoomInfoTable.SelectedIndex;
            int Room_Number = (int)dt.Rows[RowIndex][0];

            Query = "SELECT Код_помещения FROM помещение WHERE Номер_Помещения =" +
Room_Number + " AND Код_Здания = " + House_Index + ";";
            connector = new MySqlConnection(connection);

            connector.Open();
            command = new MySqlCommand(Query, connector);
            int Room_index = (int)command.ExecuteScalar();

            Query = "DELETE FROM информация_о_помещении WHERE информа-
ция_о_помещении.Код_Помещения = " + Room_index + ";";
            command = new MySqlCommand(Query, connector);
            command.ExecuteScalar();

            Query = "DELETE FROM помещение WHERE помещение.Код_Помещения = " + Room_index +
";";
```

```
            command = new MySqlCommand(Query, connector);
            command.ExecuteScalar();

            updateTable();
        }

        private void EditBtn_Click(object sender, RoutedEventArgs e)
        {
            if (RoomInfoTable.SelectedIndex == -1)
            {
                Error.Text = "Не выбрана строка!";
                return;
            }
            Error.Text = "";
            int RowIndex = RoomInfoTable.SelectedIndex;
            int Room_Number = (int)dt.Rows[RowIndex][0];

            Query = "SELECT Код_помещения FROM помещение WHERE Номер_Помещения =" +
Room_Number + " AND Код_Здания= " + House_Index + ";";
            connector = new MySqlConnection(connection);

            connector.Open();
            command = new MySqlCommand(Query, connector);
            int Room_index = (int)command.ExecuteScalar();


            AddRoom newHouse = new AddRoom(this, 2, House_Index, Room_index);
            newHouse.Show();
            connector.Close();
        }
    }
}
```

//AddRoom.xaml.cs

```
using System.Data;
using System.Collections.Generic;
using System.Windows;

using MySql.Data.MySqlClient;


namespace Kursach
{
    /// <summary>
    /// Логика взаимодействия для AddRoom.xaml
    /// </summary>
    public partial class AddRoom : Window
    {
        public string connection = "server = localhost; user=root; database=kursach; pass-
word=Dragon2012";
        private int type;
        private int House_Index;
        private int Room_Index;
        private RoomInfo the_window;

        public AddRoom(RoomInfo win,int _type, int house_index, int room_index)
        {
            InitializeComponent();

            the_window = win;
            type = _type;
            House_Index = house_index;
            Room_Index = room_index;
```

```
            MySqlConnection connector = new MySqlConnection(connection);
            DataTable dt = new DataTable();
            string Query = "SELECT Назначение_Помещения from назначение_помещения;";
            connector.Open();
            MySqlCommand command = new MySqlCommand(Query, connector);
            dt.Load(command.ExecuteReader());

            List<string> Purposes = new List<string>();

            for (int i = 0; i < dt.Rows.Count; i++)
            {
                string s = dt.Rows[i][0].ToString();
                Purposes.Add(s);
            }

            PurposeBox.DataContext = Purposes;
            PurposeBox.ItemsSource = Purposes;
            //////////////////////////


            if (type == 2)
            {
                this.Title = "Изменить помещения";
                dt = new DataTable();

                Query = "SELECT Назначение_помещения, Площадь, Высота, Этаж, Номер_помещения
" +
                    "FROM информация_о_помещении RIGHT JOIN помещение ON помеще-
ние.Код_помещения = информация_о_помещении.Код_помещения" +
                  " WHERE информация_о_помещении.Код_Помещения = " + Room_Index + ";";

                command = new MySqlCommand(Query, connector);
                dt.Load(command.ExecuteReader());

                PurposeBox.Text = (string)dt.Rows[0][0].ToString();
                SquareBox.Text = (string)dt.Rows[0][1].ToString();
                HeightBox.Text = dt.Rows[0][2].ToString();
                LevelBox.Text = dt.Rows[0][3].ToString();
                NumberBox.Text = dt.Rows[0][4].ToString();

            }

            connector.Close();

        }

        private void OkButton_Click(object sender, RoutedEventArgs e)
        {

            MySqlConnection connector = new MySqlConnection(connection);
            connector.Open();


            if (PurposeBox.Text == "" || SquareBox.Text == "" || HeightBox.Text == "" ||
LevelBox.Text == "" || NumberBox.Text == "")
            {

                test.Text = "Ошибка! Пустое поле ввода!";
                return;
            }

            string Purpose = SQLFixer(PurposeBox.Text);
            string Square = SQLFixer(SquareBox.Text);
            string Height = SQLFixer(HeightBox.Text);
```

```
            string Level = SQLFixer(LevelBox.Text);
            string Number = SQLFixer(NumberBox.Text);

            if (type == 1)
            {

                string Query = "INSERT INTO помещение (Номер_Помещения,Код_Здания) VALUES "
+
                    "(" + Number + " ," + House_Index + ");";
                MySqlCommand command = new MySqlCommand(Query, connector);
                command.ExecuteScalar();

                Query = "SELECT Код_Помещения FROM помещение ORDER BY Код_Помещения DESC;";
                command = new MySqlCommand(Query, connector);
                Room_Index = (int)command.ExecuteScalar();

                Query = "INSERT INTO информация_о_помещении (Назначение_помещения, Площадь,
Высота, Этаж, Код_помещения) VALUES " +
                    "(" + Purpose + ", " + Square + ", " + Height + ", " + Level + ", " +
Room_Index + ");";
                command = new MySqlCommand(Query, connector);
                command.ExecuteScalar();
            }
            else if (type == 2) //Изменить
            {

                string Query = "UPDATE помещение SET Номер_Помещения=" + Number + " WHERE
(Код_Помещения=" + Room_Index + ");";
                MySqlCommand command = new MySqlCommand(Query, connector);
                command.ExecuteScalar();

                Query = "UPDATE информация_о_помещении SET Назначение_помещения=" + Purpose
+ ", Площадь= " + Square + ", Высота=" + Height + ", " +
                    "Этаж=" + Level + " WHERE (Код_Помещения=" + Room_Index + ");";
                command = new MySqlCommand(Query, connector);
                command.ExecuteScalar();

            }
            this.Close();
            the_window.updateTable();
        }

        private void CancelButton_Click(object sender, RoutedEventArgs e)
        {
            this.Close();
        }
        private string SQLFixer(string a)
        {
            a = string.Concat("'", a, "'");

            return a;
        }

    }
}
```

ПРИЛОЖЕНИЕ Б

(рекомендуемое)

РЕЗУЛЬТАТ ПРОВЕРКИ НА ОРИГИНАЛЬНОСТЬ

Рисунок Б.1 – Проверка на заимствования