Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных технологий

Кафедра информационных систем и технологий

Дисциплина: Базы данных

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

СИСТЕМА УЧЕТА КНИГ В БИБЛИОТЕКЕ

БГУИР КП 1-40 01 01 009 ПЗ

Студент: гр. 081074 Дрозд М.А.

Руководитель: Бакунов А.М.

Минск 2022

**Содержание**

[Введение 3](#_Toc103647168)

[1 Постановка задачи 4](#_Toc103647169)

[1.1 Анализ предметной области 4](#_Toc103647170)

[1.2 Обзор существующих аналогов 5](#_Toc103647171)

[1.3 Информационная база задачи 7](#_Toc103647172)

[1.4 Функциональное назначение 7](#_Toc103647173)

[1.5 Выбор и обоснование средств разработки 8](#_Toc103647174)

[2 Проектирование программного средства 10](#_Toc103647175)

[2.1 Инфологический этап проектирования 10](#_Toc103647176)

[2.2 Логический этап проектирования 10](#_Toc103647177)

[2.3 Функциональная модель 12](#_Toc103647178)

[2.4 Конфигурация и состав ПО, требуемых для работы приложения 13](#_Toc103647179)

[3 Программная реализация 14](#_Toc103647180)

[3.1 Физическая модель базы данных 14](#_Toc103647181)

[3.2 Реализация проекта базы данных 14](#_Toc103647182)

[3.2.1 Создание и заполнение таблиц базы данных 14](#_Toc103647183)

[3.2.2 Описание разработанных компонентов базы данных 17](#_Toc103647184)

[3.3 Структура приложения 19](#_Toc103647185)

[4 Тестирование 21](#_Toc103647186)

[4.1 Методы и типы тестирования 21](#_Toc103647187)

[4.2 Результат тестирования 23](#_Toc103647188)

[5 Руководство пользователя 29](#_Toc103647189)

[Заключение 37](#_Toc103647190)

[Список использованных источников 38](#_Toc103647191)

[Приложение А 39](#_Toc103647192)

[Приложение Б 45](#_Toc103647193)

[Приложение В 46](#_Toc103647194)

[Приложение Г 47](#_Toc103647195)

# **Введение**

Темой курсового проекта является программное средство «Система учета книг в библиотеке», которое будет помогать библиотекарям вести учет книг.

Задачей курсового проекта является изучение понятия и принципы моделирования (построения) БД, а также способы работы с БД.

Развитие современного общества вплотную связано с развитием вычислительной техники и широкой информатизацией всех сфер человеческой деятельности. Трудно представить нашему современнику, что буквально совсем недавно человек проводил всякое вычисление не при помощи машины, а вручную. Сейчас же практически в любой сфере жизни человека присутствуют настольные персональные компьютеры, ноутбуки, планшеты, смартфоны, которые заняли достойное место в жизни людей. Они улучшают и облегчают их деятельность.

С развитием компьютерных технологий стали появляться все новые и новые варианты их использования. Будет странным увидеть человека, который использует компьютер только для вычислений, ведь современные технологии позволяют использовать компьютер в абсолютно различных направлениях

Развитие компьютерной техники привело к её внедрению практически во все области жизнедеятельности человека, одной из которых является библиотекарское дело.

Для данного курсового проекта будет разработана система учета книг для библиотеки, которая будет облегчать труд библиотекарей и уменьшать количество бумажной работы.

# **1** **Постановка задачи**

## **Анализ предметной области**

Библиотека – это учреждение культуры, организующее сбор, хранение и общественное пользование произведениями печати и другими документами. Библиотеки систематически занимаются сбором, хранением и выдачей читателям произведений печати.

Основными направлениями работы любой библиотеки являются: комплектование и организация книжного фонда; обслуживание читателей.

Комплектование фондов библиотеки состоит из систематического выявления (путем просмотра библиографических источников и литературы) нужных для данной библиотеки изданий и приобретения их. От своевременности и полноты комплектования библиотеки в значительной мере зависит уровень обслуживания читателей.

Организация книжного фонда включает вопросы учета, расстановки, хранения литературы и доставки ее читателю.

Учет библиотечного фонда – это прием поступивших в библиотеку изданий, запись прибывших и исключение по тем или иным причинам непригодных для читателей изданий в документах библиотеки, которые дают точные сведения о наличном составе фонда и помогают обеспечить его сохранность.

Правильная организация фонда облегчает читателю пользование литературой, библиотекарю - быстрое выполнение читательских требований, а также обеспечивает сохранность фондов как общественной собственности.

В библиотеке ведется картотека читателей. О каждом читателе заносятся следующие сведения:

* фамилия;
* имя;
* отчество;
* дата рождения;
* адрес;
* мобильный телефон.

Обслуживание читателей библиотеки осуществляется различным путем выдачи литературы и помощи отдельным читателям в подборе необходимой им литературы.

Срок пользования литературой для различных категорий читателей и количество выдаваемых изданий на каждом абонементе определяется администрацией, исходя из вида литературы и категории читателя. Число выдаваемых книг не ограничивается.

Библиотечная деятельность связана с учетом большого количества операций, множество книг и читателей серьезно замедляют работу библиотекарей. Сложность поиска нужной книги в каталоге занимает длительное время, и целиком опирается на компетентность работников библиотеки.

Каждая книга может присутствовать в нескольких экземплярах.

Все книги различаются по своему уникальному номеру.

БД находится на специальном компьютере в электронном виде, постоянно редактируется и изменяется работниками библиотеки. С базой данных библиотеки могут работать как руководители (администрация) библиотеки, так и работники библиотеки, заинтересованные в получении необходимой информации. При работе с системой библиотекарь должен иметь возможность решать следующие задачи:

Принимать новые книги и регистрировать их в библиотеке.

Вести учет выданных книг читателям, при этом предполагается два режима работы: выдача книг читателю и прием от него возвращаемых им книг обратно в библиотеку. При выдаче книг фиксируется, когда и какая книга была выдана данному читателю и на какой срок выдается данная книга. При приеме книги, возвращаемой читателем, проверяется соответствие названия книги, и она ставится на свое старое место в библиотеке.

Администрация библиотеки должна иметь возможность получать сведения о должниках - читателях библиотеки, которые не вернули вовремя взятые книги.

Таким образом был произведен анализ предметной области работника сферы библиотекаря.

## **1.2 Обзор существующих аналогов**

На данный момент существует несколько аналогов, находящихся в свободном доступе.

Программа OPAC-Global обладает очень широким функционалом: ведение учета книг, ведение учета читателей и каталогизация, однако для того чтобы научится этим всем пользоваться, необходимо проходимо долгое и недешевое обучение у компании, выпускающей данный продукт. Помимо этого, заказчика не устраивает то, что программа использует внутреннюю базу данных, а заказчик хочет использовать уже существующую базу данных MySQL. Рабочее окно программы OPAC-Global представлено на рис. 1.1:

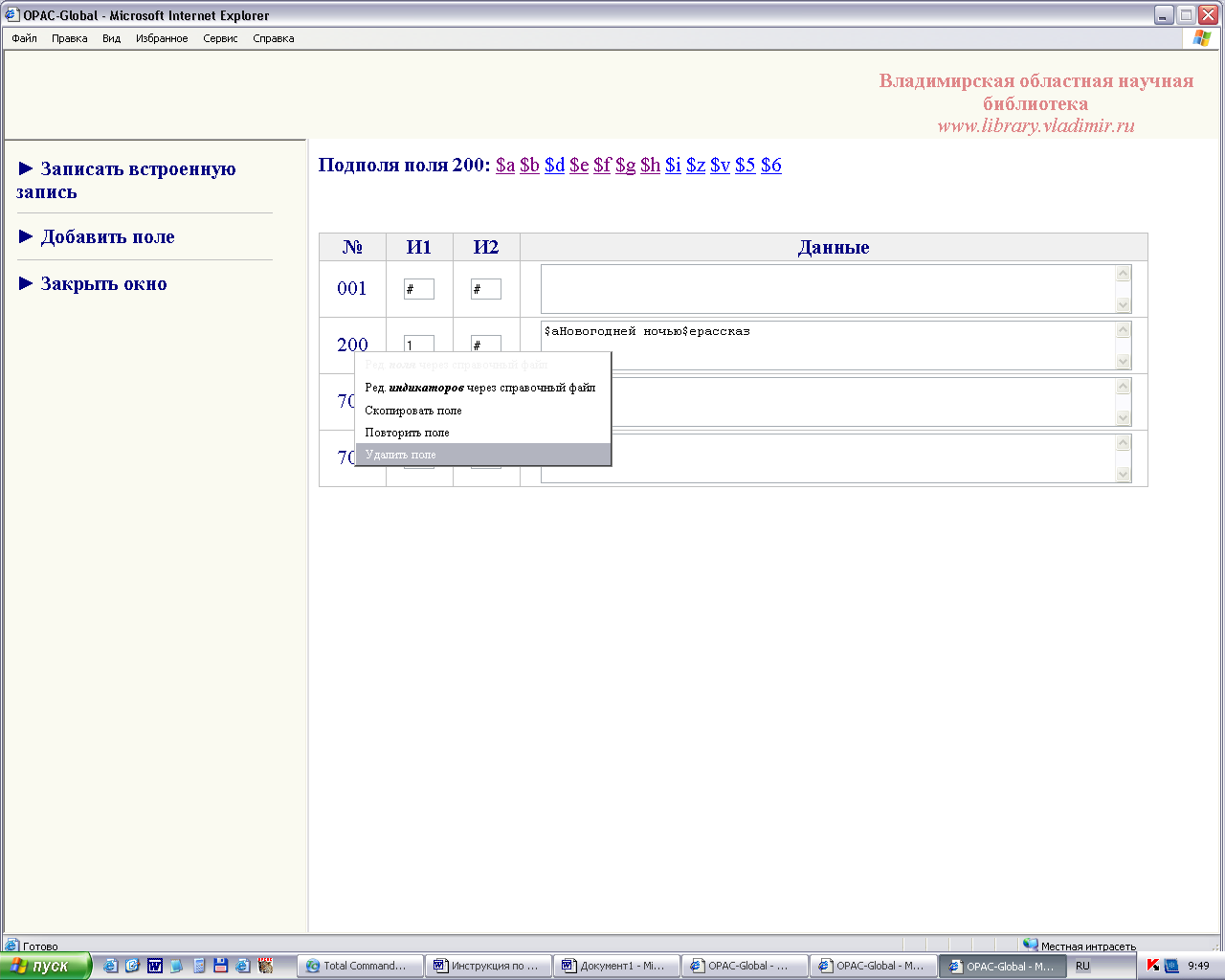


Рис. 1.1 – Рабочее окно программы OPAC-Global

Программа 1С:Библиотека имеет схожий функционал программы OPAC-Global однако из-за более простого интерфейса научиться пользоваться ей становится намного легче. Но даже эта программа не умеет подключаться к уже существующим базам данных MySQL, что делает эту программу плохим вариантом для заказчика.

Программа 1С:Библиотека, рабочее окно которой представлено на рис. 1.2:

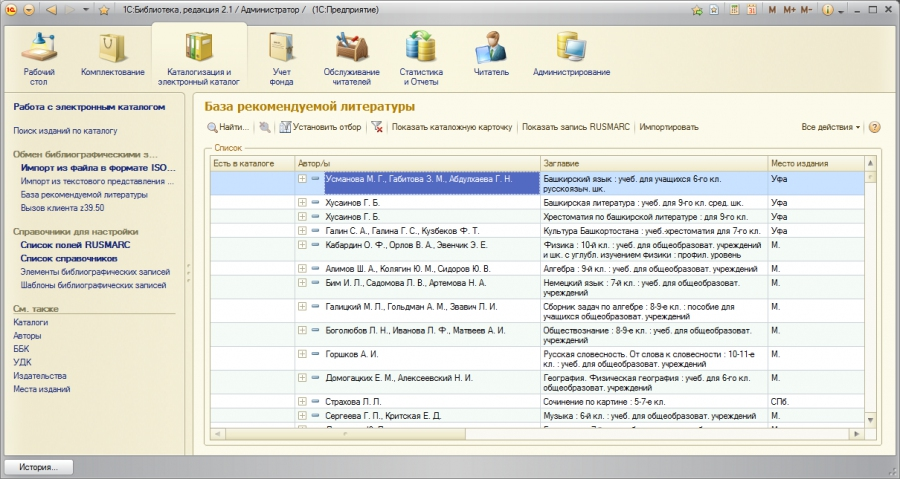


Рис. 1.2 Рабочее окно программы 1С:Библиотека

Таким образом на основе приведенных аналогов, можно сделать вывод о том, какими преимуществами аналогов должно обладать разрабатываемое средство и какие недостатки должны в нем отсутствовать. К достоинствам программного средства будут относится возможность работы с уже существующей базой данных MySQL и простой интерфейс.

## **1.3 Информационная база задачи**

Информационная база задачи состоит из входной, выходной и постоянной информации.

Входные данные:

* данные о книгах;
* данные о читателях;
* данные об условиях выдачи книги;
* критерии и данные для поиска;
* данные о датах выдачи книг.

Выходными данные:

* данные о взятых книгах читателями;
* отфильтрованные читатели по заданным критериям;
* отфильтрованные книги по заданным критериям;
* данные о должниках, не отдавших книги в срок.

Постоянными данными в программе является информация о библиотеке.

## **1.4 Функциональное назначение**

Данное приложение предназначено для хранения, редактирования, создания информации о книгах, писателях, жанрах, издательствах, языках, городах, обложках, а также возможности выдачи их читателям и учета выданных книг.

Приложения должно содержать следующие вкладки: книги, должники, читатели, авторы, языки, жанры, издатели, города, обложки. А также предусмотреть вспомогательные окна для создания и редактирования каждой из выше представленных сущностей.

Так же приложение должно реализовывать следующие функции:

* добавление, редактирование и удаление читателей;
* добавление, редактирование и удаление книг;
* возможность выдачи книги читателю;
* поиск по читателям по различным критериям;
* поиск по книгам по различным критериям;
* возможность продлить дату возврата книги читателя;
* возможность вернуть книгу читателю;
* контроль сроков возврата книги путем сравнения текущей даты и даты предположительной сдачи книги.

Программа представляет собой стандартное оконное Windows-приложение. Осуществляет взаимодействие с пользователем при помощи различных элементов управления: командных кнопок, текстовых полей для ввода значений, графических кнопок.

## **1.5 Выбор и обоснование средств разработки**

Выбор среды разработки и языка программирования всегда очень важен. Их нужно выбирать исходя из своих возможностей и возможностей среды разработки и языка, которые должны обеспечить соответствие программного средства с требованиями пользователя.

Для разработки этого программного средства нужны различные инструменты и средства, которые более подробно будут описаны ниже.

Для разрабатываемого программного продукта используются следующие средства:

* Visual Studio 2019;
* MySQL;
* С#.

В качестве среды разработки был выбран Visual Studio 2019.

Microsoft Visual Studio – бесплатная, полнофункциональная и расширяемая интегрированная среда разработки программного обеспечения, и ряд других инструментальных средств для создания современных приложений для Windows. Данная среда разработки предоставляет:

* мощные инструменты программирования, которые обеспечивают легкость написания кода, перемещения по нему и устранение ошибок;
* отладка для языка С#.

В качестве базы данных использовался MySQL.

MySQL характеризуется своей надежностью и простотой, гибкой поддержкой форматов чисел, строк переменной длины и меток времени, быстрой работой, масштабируемостью.

Поддерживаемые типы значений: TINYINT, SMALLINT, MEDIUMINT, INT, BIGINT, FLOAT, DOUBLE, DOUBLE, REAL, DECIMAL, NUMERIC, DATE, DATETIME, TIMESTAMP, TIME, YEAR, CHAR, VARCHAR, MEDIUMTEXT, BLOB, LONGBLOB, LONGTEXT, BLOB, ENUM и SET. Так же поддерживается специальное значение NULL.

Каждое значение в любом поле любой записи может быть любого из этих типов, независимо от типа, указанного при объявлении полей таблицы. Указанный при объявлении поля тип хранится для справки в его исходном написании, и используется в качестве основы для выбора предпочтений (так называемое «type affinity»: это подход, редко встречающийся в других СУБД) при выполнении неявных преобразований типов на основании похожести этого названия типа на что-либо, знакомое MySQL. В этот алгоритм зашит обширный перечень практикуемых в других СУБД вариантов названий типов данных. Если безопасного преобразования записываемого значения в предпочитаемый тип не получается, MySQL записывает значение в его исходном виде. Для получения значений из базы есть ряд функций для каждого из типов, и, если тип хранимого значения не соответствует запрашиваемому, оно тоже, по возможности, преобразуется.

В качестве языка программирования для реализации данного проекта был выбран С# – компилируемый статически типизированный язык программирования общего назначения. Именно этот язык наиболее подходит для написания системы учета книги библиотеки, так как С# успешно используется во многих областях приложения, далеко выходящих за указанные рамки, и позволяет создавать как простые приложения и утилиты, так и сложные системы.

Поддерживает такие парадигмы программирования как процедурное программирование, объектно-ориентированное программирование, обобщённое программирование, обеспечивает модульность, раздельную компиляцию, обработку исключений, абстракцию данных, объявление типов (классов) объектов, виртуальные функции. Стандартная библиотека включает, в том числе, общеупотребительные контейнеры и алгоритмы. C# сочетает свойства как высокоуровневых, так и низкоуровневых языков. В сравнении с его предшественником – языком C++, – наибольшее внимание уделено поддержке объектно-ориентированного и обобщённого программирования.

C# широко используется для разработки программного обеспечения, являясь одним из самых популярных языков программирования. Область его применения включает создание операционных систем, разнообразных прикладных программ, драйверов устройств, приложений для встраиваемых систем, высокопроизводительных серверов, а также развлекательных приложений (игр).

C# – чрезвычайно мощный язык, содержащий средства создания эффективных программ практически любого назначения, от низкоуровневых утилит и драйверов до сложных программных комплексов самого различного назначения.

В частности, поддерживаются различные стили и технологии программирования, включая традиционное директивное программирование, ООП; имеется возможность работы на низком уровне с памятью, адресами, портами; возможность создания обобщённых алгоритмов для разных типов данных, их специализация и вычисления на этапе компиляции, используя шаблоны.

# **2 Проектирование программного средства**

## **2.1 Инфологический этап проектирования**

Инфологическое проектирование – построение семантической модели предметной области, то есть информационной модели наиболее высокого уровня абстракции. Для описания семантической модели я использовал диаграмму сущность-связь.

Для программы системы учета книг в библиотеке были выделены следующие главные сущности: автор, книга, издательство, библиотекарь, администратор, читатель.

У сущности «автор» присутствуют следующие атрибуты: имя, фамилия, отчество. Сущность «издательство» содержит атрибуты город и название. Сущность «читатель» состоит из атрибутов имя, фамилия, отчество, адрес, номер телефона и дата рождения. У сущностей «администратор» и «библиотекарь» присутствуют атрибуты имя, фамилия и хэш пароля. У сущности «книга» имеются атрибуты название, автор, жанр, тип обложки, размер книги, шрифт текста и вид литературы.

Диаграмма сущность связь представлена в приложении Б.

Взаимоотношения между сущностями иллюстрируются с помощью связей. В данной диаграмме имеются следующие связи: связь один-ко-многим между «автором» и «книгой». Связь один-ко-многим между «издательством» и «книгой». Эта связь показывает, что экземпляр книги может быть издан только одним издательством, а издательство может издать сколько угодно экземпляров книг. Связь один-ко-многим между «библиотекарем» и «администратором». Она показывает, что библиотекарь может быть создан только одним администратором, но администратор может создать сколько угодно библиотекарей. Связь многие-ко-многим между «читателем» и «книгой». Она показывает, что книга за свое существование могла находиться у многих читатели, а у читателей, за свое существование, могло быть много книг. У этой связи так же атрибуты даты взятия, даты возврата, а также библиотекаря, который выдал книгу читателю.

Таким образом была спроектирована концептуальная модель БД.

## **2.2 Логический этап проектирования**

Логическое проектирование – создание абстрактной схемы базы данных, которую можно подвести под любую конкретную СУДБ. Для реляционной модели данных даталогическая модель – набор схем отношений, обычно с указанием первичных ключей, а также связей между отношениями, представляющих собой внешние ключи.

Логическая модель данных представлена на рис. 2.1:

Diagram

Description automatically generated

Рис. 2.1 Логическая модель данных

Будем рассматривать логическое проектирование БД для реляционной модели данных, так как современные СУБД – реляционные.

Разрабатываемая схема БД будет содержать следующие таблицы:

* таблица «Читатель»;
* таблица «Автор»;
* таблица «Жанр»;
* таблица «Язык»;
* таблица «Город»;
* таблица «Издательство»;
* таблица «Обложка»;
* таблица «Эра»;
* таблица «Национальность литературы»;
* таблица «Размер книги»;
* таблица «Размер шрифта»;
* таблица «Книга»;
* таблица «Администратор»;
* таблица «Библиотекарь»;
* таблица «Аренда книги»;

Таблица «Читатель» хранит информацию о читателе, его данные: имя, фамилия, отчество, дата рождения, номер телефона и адрес.

Таблица «Автор» хранит информацию об авторе, его имя, фамилию, отчество.

Таблица «Жанр» хранит всевозможные жанры.

Таблица «Язык» хранит различные языки.

Таблица «Город» хранит различные горда.

Таблица «Издательство» хранит разных издательства, их названия, а также эта таблица связана с таблицей «city». Так, у издательства есть и город.

Таблица «Обложка» хранит виды обложек.

Таблица «Эра» хранит список временных эпох, в которых была написана книга.

Таблица «Национальность литературы» хранит национальность вида литературы.

Таблица «Размер книги» хранит примерные размеры книги в мм, а также общее название для такого размера.

Таблица «Размер шрифта» хранит размер шрифта, использованного при печати, а также его название.

Таблица «Книга» соединена связью один-ко-многим с таблицами «Автор», «Жанр», «Язык», «Издательство», «Обложка», «Эра», «Национальность литературы», «Размер книги» и «Размер шрифта», а также имеет с названием книги.

Таблица «Администратор» хранит имя, фамилию и хэш пароля администратора.

Таблица «Библиотекарь» хранит имя, фамилию и хэш пароля библиотекаря, а также она связана с таблицей «Администратор», указывая на администратора, создавшего библиотекаря.

Таблица «Аренда книги» связана с таблицами «Читатель», «Книга» и «Библиотекарь», что показывает какому читателю каким библиотекарем была выдана какая книга. Также таблица содержит поля, хранящие дату взятия и дату возврата книги.

## **2.3 Функциональная модель**

Для описания функциональной модели я буду использовать диаграмму вариантов использования. Она представлена в приложении В.

В системе учета книг для библиотеки существуют следующие актеры: администратор, библиотекарь и гость. Гость может только проходить процесс авторизации. В результате которого он станет либо администратором, либо библиотекарем, в зависимости от вида аккаунта, который принадлежит пользователю.

Библиотекарь может работать с книгами и читателями: редактировать, удалять, добавлять их. У библиотекаря есть возможность искать книги и читателей по различным критериям. Так же библиотекарь может выдавать книги читателям, искать должников, продлевать и возвращать книги, находящие у читателей.

Администратор может делать все то же самое, что и библиотекарь, только помимо этого, администратор может управлять списком библиотекарей. Он может добавлять, удалять и просматривать списки библиотекарей.

## **2.4 Конфигурация и состав ПО, требуемых для работы приложения**

Программное средство представляет собой оконное Windows приложение. Для запуска требуется, чтобы компьютер удовлетворял следующим требованиям:

* Процессор Intel Core i3 или более новые модели с тактовой частотой 2 ГГц;
* Microsoft Windows 10;
* 512 МБ ОЗУ (минимум);
* 20 МБ свободного места на диске;
* Монитор VGA разрешением 1024 x 768 с поддержкой полноцветного режима (минимальные требования);
* Мышь или другое указывающее устройство.

Так же, для запуска приложения, на компьютере должен быть установлен Microsoft .NET Framework версии 4.8.

# **3 Программная реализация**

## **3.1 Физическая модель базы данных**

Физическая модель базы данных проектируется под конкретную СУБД. В моем случае это MySQL.

Эта СУБД реляционная, поэтому в ней сохраняется структура таблиц из логического моделирования, однако у полей появляются конкретные типы.

Так, численный тип – это int. Строковый – varchar (когда мы ожидаем, что у размер строки не превысит заданное ограничение) и text – когда длина строки не ограничена. А для даты и времени – datetime.

Физическая модель данных представлена в приложении Г.

Так же в СУБД MySQL принято называть все таблицы и поля на английском языке, поэтому в физической модели данных все слова переведены на английский.

## **3.2 Реализация проекта базы данных**

## **3.2.1 Создание и заполнение таблиц базы данных**

В ходе работы с БД были разработаны следующие запросы на создание таблиц:

Таблица читателей:

CREATE TABLE IF NOT EXISTS `reader` (

`id` int PRIMARY KEY NOT NULL AUTO\_INCREMENT,

`name` varchar(255) NOT NULL,

`surname` varchar(255) NOT NULL,

`patronymic` varchar(255) NOT NULL,

`birthday` DATETIME NOT NULL,

`phone\_number` varchar(255) NOT NULL,

`address` varchar(255) NOT NULL

)

COLLATE utf8\_general\_ci;

Таблица с выданными книгами:

CREATE TABLE IF NOT EXISTS `record` (

`id` int PRIMARY KEY NOT NULL AUTO\_INCREMENT,

`id\_librarian` int NOT NULL,

`id\_reader` int NOT NULL,

`id\_book` int NOT NULL,

`issue\_date` DATETIME NOT NULL,

`return\_date` DATETIME NOT NULL

)

COLLATE utf8\_general\_ci;

Таблица с книгами:

CREATE TABLE IF NOT EXISTS `book` (

`id` int PRIMARY KEY NOT NULL AUTO\_INCREMENT,

`id\_author` int NOT NULL,

`id\_language` int NOT NULL,

`id\_genre` int NOT NULL,

`id\_publishing\_house` int NOT NULL,

`id\_cover` int NOT NULL,

`id\_era` int NOT NULL,

`id\_type\_of\_literature` int NOT NULL,

`id\_book\_size` int NOT NULL,

`id\_font\_size` int NOT NULL,

`title` varchar(255) NOT NULL

)

COLLATE utf8\_general\_ci;

Таблица с авторами:

CREATE TABLE IF NOT EXISTS `author` (

`id` int PRIMARY KEY NOT NULL AUTO\_INCREMENT,

`name` varchar(255) NOT NULL,

`surname` varchar(255) NOT NULL,

`patronymic` varchar(255) NOT NULL

)

COLLATE utf8\_general\_ci;

Таблица с жанрами:

CREATE TABLE IF NOT EXISTS `genre` (

`id` int PRIMARY KEY NOT NULL AUTO\_INCREMENT,

`genre` varchar(255) NOT NULL

)

COLLATE utf8\_general\_ci;

Таблица с языками:

CREATE TABLE IF NOT EXISTS `language` (

`id` int PRIMARY KEY NOT NULL AUTO\_INCREMENT,

`language` varchar(255) NOT NULL

)

COLLATE utf8\_general\_ci;

Таблица с издательствами:

CREATE TABLE IF NOT EXISTS `publishing\_house` (

`id` int PRIMARY KEY NOT NULL AUTO\_INCREMENT,

`id\_city` int NOT NULL,

`title` varchar(255) NOT NULL

)

COLLATE utf8\_general\_ci;

Таблица с городами:

CREATE TABLE IF NOT EXISTS `city` (

`id` int PRIMARY KEY NOT NULL AUTO\_INCREMENT,

`city\_name` varchar(255) NOT NULL

)

COLLATE utf8\_general\_ci;

Таблица с обложками:

CREATE TABLE IF NOT EXISTS `cover` (

`id` int PRIMARY KEY NOT NULL AUTO\_INCREMENT,

`cover\_description` varchar(255) NOT NULL

)

COLLATE utf8\_general\_ci;

Таблица с эрами:

CREATE TABLE IF NOT EXISTS `era` (

`id` int PRIMARY KEY NOT NULL AUTO\_INCREMENT,

`era` varchar(255) NOT NULL

)

COLLATE utf8\_general\_ci;

Таблица с национальностью литературы:

CREATE TABLE IF NOT EXISTS `type\_of\_literature` (

`id` int PRIMARY KEY NOT NULL AUTO\_INCREMENT,

`type\_of\_literature` varchar(255) NOT NULL

)

COLLATE utf8\_general\_ci;

Таблица с библиотекарями:

CREATE TABLE IF NOT EXISTS `librarian` (

`id` int PRIMARY KEY NOT NULL AUTO\_INCREMENT,

`name` varchar(255) NOT NULL,

`surname` varchar(255) NOT NULL,

`password\_hash` text NOT NULL,

`id\_admin` int

)

COLLATE utf8\_general\_ci;

Таблица с администраторами:

CREATE TABLE IF NOT EXISTS `admin` (

`id` int PRIMARY KEY NOT NULL AUTO\_INCREMENT,

`name` varchar(255) NOT NULL,

`surname` varchar(255) NOT NULL,

`password\_hash` text NOT NULL

)

COLLATE utf8\_general\_ci

ENGINE=InnoDB

AUTO\_INCREMENT=1000000;

Таблица с размерами книг:

CREATE TABLE IF NOT EXISTS `book\_size` (

`id` int PRIMARY KEY NOT NULL AUTO\_INCREMENT,

`name` varchar(255) NOT NULL

)

COLLATE utf8\_general\_ci;

Таблица с размерами шрифтов:

CREATE TABLE IF NOT EXISTS `font\_size` (

`id` int PRIMARY KEY NOT NULL AUTO\_INCREMENT,

`name` varchar(255) NOT NULL

)

COLLATE utf8\_general\_ci;

Таблица с логами:

CREATE TABLE IF NOT EXISTS `logs` (

`id` int PRIMARY KEY NOT NULL AUTO\_INCREMENT,

`table\_name` varchar(255) NOT NULL,

`prev\_value` varchar(255),

`new\_value` varchar(255),

`time` DATETIME NOT NULL

)

COLLATE utf8\_general\_ci;

Создание связей в таблицах:

ALTER TABLE `book` ADD FOREIGN KEY (`id\_language`) REFERENCES `language` (`id`);

ALTER TABLE `book` ADD FOREIGN KEY (`id\_author`) REFERENCES `author` (`id`);

ALTER TABLE `publishing\_house` ADD FOREIGN KEY (`id\_city`) REFERENCES `city` (`id`);

ALTER TABLE `book` ADD FOREIGN KEY (`id\_publishing\_house`) REFERENCES `publishing\_house` (`id`);

ALTER TABLE `record` ADD FOREIGN KEY (`id\_book`) REFERENCES `book` (`id`);

ALTER TABLE `record` ADD FOREIGN KEY (`id\_reader`) REFERENCES `reader` (`id`);

ALTER TABLE `book` ADD FOREIGN KEY (`id\_genre`) REFERENCES `genre` (`id`);

ALTER TABLE `book` ADD FOREIGN KEY (`id\_cover`) REFERENCES `cover` (`id`);

ALTER TABLE `book` ADD FOREIGN KEY (`id\_era`) REFERENCES `era` (`id`);

ALTER TABLE `book` ADD FOREIGN KEY (`id\_type\_of\_literature`) REFERENCES `type\_of\_literature` (`id`);

ALTER TABLE `librarian` ADD FOREIGN KEY (`id\_admin`) REFERENCES `admin` (`id`) ON DELETE SET NULL;

ALTER TABLE `book` ADD FOREIGN KEY (`id\_book\_size`) REFERENCES `book\_size` (`id`);

ALTER TABLE `book` ADD FOREIGN KEY (`id\_font\_size`) REFERENCES `font\_size` (`id`);

Заполнение таблицы национальностей литературы:

INSERT INTO `type\_of\_literature` VALUES (1,'Русская'),(2,'Прусская'),(3,'Немецкая'),(4,'Австрийская'),(5,'Белорусская'),(6,'Японская');

Заполнение таблицы авторов:

INSERT INTO `author` VALUES (1,'Иван','Васильевич','Пушкин'),(2,'Максим','Дрозд','Алексеевич'),(3,'Александр ','Павловский','Алексеевич'),(4,'Виктор ','Ших','Адамович'),(5,'Андрей ','Гринкевич','Михайлович');

Заполнение таблицы городов:

INSERT INTO `city` VALUES (1,'Минск'),(2,'Гродно'),(3,'Брест'),(4,'Орша'),(5,'Новогрудок');

Заполнение таблицы читателей:

INSERT INTO `reader` VALUES (1,'Олег','Дорохов','Вячеславович','1994-06-17 00:00:00','+375447745648',' ул.Плотинная, 48, кв.75'),(2,'Анатолий ','Маслов ','Степанович','2000-05-15 00:00:00','ул.Маршала Соколовского, д.28, кв.18','+3754958034053'),(3,'Иван ','Егоров ','Глебович','1950-05-15 00:00:00','+375291143293','ул.Б.Марфинская, 18, кв.12');

## **3.2.2 Описание разработанных компонентов базы данных**

В ходе написания курсового проекта были разработаны триггеры процедуры и функции.

Процедура вставки дефолтного данных администратора, если не было найдено ни одного пользователя.

DROP PROCEDURE IF EXISTS createInitialUser;

CREATE PROCEDURE createInitialUser()

BEGIN

SELECT COUNT(\*)

INTO @count

FROM admin LIMIT 1;

IF @count = 0 THEN

INSERT INTO admin (name, surname, password\_hash) VALUES ('admin', 'admin', 'admin');

END IF;

END;//

Функция генерации сообщения в логи при создании библиотекаря:

DROP FUNCTION IF EXISTS creare\_log\_insert\_librarian\_statement;

CREATE FUNCTION creare\_log\_insert\_librarian\_statement(id TEXT, name TEXT, surname TEXT, password\_hash TEXT, id\_admin TEXT)

RETURNS TEXT

DETERMINISTIC

BEGIN

SELECT CONCAT("INSERT id = ", id, ", name = ", name, ", surname = ", surname, ", password\_hash = ", password\_hash, ", id\_admin = ", id\_admin)

INTO @res;

RETURN @res;

END;//

Триггер логирования события обновления библиотекаря:

DROP TRIGGER IF EXISTS log\_librarian\_updates\_insert;

CREATE TRIGGER log\_librarian\_updates\_insert

AFTER INSERT

ON librarian

FOR EACH ROW

BEGIN

INSERT INTO logs(table\_name, prev\_value, new\_value, time) VALUES (

"librarian",

null,

creare\_log\_insert\_librarian\_statement(NEW.id, NEW.name, NEW.surname, NEW.password\_hash, NEW.id\_admin),

CURRENT\_TIMESTAMP()

);

END;//

Функция генерации сообщения в логи при изменении библиотекаря:

DROP FUNCTION IF EXISTS creare\_log\_update\_librarian\_statement;

CREATE FUNCTION creare\_log\_update\_librarian\_statement(id TEXT, name TEXT, surname TEXT, password\_hash TEXT, id\_admin TEXT)

RETURNS TEXT

DETERMINISTIC

BEGIN

SELECT CONCAT("UPDATE id = ", id, ", name = ", name, ", surname = ", surname, ", password\_hash = ", password\_hash, ", id\_admin = ", id\_admin)

INTO @res;

RETURN @res;

END;//

Триггер логирования события обновления библиотекаря:

DROP TRIGGER IF EXISTS log\_librarian\_updates\_update;

CREATE TRIGGER log\_librarian\_updates\_update

BEFORE UPDATE

ON librarian

FOR EACH ROW

BEGIN

INSERT INTO logs(table\_name, prev\_value, new\_value, time) VALUES (

"librarian",

creare\_log\_update\_librarian\_statement(OLD.id, OLD.name, OLD.surname, OLD.password\_hash, OLD.id\_admin),

creare\_log\_update\_librarian\_statement(NEW.id, NEW.name, NEW.surname, NEW.password\_hash, NEW.id\_admin),

CURRENT\_TIMESTAMP()

);

END;//

Функция генерации сообщения в логи при удалении библиотекаря:

DROP FUNCTION IF EXISTS creare\_log\_delete\_librarian\_statement;

CREATE FUNCTION creare\_log\_delete\_librarian\_statement(id TEXT, name TEXT, surname TEXT, password\_hash TEXT, id\_admin TEXT)

RETURNS TEXT

DETERMINISTIC

BEGIN

SELECT CONCAT("DELETE id = ", id, ", name = ", name, ", surname = ", surname, ", password\_hash = ", password\_hash, ", id\_admin = ", id\_admin)

INTO @res;

RETURN @res;

END;//

Триггер логирования события удаления библиотекаря:

DROP TRIGGER IF EXISTS log\_librarian\_updates\_delete;

CREATE TRIGGER log\_librarian\_updates\_delete

BEFORE DELETE

ON librarian

FOR EACH ROW

BEGIN

INSERT INTO logs(table\_name, prev\_value, new\_value, time) VALUES (

"librarian",

creare\_log\_delete\_librarian\_statement(OLD.id, OLD.name, OLD.surname, OLD.password\_hash, OLD.id\_admin),

null,

CURRENT\_TIMESTAMP()

);

END;//

## **3.3 Структура приложения**

Разрабатываемое программное средство будет иметь вид классического desktop-приложения. Такие приложения также называют «оконными». В них используется графический интерфейс и ввод информации осуществляется при помощи клавиатуры и мыши. Данные приложения позволяют создавать практически любые по сложности решения.

В программном средстве реализованы следующие классы:

* AddEditBook – класс формы для создания и редактирования книг;
* AddEditReader – класс формы для создания и редактирования читателей;
* AddEditAuthors – класс формы для создания и редактирования авторов;
* AddEditPublishingHouses – класс формы для создания и редактирования издательств;
* AddEditSimpleCommon – обобщенный универсальный класс для создания и редактирования;
* Book – класс, описывающий структуру книги: автора, название, язык, обложку, издательство;
* Reader - класс, описывающий структуру читателя: имя, фамилия, отчество, адрес, телефон;
* BookIssue – класс формы для выдачи книги;
* DBChooser – класс формы, в которой пользователь должен выбрать базу данных;
* ExpandIssueDate – класс формы продления даты выдачи книги;
* Program - стандартный класс программы, который является главной точкой входа в приложение;
* Form1 - главная форма приложения;
* Args - класс для передачи входных аргументов в новый поток в параллельном программировании;
* Res - класс для передачи выходных параметров в главный поток в параллельном программировании;
* DatabaseAdapter – класс, который реализует связь с базой данных и бизнес-логикой приложения.

Это приложение было разработано в стиле объектно-ориентированного программирования, с использование элементов чистой архитектуры, а значит все классы разработанные классы соответствуют общепринятому архитектурному стилю.

# **4 Тестирование**

# **4.1 Методы и типы тестирования**

Тестирование – процесс проверки программного обеспечения с целью выявления и предотвращения дефектов, оценки соответствия требованиям заказчика.

Баг (дефект) – ошибка в коде, которая может привести к тому, что программное средство будет работать не так, как ожидается.

Тестирование программного обеспечения – это креативная и интеллектуальная работа. Разработка правильных и эффективных тестов – достаточно непростое занятие. Принципы тестирования, представленные ниже, были разработаны в последние 40 лет и являются общим руководством для тестирования в целом:

* тестирование показывает наличие дефектов. Оно может показать наличие дефектов в программном средстве, но не доказать их отсутствие. Тем не менее, важно составлять тест-кейсы, которые будут находить как можно больше багов. Таким образом, при должном тестовом покрытии, тестирование позволяет снизить вероятность наличия дефектов в программном обеспечении. В то же время, даже если дефекты не были найдены в процессе тестирования, нельзя утверждать, что их нет;
* исчерпывающее тестирование невозможно. Невозможно провести исчерпывающее тестирование, которое бы покрывало все комбинации пользовательского ввода и состояний системы, за исключения совсем уж примитивных случаев. Вместо этого необходимо использовать анализ рисков и расстановку приоритетов, что позволит более эффективно распределять усилия по обеспечению качества ПО.
* раннее тестирование. Тестирование должно начинаться как можно раньше в жизненном цикле разработки программного обеспечения, и его усилия должны быть сконцентрированы на определенных целях.
* скопление дефектов. Разные модули системы могут содержать разное количество дефектов – то есть, плотность скопления дефектов в разных элементах программного средства может отличаться. Усилия по тестированию должны распределяться пропорционально фактической плотности дефектов. В основном, большую часть критических дефектов находят в ограниченном количестве модулей. Это проявление принципа Парето: 80% проблем содержатся в 20% модулей.
* тестирование зависит от контекста. Выбор методологии, техники и типа тестирования будет напрямую зависеть от природы самого программного средства. Например, программное обеспечение для медицинских нужд требует гораздо более строгой и тщательной проверки, чем, скажем, компьютерная игра. Из тех же соображений, сайт с большой посещаемостью должен пройти через серьезное тестирование производительности, чтобы показать возможность работы в условиях высокой нагрузки.

Исполнение тестов необходимо, но не менее важны и сопровождающие действия – планирование и документирование процесса. В обязанности тестировщика входит разработка тестовых сценариев, а также подготовка тестирования и оценка его результатов. Таким образом можно выделить следующие процессы тестирования:

* планирование и управление - определение целей тестирования и описание задач тестирования для достижения этих целей; сопоставление текущей ситуации в процессе тестирования с планом и составление отчетности;
* анализ и проектирование - это процесс написания тестовых сценариев и условий на основе общих целей тестирования;
* внедрение и реализация - написание тест-кейсов, на основе написанных ранее тестовых сценариев, собирается необходимая для проведения тестов информация, подготавливается тестовое окружение и запускаются тесты;
* оценка критериев выхода и написание отчетов – проверка, было ли проведено достаточное количество тестов, достигнута ли нужная степень обеспечения качества системы; убедится в том, что нет необходимости проводить дополнительные тесты. Если все же такая необходимость есть, возможно, потребуется изменить установленный критерий выхода;
* действия по завершению тестирования - сборка, систематизация и анализ информации о результатах тестирования.

Тестирование программного обеспечения имеет следующие уровни:

* компонентное – поиск дефектов и проверка функциональности (модулей, программных средств, объектов, классов), которые можно протестировать изолированно;
* интеграционное – проверяет взаимодействие между компонентами, взаимодействие различных частей системы (ОС, файловая система, аппаратное обеспечение);
* системное – тестирование поведения тестового объекта как целостной системы или программного средства;
* приемочное, в свою очередь, подразделяется на пользовательское, эксплуатационное, контрактное и правовое, альфа и бета. Заключается в проверке работоспособности системы, частей системы или отдельных нефункциональных характеристик системы.

Тестирование программного средства проходило в ходе и после ее разработки. При тестировании использовалось компонентное, системное, интеграционное и приёмочное тестирование. Меньше всего времени было затрачено на приёмочное тестирование, потому что ежедневно просматривались требования к данному программному обеспечению и продумывались варианты оптимальной реализации. Также использовались методы тестирования чёрным и белым ящиком, что весьма помогло в поиске дефектов.

# **4.2 Результат тестирования**

В процессе тестирования в основном использовалось функциональное и нефункциональное тестирование. Были разработаны тест-кейсы, представленные в таблице 4.1. Тест кейс – это набор входных значений, предусловий выполнения, шагов и ожидаемых результатов, определяемых для покрытия определенных тестовых условий.

Таблица 4.1 – Тест Кейс

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название процесса | | Ожидаемый результат | Фактический результат | |
| 1.Запуск приложения | | 1.Ожидание открытия приложения | 1.Приложение открылось | |
| 1.Переход на вкладку «Читатели» | | 1.Ожидание открытия вкладки | 1.Вкладка открылась | |
| 1.Переход на вкладку «Должники» | | 1.Ожидание открытия вкладки | 1.Вкладка открылась | |
| 1.Переход на вкладку «Книги» | | 1.Ожидание открытия вкладки | 1.Вкладка открылась | |
| 1.Переход на вкладку «Авторы» | | 1.Ожидание открытия вкладки | 1.Вкладка открылась | |
| 1.Переход на вкладку «Языки» | | 1.Ожидание открытия вкладки | 1.Вкладка открылась | |
| 1.Переход на вкладку «Жанры» | | 1.Ожидание открытия вкладки | 1.Вкладка открылась | |
| 1.Переход на вкладку «Издатели» | | 1.Ожидание открытия вкладки | 1.Вкладка открылась | |
| 1.Переход на вкладку «Города» | | 1.Ожидание открытия вкладки | 1.Вкладка открылась | |
| 1.Переход на вкладку «Обложки» | | 1.Ожидание открытия вкладки | 1.Вкладка открылась | |
| Продолжение таблицы 4.1 | | |  | |
| 1.Переход на вкладку «Читатели»  2.Нажать на кнопку «Добавить»  3.Ввести значения во все поля  4.Нажать на кнопку «Добавить» | | 1.Ожидание открытия вкладки  2.Ожидание появления окна добавления  3.Ожидание ввода значений  4.Ожидание закрытия окна и появления новой записи | 1.Вкладка открылась  2.Окно появилось  3.Значения введены  4.Окно закрылось и новая запись появилась | |
| 1.Переход на вкладку «Авторы»  2.Нажать на кнопку «Добавить»  3.Ввести значения во все поля  4.Нажать на кнопку «Добавить» | | 1.Ожидание открытия вкладки  2.Ожидание появления окна добавления  3.Ожидание ввода значений  4.Ожидание закрытия окна и появления новой записи | 1.Вкладка открылась  2.Окно появилось  3.Значения введены  4.Окно закрылось и новая запись появилась | |
| 1.Переход на вкладку «Книги»  2.Нажать на кнопку «Добавить»  3.Ввести значения во все поля  4.Нажать на кнопку «Добавить» | | 1.Ожидание открытия вкладки  2.Ожидание появления окна добавления  3.Ожидание ввода значений  4.Ожидание закрытия окна и появления новой записи | 1.Вкладка открылась  2.Окно появилось  3.Значения введены  4.Окно закрылось и новая запись появилась | |
| 1.Переход на вкладку «Языки»  2.Нажать на кнопку «Добавить»  3.Ввести значения во все поля  4.Нажать на кнопку «Добавить» | | 1.Ожидание открытия вкладки  2.Ожидание появления окна добавления  3.Ожидание ввода значений  4.Ожидание закрытия окна | 1.Вкладка открылась  2.Окно появилось  3.Значения введены  4.Окно закрылось и новая запись появилась | |
| 1.Переход на вкладку «Жанры»  2.Нажать на кнопку «Добавить»  3.Ввести значения во все поля  4.Нажать на кнопку «Добавить» | | 1.Ожидание открытия вкладки  2.Ожидание появления окна добавления  3.Ожидание ввода значений  4.Ожидание закрытия окна | 1.Вкладка открылась  2.Окно появилось  3.Значения введены  4.Окно закрылось и новая запись появилась | |
| Продолжение таблицы 4.1 | | |  | |
| 1.Переход на вкладку «Издатели»  2.Нажать на кнопку «Добавить»  3.Ввести значения во все поля  4.Нажать на кнопку «Добавить» | | 1.Ожидание открытия вкладки  2.Ожидание появления окна добавления  3.Ожидание ввода значений  4.Ожидание закрытия окна и появления новой записи | 1.Вкладка открылась  2.Окно появилось  3.Значения введены  4.Окно закрылось и новая запись появилась | |
| 1.Переход на вкладку «Города»  2.Нажать на кнопку «Добавить»  3.Ввести значения во все поля  4.Нажать на кнопку «Добавить» | | 1.Ожидание открытия вкладки  2.Ожидание появления окна добавления  3.Ожидание ввода значений  4.Ожидание закрытия окна и появления новой записи | 1.Вкладка открылась  2.Окно появилось  3.Значения введены  4.Окно закрылось и новая запись появилась | |
| 1.Переход на вкладку «Обложки»  2.Нажать на кнопку «Добавить»  3.Ввести значения во все поля  4.Нажать на кнопку «Добавить» | | 1.Ожидание открытия вкладки  2.Ожидание появления окна добавления  3.Ожидание ввода значений  4.Ожидание закрытия окна и появления новой записи | 1.Вкладка открылась  2.Окно появилось  3.Значения введены  4.Окно закрылось и новая запись появилась | |
| 1.Переход на вкладку «Должники»  2.Выбрать любого должника  3.Нажать на кнопку «Вернуть книгу» | | 1.Ожидание открытия вкладки  2.Ожидание выбора должника  3.Ожидание исчезновения книги из списка | 1.Вкладка открылась  2.Должник выбран  3.Книга исчезла | |
| 1.Переход на вкладку «Обложки»  2.Выбрать неиспользуемую запись  3.Нажать на кнопку «Удалить» | | 1.Ожидание открытия вкладки  2.Ожидание выбора записи  3.Ожидание исчезновения записи из списка | 1.Вкладка открылась  2.Запись выбралась  3.Запись исчезла | |
| Продолжение таблицы 4.1 | | | | |
| 1.Переход на вкладку «Города»  2.Выбрать неиспользуемый город  3.Нажать на кнопку «Удалить» | | 1.Ожидание открытия вкладки  2.Ожидание выбора записи  3.Ожидание исчезновения записи из списка | 1.Вкладка открылась  2.Запись выбралась  3.Запись исчезла | |
| 1.Переход на вкладку «Книги»  2.Выбрать неиспользуемую книгу  3.Нажать на кнопку «Удалить» | | 1.Ожидание открытия вкладки  2.Ожидание выбора записи  3.Ожидание исчезновения записи из списка | 1.Вкладка открылась  2.Запись выбралась  3.Запись исчезла | |
| 1.Переход на вкладку «Издатели»  2.Выбрать неиспользуемое издательство  3.Нажать на кнопку «Удалить» | | 1.Ожидание открытия вкладки  2.Ожидание выбора записи  3.Ожидание исчезновения записи из списка | 1.Вкладка открылась  2.Запись выбралась  3.Запись исчезла | |
| 1.Переход на вкладку «Авторы»  2.Выбрать неиспользуемого автора  3.Нажать на кнопку «Удалить» | | 1.Ожидание открытия вкладки  2.Ожидание выбора записи  3.Ожидание исчезновения записи из списка | 1.Вкладка открылась  2.Запись выбралась  3.Запись исчезла | |
| 1.Переход на любую вкладку, на которой есть используемая запись  2.Выбрать используемую запись  3.Нажать на кнопку «Удалить» | | 1.Ожидание открытия вкладки  2.Ожидание выбора записи  3.Ожидание появления окна с ошибкой | 1.Вкладка открылась  2.Запись выбралась  3.Окно с ошибкой появилось | |
| 1.Переход на вкладку «Языки»  2.Выбрать неиспользуемый язык  3.Нажать на кнопку «Удалить» | | 1.Ожидание открытия вкладки  2.Ожидание выбора записи  3.Ожидание исчезновения записи из списка | 1.Вкладка открылась  2.Запись выбралась  3.Запись исчезла | |
| Продолжение таблицы 4.1 | | | | |
| 1.Переход на вкладку «Жанры»  2.Выбрать неиспользуемый жанр  3.Нажать на кнопку «Удалить» | | 1.Ожидание открытия вкладки  2.Ожидание выбора записи  3.Ожидание исчезновения записи из списка | 1.Вкладка открылась  2.Запись выбралась  3.Запись исчезла | |
| 1.Переход на вкладку «Жанры»  2.Выбрать жанр  3.Нажать на кнопку «Изменить»  4.Изменить данные  5.Нажать на кнопку «Изменить» | | 1.Ожидание открытия вкладки  2.Ожидание выбора записи  3.Ожидание появления окна редактирования  4.Ожидание ввода значений  5.Ожидание изменений данных в записи | 1.Вкладка открылась  2.Запись выбралась  3.Окно появилось  4.Данные введены  5.Данные в записи изменились | |
| 1.Переход на вкладку «Книги»  2.Выбрать книгу  3.Нажать на кнопку «Изменить»  4.Изменить данные  5.Нажать на кнопку «Изменить» | | 1.Ожидание открытия вкладки  2.Ожидание выбора записи  3.Ожидание появления окна редактирования  4.Ожидание ввода значений  5.Ожидание изменений данных в записи | 1.Вкладка открылась  2.Запись выбралась  3.Окно появилось  4.Данные введены  5.Данные в записи изменились | |
| 1.Переход на вкладку «Авторы»  2.Выбрать автора  3.Нажать на кнопку «Изменить»  4.Изменить данные  5.Нажать на кнопку «Изменить» | | 1.Ожидание открытия вкладки  2.Ожидание выбора записи  3.Ожидание появления окна редактирования  4.Ожидание ввода значений  5.Ожидание изменений данных в записи | 1.Вкладка открылась  2.Запись выбралась  3.Окно появилось  4.Данные введены  5.Данные в записи изменились | |
| Продолжение таблицы 4.1 | | | | |
| 1.Переход на вкладку «Языки»  2.Выбрать язык  3.Нажать на кнопку «Изменить»  4.Изменить данные  5.Нажать на кнопку «Изменить» | 1.Ожидание открытия вкладки  2.Ожидание выбора записи  3.Ожидание появления окна редактирования  4.Ожидание ввода значений  5.Ожидание изменений данных в записи | | | 1.Вкладка открылась  2.Запись выбралась  3.Окно появилось  4.Данные введены  5.Данные в записи изменились |
| 1.Переход на вкладку «Издатели»  2.Выбрать издателя  3.Нажать на кнопку «Изменить»  4.Изменить данные  5.Нажать на кнопку «Изменить» | 1.Ожидание открытия вкладки  2.Ожидание выбора записи  3.Ожидание появления окна редактирования  4.Ожидание ввода значений  5.Ожидание изменений данных | | | 1.Вкладка открылась  2.Запись выбралась  3.Окно появилось  4.Данные введены  5.Данные в записи изменились |
| 1.Переход на вкладку «Города»  2.Выбрать жанр  3.Нажать на кнопку «Изменить»  4.Изменить данные  5.Нажать на кнопку «Изменить» | 1.Ожидание открытия вкладки  2.Ожидание выбора записи  3.Ожидание появления окна редактирования  4.Ожидание ввода значений  5.Ожидание изменений данных в записи | | | 1.Вкладка открылась  2.Запись выбралась  3.Окно появилось  4.Данные введены  5.Данные в записи изменились |
| 1.Переход на вкладку «Обложки»  2.Выбрать обложку  3.Нажать на кнопку «Изменить»  4.Изменить данные  5.Нажать на кнопку «Изменить» | 1.Ожидание открытия вкладки  2.Ожидание выбора записи  3.Ожидание появления окна редактирования  4.Ожидание ввода значений  5.Ожидание изменений данных в записи | | | 1.Вкладка открылась  2.Запись выбралась  3.Окно появилось  4.Данные введены  5.Данные в записи изменились |

Тестирование программного средства проходило в ходе и после его разработки. При тестировании использовалось компонентное, системное, интеграционное и приёмочное тестирование. Меньше всего времени было затрачено на приёмочное тестирование, потому что ежедневно просматривались требования к данному программному обеспечению и продумывались варианты оптимальной реализации.

# **5 Руководство пользователя**

После запуска приложения появляется окно, приведенное на риc. 5.1.

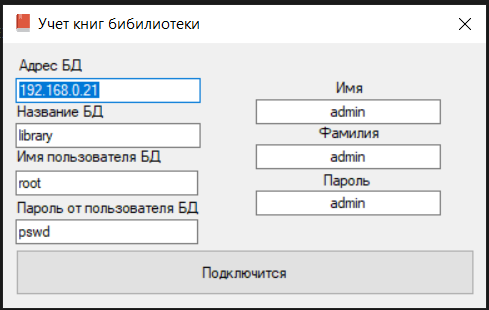
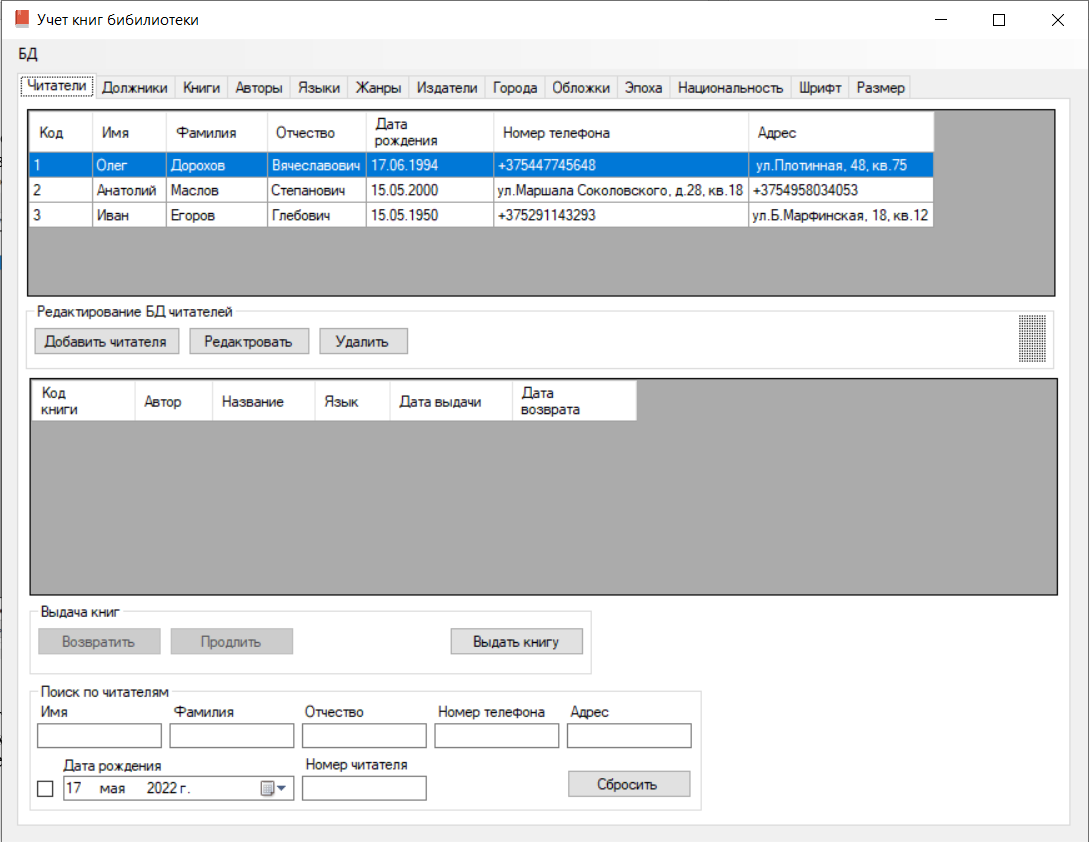


Рис. 5.1 Окно приложения для подключения к БД

При правильном вводе данных и нажатии на копку, откроется главное окно приложения, представленное на рис. 5.2. В случае, если данные введены неверно, появится окно с ошибкой, указывающей на то, в каком месте она могла произойти.



Риc. 5.2 Главное окно приложения

В главном окне мы можем наблюдать 13 вкладок. На каждой вкладке находится свой набор функций. Однако поскольку на вкладках «Языки», «Жанры», «Города», «Обложки», «Эпоха», «Национальность», «Шрифт», «Размер» все абсолютно одинаково за исключением надписей, пример работы этих вкладок можно привести на примере вкладки «Обложки», приведенная на рис. 5.3.

Graphical user interface, application, Word

Description automatically generated

Рис. 5.3 Вкладка «Обложки»

Когда никаких данных нет, единственное, что мы можем сделать – нажать на кнопку «Добавить» для добавления обложки. Появится окно, приведенное на риc. 5.4.

Graphical user interface, application

Description automatically generated

Рис. 5.4 Добавление обложки

После ввода данных достаточно нажать на кнопку «Добавить», и новая обложка окажется в списке обложек, как показано на риc. 5.5.

Graphical user interface, application, Word

Description automatically generated

Рис. 5.5 Вкладка обложек с данными

После добавления данных, кнопки «Редактировать» и «Удалить» становятся активными. При нажатии на кнопку «Удалить» выделенная обложка удалится. При нажатии на кнопку «Редактировать», появится окно редактирования с текущим значением поля, как показано на рис. 5.6.

Graphical user interface, application

Description automatically generated

Рис. 5.6 Редактирование обложек.

Для создания книги помимо обложки, города, жанра, и языка так же нужен автор и издатель. Перейдем к созданию издателя. Для этого перейдем на вкладку издателя, и нажмем на кнопку «Добавить». Появится экран, представленный на рис. 5.7.

Graphical user interface, application

Description automatically generated

Рис. 5.7 Добавление издательства.

Для добавления издательства необходимо ввести его имя, а также выбрать город из выпадающего списка. После добавления издательства оно появится в списке издательств. Как и с обложками, издательства можно удалять и редактировать. Для этого необходимо нажать на кнопки «Удалить» и «Редактировать» соответственно. Экран добавления автора представлен на рис. 5.8.

Graphical user interface

Description automatically generated

Рис. 5.8 Экран добавления автора

Для добавления автора необходимо заполнить его имя, фамилию, отчество и нажать на кнопку «Добавить». Если хоть одно поле будет пустым, будет показано сообщение об ошибке с предложением заполнить все поля.

Для добавления книги необходимо перейти на вкладку «Книги», нажать на кнопку «Добавить». Для любого параметра книги, кроме названия, буден предложен выпадающий список с соответствующими данными из ранее заполненных таблиц. Пример добавления книги представлен на рис. 5.9.

Graphical user interface, application, Word

Description automatically generated

Рис. 5.9 Добавление книги

Книги надо кому-то выдавать, поэтому необходимо добавить читателя. Делается это на странице «Читатели» по нажатию на кнопку «Добавить читателя». Появится окно, представленное на риc. 5.10.

Graphical user interface, application

Description automatically generated

Рис. 5.10 Добавление читателя

После ввода всех данных и нажатия на кнопку «Добавить», читатель появится в общем списке.

Чтобы выдать читателю книгу, необходимо сначала выбрать читателя, которому необходимо выдать книгу, а затем нажать на кнопку «Выдать книгу». Появится окно, показанное на риc. 5.11.

Graphical user interface, text

Description automatically generated

Рис. 5.11 Экран выдачи книг

Внизу есть поля для поиска по книгам. Такие поля для поиска так же есть на экране с читателями и с книгами. Поиск работает так: если текст является подстрокой искомого текста, то эта книга будет показана. При выдаче книги можно указать дату выдачи и дату возврата.

Чтобы посмотреть, какие книги выданы читателю, необходимо нажать на читателя и во втором сером поле появятся выданные ему книги. Их можно продлить или вернуть. При их продлении появится экран выбора нового времени возврата, как показано на рис. 5.12.

Graphical user interface, application

Description automatically generated

Рис. 5.12 Экран продления книги

На вкладке «Должники» находится информация о читателях и книгах, которые они на сегодняшний день должны были вернуть, но не сделали этого, как показано на рис. 5.13.

Graphical user interface, text, application

Description automatically generated

Рис. 5.13 Экран с должниками

На этом экране виден номер телефона должника, поэтому ему можно позвонить и, если он ответит, продлить книгу.

При работе из режима администратора появляются 2 дополнительных пункта меню: «Пользователи» и «Запросы». Они представлены на рис. 5.14:

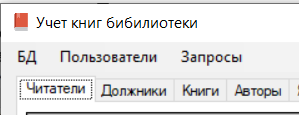


Рис. 5.14 Дополнительные пункты меню

Из вкладки «Пользователи» можно получить доступ к окну редактирования пользователей. Оно представлено на рис. 5.15:

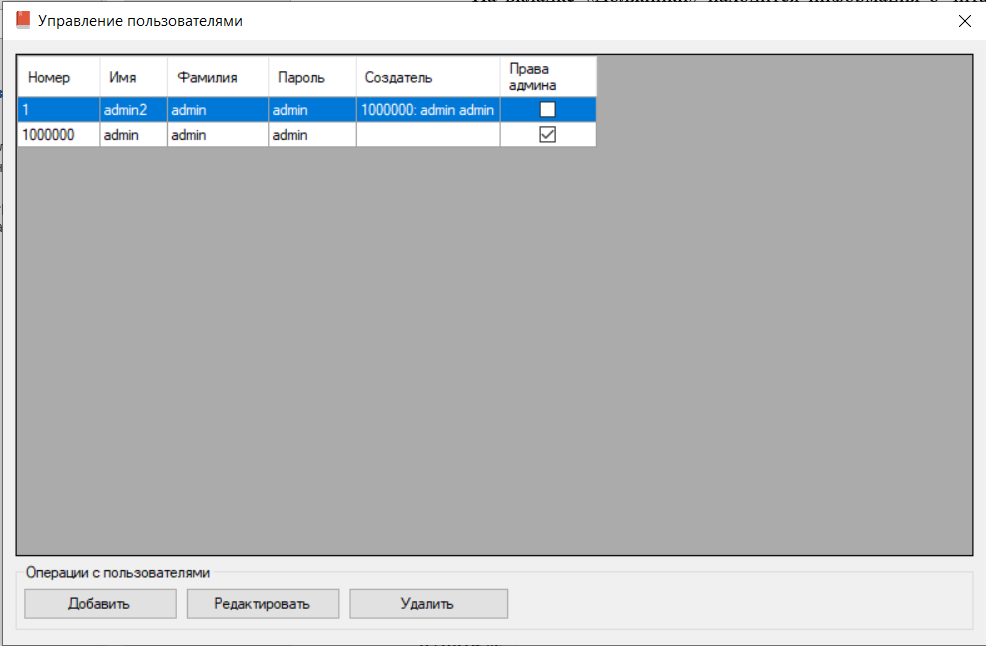


Рис 5.15 Окно редактирования пользователей

Из этого окна можно добавлять, редактировать и удалять пользователей.

Из вкладки «Запросы» администратор получает возможность исполнять любые запросы. Окно выполнения запросов представлено на рис. 5.16:

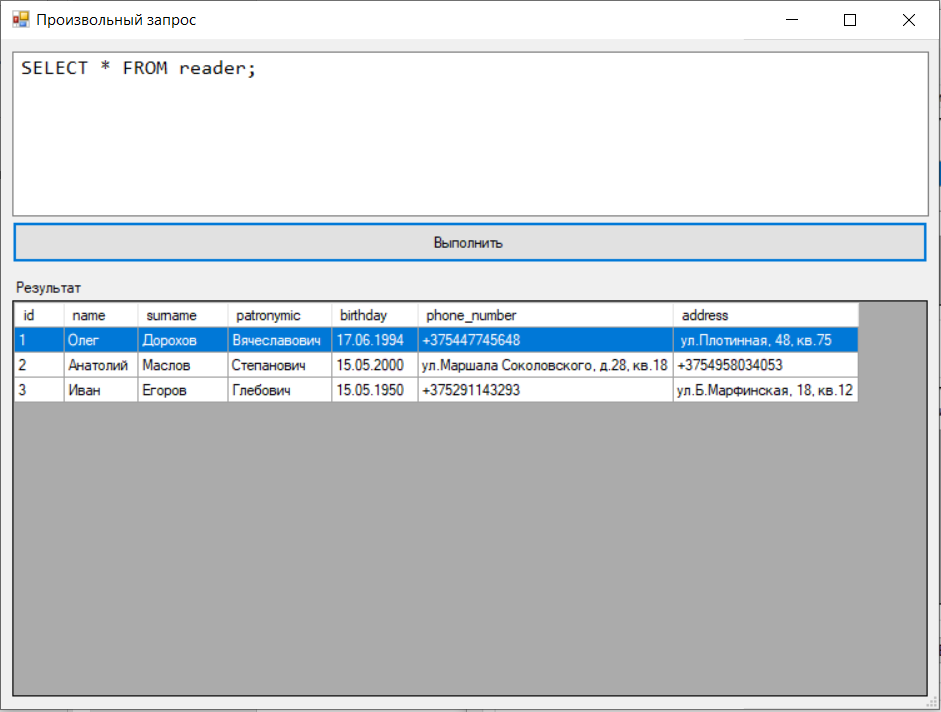


Рис. 5.16 Окно выполнения запросов

В данном окне можно выполнять все виды запросов: SELECT, DELETE, UPDATE.

Таким образом с описанным выше функционалом можно вести свою библиотеку без каких – либо ограничений.

# **Заключение**

В ходе выполнения курсового проекта было разработано десктоп-приложение «Автоматическое рабочее место библиотекаря» на языке C# с использованием технологии Windows Forms и системой управления базами данных MySQL.

Настольное приложение предназначено для автоматизации учета и контроля данных о книгах и читателях в библиотеке.

Разработанное интернет-приложение имеет следующие достоинства:

* отсутствие перегруженности интерфейса;
* отсутствие избыточного функционала;
* дружественный интерфейс;
* простота в использовании;
* высокое быстродействие программы;
* возможность работы с уже установленной и настроенной базой данных MySQL.

В ходе выполнения курсового проекта задача курсового проекта была полностью выполнена. Также были полностью выполнены все цели курсового проекта, такие как закрепление и углубление теоретических знаний и практических умений, полученных при изучении дисциплины БД и СУБД, привитие навыков самостоятельной разработки БД на SQL-сервере, а также закрепление навыков разработки удобного интерфейса пользователя, системы меню, навигации, системы помощи.

# **Список использованных источников**

[1] Бошемин, Б. Основы .NET / Б.Бошемин. – Вильямс,2016.

[2] Шилдт, Г. C# 4.0. Полное руководство / Г. Шилдт . – М.: Вильямс, 2019.

[3] Шнайер, Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке С# / Б.Шнайер. – М.: Триумф, 2003.

[4] Страуструп, Б. Полное руководство C# / Б.Страуструп. – М.: Вильямс, 2017.

[5] Троелсен, Э. Язык программирования C# 5.0 и платформа   
.NET 4.5 / Э. Троелсен. – М.: Вильямс, 2015.

[6] Павловская, Т. А. С#. Объектно-ориентированное программирование: практикум / Павловская, Т. А., Щупак. – СПб. : Питер, 2004.

[7] Керниган, Б. Язык программирования С# / Б. Керниган, Д. Ритчи. – М.: СПб.: Питер, 2002.

[8] Поль, Д. MySQL. Сборник рецептов. / Д. Поль – М.: Эксмо, 2010.

[9] Шварц, Б. MySQL по максимуму. / Б. Шварц, Т. Вадим, З. Петр – Питер, 2015.

[10] Ульман, Л. MySQL: руководство по изучению языка / Л. Ульман – М.: Эксмо, 2010.

# Приложение А

(обязательное)

Текст программы

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Data.OleDb;

using static АРМ\_билиотекаря.DatabaseAdapter;

using Excel = Microsoft.Office.Interop.Excel;

namespace АРМ\_билиотекаря

{

public partial class Form1 : Form

{

private DataTable empty = null;

private DBChooser chooser;

public int librarianId = -1;

private bool isAdmin = false;

private readonly object syncLock = new object();

private bool isResetButton = false;

public DatabaseAdapter adapter;

private bool mouseDown = false;

private Point startPos;

public void updateConnection(String address, String databaseName, String username, String password)

{

adapter.setBDPath(address, databaseName, username, password);

adapter.createTables();

updateDebtors();

}

public void updateUser(int librarianId, bool isAdmin)

{

this.librarianId = librarianId;

this.isAdmin = isAdmin;

пользователиToolStripMenuItem.Visible = isAdmin;

запросыToolStripMenuItem.Visible = isAdmin;

}

public Form1(DBChooser chooser)

{

InitializeComponent();

this.chooser = chooser;

adapter = DatabaseAdapter.getInstance();

}

private void panel1\_MouseDown(object sender, MouseEventArgs e)

{

mouseDown = true;

startPos = e.Location;

}

private void panel1\_MouseUp(object sender, MouseEventArgs e)

{

mouseDown = false;

}

private void panel1\_MouseMove(object sender, MouseEventArgs e)

{

if (mouseDown)

{

if (dataGridView2.Height + e.Y - startPos.Y < 20)

return;

if (dataGridView1.Height - e.Y + startPos.Y < 20)

return;

groupBox3.Top += e.Y - startPos.Y;

dataGridView2.Height += e.Y - startPos.Y;

dataGridView1.Height -= e.Y - startPos.Y;

dataGridView1.Top += e.Y - startPos.Y;

dataGridView2.Refresh();

groupBox3.Refresh();

dataGridView1.Refresh();

}

}

private void Form1\_Load(object sender, EventArgs e)

{

updateReadersGrid();

updateBooksGrid();

}

private void textBox1\_TextChanged(object sender, EventArgs e)

{

if (!isResetButton)

updateReadersGrid();

}

public void updateDebtors()

{

var worker = new BackgroundWorker();

worker.DoWork += BackgroundWorker1\_DoWork;

worker.RunWorkerCompleted += BackgroundWorker1\_RunWorkerCompleted;

worker.RunWorkerAsync(new Args(6));

}

public void updateReadersGrid()

{

var worker = new BackgroundWorker();

worker.DoWork += BackgroundWorker1\_DoWork;

worker.RunWorkerCompleted += BackgroundWorker1\_RunWorkerCompleted;

worker.RunWorkerAsync(new Args(2));

}

private void CheckBox1\_CheckedChanged(object sender, EventArgs e)

{

if(!isResetButton)

updateReadersGrid();

}

private void DateTimePicker1\_ValueChanged(object sender, EventArgs e)

{

if (!isResetButton)

updateReadersGrid();

}

private void Button1\_Click(object sender, EventArgs e)

{

isResetButton = true;

textBox1.Text = "";

textBox2.Text = "";

textBox3.Text = "";

textBox5.Text = "";

textBox6.Text = "";

textBox10.Text = "";

checkBox1.Checked = false;

isResetButton = false;

updateReadersGrid();

}

private void DataGridView2\_CellEnter(object sender, DataGridViewCellEventArgs e)

{

int reader\_id = Convert.ToInt32(dataGridView2[0, e.RowIndex].Value);

updateReaderBooks(reader\_id);

}

private void updateBooksGrid()

{

var worker = new BackgroundWorker();

worker.DoWork += BackgroundWorker1\_DoWork;

worker.RunWorkerCompleted += BackgroundWorker1\_RunWorkerCompleted;

worker.RunWorkerAsync(new Args(1));

//dataGridView3.DataSource = adapter.getFilteredBooks(textBox4.Text, textBox7.Text, textBox8.Text, textBox9.Text, textBox11.Text);

}

private void Button11\_Click(object sender, EventArgs e)

{

isResetButton = true;

textBox4.Text = "";

textBox7.Text = "";

textBox8.Text = "";

textBox11.Text = "";

isResetButton = false;

updateBooksGrid();

}

private void TextBox4\_TextChanged(object sender, EventArgs e)

{

if (!isResetButton)

updateBooksGrid();

}

public void updateReaderBooks(int id)

{

var worker = new BackgroundWorker();

worker.DoWork += BackgroundWorker1\_DoWork;

worker.RunWorkerCompleted += BackgroundWorker1\_RunWorkerCompleted;

worker.RunWorkerAsync(new Args(4, id));

}

public UserResolution getUserId(string name, string surname, string password)

{

return adapter.resolveUser(name, surname, password);

}

public class Args

{

public int author2;

public int language;

public int genre;

public int ph;

public int cover;

public string title;

public Book book;

public String message;

public int id;

public int user\_id;

public int debitId;

public Reader reader = null;

public DataGridView dataGridView = null;

public string tableName = null;

public string columnName = null;

public Button deleteButton;

public Button editButton;

public string data;

public int dataId;

public string errorMessage = "";

public int era;

public int type;

public int font;

public int size;

public bool isAdmin;

public string name;

public string surname;

public string password;

public int userAuthorId;

public bool isPasswordUpdate;

public Args(int id, int user\_id) : this(id)

{

this.user\_id = user\_id;

}

public Args(int id, Reader reader) : this(id)

{

this.reader = reader;

}

public Args(int id, DataGridView table, string tableName, Button deleteButton, Button editButton, string data)

{

this.id = id;

this.dataGridView = table;

this.tableName = tableName;

this.deleteButton = deleteButton;

this.editButton = editButton;

this.data = data;

}

public Args(int id, DataGridView table, string tableName, Button deleteButton, Button editButton)

{

this.id = id;

this.dataGridView = table;

this.tableName = tableName;

this.deleteButton = deleteButton;

this.editButton = editButton;

}

public Args(int id)

{

this.id = id;

}

}

class Res

{

public int arg;

public DataTable table;

public bool readerError = false;

public bool bookError = false;

public int id;

public DataGridView dataGridView;

public Button deleteButton;

public Button editButton;

public bool error = false;

public string message = "";

public Res(int argg, DataTable tablee)

{

arg = argg;

table = tablee;

}

}

private void disableEnableBookButtons()

{

if (dataGridView1.RowCount != 0)

{

button2.Enabled = true;

button12.Enabled = true;

}

else

{

button2.Enabled = false;

button12.Enabled = false;

}

}

private void deleteAuthor(int id)

{

var worker = new BackgroundWorker();

worker.DoWork += BackgroundWorker1\_DoWork;

worker.RunWorkerCompleted += BackgroundWorker1\_RunWorkerCompleted;

worker.RunWorkerAsync(new Args(18, dataGridView5, "author", button14, button15)

{

user\_id = id,

errorMessage = "Чтобы удалить автора, необходимо избавиться от его использования во всех книгах."

});

}

private void deleteLanguage(int id)

{

var worker = new BackgroundWorker();

worker.DoWork += BackgroundWorker1\_DoWork;

worker.RunWorkerCompleted += BackgroundWorker1\_RunWorkerCompleted;

worker.RunWorkerAsync(new Args(18, dataGridView6, "language", button17, button18)

{

user\_id = id,

errorMessage = "Чтобы удалить язык, необходимо избавиться от его использования во всех книгах."

});

}

private void deleteGenre(int id)

{

var worker = new BackgroundWorker();

worker.DoWork += BackgroundWorker1\_DoWork;

worker.RunWorkerCompleted += BackgroundWorker1\_RunWorkerCompleted;

worker.RunWorkerAsync(new Args(18, dataGridView7, "genre", button20, button21)

{

user\_id = id,

errorMessage = "Чтобы удалить жанр, необходимо избавиться от его использования во всех книгах."

});

}

private void deletePublishingHouse(int id)

{

var worker = new BackgroundWorker();

worker.DoWork += BackgroundWorker1\_DoWork;

worker.RunWorkerCompleted += BackgroundWorker1\_RunWorkerCompleted;

worker.RunWorkerAsync(new Args(22, dataGridView8, "publishing\_house", button23, button24)

{

user\_id = id,

errorMessage = "Чтобы удалить издательство, необходимо избавиться от его использования во всех книгах."

});

}

private void deleteCity(int id)

{

var worker = new BackgroundWorker();

worker.DoWork += BackgroundWorker1\_DoWork;

worker.RunWorkerCompleted += BackgroundWorker1\_RunWorkerCompleted;

worker.RunWorkerAsync(new Args(18, dataGridView9, "city", button26, button27)

{

user\_id = id,

errorMessage = "Чтобы удалить город, необходимо избавиться от его использования во всех издательствах."

});

}

private void deleteCover(int id)

{

var worker = new BackgroundWorker();

worker.DoWork += BackgroundWorker1\_DoWork;

worker.RunWorkerCompleted += BackgroundWorker1\_RunWorkerCompleted;

worker.RunWorkerAsync(new Args(18, dataGridView10, "cover", button29, button30)

{

user\_id = id,

errorMessage = "Чтобы удалить обложку, необходимо избавиться от ее использования во всех книгах."

});

}

private void deleteEra(int id)

{

var worker = new BackgroundWorker();

worker.DoWork += BackgroundWorker1\_DoWork;

worker.RunWorkerCompleted += BackgroundWorker1\_RunWorkerCompleted;

worker.RunWorkerAsync(new Args(18, dataGridView11, "era", button33, button32)

{

user\_id = id,

errorMessage = "Чтобы удалить эру, необходимо избавиться от ее использования во всех книгах."

});

}

# Приложение Б

(обязательное)

Диаграмма «сущность-связь»

Diagram

Description automatically generated

# Приложение В

(обязательное)

Диаграмма вариантов использования

Diagram

Description automatically generated

# Приложение Г

(обязательное)

Физическая модель данных

Graphical user interface, application, Word

Description automatically generated