МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования

«Гомельский государственный университет

имени Франциска Скорины»

Факультет математики и технологий программирования

Кафедра математических проблем управления и информатики

**Разработка приложения для автоматизации работы библиотеки**

Курсовая работа

Исполнитель

студент группы ПИ-41 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Н.М. Обухов

Научный руководитель

старший преподаватель кафедры МПУИ \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Л.И. Короткевич

Гомель 2024

РЕФЕРАТ

Курсовая работа 42 страниц, 15 таблиц, 7 рисунков, 4 приложения, 6 источников

Ключевые слова:база данных, язык программирования, приложение, визуализация данных, данные, средства разработки, запрос

Цель работы:Изучение языков программирования C# и SQL, их взаимодействие, и составление приложения для выполнения, а так же получение различных навыков при использовании этих языков программирования.

Методология проведения работы: изучение среды Visual Studio 2022 и сервера баз данных MS SQL Server, создание базы данных на сервере, разработка и отладка клиент-серверного приложения баз данных.

Полученные результаты работы: база данных на сервере MS SQL Server,

клиент-серверное приложение для автоматизации работы библиотеки.

Область применения результатов: библиотека.

Практическая значимость работы: разработанное приложение упрощает

работу учета книг и выдачи библиотеки.

Выводы:Созданные функциональности приложения позволяют упрощать работу при взаимодействии пользователя и кода. Основная задача курсовой работы достигнута.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 3

1. Назначение разработанного приложения 4

2 Описание средств разработки приложения 5

2.1 Среда разработки Visual Studio 2022 5

2.2 Система Управления Базами Данных Microsoft SQL Server 5

2.3 Среда управления SQL Server Management Studio 6

2.4 Язык программирования SQL 7

2.5 Язык программирования C# 7

3 Структура базы данных разработанного приложения 10

3.1 Состав таблиц базы данных 10

3.2 Структура таблиц базы данных 11

4 Взаимодействие пользователя с разработанным приложением 15

ЗАКЛЮЧЕНИЕ 19

ПРИЛОЖЕНИЕ А 21

Скрипт для создания структуры таблиц базы данных 21

ПРИЛОЖЕНИЕ Б 26

Код C#, выполняющий подключение к СУБД, и выполняющий ряд заданных действий для взаимодействия с СУБД 26

ПРИЛОЖЕНИЕ В 38

Код C#, выполняющий логику регистрации и авторизации 38

ПРИЛОЖЕНИЕ Г 41

Результат проверки на плагиат 41

# ВВЕДЕНИЕ

В библиотеках хранится великое число различного рода литературы, которую необходимо каким-либо оптимальным образом хранить, и знать, где её искать. Также библиотеку посещают читатели – они могут брать оттуда книги, при этом идет учет того, что они взяли.

Целью проекта является упрощение работы библиотеки, а именно: выдача книг клиентам(читателям), их хранение, каталогизация, получение и списание; учет клиентов, сотрудников, помещений; контроль корректности данных.  
Данные возможности актуальны в нынешнее время, учитывая тот факт, что в библиотеке, архиве и читальных залах хранится множество единиц различного рода литературы, информацию о которых необходимо собрать воедино и упорядочить для последующего решения проблем по поиску, выдаче и учету книг, а так же книг, которые выдали читателю.

Задачей курсовой работы является создание приложения на базе языка C# в интегрированной среде разработки Visual Studio, где работа идет с созданием GUI формы с подключением базы данных на языке SQL в MS SQL Server для хранения информации и компонентами ADO.Net для связи программы с базой данных (см. приложение А).

Для достижения вышеописанных целей необходимо решить следующие задачи:

* проанализировать требования;
* реализовать серверное подключение с базой данных;
* разработать клиентскую часть, которая будет интерфейсом пользователя, или сотрудника;
* сделать приложение простым и доступным в использовании, а также настроить ограничения на доступ к данным;
* протестировать разработанное приложение.

Разработка визуальной части приложения будет вестись на MS Visual Studio 2022. Для хранения и получения данных будет подключена СУБД на MS SQL Server

# Назначение разработанного приложения

Приложение представляет собой возможность быстрого и удобного взаимодействия с хранимыми данными. Создано для участников библиотеки (читатели, сотрудники) с целью ведения учета хранимых книг, их поставок и списаний, для быстрого доступа к необходимым книгам, а также хранения информации о выдаче книг читателям с целью несения ответственности читателем за полученную им литературу, принадлежащую учреждению (библиотеке), выдавшего книгу.

Подробная информация о книгах, выдачах и читателях станет доступна пользователю, как только он выберет соответствующий раздел, а так же от его прав доступа к данным

Подробная информация, которая будет упомянута выше, будет включать:

* личные данные читателя;
* личные данные сотрудника(-ков);
* полная информация о книгах: их название, жанр, издательство, инвентарный номер и т.п.;
* выдача книг читателям.

Примечание: информация будет доступна только пользователям, имеющим необходимый уровень доступа.

# Описание средств разработки приложения

## **Среда разработки Visual Studio 2022**

Microsoft Visual Studio  - линейка продуктов компании Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментов. Данные продукты позволяют разрабатывать как консольные приложения, так и игры и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms, UWP а также веб-сайты, веб-приложения, веб-службы как в родном, так и в управляемом кодах для всех платформ, поддерживаемых Windows, Windows Mobile, Windows CE, .NET Framework, .NET Core, .NET, MAUI, Xbox, Windows Phone .NET Compact Framework и Silverlight.

Visual Studio включает в себя редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего рефакторинга кода. Встроенный отладчик может работать как отладчик уровня исходного кода, так и отладчик машинного уровня. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения, веб-редактор, дизайнер классов и дизайнер схемы базы данных. Visual Studio позволяет создавать и подключать сторонние дополнения (плагины) для расширения функциональности практически на каждом уровне, включая добавление поддержки систем контроля версий исходного кода (как, например, Subversion и Visual SourceSafe), добавление новых наборов инструментов (например, для редактирования и визуального проектирования кода на предметно-ориентированных языках программирования) или инструментов для прочих аспектов процесса разработки программного обеспечения (например, клиент Team Explorer для работы с Team Foundation Server).

## **Система Управления Базами Данных Microsoft SQL Server**

Microsoft SQL Server - система управления реляционными базами данных (РСУБД), разработанная корпорацией Microsoft. Основной используемый язык запросов - Transact-SQL, создан совместно Microsoft и Sybase. Transact-SQL является реализацией стандарта ANSI/ISO по структурированному языку запросов (SQL) с расширениями. Используется для работы с базами данных размером от персональных до крупных баз данных масштаба предприятия.

Преимущества СУБД:

* создание клиент-серверных приложений (Многопользовательская работа (совместное обновление данных), уменьшение сетевого трафика);
* создание программ, не чувствительных к изменению структуры данных;
* Наличие в СУБД стандартных, качественно реализованных, алгоритмов обработки данных (сортировка, поиск и др.);
* Автоматическое поддержание целостности данных (автоматическое резервное копирование, контроль ссылочной целостности, поддержка механизма транзакций с возможным откатом транзакций);
* разграничение доступа к данным;
* возможность получения данных по запросам без разработки специальных программ.

Центральным аспектом MS SQL является база данных, которая представляет из себя хранилище данных, организованных определенным способом. Для организации баз данных MS SQL использует реляционную модель.

Реляционная модель – подход к управлению данными с использованием языка и структуры, согласованных с логикой первого порядка, где все данные представлены в виде кортежей, сгруппированных в отношения.

Реляционная модель базы данных:

* каждому типу объекта базы данных соответствует таблица. Строка — экземпляр объекта, столбец – атрибут (поле);
* у столбца уникальное имя в таблице. Возможны одинаковые имена столбцов, расположенных в разных таблицах;
* в таблице может содержаться любое количество строк (возможно, нулевое);
* строки таблицы не имеют определенного порядка. Нельзя выбрать строку по номеру в таблице. Нет “первой”, “последней” и т.д. строк;
* в каждой таблице есть один или несколько столбцов, значения или совокупность значений, которых во всех строках различаются. Этот столбец (столбцы) называется первичным ключом таблицы, и выбор конкретной записи таблицы можно осуществить по значению её первичного ключа;
* столбец (совокупность столбцов) одной таблицы, значения в котором совпадают со значениями столбца, являющегося первичным ключом другой таблицы, называется внешним столбцом. Совместно первичный и внешний ключи создают между таблицами отношения предок/потомок (родительская и дочерняя таблицы).

## **Среда управления SQL Server Management Studio**

SQL Server Management server – интегрированная среда для управления любой инфраструктурой SQL. Используется для настройки, доступа, администрирования, разработки всех компонентов SQL. Является стандартным инструментом для работы с MS SQL. Предоставляет удобный интерфейс пользователю для настройки, мониторинга, администрирования экземпляров SQL server и БД. Также предоставляет возможность создавать запросы и скрипты.

Компоненты среды:

* обозреватель объектов – нужен для просмотра всех объектов и управления ими;
* обозреватель шаблонов – нужен для создания файлов со стандартным текстом, которые используются для ускорения разработки скриптов, запросов, и управления ими;
* средства визуального проектирования – нужны для создания запросов, таблиц и схем баз данных;
* редакторы запросов и текста – нужны для интерактивного написания и откладки скриптов, запросов.

## **Язык программирования SQL**

SQL (Structured Query Language - «язык структурированных запросов») — декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей системой управления базами данных. Является, прежде всего, информационно-логическим языком, предназначенным для описания, изменения и извлечения данных, хранимых в реляционных базах данных. В общем случае SQL (без ряда современных расширений) считается языком программирования неполным по Тьюрингу, но вместе с тем стандарт языка спецификацией SQL/PSM предусматривает возможность его процедурных расширений.

На языке SQL описываются наборы данных, помогающие получать ответы на вопросы. При использовании SQL необходимо применять правильный синтаксис. Синтаксис - это набор правил, позволяющих правильно сочетать элементы языка. Синтаксис SQL основан на синтаксисе английского языка и имеет много общих элементов с синтаксисом языка Visual Basic для приложений (VBA).

## **Язык программирования C#**

На сегодняшний момент язык программирования C# один из самых мощных, быстро развивающихся и востребованных языков в ИТ-отрасли. В настоящий момент на нем пишутся самые различные приложения: от небольших десктопных программок до крупных веб-порталов и веб-сервисов, обслуживающих ежедневно миллионы пользователей.

C# уже не молодой язык и как и вся платформа .NET уже прошел большой путь. Первая версия языка вышла вместе с релизом Microsoft Visual Studio .NET в феврале 2002 года. Текущей версией языка является версия C# 11, которая вышла 8 ноября 2022 года вместе с релизом .NET 7.

C# является языком с Си-подобным синтаксисом и близок в этом отношении к C++ и Java. Поэтому, если вы знакомы с одним из этих языков, то овладеть C# будет легче.

C# является объектно-ориентированным и в этом плане много перенял у Java и С++. Например, C# поддерживает полиморфизм, наследование, перегрузку операторов, статическую типизацию. Объектно-ориентированный подход позволяет решить задачи по построению крупных, но в тоже время гибких, масштабируемых и расширяемых приложений. И C# продолжает активно развиваться, и с каждой новой версией появляется все больше интересных функциональностей.

Роль платформы .NET

Когда говорят C#, нередко имеют в виду технологии платформы .NET (Windows Forms, WPF, ASP.NET, Xamarin). И, наоборот, когда говорят .NET, нередко имеют в виду C#. Однако, хотя эти понятия связаны, отождествлять их неверно. Язык C# был создан специально для работы с фреймворком .NET, однако само понятие .NET несколько шире.

Как-то Билл Гейтс сказал, что платформа .NET - это лучшее, что создала компания Microsoft. Возможно, он был прав. Фреймворк .NET представляет мощную платформу для создания приложений. Можно выделить следующие ее основные черты:

Поддержка нескольких языков. Основой платформы является общеязыковая среда исполнения Common Language Runtime (CLR), благодаря чему .NET поддерживает несколько языков: наряду с C# это также VB.NET, C++, F#, а также различные диалекты других языков, привязанные к .NET, например, Delphi.NET. При компиляции код на любом из этих языков компилируется в сборку на общем языке CIL (Common Intermediate Language) - своего рода ассемблер платформы .NET. Поэтому при определенных условиях мы можем сделать отдельные модули одного приложения на отдельных языках.

Кроссплатформенность. .NET является переносимой платформой (с некоторыми ограничениями). Например, последняя версия платформы на данный момент - .NET 7 поддерживается на большинстве современных ОС Windows, MacOS, Linux. Используя различные технологии на платформе .NET, можно разрабатывать приложения на языке C# для самых разных платформ - Windows, MacOS, Linux, Android, iOS, Tizen.

Мощная библиотека классов. .NET представляет единую для всех поддерживаемых языков библиотеку классов. И какое бы приложение мы не собирались писать на C# - текстовый редактор, чат или сложный веб-сайт - так или иначе мы задействуем библиотеку классов .NET.

Разнообразие технологий. Общеязыковая среда исполнения CLR и базовая библиотека классов являются основой для целого стека технологий, которые разработчики могут задействовать при построении тех или иных приложений. Например, для работы с базами данных в этом стеке технологий предназначена технология ADO.NET и Entity Framework Core. Для построения графических приложений с богатым насыщенным интерфейсом - технология WPF и WinUI, для создания более простых графических приложений - Windows Forms. Для разработки кроссплатформенных мобильных и десктопных приложений - Xamarin/MAUI. Для создания веб-сайтов и веб-приложений - ASP.NET и т.д.

К этому стоит добавить активной развивающийся и набирающий популяность Blazor - фреймворк, который работает поверх .NET и который позволяет создавать веб-приложения как на стороне сервера, так и на стороне клиента. А в будущем будет поддерживать создание мобильных приложений и, возможно, десктоп-приложений.

Производительность. Согласно ряду тестов веб-приложения на .NET 7 в ряде категорий сильно опережают веб-приложения, построенные с помощью других технологий. Приложения на .NET 7 в принципе отличаются высокой производительностью.

Также еще следует отметить такую особенность языка C# и фреймворка .NET, как автоматическая сборка мусора. А это значит, что нам в большинстве случаев не придется, в отличие от С++, заботиться об освобождении памяти. Вышеупомянутая общеязыковая среда CLR сама вызовет сборщик мусора и очистит память.

# Структура базы данных разработанного приложения

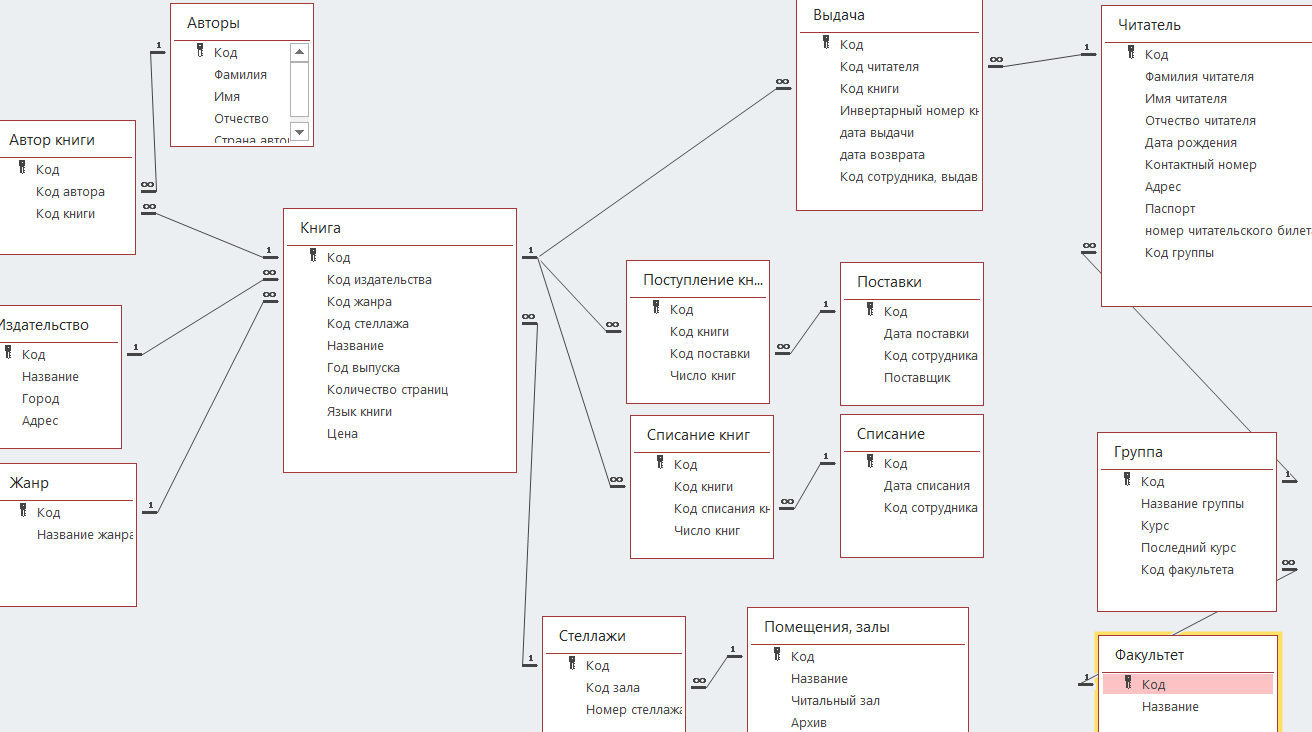
## **Состав таблиц базы данных**

Состав таблиц для базы данных библиотеки представлен в таблице 1.

**Таблица 1 – Состав таблиц базы данных**

|  |  |
| --- | --- |
| Наименование | Общие сведения о содержимом |
| 1. Издательство | издательство, выпустившее книгу |
| 1. Жанр | жанр книги |
| 1. Авторы | список всех авторов с информацией о них |
| 1. Автор книги | служит для связи между таблицами «Авторы», «Книга» |
| 1. Книга | содержит доступную информацию о книге |
| 1. Поставки | информация о поставках книг в библиотеку |
| 1. Списание | информация о списании книг |
| 1. Поступление книг | служит для связи между таблицами «Поставки», «Книга» |
| 1. Списание книг | служит для связи между таблицами «Списание», «Книга» |
| 1. Выдача и возврат | информация о выданной литературе читателю |
| 1. Стеллажи | описывает расположение книг на стеллажах |
| 1. Помещения | помещения библиотеки |
| 1. Сотрудники | сотрудники, работающие в библиотеке |
| 1. Читатель | информация о читателях библиотеки |
| 1. Группа | Информация о группах |
| 1. Факультет | Информация о факультетах |

Взаимосвязь между таблицами указана на рисунке 2.



**Рисунок 1 – Схема базы данных**

## **Структура таблиц базы данных**

В таблицах 2-17 представлена структура таблиц разработанной базы данных.

**Таблица 2 – Издательство**

|  |  |  |
| --- | --- | --- |
| Наименование поля | Формат поля | Содержимое поля |
| Код | int | код издательства |
| Название | varchar(50) | Название издательства |
| Город | varchar(40) | Город издательства |
| Адрес | varchar(80) | Адрес издательства |

**Таблица 3 – Жанр**

|  |  |  |
| --- | --- | --- |
| Наименование поля | Формат поля | Содержимое поля |
| Код | int | Код жанра |
| Название жанра | varchar(40) | Название жанра книги |

**Таблица 4 – Автор книги**

|  |  |  |
| --- | --- | --- |
| Наименование поля | Формат поля | Содержимое поля |
| Код | int | Код автора книги |
| Код книги | int | Код книги |
| Код автора | int | Код автора, написавшего книгу |

**Таблица 5 – Книга**

|  |  |  |
| --- | --- | --- |
| Наименование поля | Формат поля | Содержимое поля |
| Код | int | Код книги |
| Код издательства | int | Код издательства |
| Код жанра | int | Код жанра |
| Код стеллажа | int | Код стеллажа |
| Название | varchar(50) | Название книги |
| Год выпуска | int | Год выпуска книги |
| Число страниц | int | Число страниц в книге |
| Язык книги | varchar(20) | Язык, на котором издана книга |
| Цена | money | Цена книги при её закупке (в формате BUN) |

**Таблица 6 – Авторы**

|  |  |  |
| --- | --- | --- |
| Наименование поля | Формат поля | Содержимое поля |
| Код | int | Уникальный код Автора |
| Фамилия | varchar(40) | Фамилия автора |
| Имя | varchar(40) | Имя автора |
| Отчество | varchar(40) | Отчество автора |
| Страна автора | varchar(40) | Страна, где родился автор |

**Таблица 7 – Поступление книг**

|  |  |  |
| --- | --- | --- |
| Наименование поля | Формат поля | Содержимое поля |
| Код | int | Уникальный код поступления книг |
| Код книги | int | Код книги |
| Код поставки | int | Код поставки |
| Число книг | int | Число книг в партии |

**Таблица 8 – Поставки**

|  |  |  |
| --- | --- | --- |
| Наименование поля | Формат поля | Содержимое поля |
| Код | int | Уникальный код Поставки |
| Дата поставки | date | Дата, когда были поставлены книги |
| Код сотрудника | int | Код сотрудника, ответственного за получение партии книг |
| Поставщик | varchar(40) | Код поставщика |

**Таблица 9 – Списание книг**

|  |  |  |
| --- | --- | --- |
| Наименование поля | Формат поля | Содержимое поля |
| Код | int | Уникальный код списания книг |
| Код книги | int | Код книги |
| Код списания книг | int | Код списания книг |
| Число книг | int | Число книг, которые списаны за определенный период |

**Таблица 10 – Списание**

|  |  |  |
| --- | --- | --- |
| Наименование поля | Формат поля | Содержимое поля |
| Код | int | Уникальный код Списания |
| Дата списания | date | Дата списания книг |
| Код сотрудника | int | Код сотрудника, ответственного за списание книг |

**Таблица 11 – Выдача и возврат**

|  |  |  |
| --- | --- | --- |
| Наименование поля | Формат поля | Содержимое поля |
| Код | int | Уникальный код выдачи |
| Код читателя | int | Код читателя |
| Код книги | int | Код книги |
| Инвентарный номер | varchar(40) | Инвертарный номер книги |
| Дата выдачи | date | Дата выдачи книги читателю |
| Дата возврата | date | Дата возврата книги читателю |
| Код сотрудника, выдавшего книгу | int | Код сотрудника, ответственного за процесс выдачи книги |

**Таблица 12 – Стеллажи**

|  |  |  |
| --- | --- | --- |
| Наименование поля | Формат поля | Содержимое поля |
| Код | int | Уникальный код стеллажа |
| Код зала | int | Код зала, где находится стеллаж |
| Номер стеллажа | varchar(40) | Номер стеллажа (для идентификации) |

**Таблица 13 - Помещения**

|  |  |  |
| --- | --- | --- |
| Наименование поля | Формат поля | Содержимое поля |
| Код | int | Уникальный код помещения |
| Название | varchar(40) | Название помещения |
| Читальный зал | bit | Поле для отметки, чем является данное помещение |
| Архив | bit | Поле для отметки, чем является данное помещение |
| Абонемент | bit | Поле для отметки, чем является данное помещение |
| Адрес помещения | varchar(40) | Адрес помещения |

**Таблица 14 – Читатель**

|  |  |  |
| --- | --- | --- |
| Наименование поля | Формат поля | Содержимое поля |
| Код | int | Уникальный код читателя |
| Фамилия | varchar(40) | Фамилия читателя |
| Имя | varchar(40) | Имя читателя |
| Отчество | varchar(40) | Отчество читателя |
| Дата рождения | date | Дата рождения читателя |
| Контактный номер | varchar(40) | Номер телефона для связи с читателем |
| Адрес проживания | varchar(40) | Адрес, где живет читатель |
| Данные паспорта | char(60) | Идентификационный номер паспорта |
| Номер читательского билета | varchar(40) | Номер читательского билета |
| Код группы | int | Код группы |
| Имя для входа | varchar(100) | Имя логина читателя в базе данных |

**Таблица 15 - Сотрудники**

|  |  |  |
| --- | --- | --- |
| Наименование поля | Формат поля | Содержимое поля |
| Код | int | Уникальный код сотрудника |
| Код помещения | int | Код помещения, где работает сотрудник |
| Фамилия | varchar(40) | Фамилия сотрудника |
| Имя | varchar(40) | Имя сотрудника |
| Отчество | varchar(40) | Отчество сотрудника |
| Занимаемая должность | varchar(40) | Занимаемая сотрудником должность |
| Стаж | int | Стаж сотрудника (в годах) |

**Таблица 16 – Группа**

|  |  |  |
| --- | --- | --- |
| Наименование поля | Формат поля | Содержимое поля |
| Код | int | Уникальный код группы |
| Название | varchar(40) | Название группы |
| Курс | int | Курс группы |
| Последний курс | int | Последний курс группы |
| Код факультета | varchar(40) | Код факультета группы |

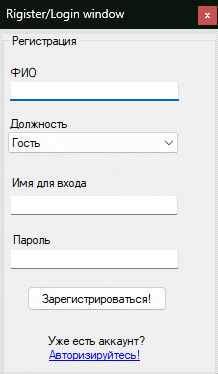
**Таблица 17 – Факультеты**

|  |  |  |
| --- | --- | --- |
| Наименование поля | Формат поля | Содержимое поля |
| Код | int | Уникальный код факультета |
| Название | varchar(40) | Название факультета |

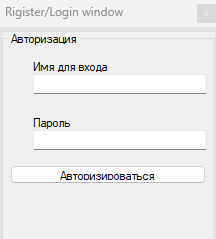
# Взаимодействие пользователя с разработанным приложением

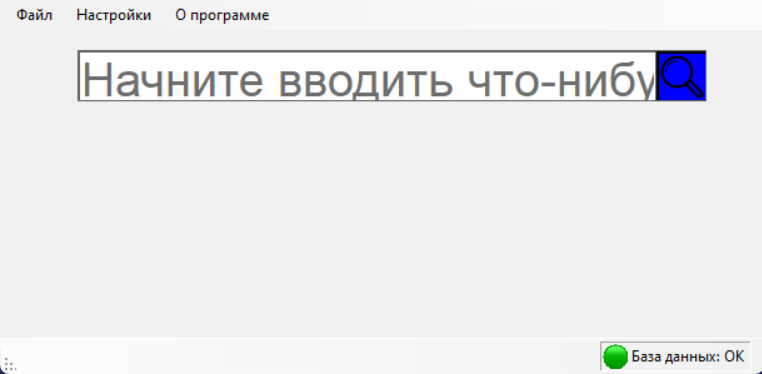
Для управления приложением с подключенной базой данных пользователю предоставляется окно GUI, на котором расположены различные элементы для отображения информации и управлением ею: текстовые поля, поля для ввода текста, выбор даты, списки (в т.ч. выпадающие), и кнопки

Предварительно, пользователь может зарегистрироваться или авторизироваться, чтобы получить доступ к дополнительным возможностям приложения. (По умолчанию активирован базовый функционал: только просмотр книг, авторов, издательств – публичная информация)

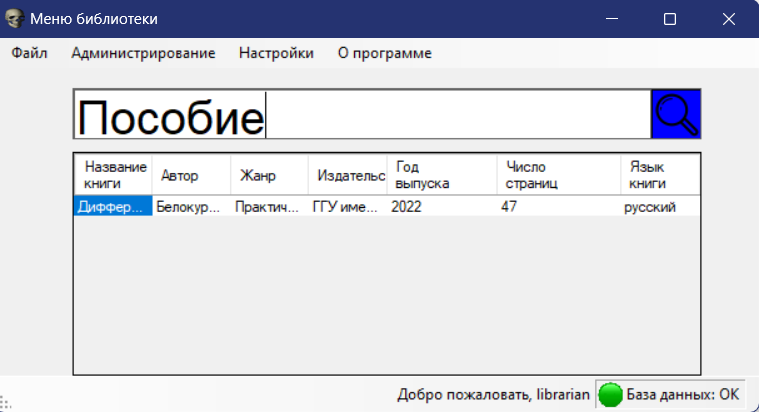


**Рисунок 2 – Регистрационная форма**

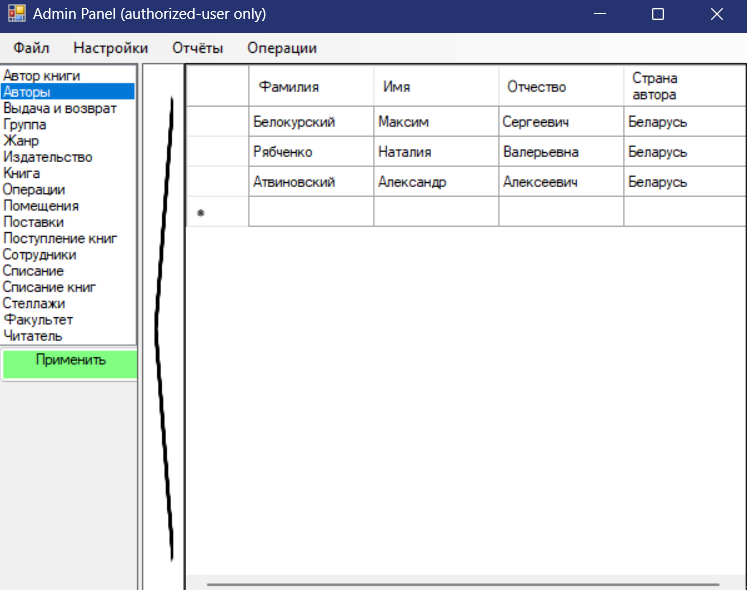
  
**Рисунок 3 – Форма авторизации пользователя**

****

**Рисунок 4 – Рабочая область пользовательского интерфейса для взаимодействия с пользователем (пользователь не вошёл в систему / нет прав**

****

**Рисунок 5 – Рабочая область пользовательского интерфейса для взаимодействия с пользователем (пользователь вошёл в систему).**

****

**Рисунок 6 – Рабочая область пользовательского интерфейса для управлением данными в базе данных (пользователь вошёл в систему, и у него есть права управлять базой данных).**

ЗАКЛЮЧЕНИЕ

Для упрощения работы с людьми и ресурсами библиотеки было создано приложение на языке С#, с подключением СУБД, в которой содержатся различные данные касательно структуры библиотеки, её содержимого, а также посещающих её клиентов (читателей). Пользователям с определёнными правами, таких как сотрудники, разрешено не только просматривать содержимое БД, но и вносить изменения, добавлять, удалять, редактировать информацию в БД.

Для успешного выполнения данной работы были изучены навыки работы с созданием оконных приложений в среде Visual Studio, язык С++, а также реляционные базы данных, в том числе язык запросов SQL.

Для достижения вышеописанных целей были выполнены следующие шаги:

* проанализировать требования;
* реализовать серверное подключение с базой данных;
* разработать клиентскую часть, которая будет интерфейсом пользователя, или сотрудника;
* сделать приложение простым и доступным в использовании, а также настроить ограничения на доступ к данным;
* протестировать разработанное приложение.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Шилд Г. C# 4.0. Полное руководство. – М.:Вильямс, 2022. -1056 с.
2. Скит Дж. C# программирование для профессионалов. – М.: Вильямс, 2019. – 608 с.
3. Грин Дж., Стиллмен Э. Изучаем C#/ - СПб.: Питер, 2021, - 816 с.
4. Васильев А.Н. Программирование для начинающих на С#. – М.: Бомбора, 2018, -592 с.
5. Фленов М.Е. Библия C#. – СПб.: БХВ-Петербург, 2020, - 512 с.
6. Лобел Л., Браст Э., Форте С. Разработка приложений на основе Microsoft SQL Server 2008. – СПб.: BHV, 2016. – 1024 с.

# ПРИЛОЖЕНИЕ А

## **Скрипт для создания структуры таблиц базы данных**

use Библиотека;

DROP PROCEDURE GetUserRoles;

DROP PROCEDURE GetUserTablePermissions;

DROP PROCEDURE SearchBooks;

DROP PROCEDURE УвеличитьКурсГрупп;

DROP TRIGGER trg\_УдалитьЧитателейПриОбновленииГруппы;

DROP FUNCTION dbo.GetPercentageUsersByGroupAndFaculty;

DROP FUNCTION dbo.GetDebtorsReport

drop function dbo.GetBookStatusAndDueDate

GO

alter table Книга

drop constraint FK\_Книга\_Издательство,

constraint FK\_Книга\_Жанр,

constraint FK\_Книга\_Стеллаж

GO

alter table [Автор книги]

drop constraint FK\_Книга\_Автор\_книги,

constraint FK\_Автор\_книги\_Авторы

GO

alter table [Поступление книг]

drop constraint FK\_Книга\_Поступления\_книг,

constraint FK\_Поступление\_книг\_Поставки

GO

alter table [Списание книг]

drop constraint FK\_Книга\_Списание\_книг,

constraint FK\_Списание\_книг\_Списание

GO

alter table [Выдача и возврат]

drop constraint FK\_Выдача\_Возврат\_Книга,

constraint FK\_Выдача\_Возврат\_Сотрудники,

constraint FK\_Выдача\_Возврат\_Читатель

GO

alter table [Стеллажи]

drop constraint FK\_Стеллажи\_Помещения

GO

alter table [Сотрудники]

drop constraint FK\_Сотрудники\_Помещения

go

alter table [Поставки]

drop constraint FK\_Поставки\_Сотрудники

go

alter table [Списание]

drop constraint FK\_Списание\_Сотрудники

go

alter table Группа

drop constraint FK\_Группа\_Факультеты

go

alter table Читатель

drop constraint FK\_Читатель\_Группы

go

drop table Операции

go

drop table Издательство

go

drop table Жанр

go

drop table Книга

go

drop table [Автор книги]

go

drop table Авторы

go

drop table [Поступление книг]

go

drop table Поставки

go

drop table [Списание книг]

go

drop table Списание

go

drop table [Выдача и возврат]

go

drop table Стеллажи

go

drop table Читатель

go

drop table [Помещения]

go

drop table Сотрудники

go

drop table Факультет

go

drop table Группа

go

Create table Издательство

(Код int not null primary key identity,

Название varchar(50),

Город varchar(40),

Адрес varchar(80) not null

)

go

create table Жанр

(Код int primary key not null identity,

[Название жанра] varchar(40) not null

)

GO

create table Книга

(Код int primary key not null identity,

[Код издательства] int,

[Код жанра] int,

[Код стеллажа] int,

[Название] varchar(50),

[Год выпуска] int,

[Число страниц] int,

[Язык книги] varchar(20) not null,

[Цена] money

)

GO

create table [Автор книги]

(Код int primary key not null identity,

[Код книги] int,

[Код автора] int

)

GO

create table Авторы

(Код int primary key not null identity,

[Фамилия] varchar(40),

[Имя] varchar(40),

[Отчество] varchar(40),

[Страна автора] varchar(40)

)

GO

create table [Поступление книг]

(Код int primary key not null identity,

[Код книги] int,

[Код поставки] Int,

[Число книг] int

)

GO

create table Поставки

(Код int primary key not null identity,

[Дата поставки] date,

[Код сотрудника] int,

[Поставщик] varchar(40)

)

GO

create table [Списание книг]

(Код int primary key not null identity,

[Код книги] int,

[Код списания книг] int,

[Число книг] int

)

GO

create table Списание

(Код int primary key not null identity,

[Дата списания] date,

[Код сотрудника] int

)

GO

create table [Выдача и возврат]

(

Код int primary key not null identity,

[Код читателя] int,

[Код книги] int,

[Инвертарный номер] varchar(40),

[Дата выдачи] date not null,

[Дата возврата] date,

[Код сотрудника, выдавшего книгу] int,

[Книга утеряна] bit

)

GO

create table Стеллажи

(Код int primary key not null identity,

[Код зала] int not null,

[Номер стеллажа] varchar(4)

)

GO

create table Помещения

(Код int primary key not null identity,

[Название] varchar(40),

[Читальный зал] bit,

[Архив] bit,

[Абонемент] bit,

[Адрес помещения] varchar(40)

)

GO

CREATE TABLE Читатель

(

Код INT PRIMARY KEY NOT NULL IDENTITY,

[Фамилия] VARCHAR(40),

[Имя] VARCHAR(40),

[Отчество] VARCHAR(40),

[Дата рождения] DATE,

[Контактный номер] VARCHAR(40),

[Адрес проживания] VARCHAR(40),

[Данные паспорта] CHAR(60),

[Номер читательского билета] VARCHAR(40),

[Код группы] INT,

[Имя для входа] VARCHAR(100)

);

create table Сотрудники

(Код int primary key not null identity,

[Код помещения] int,

[Фамилия] varchar(40),

[Имя] varchar(40),

[Отчество] varchar(40),

[Занимаемая должность] varchar(40),

[Стаж] int

)

GO

CREATE TABLE Группа

(

Код INT PRIMARY KEY NOT NULL IDENTITY,

Название VARCHAR(40) NOT NULL,

Курс INT NOT NULL,

[Последний курс] INT NOT NULL,

[Код факультета] INT NOT NULL

);

CREATE TABLE Факультет

(

Код INT PRIMARY KEY NOT NULL IDENTITY,

Название VARCHAR(40) NOT NULL

);

alter table Книга

add constraint FK\_Книга\_Издательство foreign key ([Код издательства]) references Издательство(Код),

constraint FK\_Книга\_Жанр foreign key ([Код жанра]) references Жанр(Код),

constraint FK\_Книга\_Стеллаж foreign key ([Код стеллажа]) references Стеллажи(Код)

GO

alter table [Автор книги]

add constraint FK\_Книга\_Автор\_книги foreign key ([Код книги]) references Книга(Код),

constraint FK\_Автор\_книги\_Авторы foreign key ([Код автора]) references Авторы(Код)

GO

alter table [Поступление книг]

add constraint FK\_Книга\_Поступления\_книг foreign key ([Код книги]) references Книга(Код),

constraint FK\_Поступление\_книг\_Поставки foreign key ([Код поставки]) references Авторы(Код)

GO

alter table [Списание книг]

add constraint FK\_Книга\_Списание\_книг foreign key ([Код книги]) references Поставки(Код),

constraint FK\_Списание\_книг\_Списание foreign key ([Код списания книг]) references Списание(Код)

GO

alter table [Выдача и возврат]

add constraint FK\_Выдача\_Возврат\_Книга foreign key ([Код книги]) references Книга(Код),

constraint FK\_Выдача\_Возврат\_Сотрудники foreign key ([Код сотрудника, выдавшего книгу]) references Сотрудники(Код),

constraint FK\_Выдача\_Возврат\_Читатель foreign key ([Код читателя]) references Читатель(Код)

GO

alter table [Стеллажи]

add constraint FK\_Стеллажи\_Помещения foreign key ([Код зала]) references Помещения(Код)

GO

alter table [Сотрудники]

add constraint FK\_Сотрудники\_Помещения foreign key ([Код помещения]) references Помещения(Код)

GO

alter table [Поставки]

add constraint FK\_Поставки\_Сотрудники foreign key ([Код сотрудника]) references Сотрудники(Код)

GO

alter table [Списание]

add constraint FK\_Списание\_Сотрудники foreign key ([Код сотрудника]) references Сотрудники(Код)

GO

alter table Читатель

add constraint FK\_Читатель\_Группы foreign key ([Код группы]) references Группа(Код)

GO

alter table Группа

add constraint FK\_Группа\_Факультеты foreign key ([Код факультета]) references Факультет(Код)

GO

insert into Жанр

values

('Практическое пособие'),

('Книга'),

('Статья')

insert into Издательство

values

('ГГУ имени Ф. Скорины','Гомель','Советская 104')

insert into Факультет

values

('Математики и технологий программирования')

insert into [Группа]

values

('ПИ', 4,4,1),

('ПИ',3,4,1)

insert into Читатель

values

('Обухов','Никита','Михайлович','2002-06-21','+37529#######','Свиридова ##','HB#######','number\_bilet', 1, 'librarian'),

('Макаренко','Владислав','Станиславович','2002-12-12','+37529#######','Чечерская ##','HB#######','number\_bilet', 2, null),

('Фамилия','Имя','Отчество','2003-12-12','+37529#######','Чечерская ##','HB#######','number\_bilet', 2, null)

insert into Авторы

values

('Белокурский','Максим','Сергеевич','Беларусь'),

('Рябченко','Наталия','Валерьевна','Беларусь'),

('Атвиновский','Александр','Алексеевич','Беларусь')

insert into Помещения(Название,[Адрес помещения])

values

('1й читальный зал при ГГУ им. Ф. Скорины','ул. Советская 102'),

('2й читальный зал при ГГУ им. Ф. Скорины','ул. Советская 102')

insert into Стеллажи([Код зала],[Номер стеллажа])

values

(1,1)

insert into Книга

values

(1,1,1,'Дифференциальные уравнения', 2022, 47, 'русский', 30),

(1,2,1,'Программирование', 2007, 211, 'русский', 38)

insert into [Автор книги]([Код книги],[Код автора])

values

(1,1),

(2,2)

insert into Сотрудники

values

(1,'Иванов','Петр','Ильич','Библиотекарь',3),

(1,'Семченко','Алла','Сергеевна','Уборщица',2)

insert into [Поступление книг]([Код книги],[Код поставки],[Число книг])

values

(1,1,10)

insert into [Поставки]

values

('2022-05-07',1,'ГГУ им. Ф. Скорины')

insert into [Выдача и возврат]

values

(3, 2, '1', '2022-12-15', NULL, 1, 0);

go

CREATE PROCEDURE SearchBooks

@SearchTerm VARCHAR(100)

AS

BEGIN

SELECT

Книга.Название AS 'Название книги',

Авторы.Фамилия + ' ' + Авторы.Имя + ' ' + Авторы.Отчество AS 'Автор',

Жанр.[Название жанра] AS 'Жанр',

Издательство.Название AS 'Издательство',

Книга.[Год выпуска] AS 'Год выпуска',

Книга.[Число страниц] AS 'Число страниц',

Книга.[Язык книги] AS 'Язык книги'

FROM

Книга

JOIN

Жанр ON Книга.[Код жанра] = Жанр.Код

JOIN

[Автор книги] ON [Автор книги].[Код книги] = Книга.Код

JOIN

[Авторы] ON [Авторы].Код = [Автор книги].[Код автора]

JOIN

[Издательство] ON Книга.[Код издательства] = [Издательство].Код

WHERE

Книга.Название LIKE '%' + @SearchTerm + '%'

OR Авторы.Фамилия LIKE '%' + @SearchTerm + '%'

OR Авторы.Имя LIKE '%' + @SearchTerm + '%'

OR Авторы.Отчество LIKE '%' + @SearchTerm + '%'

OR Жанр.[Название жанра] LIKE '%' + @SearchTerm + '%';

END;

go

CREATE PROCEDURE GetUserTablePermissions

@DatabaseUserName NVARCHAR(255),

@TableName NVARCHAR(255)

AS

BEGIN

SELECT dp.permission\_name

FROM sys.database\_permissions dp

JOIN sys.database\_principals dpn ON dp.grantee\_principal\_id = dpn.principal\_id

WHERE dpn.name = (SELECT dp.name AS DatabaseUserName

FROM sys.database\_principals dp

JOIN sys.server\_principals sp ON dp.sid = sp.sid

WHERE sp.name = @DatabaseUserName) AND dp.major\_id = OBJECT\_ID(@TableName)

END;

go

CREATE PROCEDURE GetUserRoles

@LoginName NVARCHAR(100)

AS

BEGIN

DECLARE @UserId INT;

SELECT @UserId = dp.principal\_id

FROM sys.database\_principals dp

JOIN sys.server\_principals sp ON dp.sid = sp.sid

WHERE sp.name = @LoginName;

SELECT role.name AS RoleName

FROM sys.database\_role\_members members

JOIN sys.database\_principals role ON members.role\_principal\_id = role.principal\_id

WHERE members.member\_principal\_id = @UserId;

END;

go

CREATE TABLE Операции

(

Код INT PRIMARY KEY NOT NULL IDENTITY,

[Таблица] VARCHAR(50) NOT NULL,

[Операция] VARCHAR(50) NOT NULL,

[Дата операции] INT NOT NULL

);

go

CREATE PROCEDURE УвеличитьКурсГрупп

AS

BEGIN

IF NOT EXISTS (SELECT 1 FROM Операции WHERE [Таблица] = 'Группа' AND [Операция] = 'Увеличение курса' AND YEAR([Дата операции]) = YEAR(GETDATE()))

BEGIN

UPDATE Группа

SET Курс = Курс + 1;

INSERT INTO Операции ([Таблица], [Операция], [Дата операции])

VALUES ('Группа', 'Увеличение курса', YEAR(GETDATE()));

END;

END;

go

CREATE TRIGGER trg\_УдалитьЧитателейПриОбновленииГруппы

ON Группа

AFTER UPDATE

AS

BEGIN

IF UPDATE([Последний курс]) OR UPDATE([Курс])

BEGIN

IF EXISTS (

SELECT 1

FROM Операции

WHERE [Таблица] = 'Группа'

AND [Операция] = 'Увеличение курса'

AND [Дата операции] = YEAR(GETDATE())

)

BEGIN

ROLLBACK;

RETURN;

END

DELETE FROM Читатель

WHERE [Код группы] IN (

SELECT G.[Код]

FROM Группа G

INNER JOIN INSERTED I ON G.[Код] = I.[Код]

WHERE G.[Курс] > G.[Последний курс]

);

-- Удаляем группу, но только если не осталось читателей

DELETE FROM Группа

WHERE [Курс] > [Последний курс];

END

END;

GO

--exec УвеличитьКурсГрупп

CREATE FUNCTION dbo.GetPercentageUsersByGroupAndFaculty

(

@StartDate DATE,

@EndDate DATE

)

RETURNS TABLE

AS

RETURN

(

WITH ВыдачиКниг AS (

SELECT

Ч.Код AS [Код читателя],

Г.[Код] AS [Код группы]

FROM

[Выдача и возврат] В

INNER JOIN Читатель Ч ON В.[Код читателя] = Ч.Код

INNER JOIN Группа Г ON Ч.[Код группы] = Г.[Код]

WHERE

В.[Дата выдачи] BETWEEN @StartDate AND @EndDate

),

ЧитателиСВыдачами AS (

SELECT

Г.[Код факультета],

Г.[Код] AS [Код группы],

COUNT(DISTINCT Ч.[Код]) AS С\_выдачами

FROM

Читатель Ч

INNER JOIN ВыдачиКниг ВК ON Ч.Код = ВК.[Код читателя]

INNER JOIN Группа Г ON Ч.[Код группы] = Г.[Код]

WHERE

Ч.[Код группы] IS NOT NULL

GROUP BY

Г.[Код факультета],

Г.[Код]

)

SELECT

Ф.[Название] AS [Название факультета],

Г.[Название] AS [Название группы],

Г.[Курс],

ISNULL(ЧитателиСВыдачами.С\_выдачами, 0) AS [С выдачами],

ISNULL(Читатели.Общее\_количество, 0) AS [Общее количество],

IIF(ISNULL(Читатели.Общее\_количество, 0) = 0, 0, 100 \* CAST(ISNULL(ЧитателиСВыдачами.С\_выдачами, 0) AS FLOAT) / CAST(ISNULL(Читатели.Общее\_количество, 0) AS FLOAT)) AS [Процент пользователей от читателей]

FROM

Факультет Ф

INNER JOIN Группа Г ON Ф.[Код] = Г.[Код факультета]

LEFT JOIN ЧитателиСВыдачами ON Г.[Код] = ЧитателиСВыдачами.[Код группы] AND Ф.[Код] = ЧитателиСВыдачами.[Код факультета]

LEFT JOIN (

SELECT

Г.[Код факультета],

Г.[Код] AS [Код группы],

COUNT(DISTINCT Ч.[Код]) AS Общее\_количество

FROM

Читатель Ч

LEFT JOIN Группа Г ON Ч.[Код группы] = Г.[Код]

WHERE

Ч.[Код группы] IS NOT NULL

GROUP BY

Г.[Код факультета],

Г.[Код]

) AS Читатели ON Г.[Код] = Читатели.[Код группы] AND Ф.[Код] = Читатели.[Код факультета]

);

go

CREATE FUNCTION dbo.GetDebtorsReport(@StartDate DATETIME2, @EndDate DATETIME2)

RETURNS TABLE

AS

RETURN

(

WITH ВыдачиКниг AS (

SELECT

Г.[Код факультета],

Ч.[Код группы],

MONTH(В.[Дата выдачи]) AS [Месяц],

CASE

WHEN MONTH(В.[Дата выдачи]) BETWEEN 1 AND 6 THEN '1-й семестр'

WHEN MONTH(В.[Дата выдачи]) BETWEEN 7 AND 12 THEN '2-й семестр'

END AS [Семестр],

YEAR(В.[Дата выдачи]) AS [Год],

COUNT(\*) AS [Количество должников]

FROM

Читатель Ч

INNER JOIN [Выдача и возврат] В ON Ч.[Код] = В.[Код читателя] AND В.[Дата возврата] IS NULL -- Должники

INNER JOIN Группа Г ON Ч.[Код группы] = Г.[Код]

WHERE

В.[Дата выдачи] BETWEEN @StartDate AND @EndDate

GROUP BY

Г.[Код факультета],

Ч.[Код группы],

MONTH(В.[Дата выдачи]),

CASE

WHEN MONTH(В.[Дата выдачи]) BETWEEN 1 AND 6 THEN '1-й семестр'

WHEN MONTH(В.[Дата выдачи]) BETWEEN 7 AND 12 THEN '2-й семестр'

END,

YEAR(В.[Дата выдачи])

)

SELECT

Ф.[Название] AS [Название факультета],

Г.[Название] AS [Название группы],

Г.[Курс],

VK.[Месяц],

VK.[Семестр],

VK.[Год],

VK.[Количество должников]

FROM

ВыдачиКниг VK

INNER JOIN Группа Г ON VK.[Код группы] = Г.[Код]

INNER JOIN Факультет Ф ON VK.[Код факультета] = Ф.[Код]

);

go

CREATE FUNCTION dbo.GetBookStatusAndDueDate

(

@ReaderID INT,

@BookID INT

)

RETURNS TABLE

AS

RETURN

(

SELECT

CASE

WHEN EXISTS (

SELECT 1

FROM [Выдача и возврат]

WHERE [Код читателя] = @ReaderID

AND [Код книги] = @BookID

AND [Дата возврата] IS NULL

) THEN 'Книга на руках'

ELSE 'Книга сдана'

END AS [Статус книги],

DATEADD(MONTH, 1, [Выдача и возврат].[Дата выдачи]) AS [Ближайшая дата сдачи]--MIN(ISNULL([Дата возврата], '9999-12-31'))

FROM [Выдача и возврат]

WHERE [Код читателя] = @ReaderID

AND [Код книги] = @BookID

);

go

select \* from dbo.GetBookStatusAndDueDate(1,1)

--o1

SELECT \* FROM dbo.GetPercentageUsersByGroupAndFaculty('2020-01-01', '2024-12-31');

--o2

SELECT \* FROM dbo.GetDebtorsReport('2020-01-01', '2024-12-31');

--o3

select \* from dbo.GetBookStatusAndDueDate(3,2)

--y++

exec УвеличитьКурсГрупп

# ПРИЛОЖЕНИЕ Б

## **Код C#, выполняющий подключение к СУБД, и выполняющий ряд заданных действий для взаимодействия с СУБД**

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.SqlClient;

using System.Drawing;

using System.Globalization;

using System.IO;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace LibA

{

public partial class AdminPanel : Form

{

private static AdminPanel instance;

private static bool isClosed = true;

public static AdminPanel Instance

{

get

{

if (instance == null || isClosed)

{

instance = new AdminPanel();

instance.FormClosed += (sender, e) =>

{

isClosed = true;

};

isClosed = false;

}

return instance;

}

}

private AdminPanel()

{

InitializeComponent();

this.Shown += LoadTablesAsync;

}

private async void LoadTablesAsync(object sender, EventArgs e)

{

string[] tables = null;

try

{

tables = await DBWorker.BdGetDataMSSQL("SELECT table\_name FROM information\_schema.tables WHERE table\_type = 'BASE TABLE' order by table\_name ASC");

}

catch

{

this.Close();

return;

}

if (tables != null)

{

Tables.Items.AddRange(tables);

}

Tables.Height = Tables.ItemHeight \* (Tables.Items.Count + 1);

buttonTransact.Location = new Point(

Tables.Left,

Tables.Bottom

);

buttonRollback.Location = new Point(

buttonTransact.Left,

buttonTransact.Bottom

);

this.Height = dataGridViewMain.Bottom;

this.Width = dataGridViewMain.Right + (this.Width - this.ClientSize.Width);

}

private async void Tables\_SelectedIndexChanged(object sender, EventArgs e)

{

if (DBWorker.OldTable != null && !AreTablesEqual(dataGridViewMain.DataSource as DataTable, DBWorker.OldTable))

{

DialogResult dgv = MessageBox.Show("У вас есть несохранённые изменения. Вы хотите сохранить их?", "Предупреждение", MessageBoxButtons.YesNo, MessageBoxIcon.Hand);

if (dgv == DialogResult.Yes)

{

buttonTransact\_Click(sender, e);

return;

}

}

string selectedTable = Tables.Items[Tables.SelectedIndex > 0 ? Tables.SelectedIndex : 0].ToString();

DataTable datatable = await DBWorker.GetDataTable(selectedTable);

if (datatable != null)

{

buttonRollback.Enabled = false;

buttonRollback.BackColor = Color.Gray;

DBWorker.OldTable = datatable.Copy();

dataGridViewMain.DataSource = datatable;

dataGridViewMain.Enabled = true;

dataGridViewMain.Visible = true;

dataGridViewMain.Columns[0].Visible = false;

dataGridViewMain.AutoGenerateColumns = true;

}

}

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

{

this.Close();

this.Dispose();

}

private bool AreTablesEqual(DataTable table1, DataTable table2)

{

try

{

if (table1 == null || table2 == null)

{

return table1 == table2;

}

if (table1.Rows.Count != table2.Rows.Count || table1.Columns.Count != table2.Columns.Count)

{

return false;

}

for (int i = 0; i < table1.Rows.Count; i++)

{

for (int j = 0; j < table1.Columns.Count; j++)

{

if (!object.Equals(table1.Rows[i][j], table2.Rows[i][j]))

{

return false;

}

}

}

}

catch

{

return false;

}

return true;

}

private async void buttonTransact\_Click(object sender, EventArgs e)

{

try

{

await DBWorker.BeginTransaction(dataGridViewMain);

buttonRollback.Enabled = true;

buttonRollback.BackColor = System.Drawing.Color.FromArgb(((int)(((byte)(255)))), ((int)(((byte)(128)))), ((int)(((byte)(128)))));

MessageBox.Show("Изменения сохранены в базе данных.");

}

catch (Exception er)

{

MessageBox.Show("Непредвиденная ошибка при обновлении данных:\n" + er.Message);

}

}

private async void buttonRollback\_Click(object sender, EventArgs e)

{

try

{

dataGridViewMain.DataSource = DBWorker.OldTable;

await DBWorker.BeginTransaction(DBWorker.OldTable);

buttonRollback.Enabled = false;

buttonRollback.BackColor = Color.Gray;

MessageBox.Show("Откат выполнен успешно");

}

catch

{

MessageBox.Show("Ошибка отката");

}

}

private void dataGridViewMain\_CellEndEdit(object sender, DataGridViewCellEventArgs e)

{

try

{

if (dataGridViewMain.Columns[e.ColumnIndex] is DataGridViewTextBoxColumn column &&

column.DefaultCellStyle.Format == "dd.MM.yyyy")

{

if (DateTime.TryParseExact(

dataGridViewMain.Rows[e.RowIndex].Cells[e.ColumnIndex].Value.ToString(),

"dd.MM.yyyy",

CultureInfo.InvariantCulture,

DateTimeStyles.None,

out DateTime date))

{

dataGridViewMain.Rows[e.RowIndex].Cells[e.ColumnIndex].Value = date.ToString("dd.MM.yyyy");

}

else

{

MessageBox.Show("Неверный формат даты!");

}

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при редактировании: {ex.Message}");

}

}

private async void lPane\_Panel2\_Click(object sender, EventArgs e)

{

int targetWidth;

if (lPane.Panel1Collapsed)

{

targetWidth = 200;

lPane.Panel1Collapsed = !lPane.Panel1Collapsed;

await AnimateSplitter(targetWidth);

}

else

{

targetWidth = 40;

await AnimateSplitter(targetWidth);

lPane.Panel1Collapsed = !lPane.Panel1Collapsed;

}

lPane.Panel2.BackgroundImage.RotateFlip(RotateFlipType.RotateNoneFlipX);

lPane.Refresh();

}

private async Task AnimateSplitter(int targetWidth)

{

const int AnimationDuration = 100;

const int AnimationSteps = 10;

int currentWidth = lPane.Width;

int step = (targetWidth - currentWidth) / AnimationSteps;

for (int i = 0; i < AnimationSteps; i++)

{

currentWidth += step;

lPane.Width = currentWidth;

await Task.Delay(AnimationDuration / AnimationSteps);

}

lPane.Width = targetWidth;

dataGridViewMain.Dock = DockStyle.Fill;

}

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

{

Program.MakeFocus(SettingsPane.Instance);

}

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

{

ConnectionManager.Instance.Disconnect();

this.Close();

}

private void выходToolStripMenuItem\_Click\_1(object sender, EventArgs e)

{

this.Close();

}

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

{

DynamicLinkerAndReports dln = new DynamicLinkerAndReports();

dln.MakeData("SELECT \* FROM dbo.GetPercentageUsersByGroupAndFaculty(@startdate, @enddate)");

dln.ShowDialog();

}

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

{

DynamicLinkerAndReports dln = new DynamicLinkerAndReports();

dln.MakeData("SELECT \* FROM dbo.GetDebtorsReport(@startdate, @enddate)");

dln.ShowDialog();

}

private async void должникиToolStripMenuItem\_Click(object sender, EventArgs e)

{

try

{

int selectedRow = 0;

using (DynamicLinkerAndReports dln = new DynamicLinkerAndReports())

{

DataTable data = await DBWorker.GetDataTable("SELECT \* from Читатель");

dln.MakeData(data);

dln.ShowDialog();

selectedRow = dln.selectedRowCode;

}

if (selectedRow == 0)

return;

using (DynamicLinkerAndReports dln = new DynamicLinkerAndReports())

{

DataTable data = await DBWorker.GetDataTable($"select \* from dbo.GetBookStatusAndDueDate({selectedRow})");

dln.MakeData(data);

dln.ShowDialog();

}

}

catch { }

}

private async void перевестиГруппыНаСледующийГодToolStripMenuItem\_Click(object sender, EventArgs e)

{

await DBWorker.ExecProcedure("УвеличитьКурсГрупп");

}

private async void dataGridViewMain\_CellMouseDoubleClick(object sender, DataGridViewCellMouseEventArgs e)

{

string tableName = null;

if (dataGridViewMain.DataSource is DataTable dataTable1)

{

tableName = dataTable1.TableName;

}

if (e.RowIndex >= 0 && e.ColumnIndex >= 0)

{

string columnName = dataGridViewMain.Columns[e.ColumnIndex].Name;

if (columnName.ToLower().StartsWith("код") || columnName.ToLower().StartsWith("id"))

{

int selectedRow = 0;

DataTable dataTable = await GetDataForTable(columnName, tableName);

using (DynamicLinkerAndReports linkerAndReports = new DynamicLinkerAndReports())

{

linkerAndReports.MakeData(dataTable);

linkerAndReports.ShowDialog();

selectedRow = linkerAndReports.selectedRowCode;

}

if (selectedRow != 0)

{

dataGridViewMain.EndEdit();

dataGridViewMain.Rows[e.RowIndex].Cells[e.ColumnIndex].Value = selectedRow;

dataGridViewMain.InvalidateCell(e.ColumnIndex, e.RowIndex);

}

}

}

}

private async Task<DataTable> GetDataForTable(string columnName, string selectedTable)

{

DataTable dataTable = new DataTable();

string lowerColumnName = columnName.ToLower();

string[] tablesSM = await DBWorker.BdGetDataMSSQL("SELECT table\_name FROM information\_schema.tables WHERE table\_type = 'BASE TABLE' order by table\_name ASC");

List<string> tables = new List<string>(tablesSM);

string matchingTableName = DBWorker.FindMatchingTableName(lowerColumnName, tables, selectedTable);

if (!string.IsNullOrEmpty(matchingTableName))

{

dataTable = await DBWorker.GetDataTable(matchingTableName);

}

return dataTable;

}

private async void загрузкаПервокурсниковToolStripMenuItem\_Click(object sender, EventArgs e)

{

OpenFileDialog openFileDialog = new OpenFileDialog();

openFileDialog.InitialDirectory = "desktop";

openFileDialog.Filter = "CSV|\*.csv|txt|\*.txt|CSV или TXT|\*.\*";

openFileDialog.Title = "Открытие файла...";

if (openFileDialog.ShowDialog() == DialogResult.OK)

{

string selectedFileName = openFileDialog.FileName;

using (StreamReader reader = new StreamReader(selectedFileName))

{

bool firstLine = true;

while (!reader.EndOfStream)

{

var line = reader.ReadLine().Split(';');

if (firstLine)

{

firstLine = false;

continue;

}

var f = line[1];

var i = line[2];

var o = line[3];

var date = DateTime.ParseExact(line[4], "dd.MM.yyyy", CultureInfo.InvariantCulture);

var number = line[5];

var adress = line[6];

var passport = line[7];

var readerId = line[8];

var gName = line[9];

var login = line[10];

string checkReaderQuery = $"Select 1 Where Exists (Select 1 From Читатель Where Фамилия = '{f}' and Имя = '{i}' and Отчество = '{o}')";

string insertReaderQuery = $"INSERT INTO Читатель VALUES (@Фамилия, @Имя, @Отчество, @Дата\_рождения, @Контактный\_номер, @Адрес\_проживания, @Данные\_паспорта, @Номер\_читательского\_билета, @Код\_группы, @Имя\_для\_входа)";

string updateReaderQuery = $"UPDATE Читатель SET [Фамилия] = @Фамилия, [Имя] = @Имя, [Отчество] = @Отчество, [Дата рождения]=@Дата\_рождения, [Контактный номер]=@Контактный\_номер, [Адрес проживания]=@Адрес\_проживания,[Данные паспорта]=@Данные\_паспорта,[Номер читательского билета]=@Номер\_читательского\_билета,[Код группы]=@Код\_группы,[Имя для входа] = @Имя\_для\_входа where [Фамилия] = @Фамилия AND [Имя] = @Имя AND [Отчество] = @Отчество";

using (SqlConnection connection = await ConnectionManager.Instance.OpenConnection())

{

using (SqlCommand checkReaderCommand = new SqlCommand(checkReaderQuery, connection))

{

object result = checkReaderCommand.ExecuteScalar();

int readerCount = result == null ? 0 : (int)result;

int groupId = GetOrCreateGroup(connection, gName);

if ((int)readerCount == 0)

{

using (SqlCommand insertCommand = new SqlCommand(insertReaderQuery, connection))

{

insertCommand.Parameters.AddWithValue("@Фамилия", f);

insertCommand.Parameters.AddWithValue("@Имя", i);

insertCommand.Parameters.AddWithValue("@Отчество", o);

insertCommand.Parameters.AddWithValue("@Дата\_рождения", date);

insertCommand.Parameters.AddWithValue("@Контактный\_номер", number);

insertCommand.Parameters.AddWithValue("@Адрес\_проживания", adress);

insertCommand.Parameters.AddWithValue("@Данные\_паспорта", passport);

insertCommand.Parameters.AddWithValue("@Номер\_читательского\_билета", readerId);

insertCommand.Parameters.AddWithValue("@Код\_группы", groupId);

insertCommand.Parameters.AddWithValue("@Имя\_для\_входа", login);

insertCommand.ExecuteNonQuery();

}

}

else

{

DialogResult updateResult = MessageBox.Show($"Читатель {f} {i} {o} уже существует. Обновить информацию?\n\nВнимание, это удалит сущетсвующего пользователя!", "Обновление читателя", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

if (updateResult == DialogResult.Yes)

{

using (SqlCommand updateCommand = new SqlCommand(updateReaderQuery, connection))

{

updateCommand.Parameters.AddWithValue("@Фамилия", f);

updateCommand.Parameters.AddWithValue("@Имя", i);

updateCommand.Parameters.AddWithValue("@Отчество", o);

updateCommand.Parameters.AddWithValue("@Дата\_рождения", date);

updateCommand.Parameters.AddWithValue("@Контактный\_номер", number);

updateCommand.Parameters.AddWithValue("@Адрес\_проживания", adress);

updateCommand.Parameters.AddWithValue("@Данные\_паспорта", passport);

updateCommand.Parameters.AddWithValue("@Номер\_читательского\_билета", readerId);

updateCommand.Parameters.AddWithValue("@Код\_группы", groupId);

updateCommand.Parameters.AddWithValue("@Имя\_для\_входа", login);

updateCommand.ExecuteNonQuery();

}

}

}

}

}

}

}

}

MessageBox.Show("Данные успешно обработаны и добавлены в базу данных.", "Успех", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

private int GetOrCreateGroup(SqlConnection connection, string groupName)

{

string selectGroupQuery = "SELECT Код FROM Группа WHERE Название = @GroupName AND Курс = 1";

using (SqlCommand selectGroupCommand = new SqlCommand(selectGroupQuery, connection))

{

selectGroupCommand.Parameters.AddWithValue("@GroupName", groupName);

object result = selectGroupCommand.ExecuteScalar();

if (result != null)

{

return (int)result;

}

else

{

string insertGroupQuery = "INSERT INTO Группа (Название, Курс, [Последний курс], [Код факультета]) VALUES (@GroupName, 1, 2, NULL); SELECT SCOPE\_IDENTITY();";

using (SqlCommand insertGroupCommand = new SqlCommand(insertGroupQuery, connection))

{

insertGroupCommand.Parameters.AddWithValue("@GroupName", groupName);

return Convert.ToInt32(insertGroupCommand.ExecuteScalar());

}

}

}

}

}

}

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.SqlClient;

using System.IO.Packaging;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace LibA

{

internal class DBWorker

{

private static DataTable oldTable;

public static DataTable OldTable

{

set { oldTable = value; }

get { return oldTable!; }

}

public static async Task<string[]> BdGetDataMSSQL(string commandText)

{

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

try

{

using (SqlConnection connection = await ConnectionManager.Instance.OpenConnection())

{

using (SqlCommand command = new SqlCommand(commandText, connection))

{

using (var reader = await command.ExecuteReaderAsync())

{

while (await reader.ReadAsync())

{

result.Add(reader.GetString(0));

}

}

}

connection.Close();

}

}

catch (Exception e)

{

Console.WriteLine(e.ToString());

throw;

}

return result.ToArray();

}

public static DataTable GetDataTable(SqlCommand command) {

using (SqlDataAdapter adapter = new SqlDataAdapter(command))

{

DataTable dataTable = new();

adapter.Fill(dataTable);

return dataTable;

}

}

public static async Task<DataTable> GetDataTable(string tableName)

{

string commandText = null;

if (tableName.ToLower().StartsWith("select"))

commandText = tableName;

else

commandText = $"SELECT \* FROM [{tableName}]";

try

{

using (SqlCommand command = new SqlCommand(commandText, await ConnectionManager.Instance.OpenConnection()))

{

using (SqlDataAdapter adapter = new SqlDataAdapter(command))

{

DataTable dataTable = new(tableName);

adapter.Fill(dataTable);

return dataTable;

}

}

}

catch (Exception e)

{

MessageBox.Show(e.ToString());

}

return null;

}

public static async Task ExecProcedure(string procname, params SqlParameter[] parameters) {

try

{

using (SqlCommand command = new SqlCommand(procname, await ConnectionManager.Instance.OpenConnection()))

{

command.CommandType = CommandType.StoredProcedure;

foreach (var parameter in parameters)

{

command.Parameters.Add(parameter);

}

await command.ExecuteNonQueryAsync();

}

MessageBox.Show($"Выполнение процедуры {procname} успешно");

}

catch (Exception e)

{

MessageBox.Show($"Ошибка выполнения процедуры {procname}: {e.Message}");

}

}

public static async Task BeginTransaction(DataGridView dgv)

{

DataTable dataTable = dgv.DataSource as DataTable;

oldTable = dataTable;

await BeginTransaction(dataTable);

}

public static async Task BeginTransaction(DataTable dt)

{

using (SqlConnection connection = await ConnectionManager.Instance.OpenConnection())

{

IEnumerable<string> userPermissions = await CheckTablePermissions(dt.TableName, connection);

if (!userPermissions.Except(new[] { "SELECT" }, StringComparer.OrdinalIgnoreCase).Any())

{

MessageBox.Show("У вас нет прав на выполнение операций в данной таблице.");

throw new Exception();

}

using (SqlTransaction transaction = connection.BeginTransaction())

{

if (dt != null)

{

try

{

using (SqlDataAdapter adapter = new SqlDataAdapter($"SELECT \* FROM [{dt.TableName}]", connection))

{

adapter.SelectCommand.Transaction = transaction;

SqlCommandBuilder commandBuilder = new SqlCommandBuilder(adapter);

if (userPermissions.Contains("INSERT", StringComparer.OrdinalIgnoreCase))

adapter.InsertCommand = commandBuilder.GetInsertCommand();

if (userPermissions.Contains("UPDATE", StringComparer.OrdinalIgnoreCase))

adapter.UpdateCommand = commandBuilder.GetUpdateCommand();

if (userPermissions.Contains("DELETE", StringComparer.OrdinalIgnoreCase))

adapter.DeleteCommand = commandBuilder.GetDeleteCommand();

adapter.Update(dt);

transaction.Commit();

}

}

catch

{

transaction.Rollback();

//MessageBox.Show($"Ошибка при обновлении базы данных");

throw;

}

}

}

}

}

public static async Task RollbackTransaction()

{

try

{

await BeginTransaction(oldTable);

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при транзакции: {ex.Message}");

}

}

private static async Task<IEnumerable<string>> CheckTablePermissions(string tableName, SqlConnection connection)

{

var builder = new SqlConnectionStringBuilder(connection.ConnectionString);

string currentUsername = builder.UserID;

if (string.IsNullOrEmpty(currentUsername))

{

MessageBox.Show("Не удалось определить текущего пользователя.");

return Enumerable.Empty<string>();

}

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

try

{

if (await CheckUserRights(connection))

return new List<string> { "SELECT", "INSERT", "UPDATE", "DELETE", "ALTER", "CREATE", "DROP", "EXECUTE", "GRANT", "REFERENCES", "VIEW DEFINITION", /\* и другие операции \*/ };

string getPermissionsQuery = "EXEC GetUserTablePermissions @DatabaseUserName, @TableName";

using (SqlCommand permissionsCommand = new SqlCommand(getPermissionsQuery, connection))

{

permissionsCommand.Parameters.AddWithValue("@DatabaseUserName", currentUsername);

permissionsCommand.Parameters.AddWithValue("@TableName", tableName);

using (SqlDataReader permissionsReader = await permissionsCommand.ExecuteReaderAsync())

{

while (permissionsReader.Read())

{

string permission = permissionsReader.GetString(0);

userPermissions.Add(permission);

}

}

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при проверке прав пользователя: {ex.Message}");

return Enumerable.Empty<string>();

}

return userPermissions;

}

public static string FindMatchingTableName(string columnName, List<string> tableNames, string selectedTableName)

{

string matchingTableName = string.Empty;

double maxSimilarity = 0;

foreach (string tableName in tableNames)

{

// Проверяем, что текущая таблица не совпадает с выбранной таблицей

if (!tableName.Equals(selectedTableName, StringComparison.OrdinalIgnoreCase))

{

double similarity = CalculateSimilarity(columnName, tableName.ToLower());

if (similarity > maxSimilarity)

{

maxSimilarity = similarity;

matchingTableName = tableName;

}

}

}

return matchingTableName;

}

private static int CalculateLevenshteinDistance(string str1, string str2)

{

int[,] matrix = new int[str1.Length + 1, str2.Length + 1];

for (int i = 0; i <= str1.Length; i++)

{

matrix[i, 0] = i;

}

for (int j = 0; j <= str2.Length; j++)

{

matrix[0, j] = j;

}

for (int i = 1; i <= str1.Length; i++)

{

for (int j = 1; j <= str2.Length; j++)

{

int cost = (str1[i - 1] == str2[j - 1]) ? 0 : 1;

matrix[i, j] = Math.Min(

Math.Min(matrix[i - 1, j] + 1, matrix[i, j - 1] + 1),

matrix[i - 1, j - 1] + cost

);

}

}

return matrix[str1.Length, str2.Length];

}

private static double CalculateSimilarity(string str1, string str2)

{

int distance = CalculateLevenshteinDistance(str1, str2);

int maxLength = Math.Max(str1.Length, str2.Length);

if (maxLength == 0)

{

return 1.0; // Специальный случай: две пустые строки считаются полностью схожими

}

else

{

return 1.0 - (double)distance / maxLength;

}

}

public static async Task<bool> CheckUserRights(SqlConnection connection)

{

var builder = new SqlConnectionStringBuilder(connection.ConnectionString);

string userID = builder.UserID;

string getRolesQuery = "EXEC GetUserRoles @LoginName";

using (SqlCommand rolesCommand = new SqlCommand(getRolesQuery, connection))

{

rolesCommand.Parameters.AddWithValue("@LoginName", userID);

using (SqlDataReader rolesReader = await rolesCommand.ExecuteReaderAsync())

{

while (rolesReader.Read())

{

string roleName = rolesReader.GetString(0);

if (Enum.TryParse<DatabaseRoles>(roleName, out DatabaseRoles role))

{

if (role != DatabaseRoles.None)

{

return true;

}

}

}

return false;

}

}

}

public enum DatabaseRoles

{

None,

db\_securityadmin,

db\_owner,

db\_denydatawriter,

db\_denydatareader,

db\_ddladmin,

db\_datawriter,

db\_datareader,

db\_backupoperator,

db\_accessadmin,

db\_executor,

db\_owner\_sid,

db\_securityadmin\_sid,

db\_accessadmin\_sid,

db\_backupoperator\_sid,

db\_ddladmin\_sid,

db\_datareader\_sid,

db\_datawriter\_sid,

db\_denydatareader\_sid

}

}

}

using System;

using System.Data;

using System.Data.SqlClient;

using System.Windows.Forms;

namespace LibA

{

public partial class DynamicLinkerAndReports : Form

{

public int selectedRowCode;

private string sqlcommand;

public DynamicLinkerAndReports()

{

InitializeComponent();

}

public void MakeData(string sqlcommand)

{

uPane.Panel1.Show();

groupBox1.Visible= true;

groupBox1.Dock = DockStyle.Fill;

this.sqlcommand = sqlcommand;

}

public void MakeData(DataTable dt)

{

dataGridViewMain.DataSource = dt;

dataGridViewMain.Visible = true;

dataGridViewMain.RowHeadersVisible = false;

dataGridViewMain.AutoSizeRowsMode = DataGridViewAutoSizeRowsMode.AllCells;

foreach (DataGridViewRow row in dataGridViewMain.Rows)

{

int rowIndex = row.Index;

row.Height = row.GetPreferredHeight(rowIndex, DataGridViewAutoSizeRowMode.AllCells, true);

}

//dataGridViewMain.AutoSizeRowsMode = DataGridViewAutoSizeRowsMode.AllHeaders;

}

private void dataGridViewMain\_CellDoubleClick(object sender, DataGridViewCellEventArgs e)

{

if (e.RowIndex >= 0)

{

object codeValue = dataGridViewMain.Rows[e.RowIndex].Cells["Код"].Value;

if (codeValue != null && int.TryParse(codeValue.ToString(), out int code))

{

selectedRowCode = code;

this.Close();

}

}

}

private async void button1\_Click(object sender, System.EventArgs e)

{

DateTime date1 = dateTimePicker1.Value;

DateTime date2 = dateTimePicker2.Value;

if (date1 <= date2)

{

string formattedDate1 = dateTimePicker1.Value.ToString("yyyy-MM-dd");

string formattedDate2 = dateTimePicker2.Value.ToString("yyyy-MM-dd");

DataTable dt = new();

using (SqlCommand comm = new SqlCommand(this.sqlcommand, await ConnectionManager.Instance.OpenConnection()))

{

comm.Parameters.AddWithValue("@startdate", formattedDate1);

comm.Parameters.AddWithValue("@enddate", formattedDate2);

dt = DBWorker.GetDataTable(comm);

MakeData(dt);

}

}

}

}

}

using System;

using System.Data;

using System.Data.SqlClient;

using System.Net.Sockets;

using System.Runtime.CompilerServices;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Xml;

namespace LibA

{

public partial class SettingsPane : Form

{

private static SettingsPane instance;

private static bool isClosed = true;

public static SettingsPane Instance

{

get

{

if (instance == null || isClosed)

{

instance = new SettingsPane();

instance.FormClosed += (sender, e) =>

{

isClosed = true;

};

isClosed = false;

}

return instance;

}

}

public SettingsPane(){

InitializeComponent();

dataSource.Text = Properties.Settings.Default.dbConnSourceAddr.ToString();

port.Text = Properties.Settings.Default.DBPort;

AcceptButton = saveState;

Focus();

Text = "Настройки";

}

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

{

Properties.Settings.Default.dbConnSourceAddr = dataSource.Text.Replace(',','.').Replace(" ","");

Properties.Settings.Default.DBPort = port.Text;

//Properties.Settings.Default.dbConnSourceInstName = servName.Text;

try

{

using (TcpClient tcpClient = new TcpClient())

{

tcpClient.Connect(Properties.Settings.Default.dbConnSourceAddr, Int32.Parse(port.Text));

tcpClient.Close();

}

Properties.Settings.Default.dbConnSourceAddr = dataSource.Text;

//Properties.Settings.Default.dbConnSourceInstName = servName.Text;

Properties.Settings.Default.Save();

System.Windows.Forms.MessageBox.Show("Данные обновлены");

this.Close();

}

catch

{

System.Windows.Forms.MessageBox.Show("Ошибка!");

dataSource.Text = Properties.Settings.Default.dbConnSourceAddr.ToString();

//servName.Text = Properties.Settings.Default.dbConnSourceInstName.ToString();

}

}

}

}

using System;

using System.Data;

using System.Data.SqlClient;

using System.Drawing;

using System.Windows.Forms;

namespace LibA

{

public partial class UserPanel : Form

{

private AuthForm authForm;

public UserPanel()

{

this.Size = new Size(200, 200);

this.Refresh();

this.AutoSize = false;

InitializeComponent();

Screen screen = Screen.FromControl(this);

this.Location = new Point(screen.WorkingArea.Right - this.Width, screen.WorkingArea.Bottom - this.Height);

ConnectionManager.Instance.Disconnection += (sender, e) =>

{

администрированиеToolStripMenuItem.Visible = false;

};

}

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

{

DialogResult = MessageBox.Show("Developer: Никита Обухов\nemail: nikitoniy2468@gmail.com\nAll rights reserved.", "About", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

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

{

authForm = new AuthForm(this);

authForm.RegAuthSuccess += ChangeStatus;

authForm.WhichWindow(WindowType.REGISTER);

authForm.ShowDialog();

}

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

{

authForm = new AuthForm(this);

authForm.RegAuthSuccess += ChangeStatus;

authForm.HasRights += (sender, e) =>

{

администрированиеToolStripMenuItem.Visible = true;

};

authForm.WhichWindow(WindowType.AUTHORIZE);

authForm.ShowDialog();

}

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

{

Environment.Exit(0);

}

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

{

ConnectionManager.Instance.Disconnect();

this.statText.Text = "";

}

public void newTSSLabel(string ctext)

{

this.statText.Text = ctext;

}

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

{

Program.MakeFocus(SettingsPane.Instance);

}

private void выходToolStripMenuItem1\_Click(object sender, EventArgs e)

{

Close();

Dispose();

}

private void администрированиеToolStripMenuItem\_Click(object sender = null, EventArgs e = null)

{

Program.MakeFocus(AdminPanel.Instance);

}

private async void IsDBAliveTimer\_Tick(object sender, EventArgs e)

{

try

{

bool isDBAlive = await ConnectionManager.CheckDBConnectionAsync();

if (isDBAlive)

{

this.DBStat.Image = Properties.Resources.ok;

this.DBStat.Text = "База данных: ОК";

IsDBAliveTimer.Interval = 60 \* 1000;

}

else

{

this.DBStat.Image = Properties.Resources.error;

this.DBStat.Text = "База данных: ОШИБКА";

IsDBAliveTimer.Interval = 10 \* 1000;

}

}

catch (Exception ex)

{

Console.WriteLine($"Ошибка при проверке базы данных: {ex.Message}");

}

}

private async void ChangeStatus(object sender, EventArgs e)

{

using (SqlConnection c = await ConnectionManager.Instance.OpenConnection())

{

var b = new SqlConnectionStringBuilder(c?.ConnectionString);

statText.Text = $"Добро пожаловать, {b?.UserID}";

}

}

private void UserPanel\_SizeChanged(object sender = null, EventArgs e = null)

{

int leftMargin = (int)(Width \* 0.1);

int topMargin = (int)(Height \* 0.05);

int rightMargin = (int)(Width \* 0.1);

int topMarginWithMenuStrip = topMargin + menuStrip1.Height;

searchPanel.Location = new Point(leftMargin, topMarginWithMenuStrip);

searchPanel.Width = Width - leftMargin - rightMargin;

UpdateSearchPanelLayout();

}

private void searchInput\_Enter(object sender, EventArgs e)

{

if (searchInput.Text == "Начните вводить что-нибудь...")

{

searchInput.Text = "";

searchInput.ForeColor = SystemColors.WindowText;

UpdateSearchPanelLayout();

}

}

private void searchInput\_Leave(object sender, EventArgs e)

{

if (string.IsNullOrWhiteSpace(searchInput.Text))

{

searchInput.Text = "Начните вводить что-нибудь...";

searchInput.ForeColor = SystemColors.GrayText;

UpdateSearchPanelLayout();

}

}

private void UpdateSearchPanelLayout()

{

int panelWidth = searchPanel.Width;

int panelHeight = searchPanel.Height;

int doSearchSize = panelHeight;

doSearch.Size = new Size(doSearchSize, doSearchSize);

doSearch.Location = new Point(panelWidth - doSearchSize, 0);

int searchInputWidth = panelWidth - doSearchSize;

int searchInputHeight = panelHeight;

searchInput.Size = new Size(searchInputWidth, searchInputHeight);

searchInput.Location = new Point(0, 0);

float fontSize = searchInputHeight \* 0.7f;

if (fontSize > searchInputHeight)

fontSize = searchInputHeight;

Font font = new Font("Arial", fontSize);

searchInput.Font = font;

searchInput.TextAlign = HorizontalAlignment.Left;

searchInput.Select(0, 0);

int dataGridViewMargin = 10;

int dataGridViewTopMargin = searchPanel.Bottom + dataGridViewMargin;

dataGridViewMain.Location = new Point(searchPanel.Left, searchPanel.Bottom + 10);

dataGridViewMain.Width = searchPanel.Width;

dataGridViewMain.Height = this.ClientSize.Height - dataGridViewMain.Top - statusStrip1.Height;

}

private async void doSearch\_Click(object sender, EventArgs e)

{

using (SqlCommand command = new SqlCommand("SearchBooks", await ConnectionManager.Instance.OpenConnection()))

{

try

{

command.CommandType = CommandType.StoredProcedure;

if (searchInput.Text == string.Empty || searchInput.Text.Equals("Начните вводить что-нибудь..."))

throw new Exception();

command.Parameters.AddWithValue("@SearchTerm", searchInput.Text);

SqlDataAdapter adapter = new SqlDataAdapter(command);

DataTable dataTable = new DataTable();

adapter.Fill(dataTable);

dataGridViewMain.DataSource = dataTable;

dataGridViewMain.Visible = true;

dataGridViewMain.RowHeadersVisible = false;

dataGridViewMain.Columns["Год выпуска"].AutoSizeMode = DataGridViewAutoSizeColumnMode.AllCells;

dataGridViewMain.Columns["Число страниц"].AutoSizeMode = DataGridViewAutoSizeColumnMode.AllCells;

foreach (DataGridViewColumn column in dataGridViewMain.Columns)

{

if (column.AutoSizeMode != DataGridViewAutoSizeColumnMode.AllCells)

{

column.AutoSizeMode = DataGridViewAutoSizeColumnMode.Fill;

}

}

}

catch

{

dataGridViewMain.Visible = false;

}

}

}

}

}

# ПРИЛОЖЕНИЕ В

## **Код C#, выполняющий логику регистрации и авторизации**

namespace LibA

{

public enum WindowType

{

REGISTER,

AUTHORIZE

}

public partial class AuthForm : Form

{

public event EventHandler RegAuthSuccess;

public event EventHandler HasRights;

//private ProgressBar progressBar;

public AuthForm(UserPanel userPanel)

{

InitializeComponent();

this.AutoSizeMode = AutoSizeMode.GrowAndShrink;

this.AutoSize = true;

}

public void WhichWindow(WindowType winType)

{

this.AutoSize = true;

if (winType == WindowType.REGISTER)

{

this.AcceptButton = button1;

authbox.Visible = false;

regbox.Visible = true;

regbox.Location = new Point(5, 5);

this.Text = "Окно регистрации";

}

else if (winType == WindowType.AUTHORIZE)

{

this.AcceptButton = button2;

regbox.Visible = false;

authbox.Visible = true;

authbox.Location = new Point(5, 5);

this.Text = "Окно авторизации";

}

}

private async void button1\_Click(object sender, EventArgs e)

{

try

{

string transfer = textBox1.Text;

transfer = transfer.Replace(' ', '\_').TrimEnd('\_');

transfer += "\_" + comboBox1.Text;

//AddProgressBarToRegBox();

//await Task.Delay(10000);

string response = await ConnectionManager.Instance.SendRegData(transfer, textBox2.Text, textBox5.Text);

if (response == "200")

{

ConnectionManager.Instance.SetupConnectionString(textBox2.Text, textBox5.Text);

await ConnectionManager.Instance.OpenConnection();

RegAuthSuccess?.Invoke(this, EventArgs.Empty);

}

else

{

throw new Exception($"{response.Split('|')[1]}");

}

}

catch (SqlException er)

{

MessageBox.Show($"Невозможно добавить такого пользователя\n" + er.Message, "RegErr", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

catch (Exception er)

{

MessageBox.Show($"Невозможно добавить такого пользователя\n" + er.Message, "RegErr", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void linkLabel1\_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)

{

WhichWindow(WindowType.AUTHORIZE);

}

private async void button2\_Click(object sender, EventArgs e)

{

try

{

ConnectionManager.Instance.SetupConnectionString(textBox3.Text, textBox4.Text);

using (SqlConnection connection = await ConnectionManager.Instance.OpenConnection())

{

if (connection == null)

throw new Exception();

if (connection.State is ConnectionState.Open)

{

MessageBox.Show("Успешно подключено", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

else throw new Exception();

if (await DBWorker.CheckUserRights(await ConnectionManager.Instance.OpenConnection()))

{

HasRights?.Invoke(this, EventArgs.Empty);

}

RegAuthSuccess?.Invoke(this, EventArgs.Empty);

}

this.Close();

}

catch (Exception ex)

{

//MessageBox.Show("Ошибка подключения: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Warning);

//textBox3.Text = "";

textBox4.Text = "";

textBox3.Focus();

}

}

}

}

# ПРИЛОЖЕНИЕ В

## **Код C#, выполняющий регистрацию пользователя на стороне сервера**

## namespace SqlCreateUser

{

public partial class SqlServerUserRegister : ServiceBase

{

private TcpListener tcpListener;

private CancellationTokenSource cancellationTokenSource;

private const string CRYPTOKEY = "ThisIsASecretKey1234567890123456";

public SqlServerUserRegister()

{

InitializeComponent();

}

protected override void OnStart(string[] args)

{

cancellationTokenSource = new CancellationTokenSource();

tcpListener = new TcpListener(IPAddress.Any, 8888);

tcpListener.Start();

\_ = AcceptConnectionsAsync(cancellationTokenSource.Token);

}

protected override void OnStop()

{

cancellationTokenSource?.Cancel();

tcpListener?.Stop();

}

private async Task AcceptConnectionsAsync(CancellationToken cancellationToken)

{

try

{

while (!cancellationToken.IsCancellationRequested)

{

TcpClient tcpClient = await tcpListener.AcceptTcpClientAsync();

\_ = HandleClientAsync(tcpClient, cancellationToken);

}

}

catch (Exception ex)

{

// Обработка ошибок при остановке службы

Console.WriteLine(ex.Message);

}

}

private async Task HandleClientAsync(TcpClient tcpClient, CancellationToken cancellationToken)

{

string responce = null;

try

{

NetworkStream networkStream = tcpClient.GetStream();

byte[] data = new byte[1024];

int bytesRead = await networkStream.ReadAsync(data, 0, data.Length, cancellationToken);

string receivedData = Encoding.UTF8.GetString(data, 0, bytesRead);

string[] credentials = receivedData.Split(',');

if (credentials.Length == 3)

{

string userName = credentials[0].Trim();

string login = DecryptString(credentials[1].Trim());

string password = DecryptString(credentials[2].Trim());

string connectionString = $"Data Source=localhost;Initial Catalog=Библиотека;Integrated Security=True";

string query = $"CREATE LOGIN {login} WITH PASSWORD = '{password}'; " +

$"CREATE USER {userName} FOR LOGIN {login}; " +

$"GRANT EXECUTE ON [SearchBooks] TO {userName}";

try

{

using (SqlConnection connectionTRUSTABLE = new SqlConnection(connectionString))

{

SqlCommand command = new SqlCommand(query, connectionTRUSTABLE);

connectionTRUSTABLE.Open();

await command.ExecuteNonQueryAsync(cancellationToken);

string[] ui = userName.Split('\_');

command.CommandText = "INSERT INTO Читатель ([Фамилия], [Имя], [Отчество], [Имя для входа]) VALUES (@Фамилия, @Имя, @Отчество, @ИмяДляВхода)";

command.Parameters.Clear();

command.Parameters.AddWithValue("@Фамилия", ui[0]);

command.Parameters.AddWithValue("@Имя", ui[1]);

command.Parameters.AddWithValue("@Отчество", ui[2]);

command.Parameters.AddWithValue("@ИмяДляВхода", login);

await command.ExecuteNonQueryAsync(cancellationToken);

}

}

catch (Exception e)

{

EventLog.WriteEntry("SQLSlujba", "Ошибка в создании записи.\n" + e.Message, EventLogEntryType.Error);

responce = $"500|{e.Message}";

}

if (responce is null)

responce = "200";

EventLog.WriteEntry("SQLSlujba", "Ответ сервера:\n" + responce, EventLogEntryType.Warning);

byte[] responseData = Encoding.UTF8.GetBytes(responce);

await networkStream.WriteAsync(responseData, 0, responseData.Length);

await networkStream.FlushAsync();

networkStream.Close();

tcpClient.Close();

}

}

catch (Exception ex)

{

Console.WriteLine("finally:" + ex.Message);

}

}

public static string EncryptString(string plainText)

{

using (Aes aesAlg = Aes.Create())

{

aesAlg.Key = Encoding.UTF8.GetBytes(CRYPTOKEY);

aesAlg.IV = new byte[aesAlg.BlockSize / 8];

ICryptoTransform encryptor = aesAlg.CreateEncryptor(aesAlg.Key, aesAlg.IV);

using (MemoryStream msEncrypt = new MemoryStream())

{

using (CryptoStream csEncrypt = new CryptoStream(msEncrypt, encryptor, CryptoStreamMode.Write))

using (StreamWriter swEncrypt = new StreamWriter(csEncrypt))

{

swEncrypt.Write(plainText);

}

return Convert.ToBase64String(msEncrypt.ToArray());

}

}

}

public static string DecryptString(string cipherText)

{

using (Aes aesAlg = Aes.Create())

{

aesAlg.Key = Encoding.UTF8.GetBytes(CRYPTOKEY);

aesAlg.IV = new byte[aesAlg.BlockSize / 8];

ICryptoTransform decryptor = aesAlg.CreateDecryptor(aesAlg.Key, aesAlg.IV);

using (MemoryStream msDecrypt = new MemoryStream(Convert.FromBase64String(cipherText)))

{

using (CryptoStream csDecrypt = new CryptoStream(msDecrypt, decryptor, CryptoStreamMode.Read))

using (StreamReader srDecrypt = new StreamReader(csDecrypt))

{

return srDecrypt.ReadToEnd();

}

}

}

}

}

# ПРИЛОЖЕНИЕ Д

## **Результат проверки на плагиат**