|  |
| --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ |
| ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ |
| «Национальный исследовательский ядерный университет «МИФИ» |
| **Технологический институт –** |
| филиал федерального государственного автономного образовательного учреждения высшего |
| образования «Национальный исследовательский ядерный университет «МИФИ» |

**(ТИ НИЯУ МИФИ)**

**Кафедра информационных технологий и прикладной математики**

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе

по дисциплине «Разработка программных модулей»

**Разработка клиент-серверного приложения для автоматизации работы сервисного центра**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | | |  |
| **Руководитель**  преподаватель |  |  |  | А.А.Шляпкин |
| *(должность)* |  | *(подпись)* |  | *(И.О. Фамилия)* |
|  |  |  |  |  |
| **Студент**  ИСП-311 |  |  |  | В.Н.Котельников |
| *(группа)* |  | *(подпись)* |  | *(И.О. Фамилия)* |

Курсовая работа защищена с оценкой «\_\_\_\_\_\_\_\_\_\_\_\_» «\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_20\_\_ г.

Лесной 2024

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc167048855)

[1 Анализ предметной области 4](#_Toc167048856)

[1.1 Постановка задачи 4](#_Toc167048857)

[1.2 Описание бизнес-процесса 5](#_Toc167048858)

[2 Анализ инструментальных средств разработки 6](#_Toc167048859)

[3 Разработка логической модели базы данных 7](#_Toc167048860)

[4 Создание базы данных и разработка клиентского приложения 9](#_Toc167048861)

[4.1 Создание базы данных 9](#_Toc167048862)

[4.2 Взаимосвязь базы данных и приложения 10](#_Toc167048863)

[4.3 Программирование клиентского приложения 10](#_Toc167048864)

[5 Проверка работоспособности приложения 18](#_Toc167048865)

[ЗАКЛЮЧЕНИЕ 28](#_Toc167048866)

[Список использованных источников 29](#_Toc167048867)

[ПРИЛОЖЕНИЕ А (Обязательное). Структура таблиц БД «sc\_x99» 30](#_Toc167048868)

[ПРИЛОЖЕНИЕ Б (Обязательное). Программный код на языке C# 32](#_Toc167048869)

# **ВВЕДЕНИЕ**

Современная информационная технология ставит перед разработчиками задачу создания эффективных и удобных программных модулей, способных решать разнообразные задачи и обеспечивать эффективное взаимодействие с данными. В рамках курсовой работы по модулю "Разработка программных модулей" была поставлена задача разработать базу данных и клиент-серверное приложение, способное обеспечить хранение и обработку информации, а также обеспечить доступ к данным для конечных пользователей.

Целью данного проекта является создание программного решения, которое позволит эффективно управлять данными и обеспечить пользователей надежным и удобным интерфейсом для работы с информацией. В рамках работы была создана база данных, а также разработано клиент-серверное приложение, которое предоставляет доступ к этим данным через сеть.

Для реализации проекта, необходимо выполнить задачи:

* Изучить предметную область;
* Спроектировать базу данных;
* Создать базу данных;
* Разработать клиентскую часть приложения;
* Протестировать приложение.

# **Анализ предметной области**

## **Постановка задачи**

В рамках курсовой работы необходимо разработать систему, которая бы осуществляла:

1. **Учет клиентов и заявок**:
   * Регистрация новых клиентов в системе сервисного центра.
   * Ведение базы данных клиентов с контактной информацией и историей обращений.
   * Создание и учет заявок на обслуживание от клиентов.
2. **Планирование и управление задачами**:
   * Организация и распределение задач между сотрудниками сервисного центра.
   * Отслеживание статуса выполнения задач и сроков их выполнения.
   * Оповещение сотрудников о новых задачах или изменениях в статусе текущих.
3. **Финансовый учет**:
   * Возможность выставления счетов клиентам за оказанные услуги.
   * Возможность изменения цен на услуги.
4. **Аутентификация и безопасность**:
   * Реализация механизмов аутентификации пользователей и контроля доступа к функционалу приложения.
   * Защита конфиденциальной информации клиентов и компании.
5. **Удобный интерфейс пользователя**:
   * Создание интуитивно понятного пользовательского интерфейса для удобного взаимодействия с приложением.
   * Оптимизация процесса ввода данных и выполнения операций для повышения производительности сотрудников.

Необходимо реализовать ролевую модель доступа к приложению, учитывающую возможность работы с ним администраторов, операторов и мастеров.

## **Описание бизнес-процесса**

Цель приложения для сервисного центра - обеспечить эффективное управление процессом обслуживания клиентов, автоматизировать учет заявок, оптимизировать работу персонала и повысить качество обслуживания. Ключевые задачи включают в себя управление клиентами и заявками, планирование работ.

Новые сотрудники сервисного центра добавляются Администратором, указываются их роль, имя, фамилия, телефон, пароль, статус работы. Также, Администратор может менять любые данные работника при необходимости.

Администратор может добавлять новые услуги сервисного центра и менять их цену и название по необходимости.

Кроме того, администратор имеет доступ ко всему функционалу приложения и в определенных случаях может выполнять роль оператора.

Оператор принимает заказы по телефону и назначает мастера для выполнения заказа. В функционал оператора входит добавление новых клиентов, добавление новых заказов и услуг к ним, изменение даты выполнения заказа и передача заказа другому мастеру.

Мастеру передает заказ оператор по телефону или в мессенджере, однако, мастер также имеет доступ к приложению, где он может просматривать заказы и клиентов. Мастер отчитывается оператору о проделанной работе, после чего оператор закрывает дату заказа и добавляет к заказу выполненные оператором услуги.

# **Анализ инструментальных средств разработки**

В данной работе для хранения базы данных была использована база данных MSSQL (Microsoft SQL Server). Эта система управления базами данных (СУБД) предлагает высокую производительность и надежность. Она обладает расширенными возможностями по управлению данными, включая поддержку сложных запросов и масштабируемость. Кроме того, MSSQL имеет интеграцию с другими продуктами Microsoft и обширную документацию, что упрощает разработку и поддержку приложений.

Для данной курсовой работы использовался язык программирования C#. Выбор этого языка обусловлен его широким спектром возможностей и простотой в изучении. C# предоставляет мощные средства для разработки различных типов приложений, включая веб-приложения, настольные приложения и мобильные приложения. Кроме того, C# является основным языком для разработки приложений на платформе .NET, что обеспечивает высокую совместимость и интеграцию с другими технологиями Microsoft.

В качестве среды программирования была выбрана Visual Studio 2022. Это интегрированная среда разработки, которая предоставляет широкий набор инструментов для разработки приложений на языке C#. Visual Studio обладает удобным интерфейсом, обширными возможностями отладки и интегрированным поддержкой различных технологий, что делает ее популярным выбором среди разработчиков.

# **Разработка логической модели базы данных**

Логическая модель данных охватывает все сущности, их атрибуты и связи между ними, отражая бизнес-информацию и правила. По завершении проектирования логической модели она применяется для создания физической модели базы данных.

Исходя из анализа предметной области, можно выделить следующие сущности: Сотрудник, Клиент, Заказ, Услуга.

Логическая модель, созданная для целевой базы данных, соответствует третьей нормальной форме (3НФ) и отвечает следующим требованиям: каждая запись уникальна, имена столбцов (атрибутов) уникальны, отсутствуют составные столбцы, значения атрибутов атомарны, не ключевые поля функционально зависят только от первичного ключа. Полученная логическая модель представлена на Рисунке 1.

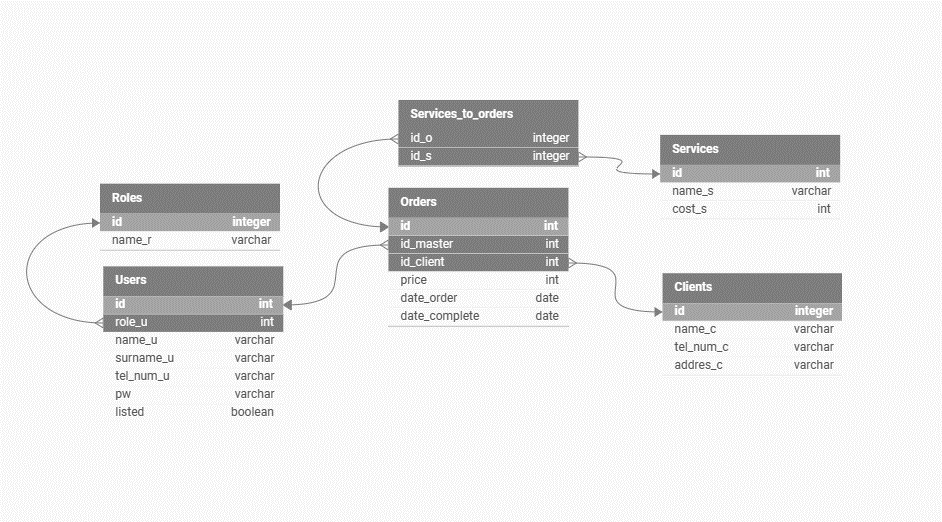


Рисунок 1 – Логическая модель БД «sc\_x99»

# **Создание базы данных и разработка клиентского приложения**

## **Создание базы данных**

После разработки логической модели базы данных на платформе dbdesigner.net и ее последующего развертывания на локальном сервере MSSQL, была создана база данных, включающая 6 таблиц: Clients, Orders, Roles, Services, Services\_to\_Orders, Users, эти таблицы можно увидеть на Рисунке 2.

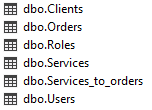


Рисунок 2 – Таблицы БД «sc\_x99»

Таблица Эти таблицы отражают следующую информацию:

* Таблица Clients содержит данные о клиентах, включая их имя, телефон и адрес.
* Таблица Orders содержит информацию о заказах, такую как идентификатор мастера, идентификатор клиента, стоимость, дату размещения заказа и дату выполнения.
* В таблице Roles хранятся роли пользователей.
* Таблица Services содержит информацию о предоставляемых сервисах, таких как название и цена.
* Таблица Services\_to\_Orders является связующим звеном между заказами и услугами.
* Таблица Users содержит данные о сотрудниках сервисного центра, включая их роль, имя, фамилию, телефон, пароль и статус работы.

Кроме того, в системе предусмотрены роли администратора, оператора и мастера, каждая из которых имеет свои ограничения по функционалу.

Структура созданных таблиц представлена в Приложении А (Таблицы А.1-А.6).

Полученная база данных полностью соответствует требованиям.

## **Взаимосвязь базы данных и приложения**

Взаимодействие между базой данных и приложением MSSQL реализуется с использованием базовых компонентов библиотеки System.Data.SqlClient: SqlConnection, SqlCommand, SqlDataAdapter, SqlDataReader. Эти компоненты обеспечивают коммуникацию с базой данных путем использования встроенных функций и свойств, а также динамического изменения соответствующих данных.

## **Программирование клиентского приложения**

После завершения проектирования физической модели базы данных необходимо приступить к разработке клиентского приложения. Этот процесс включает следующие этапы:

1. Создание пользовательского интерфейса:
   * Разработка графического интерфейса, включая формы и компоненты, через которые пользователи будут взаимодействовать с приложением.
2. Программирование функциональности для работы с базой данных:
   * Написание кода для осуществления операций с данными в базе, таких как сортировка, поиск, вставка, удаление и изменение.
   * Разработка элементов управления, обеспечивающих взаимодействие с этой функциональностью.

В результате этих этапов будет создано полнофункциональное клиентское приложение, готовое к использованию пользователями для управления данными.

Внешний вид формы оператора представлен на Рисунке 3.

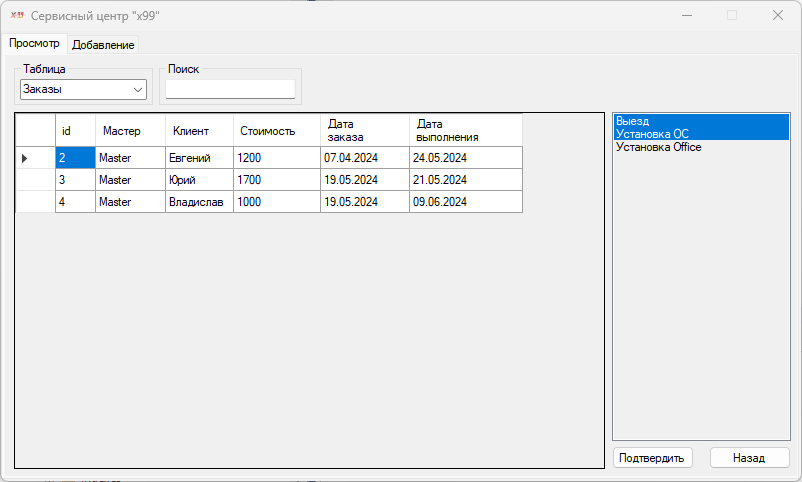


Рисунок 3 – Внешний вид формы оператора

Для оператора предусмотрено добавление данных о заказах и клиентах на второй вкладке формы оператора, которая представлена на Рисунках 4-5.

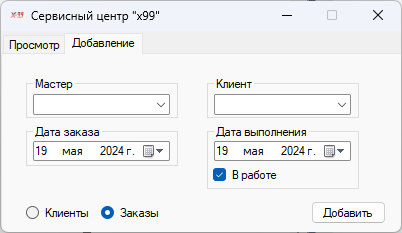


Рисунок 4 – Внешний вид формы добавления заказа

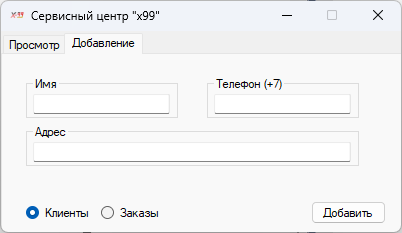


Рисунок 5 – Внешний вид формы добавления клиента

Для оператора предусмотрен функционал изменения даты выполнения заказа и мастера, внешний вид на Рисунке 6.

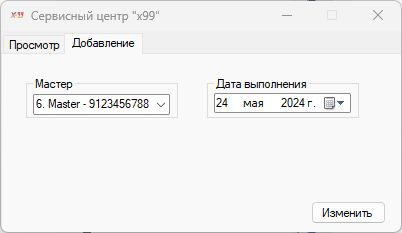


Рисунок 6 – Внешний вид формы изменения даты выполнения заказа и мастера

В приложении возможна только авторизация, регистрация выполняется администратором. Внешний вид формы авторизации на Рисунке 7.

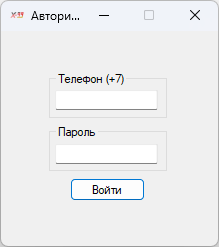


Рисунок 7 – Внешний вид формы авторизации

Для мастера предусмотрен только функционал просмотра клиентов и заказов. Внешний вид на Рисунке 8.

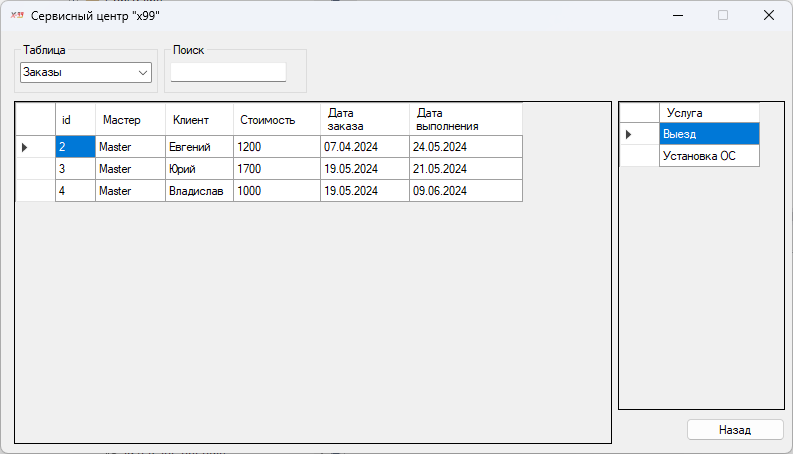


Рисунок 8 – Внешний вид формы мастера

Для администратора тоже предусмотрен функционал просмотра таблиц, но их список расширен до всех таблиц. Внешний вид на Рисунке 9.

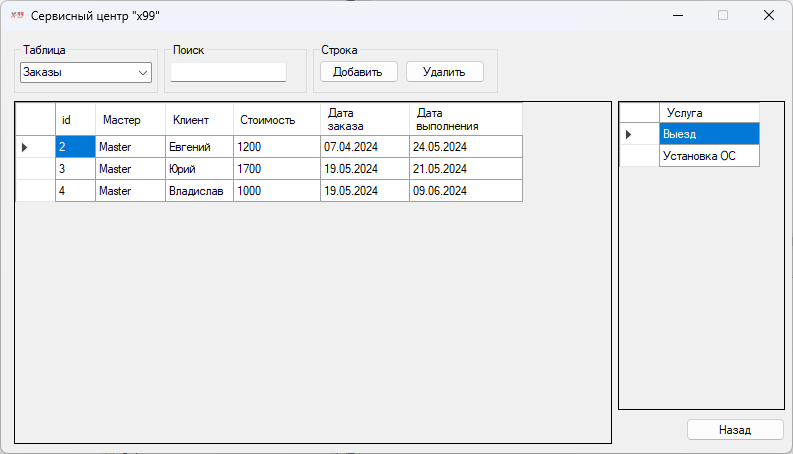


Рисунок 9 – Внешний вид формы администратора

Также, для администратора предусмотрен функционал ручного добавления и изменения значений в таблице. Внешний вид формы добавления схож с внешним видом формы изменения, поэтому на Рисунках 10-15 представлен внешний вид формы изменения.

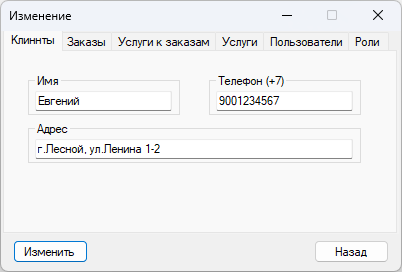


Рисунок 10 – Внешний вид формы изменения клиентов

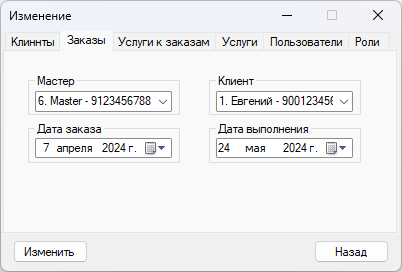


Рисунок 11 – Внешний вид формы изменения заказов

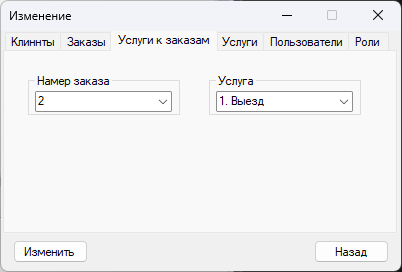


Рисунок 12 – Внешний вид формы изменения услуг к заказам

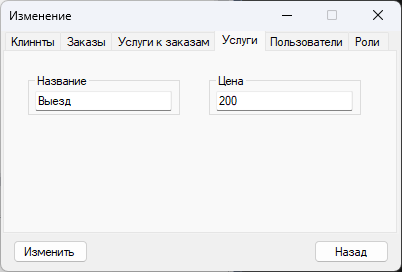


Рисунок 13 – Внешний вид формы изменения услуг

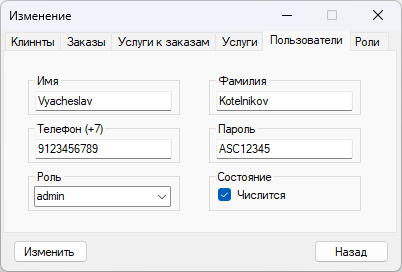


Рисунок 14 – Внешний вид формы изменения пользователей

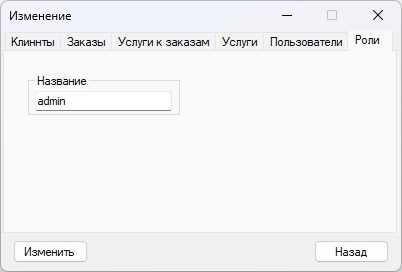


Рисунок 15 – Внешний вид формы изменения ролей

В приложении создано 4 формы:

1. AuthorizationForm – форма авторизации, начальная;
2. MainForm – форма администратора и мастера;
3. OperatorForm – форма оператора;
4. EditAddForm – форма добавления и изменения строк для администратора.

Связи форм представлены на Рисунке 16.

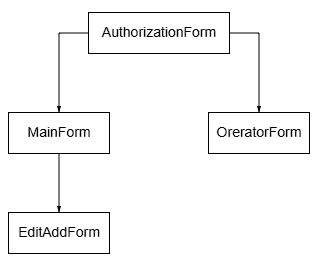


Рисунок 16 – Связи форм

Программный код для всех форм представлен в Приложении Б

# **Проверка работоспособности приложения**

При запуске приложения открывается форма авторизации, где пользователь должен ввести номер телефона и пароль. Внешний вид формы на Рисунке 17.

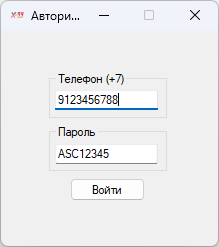


Рисунок 17 – Внешний вид формы авторизации

В случае, когда пользователь с таким логином и паролем не существует, пользователь увидит сообщение, как на Рисунке 18.

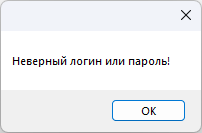


Рисунок 18 – Сообщение о неверном логине или пароле

Если в базе данных есть мастер с таким логином и паролем, пользователь попадет на форму мастера, внешний вид которой на Рисунке 19.

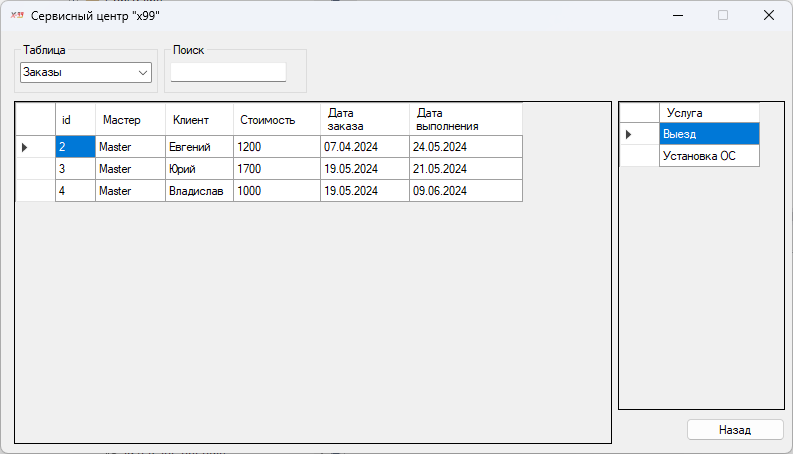


Рисунок 19 – Внешний вид формы мастера

Пользователь на форме мастера может выбирать таблицу для просмотра с помощью выпадающего списка, внешний вид которого представлен на Рисунке 20.

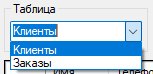


Рисунок 20 – Внешний вид выпадающего списка таблиц мастера

На таблице «Заказы» пользователь увидит все услуги к заказу по клику на любую ячейку таблицы. Внешний вид списка услуг к заказу представлен на Рисунке 21.

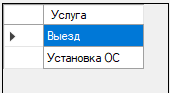


Рисунок 21 – Внешний вид списка услуг к заказу

Также, в текстовое поле поиска пользователь может ввести данные, которые он хочет найти в таблице. Демонстрация функционала на Рисунке 22.

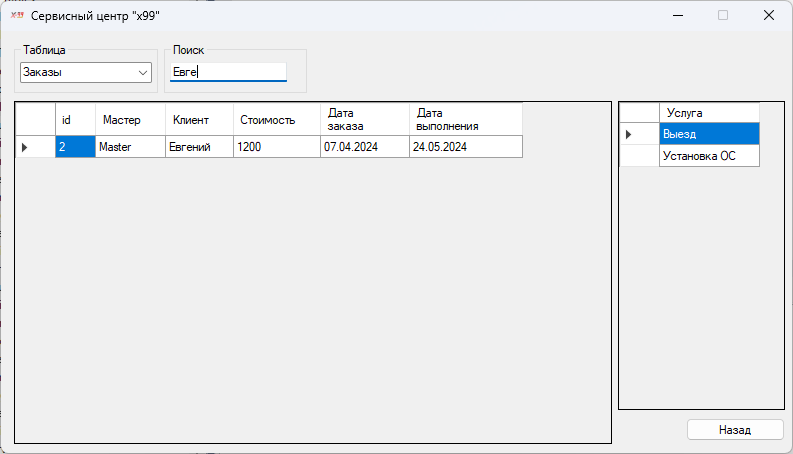


Рисунок 22 – Демонстрация функционала поиска

На этом функционал мастера закончен.

Если пользователь зайдет под логином и паролем оператора, он попадет на форму оператора, внешний вид которой представлен на Рисунке 23.

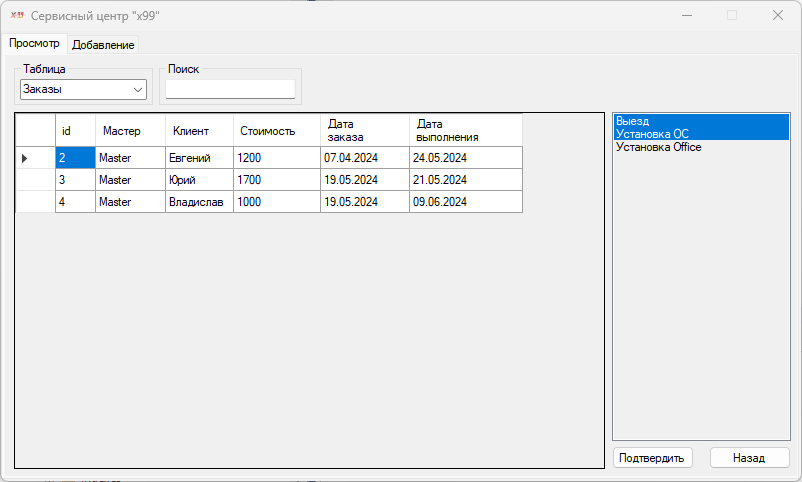


Рисунок 23 – Внешний вид формы оператора

На этой форме пользователь может выбирать таблицу в выпадающем меню, как на Рисунке 20, а также производить поиск по таблице, как на Рисунке 22.

При нажатии на любую ячейку таблицы «Заказы» пользователь может добавить или убрать услугу из заказа, после нажатия кнопки «Подтвердить» его стоимость изменится. Демонстрация этого функционала на Рисунке 23.

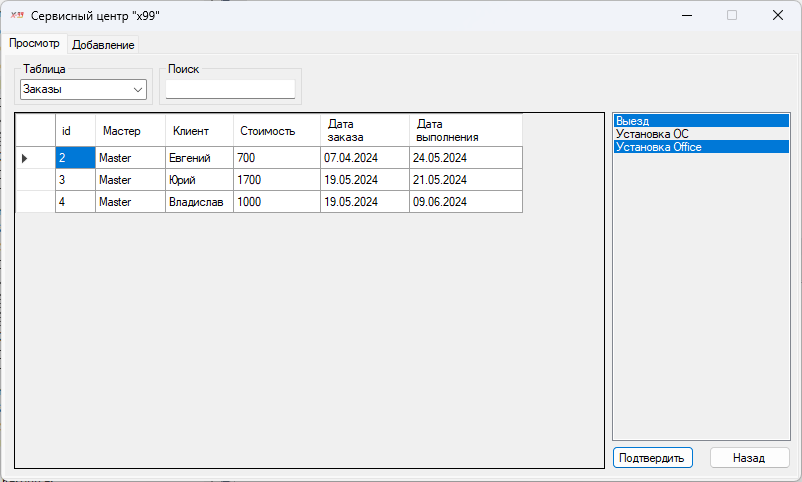


Рисунок 23 – Демонстрация функционала редактирования услуг к заказу

При переходе на вкладку «Добавления» открывается интерфейс добавления клиентов и заказов. Добавление после заполнения всех текстовых полей, выпадающих списков и элементов выбора даты осуществляется путем нажатия кнопки «Добавить». При добавлении заказа предусмотрен функционал, позволяющий по выбору флажка добавить дату выполнения заказа позже. Внешний вид вкладок на рисунках 24-25.

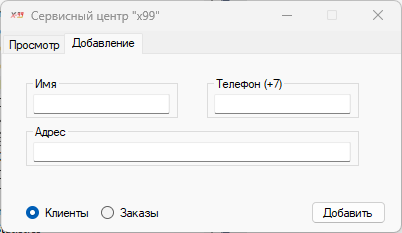


Рисунок 24 – Внешний вид вкладки добавления клиентов

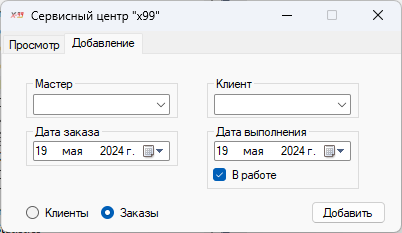


Рисунок 25 – Внешний вид вкладки добавления заказов

После добавления клиента или заказа пользователю предстает форма оператора, где отображена добавленная им запись.

При двойном нажатии на ячейку таблицы «Заказы» открывается вкладка, на которой пользователь может изменить дату выполнения заказа или мастера, заполнив выпадающий список и элемент выбора даты, после чего, нажав кнопку «Изменить». Внешний вид вкладки на Рисунке 26.

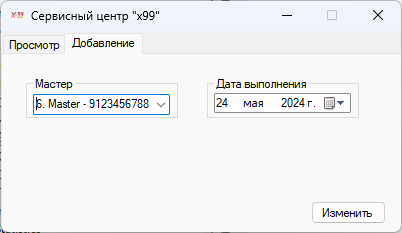


Рисунок 26 – Внешний вид вкладки изменения даты выполнения заказа и мастера

После нажатия кнопки «Изменить» откроется первая вкладка формы, где существующий заказ будет обновлен данными, введенными при изменении.

На этом функционал оператора окончен.

Если пользователь зайдет под логином и паролем администратора, откроется форма администратора, внешний вид которой представлен на Рисунке 27.

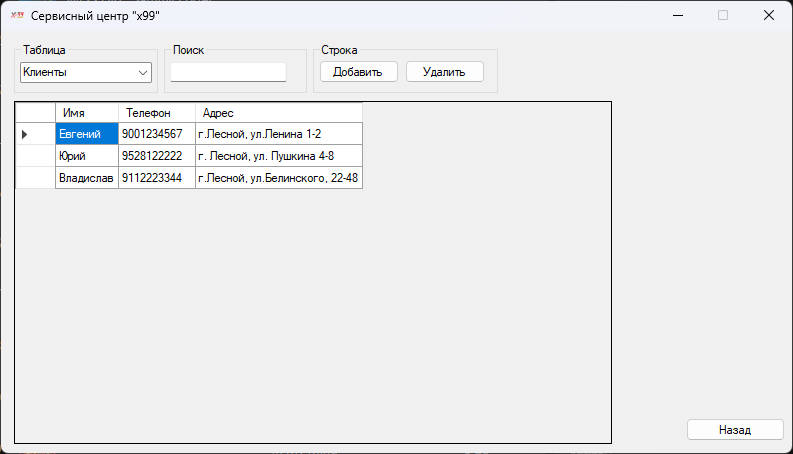


Рисунок 27 – Внешний вид формы администратора

Пользователь может выбирать таблицы через выпадающий список, представленный на рисунке 28.

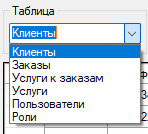


Рисунок 28. Выпадающий список выбора таблицы

Пользователь может произвести поиск, как показано на Рисунке 22.

На таблице «Заказы» пользователь увидит все услуги к заказу по клику на любую ячейку таблицы, как на рисунке 21.

По двойному нажатию на любую таблицу откроется форма изменения записи, как на Рисунках 10-15. На этой форме, заполнив все текстовые поля, выпадающие списки и элементы выбора даты и времени, и нажав на кнопку «Изменить», в таблице изменится запись.

При нажатии на кнопку «Удалить» пользователь увидит окно подтверждения удаления, внешний вид которого представлен на Рисунке 29, и, если пользователь подтвердит удаление, выбранная запись пропадет из таблицы.

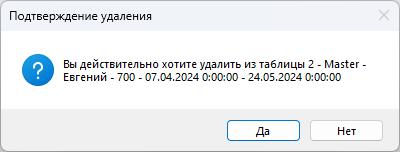


Рисунок 29 – Внешний вид окна подтверждения удаления

При нажатии на кнопку «Добавить» откроется форма добавления записи в выбранную таблицу, внешний вид которой представлен для Рисунках 30-35. На этой форме, заполнив все текстовые поля, выпадающие списки и элементы выбора даты и времени, и нажав на кнопку «Добавить», в таблицу добавиться новая запись.

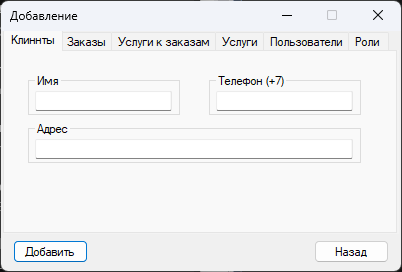


Рисунок 30 – Внешний вид формы добавления клиентов

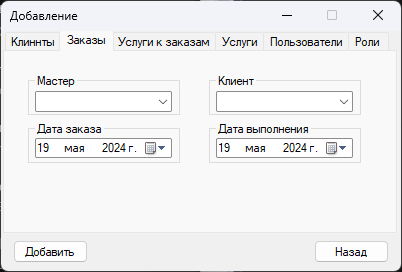


Рисунок 31 – Внешний вид формы добавления заказов

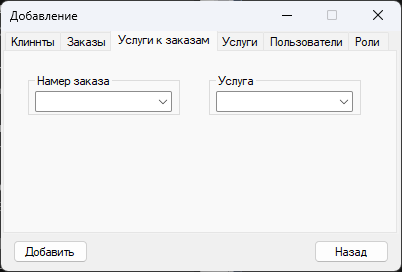


Рисунок 32 – Внешний вид формы добавления услуг к заказам

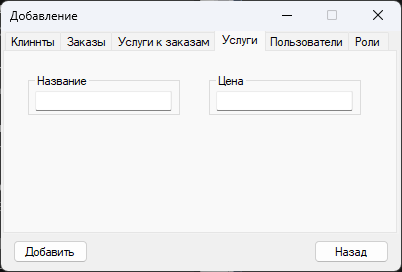


Рисунок 33 – Внешний вид формы добавления услуг

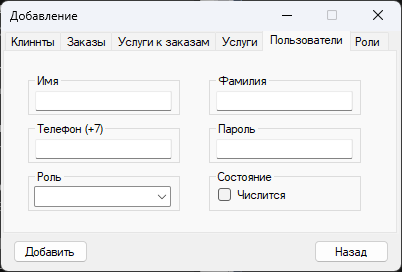


Рисунок 34 – Внешний вид формы добавления пользователей

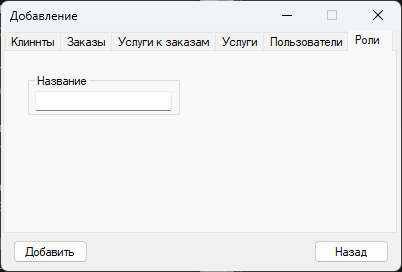


Рисунок 35 – Внешний вид формы добавления ролей

В данной главе проведено тестирование основного функционала приложения: управляющие элементы успешно выполняют назначенные им задачи, запросы выполняются без проблем, приложение не вылетает при ошибках.

# **ЗАКЛЮЧЕНИЕ**

В результате проделанной работы была разработана логическая модель базы данных и создано клиентское приложение на языке C#. Все задачи выполнены успешно, и цель достигнута.

Приложение соответствует предъявленным требованиям и успешно выполняет их. Проверка на работоспособность прошла успешно, не выявлено явных ошибок, что подтверждает готовность приложения к использованию.

В будущем возможно добавление нового функционала в приложение, например, учета деталей, системы скидок для постоянных клиентов и т.п. Структура базы данных создана таким образом, что позволяет легко вносить изменения и дополнять приложение, минимально вмешиваясь в его основные компоненты и базу данных.

# **Список использованных источников**

1. Metanit, URL: <https://metanit.com/sharp/windowsforms/> (07.03.2024)
2. Microsoft Learn, URL: https://learn.microsoft.com/ru-ru/ (01.04.2024)
3. Stack Overflow, URL: <https://stackoverflow.com/> (14.04.2024)
4. Иэн Гриффитс – Программируем на C# 8.0: Питер, 2021, 944стр.

# **ПРИЛОЖЕНИЕ А (Обязательное). Структура таблиц БД «sc\_x99»**

Таблица А.1 – Поля таблицы Clients

|  |  |  |
| --- | --- | --- |
| **Название поля** | **Тип поля** | **Комментарий** |
| Id | int | ID клиента  Первичный ключ |
| name\_c | varchar(15) | Имя |
| tel\_num\_c | varchar(10) | Телефон |
| addres\_c | varchar(50) | Адрес |

Таблица А.2 – Поля таблицы Orders

|  |  |  |
| --- | --- | --- |
| **Название поля** | **Тип поля** | **Комментарий** |
| id | int | ID заказа  Первичный ключ |
| id\_master | int | ID мастера  Внешний ключ (Users.id) |
| id\_client | int | ID клиента  Внешний ключ (Clients.id) |
| price | int | Стоимость |
| date\_order | date | Дата заказа |
| date\_complete | date | Дата выполнения |

Таблица А.3 – Поля таблицы Roles

|  |  |  |
| --- | --- | --- |
| **Название поля** | **Тип поля** | **Комментарий** |
| id | int | ID роли  Первичный ключ |
| name\_r | varchar(5) | Название |

Таблица А.4 – Поля таблицы Services

|  |  |  |
| --- | --- | --- |
| **Название поля** | **Тип поля** | **Комментарий** |
| id | int | ID услуги  Первичный ключ |
| name\_s | varchar(40) | Название |
| cost\_s | int | Цена |

Таблица А.5 – Поля таблицы Services\_to\_orders

|  |  |  |
| --- | --- | --- |
| **Название поля** | **Тип поля** | **Комментарий** |
| id\_o | int | ID заказа  Внешний ключ (Orders.id) |
| id\_s | int | ID услуги  Внешний ключ (Services.id) |

Таблица А.6 – Поля таблицы Users

|  |  |  |
| --- | --- | --- |
| **Название поля** | **Тип поля** | **Комментарий** |
| id | int | ID пользователя  Первичный ключ |
| role\_u | int | Роль  Внешний ключ (Roles.id) |
| name\_u | varchar(15) | Имя |
| surname\_u | varchar(15) | Фамилия |
| tel\_num\_u | varchar(10) | Телефон |
| pw | varchar(8) | Пароль |
| listed | bit | Состояние работы |

# **ПРИЛОЖЕНИЕ Б (Обязательное). Программный код на языке C#**

**Листинг формы авторизации**

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.SqlClient;

using System.Data.Common;

namespace test\_subd

{

public partial class frmAuthorization: Form

{

static SqlConnection connect = new SqlConnection(Properties.Settings.Default.connectionString);

public frmAuthorization()

{

InitializeComponent();

}

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

{}

private void btnConnect\_Click(object sender, EventArgs e) // Войти

{

string login = tbLogin.Text.Trim();

string password = tbPassword.Text.Trim();

try

{

connect.Open();

// Проверка логина и пароля

string query = "SELECT COUNT(\*) FROM Users WHERE tel\_num\_u = @Username AND pw = @Password";

SqlCommand command = new SqlCommand(query, connect);

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

command.Parameters.AddWithValue("@Password", password);

int count = (int) command.ExecuteScalar();

if (count > 0)

{

// Получаем роль пользователя

string roleCheckQuery = "SELECT role\_u FROM Users WHERE tel\_num\_u = @Username AND pw = @Password";

SqlCommand roleCheckCommand = new SqlCommand(roleCheckQuery, connect);

roleCheckCommand.Parameters.AddWithValue("@Username", login);

roleCheckCommand.Parameters.AddWithValue("@Password", password);

// Используем ExecuteScalar для получения единственного значения

object roleObj = roleCheckCommand.ExecuteScalar();

// Проверяем, что значение роли не равно NULL

if (roleObj != DBNull.Value)

{

// Преобразуем значение роли в int

int role = Convert.ToInt32(roleObj);

MessageBox.Show("Авторизация успешна!");

this.Hide();

// Здесь можно перейти к другой форме или выполнить другие действия

if (role == 2)

{

Form4 fm = new Form4(connect);

fm.ShowDialog();

}

else

{

Form2 fm = new Form2(connect, role);

fm.ShowDialog();

}

}

else

{

MessageBox.Show("Роль пользователя не найдена!");

}

}

else

{

MessageBox.Show("Неверный логин или пароль!");

}

}

catch (Exception ex)

{

MessageBox.Show("Ошибка: " + ex.Message);

}

finally

{

connect.Close();

}

}

}

}

**Листинг формы администратора и мастера**

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.SqlClient;

using System.Runtime.Remoting.Contexts;

using Microsoft.SqlServer.Server;

using static System.Windows.Forms.DataFormats;

using System.Data.Common;

using static System.Windows.Forms.VisualStyles.VisualStyleElement;

using System.Reflection.Emit;

namespace test\_subd

{

public partial class Form2: Form

{

SqlConnection connect;

int roleForm;

public Form2(SqlConnection cnct, int rl)

{

connect = cnct;

roleForm = rl;

InitializeComponent();

}

private void Form2\_Load(object sender, EventArgs e)

{

try

{

dataGridView1.AllowUserToAddRows = false;

dataGridView2.AllowUserToAddRows = false;

// Добавляем обработчик события CellDoubleClick

dataGridView1.CellDoubleClick += dataGridView1\_CellDoubleClick;

dataGridView1.CellClick += dataGridView1\_CellClick;

if (roleForm != 1)

{

groupBox3.Visible = false;

comboBox1.Items.RemoveAt(3);

comboBox1.Items.RemoveAt(3);

comboBox1.Items.RemoveAt(3);

comboBox1.Items.RemoveAt(2);

}

SqlConnection sqlConnect = new SqlConnection(Properties.Settings.Default.connectionString);

sqlConnect.Open();

SqlCommand showTable = new SqlCommand();

showTable.CommandText = $ "SELECT \* FROM Клиенты";

showTable.Connection = sqlConnect;

// SqlAdapter - прослойка между источником данных и базой данных

SqlDataAdapter adapter = new SqlDataAdapter(showTable);

DataSet dataSet = new DataSet();

// заполняем источник данных полученными из адаптера записями

adapter.Fill(dataSet);

dataGridView1.DataSource = dataSet.Tables[0];

dataGridView1.Columns[0].Visible = false;

sqlConnect.Close();

UpdatePrice();

dataGridView1.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.AllCells;

}

catch (Exception ex)

{

MessageBox.Show("Ошибка: " + ex.Message);

}

}

private void label2\_Click(object sender, EventArgs e)

{}

private void UpdatePrice()

{

// Создаем SQL-запрос

string sql = @ "

UPDATE Orders

SET Price = (SELECT SUM(s.cost\_s) FROM Services\_to\_orders so INNER JOIN Services s ON so.id\_s = s.ID WHERE so.id\_o = Orders.ID)

";

// Создаем команду и связываем ее с подключением и SQL-запросом

using(SqlCommand command = new SqlCommand(sql, connect))

{

// Выполняем SQL-запрос

int rowsAffected = command.ExecuteNonQuery();

// Выводим информацию о количестве обновленных строк

Console.WriteLine("Обновлено строк: " + rowsAffected);

}

}

public void LoadDataIntoDataGridView()

{

try

{

SqlConnection sqlConnect = new SqlConnection(Properties.Settings.Default.connectionString);

SqlCommand logRequst = new SqlCommand();

UpdatePrice();

switch (comboBox1.SelectedIndex)

{

case 0:

logRequst.CommandText = $ "SELECT \* FROM Клиенты";

break;

case 1:

logRequst.CommandText = $ "SELECT \* FROM Заказы";

break;

case 2:

logRequst.CommandText = $ "SELECT \* FROM Услугикзаказам";

break;

case 3:

logRequst.CommandText = $ "SELECT \* FROM Услуги";

break;

case 4:

logRequst.CommandText = $ "SELECT \* FROM Пользователи";

break;

case 5:

logRequst.CommandText = $ "SELECT \* FROM Роли";

break;

}

logRequst.Connection = sqlConnect;

// SqlAdapter - прослойка между источником данных и базой данных

SqlDataAdapter adapter = new SqlDataAdapter(logRequst);

DataSet dataSet = new DataSet();

// заполняем источник данных полученными из адаптера записями

adapter.Fill(dataSet);

dataGridView1.DataSource = dataSet.Tables[0];

tbDataSearch.Clear();

if (comboBox1.SelectedItem != "Заказы")

{

dataGridView2.Visible = false;

if (comboBox1.SelectedItem != "Услуги к заказам")

{

dataGridView1.Columns[0].Visible = false;

}

}

else

{

dataGridView2.Visible = true;

dataGridView1.Columns[0].Visible = true;

}

dataGridView1.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.AllCells;

sqlConnect.Close();

}

catch (Exception ex)

{

MessageBox.Show("Ошибка: " + ex.Message);

}

}

private void comboBox1\_SelectedIndexChanged(object sender, EventArgs e)

{

LoadDataIntoDataGridView();

}

private void dataGridView1\_CellClick(object sender, DataGridViewCellEventArgs e)

{

try

{

if (comboBox1.SelectedItem == "Заказы")

{

SqlConnection sqlConnect = new SqlConnection(Properties.Settings.Default.connectionString);

SqlCommand logRequst1 = new SqlCommand();

dataGridView2.Visible = true;

dataGridView1.Columns[0].Visible = true;

DataGridViewRow selectedRow = dataGridView1.CurrentRow;

logRequst1.CommandText = $ "SELECT name\_s AS 'Услуга' FROM Services\_to\_orders JOIN Services ON Services\_to\_orders.id\_s = Services.id WHERE id\_o = {selectedRow.Cells[0].Value}";

logRequst1.Connection = sqlConnect;

// SqlAdapter - прослойка между источником данных и базой данных

SqlDataAdapter adapter1 = new SqlDataAdapter(logRequst1);

DataSet dataSet1 = new DataSet();

// заполняем источник данных полученными из адаптера записями

adapter1.Fill(dataSet1);

dataGridView2.DataSource = dataSet1.Tables[0];

}

}

catch (Exception ex)

{

MessageBox.Show("Ошибка: " + ex.Message);

}

}

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

{

this.Close();

Form mainForm = Application.OpenForms["frmAuthorization"];

mainForm.Show();

}

private void label4\_Click(object sender, EventArgs e)

{}

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

{

try

{

int boxItem = comboBox1.SelectedIndex;

// Переменная для определеня назначения Form3

bool typeform = true;

this.Hide();

Form3 fm = new Form3(connect, boxItem, roleForm, typeform, "0", "0");

fm.Text = "Добавление";

fm.ShowDialog();

//connect.Close();

}

catch (Exception ex)

{

MessageBox.Show("Ошибка: " + ex.Message);

}

}

private void dataGridView1\_CellDoubleClick(object sender, DataGridViewCellEventArgs e)

{

try

{

if (roleForm == 1)

{

int boxItem = comboBox1.SelectedIndex;

// Переменная для определения назначения Form3

bool typeform = false;

this.Hide();

// Проверяем, что индекс строки действителен

if (e.RowIndex >= 0 && e.RowIndex < dataGridView1.Rows.Count)

{

DataGridViewRow selectedRow = dataGridView1.CurrentRow;

// Получаем значения из нужных столбцов по индексу столбца

string selID = selectedRow.Cells[0].Value.ToString();

string selIDs = selectedRow.Cells[1].Value.ToString();

// Создаем экземпляр Form3 и передаем необходимые значения

Form3 fm = new Form3(connect, boxItem, roleForm, typeform, selID, selIDs);

fm.Text = "Изменение";

fm.ShowDialog();

}

}

}

catch (Exception ex)

{

MessageBox.Show("Ошибка: " + ex.Message);

}

}

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

{

try

{

// Получаем выбранную строку

DataGridViewRow selectedRow = dataGridView1.CurrentRow;

// Формируем текст для подтверждения удаления

string confirmationText = $ "Вы действительно хотите удалить из таблицы ";

// Получаем значения всех столбцов выбранной строки и добавляем их к тексту подтверждения

for (int i = 0; i < selectedRow.Cells.Count; i++)

{

confirmationText += $ "{selectedRow.Cells[i].Value.ToString()}";

if (i < selectedRow.Cells.Count - 1) confirmationText += " - ";

}

// Показываем окно подтверждения

DialogResult result = MessageBox.Show(confirmationText, "Подтверждение удаления", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

// Если пользователь подтвердил удаление, выполняем операцию

if (result == DialogResult.Yes)

{

// Устанавливаем полученное значение в Label

string id = selectedRow.Cells[0].Value.ToString();

string id\_s = selectedRow.Cells[1].Value.ToString();

SqlCommand logRequest = new SqlCommand();

logRequest.Connection = connect;

switch (comboBox1.SelectedIndex)

{

case 0:

logRequest.CommandText = $ "DELETE FROM Clients WHERE ID = {id}";

break;

case 1:

logRequest.CommandText = $ "DELETE FROM Orsers WHERE ID = {id}";

break;

case 2:

// Получаем название услуги из выбранной строки DataGridView

string serviceName = dataGridView1.CurrentRow.Cells[1].Value.ToString();

// Создаем SQL-запрос для получения ID услуги по названию услуги

string sql = "SELECT ID FROM Services WHERE Name\_s = @ServiceName";

// Создаем команду SQL с параметром

using(SqlCommand cmd = new SqlCommand(sql, connect))

{

// Добавляем параметр @ServiceName и устанавливаем его значение

cmd.Parameters.AddWithValue("@ServiceName", serviceName);

// Открываем соединение с базой данных

//connect.Open();

// Выполняем запрос и получаем ID услуги

int serviceId = (int) cmd.ExecuteScalar();

logRequest.CommandText = $ "DELETE FROM Services\_to\_orders WHERE ID\_O = {id} AND ID\_S = {serviceId}";

// Закрываем соединение

//connect.Close();

}

break;

case 3:

logRequest.CommandText = $ "DELETE FROM Services WHERE ID = {id}";

break;

case 4:

logRequest.CommandText = $ "DELETE FROM Users WHERE ID = {id}";

break;

case 5:

logRequest.CommandText = $ "DELETE FROM Roles WHERE ID = {id}";

break;

}

logRequest.ExecuteNonQuery();

}

UpdatePrice();

LoadDataIntoDataGridView();

}

catch (Exception ex)

{

MessageBox.Show("Ошибка: " + ex.Message);

}

}

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

{

try

{

// Получаем названия всех столбцов DGV1

var columnNames = dataGridView1.Columns.Cast < DataGridViewColumn > ().Select(x => x.HeaderText).ToList();

// Создаем строку фильтрации

string filterExpression = string.Empty;

foreach(var columnName in columnNames)

{

if (!string.IsNullOrEmpty(filterExpression)) filterExpression += " OR ";

filterExpression += $ "CONVERT([{columnName}], 'System.String') LIKE '%{tbDataSearch.Text}%'";

}

// Применяем фильтр к строкам DGV1

(dataGridView1.DataSource as DataTable).DefaultView.RowFilter = filterExpression;

}

catch (Exception ex)

{

MessageBox.Show("Ошибка: " + ex.Message);

}

}

}

}

**Листинг формы изменения и добавления**

using System;

using System.Collections;

using System.Collections.Generic;

using System.Data;

using System.Data.SqlClient;

using System.Linq;

using System.Reflection;

using System.Windows.Forms;

using static System.Windows.Forms.VisualStyles.VisualStyleElement;

using static System.Windows.Forms.VisualStyles.VisualStyleElement.Button;

namespace test\_subd

{

public partial class Form3: Form

{

private SqlConnection connect;

private int selectedTable;

private int rlu;

private bool tfAdd;

private int selectedId;

private string selectedIds;

public Form3(SqlConnection cnct, int boxItem, int role, bool typeForm, string selID, string selIDs)

{

connect = cnct;;

InitializeComponent();

rlu = role;

selectedTable = boxItem;

tfAdd = typeForm;

selectedId = Convert.ToInt32(selID);

selectedIds = selIDs;

}

private void Form3\_Load(object sender, EventArgs e)

{

try

{

if (tfAdd == false)

{

button1.Text = "Изменить";

//Заполняем вкладки для изменения

switch (selectedTable)

{

case 0:

using(SqlCommand command = new SqlCommand())

{

command.Connection = connect;

command.CommandText = $ "SELECT name\_c, tel\_num\_c, addres\_c FROM Clients WHERE id = @selectedId";

command.Parameters.AddWithValue("@selectedId", selectedId);

using(SqlDataReader reader = command.ExecuteReader())

{

if (reader.Read())

{

// Заполняем элементы формы данными из базы данных

tbClientsNane\_c.Text = reader["name\_c"].ToString();

tbClientsTel\_num\_c.Text = reader["tel\_num\_c"].ToString();

tbClientsAdress\_c.Text = reader["addres\_c"].ToString();

}

}

}

break;

case 1:

string queryOrders = $ "SELECT id\_master, id\_client, date\_order, date\_complete FROM Orders WHERE id = @selectedId";

using(SqlCommand command = new SqlCommand(queryOrders, connect))

{

command.Parameters.AddWithValue("@selectedId", selectedId);

using(SqlDataReader reader = command.ExecuteReader())

{

if (reader.Read())

{

int id\_master = reader.GetInt32(0);

int id\_client = reader.GetInt32(1);

DateTime date\_order = reader.GetDateTime(2);

DateTime date\_complete = reader.GetDateTime(3);

// Закрыть текущий DataReader

reader.Close();

// Заполнение элементов формы из базы данных для мастера

string queryMaster = $ "SELECT name\_u, tel\_num\_u FROM Users WHERE id = @id\_master";

using(SqlCommand masterCommand = new SqlCommand(queryMaster, connect))

{

masterCommand.Parameters.AddWithValue("@id\_master", id\_master);

using(SqlDataReader masterReader = masterCommand.ExecuteReader())

{

if (masterReader.Read())

{

string name\_master = masterReader.GetString(0);

string telNum\_master = masterReader.GetString(1);

cbOrdersId\_maser.Text = $ "{id\_master}. {name\_master} - {telNum\_master}";

}

}

}

// Заполнение элементов формы из базы данных для клиента

string queryClient = $ "SELECT name\_c, tel\_num\_c FROM Clients WHERE id = @id\_client";

using(SqlCommand clientCommand = new SqlCommand(queryClient, connect))

{

clientCommand.Parameters.AddWithValue("@id\_client", id\_client);

using(SqlDataReader clientReader = clientCommand.ExecuteReader())

{

if (clientReader.Read())

{

string name\_client = clientReader.GetString(0);

string telNum\_client = clientReader.GetString(1);

cbOrdersId\_client.Text = $ "{id\_client}. {name\_client} - {telNum\_client}";

}

}

}

// Заполнение даты из базы данных

dtpOrdersDate\_order.Value = date\_order;

dtpOrdersDate\_complete.Value = date\_complete;

}

}

}

break;

case 2:

string querySelectedOrder = "SELECT id\_o, id\_s FROM Services\_to\_orders " + "INNER JOIN Services ON Services\_to\_orders.id\_s = Services.id " + "WHERE id\_o = @selectedId\_o AND name\_s = @selectedName\_s";

using(SqlCommand command = new SqlCommand(querySelectedOrder, connect))

{

command.Parameters.AddWithValue("@selectedId\_o", selectedId);

command.Parameters.AddWithValue("@selectedName\_s", selectedIds);

using(SqlDataReader reader = command.ExecuteReader())

{

if (reader.Read())

{

int id\_o = reader.GetInt32(0);

int id\_s = reader.GetInt32(1);

// Закрыть текущий DataReader

reader.Close();

// Заполнение элементов формы из базы данных для id\_o

cbS\_t\_oId\_o.Text = id\_o.ToString();

// Заполнение элементов формы из базы данных для id\_s

cbS\_t\_oId\_s.Text = $ "{id\_s}. {selectedIds}";

}

}

}

break;

case 3:

using(SqlCommand command = new SqlCommand())

{

command.Connection = connect;

command.CommandText = $ "SELECT name\_s, cost\_s FROM Services WHERE id = @selectedId";

command.Parameters.AddWithValue("@selectedId", selectedId);

using(SqlDataReader reader = command.ExecuteReader())

{

if (reader.Read())

{

// Заполняем элементы формы данными из базы данных

tbServicesName\_s.Text = reader["name\_s"].ToString();

tbServicesCost\_s.Text = reader["cost\_s"].ToString();

}

}

}

break;

case 4:

// SQL-запрос для выборки данных пользователя по его id

string query = "SELECT roles.name\_r, name\_u, surname\_u, tel\_num\_u, pw, listed FROM Users " + "INNER JOIN roles ON Users.role\_u = roles.id " + "WHERE Users.id = @selectedId";

using(SqlCommand command = new SqlCommand(query, connect))

{

command.Parameters.AddWithValue("@selectedId", selectedId);

using(SqlDataReader reader = command.ExecuteReader())

{

if (reader.HasRows)

{

while (reader.Read())

{

cbUsersRole\_u.Text = reader["name\_r"].ToString(); // Обратите внимание на изменение здесь

tbUsersName\_u.Text = reader["name\_u"].ToString();

tbUsersSurname\_u.Text = reader["surname\_u"].ToString();

tbUsersTel\_num\_u.Text = reader["tel\_num\_u"].ToString();

tbUsersPw.Text = reader["pw"].ToString();

chbUsersListed.Checked = Convert.ToBoolean(reader["listed"]);

}

}

else

{

Console.WriteLine("No rows found.");

}

}

}

break;

case 5:

using(SqlCommand command = new SqlCommand())

{

command.Connection = connect;

command.CommandText = $ "SELECT name\_r FROM Roles WHERE id = @selectedId";

command.Parameters.AddWithValue("@selectedId", selectedId);

using(SqlDataReader reader = command.ExecuteReader())

{

if (reader.Read())

{

// Заполняем элементы формы данными из базы данных

tbRolesName\_r.Text = reader["name\_r"].ToString();

}

}

}

break;

}

}

else

{

button1.Text = "Добавить";

}

if (rlu != 1)

{

tabControl1.TabPages[3].Hide();

tabControl1.TabPages[4].Hide();

tabControl1.TabPages[5].Hide();

}

switch (selectedTable)

{

case 0:

tabControl1.SelectedIndex = selectedTable;

break;

case 1:

tabControl1.SelectedIndex = selectedTable;

// Создаем словарь для хранения значений из таблицы Users

Dictionary < int, Tuple < string, string >> usersDictionary = new Dictionary < int, Tuple < string, string >> ();

string queryUsers = "SELECT id, name\_u, tel\_num\_u FROM Users";

using(SqlCommand command = new SqlCommand(queryUsers, connect))

{

// Выполняем запрос и получаем результаты

using(SqlDataReader reader = command.ExecuteReader())

{

// Перебираем строки результата

while (reader.Read())

{

// Получаем значения столбцов id, name\_u и tel\_num\_u

int id = reader.GetInt32(0);

string name = reader.GetString(1);

string telNum = reader.GetString(2);

// Добавляем значения в словарь в виде кортежа

usersDictionary.Add(id, Tuple.Create(name, telNum));

}

}

}

foreach(var kvp in usersDictionary)

{

cbOrdersId\_maser.Items.Add($ "{kvp.Key}. {kvp.Value.Item1} - {kvp.Value.Item2}");

}

// Создаем словарь для хранения значений из таблицы Clients

Dictionary < int, Tuple < string, string >> clientsDictionary = new Dictionary < int, Tuple < string, string >> ();

string queryClients = "SELECT id, name\_c, tel\_num\_c FROM Clients";

using(SqlCommand command = new SqlCommand(queryClients, connect))

{

// Выполняем запрос и получаем результаты

using(SqlDataReader reader = command.ExecuteReader())

{

// Перебираем строки результата

while (reader.Read())

{

// Получаем значения столбцов id, name\_c и tel\_num\_c

int id = reader.GetInt32(0);

string name = reader.GetString(1);

string telNum = reader.GetString(2);

// Добавляем значения в словарь в виде кортежа

clientsDictionary.Add(id, Tuple.Create(name, telNum));

}

}

}

foreach(var kvp in clientsDictionary)

{

cbOrdersId\_client.Items.Add($ "{kvp.Key}. {kvp.Value.Item1} - {kvp.Value.Item2}");

}

break;

case 2:

tabControl1.SelectedIndex = selectedTable;

// Создаем словарь для хранения значений из таблицы Services

Dictionary < int, string > servicesDictionary = new Dictionary < int, string > ();

// Создаем команду для выполнения запроса к таблице Services

string queryServices = "SELECT id, name\_s FROM Services";

using(SqlCommand command = new SqlCommand(queryServices, connect))

{

// Выполняем запрос и получаем результаты

using(SqlDataReader reader = command.ExecuteReader())

{

// Перебираем строки результата

while (reader.Read())

{

// Получаем значения столбцов id и name\_s

int id = reader.GetInt32(0);

string name\_s = reader.GetString(1);

// Добавляем значения в словарь

servicesDictionary.Add(id, name\_s);

}

}

}

// Выводим содержимое словаря в cbS\_t\_oId\_s

foreach(var kvp in servicesDictionary)

{

cbS\_t\_oId\_s.Items.Add($ "{kvp.Key}. {kvp.Value}");

}

// Создаем список для хранения id заказов

List < int > ordersList = new List < int > ();

// Создаем команду для выполнения запроса к таблице Orders

string query = "SELECT id FROM Orders";

using(SqlCommand command = new SqlCommand(query, connect))

{

// Выполняем запрос и получаем результаты

using(SqlDataReader reader = command.ExecuteReader())

{

// Перебираем строки результата и добавляем id заказов в список

while (reader.Read())

{

int orderId = reader.GetInt32(0);

ordersList.Add(orderId);

}

}

}

// Добавляем элементы списка в ComboBox

foreach(int orderId in ordersList)

{

cbS\_t\_oId\_o.Items.Add(orderId);

}

break;

case 3:

tabControl1.SelectedIndex = selectedTable;

break;

case 4:

tabControl1.SelectedIndex = selectedTable;

// Создаем словарь для хранения значений из таблицы Roles

Dictionary < int, string > rolesDictionary = new Dictionary < int, string > ();

// Создаем команду для выполнения запроса к таблице Roles

string queryRoles = "SELECT id, name\_r FROM Roles";

using(SqlCommand command = new SqlCommand(queryRoles, connect))

{

// Выполняем запрос и получаем результаты

using(SqlDataReader reader = command.ExecuteReader())

{

// Перебираем строки результата

while (reader.Read())

{

// Получаем значения столбцов id и name\_r

int id = reader.GetInt32(0);

string name\_r = reader.GetString(1);

// Добавляем значения в словарь

rolesDictionary.Add(id, name\_r);

}

}

}

// Выводим содержимое словаря в cbUsersRole\_u

foreach(var kvp in rolesDictionary)

{

cbUsersRole\_u.Items.Add($ "{kvp.Key}. {kvp.Value}");

}

break;

case 5:

tabControl1.SelectedIndex = selectedTable;

break;

}

for (int i = 0; i < tabControl1.TabCount; i++)

{

// Устанавливаем Enabled = false для всех вкладок, кроме выбранной

if (i != selectedTable)

{

tabControl1.TabPages[i].Enabled = false;

}

}

}

catch (Exception ex)

{

MessageBox.Show("Ошибка: " + ex.Message);

}

}

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

{

try

{

using(SqlConnection conn = new SqlConnection(Properties.Settings.Default.connectionString))

{

conn.Open();

SqlCommand logRequest = new SqlCommand();

logRequest.Connection = conn;

switch (selectedTable)

{

case 0:

if (tfAdd == true)

{

logRequest.CommandText = $ "INSERT INTO Clients (name\_c, tel\_num\_c, addres\_c) " + $ "VALUES ('{tbClientsNane\_c.Text}', '{tbClientsTel\_num\_c.Text}', '{tbClientsAdress\_c.Text}')";

MessageBox.Show("Строка добавлена");

}

else

{

logRequest.CommandText = $ "UPDATE Clients SET name\_c = '{tbClientsNane\_c.Text}', tel\_num\_c = '{tbClientsTel\_num\_c.Text}', addres\_c = '{tbClientsAdress\_c.Text}' WHERE id = {selectedId}";

MessageBox.Show("Строка изменена");

}

break;

case 1:

if (tfAdd == true)

{

int indexOfDotMaster = cbOrdersId\_maser.Text.IndexOf('.');

string id\_master = cbOrdersId\_maser.Text.Substring(0, indexOfDotMaster);

int indexOfDotClient = cbOrdersId\_client.Text.IndexOf('.');

string id\_client = cbOrdersId\_client.Text.Substring(0, indexOfDotClient);

logRequest.CommandText = $ "INSERT INTO Orders (id\_master, id\_client, date\_order, date\_complete) VALUES ({id\_master}, {id\_client}, " + $ "'{dtpOrdersDate\_order.Value.ToString("

yyyy - MM - dd ")}', '{dtpOrdersDate\_complete.Value.ToString("

yyyy - MM - dd ")}')";

MessageBox.Show("Строка добавлена");

}

else

{

int indexOfDotMaster = cbOrdersId\_maser.Text.IndexOf('.');

string id\_master = cbOrdersId\_maser.Text.Substring(0, indexOfDotMaster);

int indexOfDotClient = cbOrdersId\_client.Text.IndexOf('.');

string id\_client = cbOrdersId\_client.Text.Substring(0, indexOfDotClient);

logRequest.CommandText = $ "UPDATE Orders SET id\_master = {id\_master}, id\_client = {id\_client}, " + $ "date\_order = '{dtpOrdersDate\_order.Value.ToString("

yyyy - MM - dd ")}', " + $ "date\_complete = '{dtpOrdersDate\_complete.Value.ToString("

yyyy - MM - dd ")}' " + $ "WHERE id = {selectedId}";

MessageBox.Show("Строка изменена");

}

break;

case 2:

if (tfAdd == true)

{

int indexOfDot\_s = cbS\_t\_oId\_s.Text.IndexOf('.');

string id\_o = cbS\_t\_oId\_o.Text;

string id\_s = cbS\_t\_oId\_s.Text.Substring(0, indexOfDot\_s);

logRequest.CommandText = $ "INSERT INTO Services\_to\_orders (id\_o, id\_s) VALUES ({id\_o}, {id\_s})";

MessageBox.Show("Строка добавлена");

}

else

{

int indexOfDot\_s = cbS\_t\_oId\_s.Text.IndexOf('.');

string id\_o = cbS\_t\_oId\_o.Text;

string id\_s = cbS\_t\_oId\_s.Text.Substring(0, indexOfDot\_s);

logRequest.CommandText = $ "UPDATE Services\_to\_orders SET id\_o = {id\_o}, id\_s = {id\_s} WHERE id\_o = {selectedId} AND id\_s = {selectedIds}";

MessageBox.Show("Строка изменена");

}

break;

case 3:

if (tfAdd == true)

{

logRequest.CommandText = $ "INSERT INTO Services (name\_s, cost\_s) VALUES ('{tbServicesName\_s.Text}', {tbServicesCost\_s.Text})";

MessageBox.Show("Строка добавлена");

}

else

{

logRequest.CommandText = $ "UPDATE Services SET name\_s = '{tbServicesName\_s.Text}', cost\_s = {tbServicesCost\_s.Text} WHERE id = {selectedId}";

MessageBox.Show("Строка изменена");

}

break;

case 4:

if (tfAdd == true)

{

int indexOfDot = cbUsersRole\_u.Text.IndexOf('.');

string role\_u = cbUsersRole\_u.Text.Substring(0, indexOfDot);

logRequest.CommandText = $ "INSERT INTO Users (role\_u, name\_u, surname\_u, tel\_num\_u, pw, listed) VALUES ('{role\_u}'," + $ " '{tbUsersName\_u.Text}', '{tbUsersSurname\_u.Text}', '{tbUsersTel\_num\_u.Text}', '{tbUsersPw.Text}', {(chbUsersListed.Checked ? 1 : 0)})";

MessageBox.Show("Строка добавлена");

}

else

{

int indexOfDot = cbUsersRole\_u.Text.IndexOf('.');

string role\_u = cbUsersRole\_u.Text.Substring(0, indexOfDot);

logRequest.CommandText = $ "UPDATE Users SET role\_u = '{role\_u}', name\_u = '{tbUsersName\_u.Text}', " + $ "surname\_u = '{tbUsersSurname\_u.Text}', tel\_num\_u = '{tbUsersTel\_num\_u.Text}', " + $ "pw = '{tbUsersPw.Text}', listed = {(chbUsersListed.Checked ? 1 : 0)} WHERE id = {selectedId}";

MessageBox.Show("Строка изменена");

}

break;

case 5:

if (tfAdd == true)

{

logRequest.CommandText = $ "INSERT INTO Roles (name\_r) VALUES ('{tbRolesName\_r.Text}')";

MessageBox.Show("Строка добавлена");

}

else

{

logRequest.CommandText = $ "UPDATE Roles SET name\_r = '{tbRolesName\_r.Text}' WHERE id = {selectedId}";

MessageBox.Show("Строка изменена");

}

break;

}

logRequest.ExecuteNonQuery();

conn.Close();

}

}

catch (Exception ex)

{

MessageBox.Show("Ошибка: " + ex.Message);

}

}

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

{

this.Close();

Form2 fm = Application.OpenForms["Form2"] as Form2;

fm.Show();

fm.LoadDataIntoDataGridView();

}

private void dateTimePicker1\_ValueChanged(object sender, EventArgs e)

{}

}

}

**Листинг формы оператора**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Data.SqlClient;

using System.Drawing;

using System.Linq;

using System.Net.Sockets;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace test\_subd

{

public partial class Form4: Form

{

SqlConnection connect;

private bool tfDateEdit;

DataGridViewRow selectedRowTemp;

public Form4(SqlConnection cnct)

{

connect = cnct;

InitializeComponent();

}

private void Form4\_Load\_1(object sender, EventArgs e)

{

try

{

dataGridView1.AllowUserToAddRows = false;

dataGridView1.CellClick += dataGridView1\_CellClick;

dataGridView1.CellDoubleClick += DataGridView1\_CellDoubleClick;

tabControl1.SelectedIndexChanged += tabControl1\_SelectedIndexChanged;

tabControl1.MouseClick += TabControl\_MouseClick;

SqlConnection sqlConnect = new SqlConnection(Properties.Settings.Default.connectionString);

sqlConnect.Open();

SqlCommand showTable = new SqlCommand();

showTable.CommandText = $ "SELECT \* FROM Клиенты";

showTable.Connection = sqlConnect;

// SqlAdapter - прослойка между источником данных и базой данных

SqlDataAdapter adapter = new SqlDataAdapter(showTable);

DataSet dataSet = new DataSet();

// заполняем источник данных полученными из адаптера записями

adapter.Fill(dataSet);

dataGridView1.DataSource = dataSet.Tables[0];

dataGridView1.Columns[0].Visible = false;

sqlConnect.Close();

UpdatePrice();

dataGridView1.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.AllCells;

// Создаем словарь для хранения значений из таблицы Users

Dictionary < int, Tuple < string, string >> usersDictionary = new Dictionary < int, Tuple < string, string >> ();

string queryUsers = "SELECT id, name\_u, tel\_num\_u FROM Users WHERE listed = 1 AND role\_u = 3";

using(SqlCommand command = new SqlCommand(queryUsers, connect))

{

// Выполняем запрос и получаем результаты

using(SqlDataReader reader = command.ExecuteReader())

{

// Перебираем строки результата

while (reader.Read())

{

// Получаем значения столбцов id, name\_u и tel\_num\_u

int id = reader.GetInt32(0);

string name = reader.GetString(1);

string telNum = reader.GetString(2);

// Добавляем значения в словарь в виде кортежа

usersDictionary.Add(id, Tuple.Create(name, telNum));

}

}

}

foreach(var kvp in usersDictionary)

{

cbOrdersId\_maser.Items.Add($ "{kvp.Key}. {kvp.Value.Item1} - {kvp.Value.Item2}");

}

// Создаем словарь для хранения значений из таблицы Clients

Dictionary < int, Tuple < string, string >> clientsDictionary = new Dictionary < int, Tuple < string, string >> ();

string queryClients = "SELECT id, name\_c, tel\_num\_c FROM Clients";

using(SqlCommand command = new SqlCommand(queryClients, connect))

{

// Выполняем запрос и получаем результаты

using(SqlDataReader reader = command.ExecuteReader())

{

// Перебираем строки результата

while (reader.Read())

{

// Получаем значения столбцов id, name\_c и tel\_num\_c

int id = reader.GetInt32(0);

string name = reader.GetString(1);

string telNum = reader.GetString(2);

// Добавляем значения в словарь в виде кортежа

clientsDictionary.Add(id, Tuple.Create(name, telNum));

}

}

}

foreach(var kvp in clientsDictionary)

{

cbOrdersId\_client.Items.Add($ "{kvp.Key}. {kvp.Value.Item1} - {kvp.Value.Item2}");

}

}

catch (Exception ex)

{

MessageBox.Show("Ошибка: " + ex.Message);

}

}

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

{

try

{

// Получаем названия всех столбцов DGV1

var columnNames = dataGridView1.Columns.Cast < DataGridViewColumn > ().Select(x => x.HeaderText).ToList();

// Создаем строку фильтрации

string filterExpression = string.Empty;

foreach(var columnName in columnNames)

{

if (!string.IsNullOrEmpty(filterExpression)) filterExpression += " OR ";

filterExpression += $ "CONVERT([{columnName}], 'System.String') LIKE '%{tbDataSearch.Text}%'";

}

// Применяем фильтр к строкам DGV1

(dataGridView1.DataSource as DataTable).DefaultView.RowFilter = filterExpression;

}

catch (Exception ex)

{

MessageBox.Show("Ошибка: " + ex.Message);

}

}

private void comboBox1\_SelectedIndexChanged(object sender, EventArgs e)

{

LoadDataIntoDataGridView();

}

private void UpdatePrice()

{

// Создаем SQL-запрос

string sql = @ "

UPDATE Orders

SET Price = (SELECT SUM(s.cost\_s) FROM Services\_to\_orders so INNER JOIN Services s ON so.id\_s = s.ID WHERE so.id\_o = Orders.ID)

";

// Создаем команду и связываем ее с подключением и SQL-запросом

using(SqlCommand command = new SqlCommand(sql, connect))

{

// Выполняем SQL-запрос

int rowsAffected = command.ExecuteNonQuery();

// Выводим информацию о количестве обновленных строк

Console.WriteLine("Обновлено строк: " + rowsAffected);

}

}

public void LoadDataIntoDataGridView()

{

try

{

SqlConnection sqlConnect = new SqlConnection(Properties.Settings.Default.connectionString);

SqlCommand logRequst = new SqlCommand();

UpdatePrice();

switch (comboBox1.SelectedIndex)

{

case 0:

logRequst.CommandText = $ "SELECT \* FROM Клиенты";

break;

case 1:

logRequst.CommandText = $ "SELECT \* FROM Заказы";

break;

}

logRequst.Connection = sqlConnect;

// SqlAdapter - прослойка между источником данных и базой данных

SqlDataAdapter adapter = new SqlDataAdapter(logRequst);

DataSet dataSet = new DataSet();

// заполняем источник данных полученными из адаптера записями

adapter.Fill(dataSet);

dataGridView1.DataSource = dataSet.Tables[0];

tbDataSearch.Clear();

if (comboBox1.SelectedItem != "Заказы")

{

button3.Visible = false;

listBox1.Visible = false;

dataGridView1.Columns[0].Visible = false;

}

else

{

button3.Visible = true;

listBox1.Visible = true;

dataGridView1.Columns[0].Visible = true;

}

dataGridView1.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.AllCells;

sqlConnect.Close();

}

catch (Exception ex)

{

MessageBox.Show("Ошибка: " + ex.Message);

}

}

private void dataGridView1\_CellClick(object sender, DataGridViewCellEventArgs e)

{

try

{

if (comboBox1.SelectedItem == "Заказы" && e.RowIndex >= 0)

{

// Очищаем выделение в ListBox

//listBox1.ClearSelected();

// Соединение с базой данных

using(SqlConnection sqlConnect = new SqlConnection(Properties.Settings.Default.connectionString))

{

// Открытие соединения

sqlConnect.Open();

// Получение всех услуг из таблицы Services

string servicesQuery = "SELECT name\_s FROM Services";

// Создание команды для получения всех услуг

using(SqlCommand servicesCommand = new SqlCommand(servicesQuery, sqlConnect))

{

// Выполнение запроса и получение результатов

using(SqlDataReader servicesReader = servicesCommand.ExecuteReader())

{

// Очищаем ListBox перед заполнением новыми данными

listBox1.Items.Clear();

// Заполняем ListBox всеми значениями name\_s

while (servicesReader.Read())

{

listBox1.Items.Add(servicesReader["name\_s"].ToString());

}

}

}

// Получение выделенной строки в DataGridView

DataGridViewRow selectedRow = dataGridView1.CurrentRow;

// Запрос к базе данных для получения услуг для выбранного заказа

string query = $ "SELECT name\_s FROM Services\_to\_orders JOIN Services ON Services\_to\_orders.id\_s = Services.id WHERE id\_o = {selectedRow.Cells[0].Value}";

// Создание команды

using(SqlCommand command = new SqlCommand(query, sqlConnect))

{

// Выполнение запроса и получение результатов

using(SqlDataReader reader = command.ExecuteReader())

{

// Отмечаем в ListBox те элементы, которые соответствуют услугам для выбранного заказа

while (reader.Read())

{

string serviceName = reader["name\_s"].ToString();

int index = listBox1.FindStringExact(serviceName);

if (index != ListBox.NoMatches)

{

listBox1.SetSelected(index, true);

}

}

}

}

}

}

}

catch (Exception ex)

{

MessageBox.Show("Ошибка: " + ex.Message);

}

}

private void button3\_Click(object sender, EventArgs e)

{

// Получаем id выбранного заказа из DataGridView

int orderId = (int) dataGridView1.CurrentRow.Cells[0].Value;

// Очищаем все связанные услуги для данного заказа

ClearServicesForOrder(orderId);

// Добавляем новые связи для выбранных услуг

foreach(var item in listBox1.SelectedItems)

{

string serviceName = item.ToString();

// Получаем id услуги по ее названию

int serviceId = GetServiceId(serviceName);

// Добавляем новую связь в таблицу Services\_to\_orders

AddServiceToOrder(orderId, serviceId);

}

LoadDataIntoDataGridView();

}

private void ClearServicesForOrder(int orderId)

{

string deleteQuery = "DELETE FROM Services\_to\_orders WHERE id\_o = @orderId";

using(SqlCommand command = new SqlCommand(deleteQuery, connect))

{

command.Parameters.AddWithValue("@orderId", orderId);

command.ExecuteNonQuery();

}

}

private int GetServiceId(string serviceName)

{

string selectQuery = "SELECT id FROM Services WHERE name\_s = @serviceName";

using(SqlCommand command = new SqlCommand(selectQuery, connect))

{

command.Parameters.AddWithValue("@serviceName", serviceName);

// Выполняем запрос и возвращаем результат

object result = command.ExecuteScalar();

if (result != null)

{

return Convert.ToInt32(result);

}

else

{

// Возвращаем -1 в случае, если услуга с заданным именем не найдена

return -1;

}

}

}

private void AddServiceToOrder(int orderId, int serviceId)

{

string insertQuery = "INSERT INTO Services\_to\_orders (id\_o, id\_s) VALUES (@orderId, @serviceId)";

using(SqlCommand command = new SqlCommand(insertQuery, connect))

{

command.Parameters.AddWithValue("@orderId", orderId);

command.Parameters.AddWithValue("@serviceId", serviceId);

command.ExecuteNonQuery();

}

}

private void tabControl1\_SelectedIndexChanged(object sender, EventArgs e)

{

if (tabControl1.SelectedIndex == 0) // Проверяем, что выбрана первая вкладка

{

this.Width = 816; // Устанавливаем ширину формы

this.Height = 489; // Устанавливаем высоту формы

}

if (tabControl1.SelectedIndex == 1) // Проверяем, что выбрана первая вкладка

{

this.Width = 416; // Устанавливаем ширину формы

this.Height = 240; // Устанавливаем высоту формы

radioButton1.Checked = true;

}

ClearAllElements();

}

private void tabPage2\_Click(object sender, EventArgs e)

{}

private void dateTimePicker1\_ValueChanged(object sender, EventArgs e)

{}

private void radioButton2\_CheckedChanged(object sender, EventArgs e)

{

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

panelClients.Location = new Point(392, 274);

}

private void radioButton1\_CheckedChanged(object sender, EventArgs e)

{

panelOrders.Location = new Point(392, 274);

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

}

//Очистка элементов

private void ClearAllElements()

{

tbClientsAdress\_c.Text = string.Empty;

tbClientsNane\_c.Text = string.Empty;

tbClientsTel\_num\_c.Text = string.Empty;

cbOrdersId\_client.Text = string.Empty;

cbOrdersId\_maser.Text = string.Empty;

comboBox2.Text = string.Empty;

checkBox1.Checked = true;

}

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

{

SqlCommand logRequest = new SqlCommand();

logRequest.Connection = connect;

if (tfDateEdit == false)

{

if (radioButton1.Checked == true)

{

logRequest.CommandText = $ "INSERT INTO Clients (name\_c, tel\_num\_c, addres\_c) " + $ "VALUES ('{tbClientsNane\_c.Text}', '{tbClientsTel\_num\_c.Text}', '{tbClientsAdress\_c.Text}')";

MessageBox.Show("Строка добавлена");

}

if (radioButton2.Checked == true)

{

int indexOfDotClient = cbOrdersId\_client.Text.IndexOf('.');

string id\_client = cbOrdersId\_client.Text.Substring(0, indexOfDotClient);

if (checkBox1.Checked == false)

{

int indexOfDotMaster = cbOrdersId\_maser.Text.IndexOf('.');

string id\_master = cbOrdersId\_maser.Text.Substring(0, indexOfDotMaster);

logRequest.CommandText = $ "INSERT INTO Orders (id\_master, id\_client, date\_order, date\_complete) VALUES ({id\_master}, {id\_client}, " + $ "'{dtpOrdersDate\_order.Value.ToString("

yyyy - MM - dd ")}', '{dtpOrdersDate\_complete.Value.ToString("

yyyy - MM - dd ")}')";

MessageBox.Show("Строка добавлена");

}

else

{

int indexOfDotMaster = cbOrdersId\_maser.Text.IndexOf('.');

string id\_master = cbOrdersId\_maser.Text.Substring(0, indexOfDotMaster);

logRequest.CommandText = $ "INSERT INTO Orders (id\_master, id\_client, date\_order) VALUES ({id\_master}, {id\_client}, " + $ "'{dtpOrdersDate\_order.Value.ToString("

yyyy - MM - dd ")}')";

MessageBox.Show("Строка добавлена");

}

}

}

else

{

int indexOfDotMaster = comboBox2.Text.IndexOf('.');

string id\_master = comboBox2.Text.Substring(0, indexOfDotMaster);

logRequest.CommandText = $ "UPDATE Orders SET id\_master = {id\_master}, date\_complete = '{dateTimePicker1.Value.ToString("

yyyy - MM - dd ")}' " + $ "WHERE id = {selectedRowTemp.Cells[0].Value}";

MessageBox.Show("Строка изменена");

tabControl1.SelectedIndex = 0;

}

logRequest.ExecuteNonQuery();

LoadDataIntoDataGridView();

ClearAllElements();

}

private void TabControl\_MouseClick(object sender, MouseEventArgs e)

{

tfDateEdit = false;

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

panelOrders.Location = new Point(392, 274);

panelComplete.Location = new Point(392, 274);

radioButton1.Checked = true;

radioButton1.Visible = true;

radioButton2.Visible = true;

button2.Text = "Добавить";

}

private void DataGridView1\_CellDoubleClick(object sender, DataGridViewCellEventArgs e)

{

// Проверяем, что двойной щелчок произошел в пятом столбце

if (e.RowIndex >= 0 && comboBox1.SelectedItem == "Заказы") // Убедитесь, что строка действительная

{

tfDateEdit = true;

button2.Text = "Изменить";

tabControl1.SelectedIndex = 1;

radioButton1.Visible = false;

radioButton2.Visible = false;

panelClients.Location = new Point(392, 274);

panelOrders.Location = new Point(392, 274);

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

// Создаем словарь для хранения значений из таблицы Users

Dictionary < int, Tuple < string, string >> usersDictionary = new Dictionary < int, Tuple < string, string >> ();

string queryUsers = "SELECT id, name\_u, tel\_num\_u FROM Users WHERE listed = 1 AND role\_u = 3";

using(SqlCommand command = new SqlCommand(queryUsers, connect))

{

// Выполняем запрос и получаем результаты

using(SqlDataReader reader = command.ExecuteReader())

{

// Перебираем строки результата

while (reader.Read())

{

// Получаем значения столбцов id, name\_u и tel\_num\_u

int id = reader.GetInt32(0);

string name = reader.GetString(1);

string telNum = reader.GetString(2);

// Добавляем значения в словарь в виде кортежа

usersDictionary.Add(id, Tuple.Create(name, telNum));

}

}

}

foreach(var kvp in usersDictionary)

{

comboBox2.Items.Add($ "{kvp.Key}. {kvp.Value.Item1} - {kvp.Value.Item2}");

}

selectedRowTemp = dataGridView1.CurrentRow;

// Заполняем элементы из БД

string queryOrders = $ "SELECT id\_master, id\_client, date\_order, date\_complete FROM Orders WHERE id = @selectedId";

using(SqlCommand command = new SqlCommand(queryOrders, connect))

{

command.Parameters.AddWithValue("@selectedId", selectedRowTemp.Cells[0].Value);

using(SqlDataReader reader = command.ExecuteReader())

{

if (reader.Read())

{

int id\_master = reader.GetInt32(0);

//int id\_client = reader.GetInt32(1);

//DateTime date\_order = reader.GetDateTime(2);

if (!reader.IsDBNull(3)) // Проверяем, не равно ли значение NULL

{

DateTime date\_complete = reader.GetDateTime(3);

dateTimePicker1.Value = date\_complete;

}

else

{

DateTime date\_complete = DateTime.Now;

dateTimePicker1.Value = date\_complete;

}

// Закрыть текущий DataReader

reader.Close();

// Заполнение элементов формы из базы данных для мастера

string queryMaster = $ "SELECT name\_u, tel\_num\_u FROM Users WHERE id = @id\_master";

using(SqlCommand masterCommand = new SqlCommand(queryMaster, connect))

{

masterCommand.Parameters.AddWithValue("@id\_master", id\_master);

using(SqlDataReader masterReader = masterCommand.ExecuteReader())

{

if (masterReader.Read() && selectedRowTemp.Cells[1].Value != null)

{

string name\_master = masterReader.GetString(0);

string telNum\_master = masterReader.GetString(1);

comboBox2.Text = $ "{id\_master}. {name\_master} - {telNum\_master}";

}

}

}

}

}

}

// Создаем временный список для хранения уникальных элементов

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

// Проходим по всем элементам ComboBox

foreach(var item in comboBox2.Items)

{

// Если элемент еще не добавлен в список уникальных элементов, добавляем его

if (!uniqueItems.Contains(item.ToString()))

{

uniqueItems.Add(item.ToString());

}

}

// Очищаем ComboBox

comboBox2.Items.Clear();

// Добавляем уникальные элементы обратно в ComboBox

foreach(var item in uniqueItems)

{

comboBox2.Items.Add(item);

}

}

}

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

{

this.Close();

Form mainForm = Application.OpenForms["frmAuthorization"];

mainForm.Show();

}

}

}