Содержание

Введение 5

1. Анализ технического задания 6

1.1 Описание предметной области 9

1.2 Поиск и подбор аналогов 9

2. Разработка моделей данных 12

2.1 Концептуальная модель данных 13

2.2 Логическая модель данных 14

2.3 Физическая модель данных 16

3. Разработка и реализация АИС 20

3.1 SQL запросы 20

3.2 Руководство пользователя 26

3.3 Руководство программиста 38

4. Тестирование 45

Заключение 53

Список литературы 54

Приложение 1. Модели данных 55

Приложение 2. Текст программы 58

Введение

Военкомат занимает ключевую роль в обеспечении оборонной подготовки, мобилизации и призыва на военную службу. В условиях быстрого развития технологий и увеличения объёма данных становится очевидной необходимость внедрения информационных технологий и систем автоматизации. Одним из основных инструментов в этой сфере является использование баз данных.

Базы данных позволяют автоматизировать такие процессы как регистрация граждан, их учёт, организация медицинских осмотров, отсрочек и военной службы. Это способствует оптимизации работы военкомата, ускоряет принятие решений, повышает оперативность и минимизирует вероятность ошибок.

Таким образом, актуальной задачей является разработка информационной системы для военного комиссариата на основе реляционной базы данных. Цель курсовой работы — создать систему, обеспечивающую эффективное хранение и обработку данных, а также упрощающую учет граждан, медицинские осмотры, отсрочки и контроль за военной службой. Это решение позволит повысить качество обслуживания и эффективность работы военного комиссариата.

1. Анализ технического задания

При разработке информационной системы для военкомата необходимо тщательно выбирать технологии, которые удовлетворяют требованиям проекта. В рамках этого анализа будут рассмотрены ключевые инструменты, такие как языки программирования (C#, C++), среды разработки (Visual Studio и Eclipse) и системы управления базами данных (SQLite и MySQL). Сравнение этих технологий позволит определить оптимальные решения для обеспечения надёжности, безопасности и эффективности системы.

1. Языки программирования

С++ это язык программирования низкого уровня, обеспечивающий высокую производительность за счёт точного контроля над памятью и системными ресурсами. Он подходит для создания системного ПО и высоконагруженных приложений. Однако для информационных систем военкомата такой подход может быть излишним, поскольку задачи, которые предстоит решить, не требуют столь высокой вычислительной мощности. Вдобавок, разработка на С++ требует более тщательного управления памятью, что усложняет процесс разработки.

С# является более удобным для разработки языком программирования высокого уровня, особенно когда речь идёт о создании корпоративных приложений, таких как информационная система военкомата. Также тесная интеграция с платформой .NET, упрощает разработку, особенно для приложений, ориентированных на работу с данными. Автоматическое управление памятью в C# помогает избежать ошибок, связанных с утечками памяти, что делает язык более безопасным и удобным для разработки.

2. Среды разработки

Visual Studio – это мощная интегрированная среда разработки, которая предлагает широкий набор инструментов для отладки, тестирования и разработки приложений. Она идеально подходит для работы с С# и платформой .NET, что делает её отличным выбором для разработки информационных систем военкомата. Среда поддерживает множество языков программирования и обладает хорошей интеграцией с различными сервисами, что повышает производительность разработки.

Eclipse – бесплатная и кроссплатформенная среда разработки с открытым исходным кодом. Хорошо подходит для разработки на Java и других языках. Тем не менее, Eclipse предоставляет возможность расширять функциональность с помощью плагинов, что делает её гибким инструментом для разнообразных нужд.

3. Системы управления базами данных

SQLite - это встроенная и легковесная система управления базами данных, которая не требует установки отдельного сервера. Это идеальный выбор для локальных приложений с небольшими объёмами данных, как, например, для базы данных комиссариата, где важно работать с данными на уровне отдельных территориальных подразделений. SQLite проста в использовании, имеет компактную архитектуру и идеально подходит для небольших проектов. Однако её масштабируемость ограничена, и она не предназначена для работы с большими объёмами данных или высоконагруженными приложениями. [2]

MySQL является одной из самых популярных систем управления базами данных с открытым исходным кодом. Она отличается высокой производительностью и масштабируемостью, что делает её подходящим для крупных проектов с большими объёмами данных. MySQL поддерживает сложные запросы и транзакции, что позволяет эффективно работать с большими базами данных интегрировать систему с другими информационными системами. Эта система будет полезна, если информационная система требует обработки значительных объёмов данных и взаимодействия с другими базами данных или системами.

Результат анализа характеристик можно представить в виде таблиц: 1 – таблица сравнения языков программирования, 2 – таблица сравнения сред программирования, 3 – таблица сравнения СУБД.

Таблица 1 – Сравнение языков программирования

|  |  |  |
| --- | --- | --- |
| Критерий | С++ | С# |
| Производительность | Высокая | Средняя |
| Сложность | Сложная | Простая |
| Безопасность | Низкая | Высокая |
| Гибкость | Высокая | Средняя |
| Интеграция | Ограниченная | Отличная с .NET |

Таблица 2 – Сравнение сред программирования

|  |  |  |
| --- | --- | --- |
| Критерий | Visual Studio | Eclipse |
| Производительность | Высокая | Высокая |
| Сложность | Средняя | Средняя |
| Безопасность | Высокая | Высокая |
| Гибкость | Высокая | Высокая |
| Интеграция | Экосистема MS | Кроссплатформенная |

Таблица 3 – Сравнение СУБД

|  |  |  |
| --- | --- | --- |
| Критерий | SQLite | MySQL |
| Производительность | Высокая | Высокая |
| Сложность | Простая | Средняя |
| Безопасность | Высокая | Высокая |
| Гибкость | Низкая | Высокая |
| Интеграция | Локальное хранилище | Веб-разработка |

В данном случае C#, Visual Studio и SQLite являются оптимальным выбором. Они предлагают следующе преимущества:

* C# прост в использовании имеет высокую производительность и отличную совместимость с .NET
* Visual Studio мощная и интуитивно понятная интегрированная среда разработки, поддерживает множество языков и имеет богатый набор инструментов.
* SQLite легковесная и простая в использовании СУБД, идеально подходящая для встроенных систем и приложений с локальным хранением данных.

Эти инструменты обеспечат стабильную, безопасную и эффективную работу системы управления базой данных для военкомата.

1.1 Описание предметной области

Предметная область – это реального мира, данные о которых мы хотим отразить в базе данных. В данной курсовой работе рассматривается предметная область «Военкомат».

Основной задачей является проектирование и разработка приложения для автоматизации процессов учёта граждан, подлежащих воинскому учёту. Предполагаемая база данных должна обеспечивать эффективное хранение и обработку данных о гражданах, и другой связанной с эти информацией.

Основные цели военкомата:

* Регистрация граждан;
* Прохождение медосмотров;
* Оформление отсрочек;
* Регистрация службы.

1.2 Подбор аналогов

При поиске и подборе аналогов баз данных для военного комиссариата, может быть полезно обратить внимание на решения, которые предлагают схожий функционал

1.2.1 ВоенкоматАТ-Про (MilitaryCom AT Pro)

Достоинства:

* Удобный интерфейс для ввода и обработки данных о призывниках и военнообязанных.

Недостатки:

* Высокие затраты на внедрение и настройку системы;
* Потребность в обучении сотрудников для правильной работы с программой;
* Ограниченная гибкость при изменениях в законодательстве или в процессах работы военкомата.

Особенности:

* Модуль учёта граждан, подлежащих призыву;
* Поддержка интеграции с медицинскими учреждениями для проверки данных о здоровье;
* Инструменты для планирования и координации работы с призывниками.

1.2.2 MilitaryCall Cloud

Достоинства:

* Облачная архитектура, обеспечивающая доступ к системе из любой точки с интернет-соединением;
* Гибкость системы для интеграции с другими информационными системами.

Недостатки:

* Необходимость постоянного интернет-соединения;
* Зависимость от технической поддержки облачного провайдера;
* При работе с устаревшими системами, необходима дополнительная настройка интеграции.

Особенности:

* Встроенная аналитика и отчёты для оценки призывной кампании и эффективности работы;
* Возможность интеграции с медицинскими учреждениями для получения и обработки медицинских заключений;
* Веб-интерфейс для удобного доступа к системе.

2. Разработка моделей данных

Данный этап является важным этапом при создании АИС. Здесь выделяются сущности, атрибуты сущностей и связи между сущностями. На основе полученной диаграммы “Сущность – связь” или логической модели строятся функциональные модели системы и диаграмма потоков данных. Для создания базы данных, нужно логическую модель представить в виде физической. [1]

Разработка моделей данных включает 3 этапа:

1) Концептуальная модель:

* Определение сущностей (объектов, понятий) предметной области и их взаимодействия;
* Выявление атрибутов сущностей и описание их характеристик;
* Построение диаграммы сущность-связь (ER-диаграммы) для визуального представления модели.

2) Логическая модель:

* Определение типов данных, ключей, индексов, ограничений целостности;
* Трансформация концептуальной модели в структуру, подходящую для конкретной СУБД;
* Нормализация данных для устранения избыточности и аномалий;
* Спецификация отношений между сущностями (таблицами).

3) Физическая модель:

* Оптимизация логической модели для конкретной СУБД и аппаратной платформы;
* Определение физического хранения данных: файлов, томов, буферов и т.д;
* Результатом является физическая схема базы данных, готовая для реализации.

2.1 Концептуальная модель данных

Эта модель является первым и наиболее абстрактным уровнем проектирования базы данных. На этом этапе определяются ключевые сущности предметной области, их атрибуты и взаимосвязи.

Сущность «Военная служба» имеет атрибуты: Начало службы, окончание службы, подразделение, звание, статус службы.

Сущность «Гражданин» имеет атрибуты: Фамилия, имя, отчество, дата рождения, образование, адрес, статус.

Сущность «Медосмотр» имеет атрибуты: Вид осмотра, дата осмотра, результат, врач.

Сущность «Отсрочка» имеет атрибуты: Основание отсрочки, дата начала, дата окончания, статус.

Сущность «Врач» имеет атрибуты: ФИО.

Сущность «Вид осмотра» имеет атрибуты: Наименование.

Сущность «Основание отсрочки» имеет атрибуты: Наименование.

Можно выделить следующие связи между сущностями:

* Между сущностями «Гражданин» и «Военная служба» существует связь «один-ко-многим», гражданин может иметь несколько сроков службы.
* Между сущностями «Гражданин» и «Отсрочка» существует связь «один-ко-многим», гражданин может иметь несколько отсрочек.
* Между сущностями «Гражданин» и «Медосмотр» существует связь «один-ко-многим», гражданин может проходить несколько медосмотров.
* Между сущностями «Медосмотр» и «Вид осмотра» существует связь «многие к-одному», множество медосмотров может быть одного вида.
* Между сущностями «Медосмотр» и «Врач» существует связь «многие к-одному», множество медосмотров может проводить один врач.
* Между сущностями «Отсрочка» и «Основание отсрочки» существует связь «многие к-одному», несколько отсрочек может иметь одно основание.

На рисунке 1 изображена концептуальная модель базы данных.

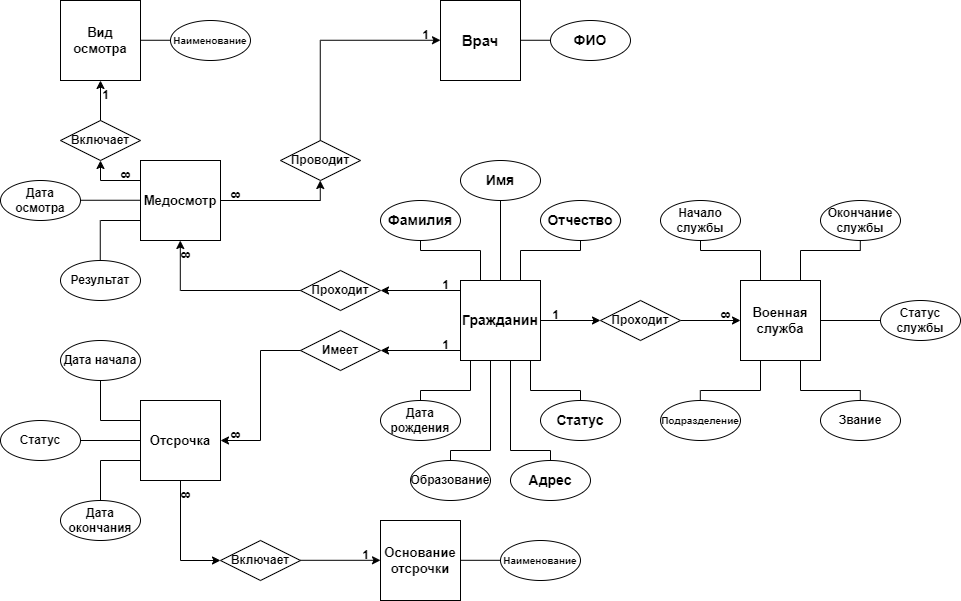


Рисунок 1 – Концептуальная модель данных

2.2 Логическая модель данных

Логическая модель данных является следующим этапом после концептуальной модели. На этом этапе концептуальная модель трансформируется в структуру, подходящую для конкретной системы управления базами данных (СУБД).

Основные сущности и их атрибуты:

2.2.1 Военная служба:

* ID – уникальный идентификатор
* ID\_Гражданина – связь с гражданином
* Начало службы
* Окончание службы
* Подразделение
* Звание
* Статус службы

2.2.2 Гражданин:

* ID – уникальный идентификатор
* Фамилия
* Имя
* Отчество
* Дата рождения
* Образование
* Адрес
* Статус

2.2.3 Медосмотр:

* ID – уникальный идентификатор
* ID\_Гражданина – связь с гражданином
* ID\_Вид осмотра
* Дата осмотра
* Результат
* Врач

2.2.4 Отсрочка:

* ID – уникальный идентификатор
* ID\_Гражданина – связь с гражданином
* ID\_Основание отсрочки – связь с основными отсрочками
* Дата начала
* Дата окончания
* Статус

2.2.5 Основание отсрочки:

* ID – уникальный идентификатор
* Название

2.2.6 Вид осмотра:

* ID – уникальный идентификатор
* Название

2.2.7 Врач:

* ID – уникальный идентификатор
* ФИО

На основе этих данных была составлена логическая модель данных, представленная на рисунке 2.

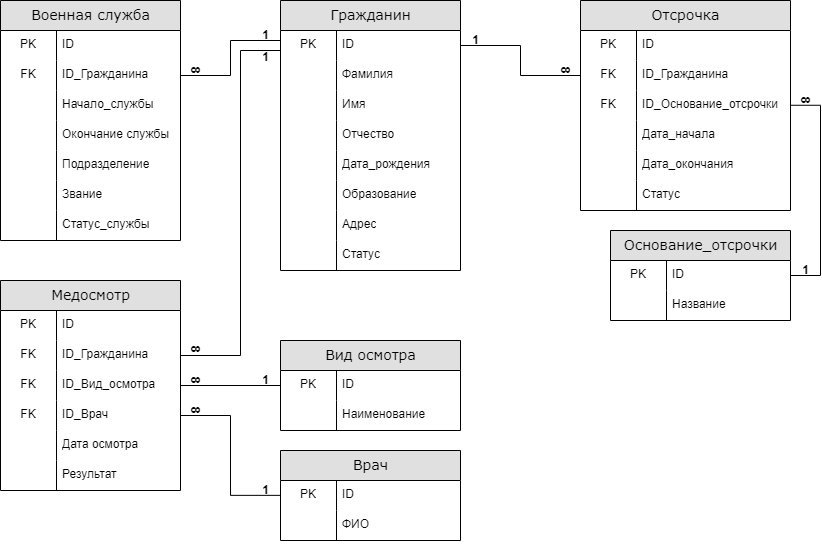


Рисунок 2 – Логическая модель данных

2.3 Физическая модель данных

Этот этап проектирования базы данных, предполагаемый реализацию логической модели с учётом конкретной системы управления базами данных (СУБД) и требований к производительности системы. [3]

К основным функциям физической модели данных относится:

1) Определение физических характеристик таблиц и полей.

2) Проектирование физической структуры базы данных.

3) Оптимизация производительности.

На рисунке 3 изображена физическая модель данных.

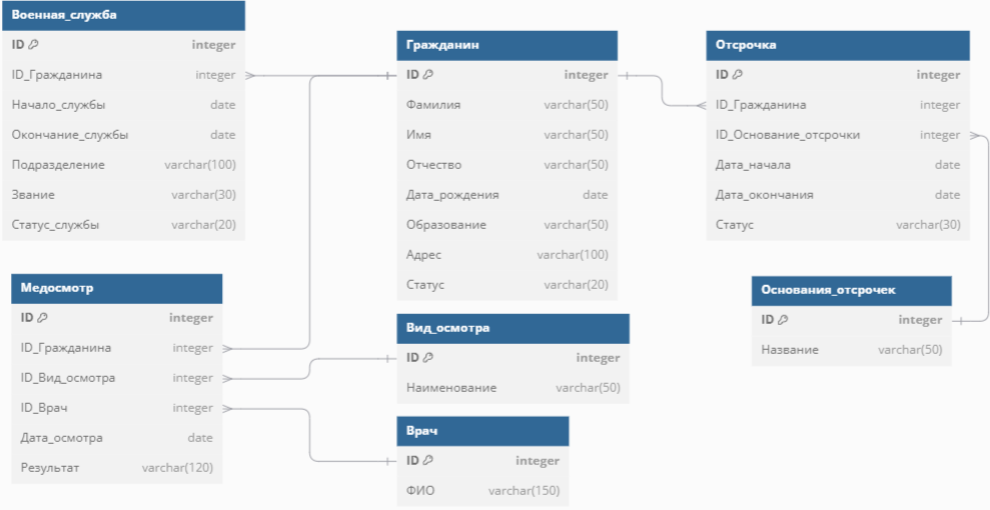


Рисунок 3 – Физическая модель данных

Физическая модель данных состоит из следующих таблиц:

2.3.1 Таблица ʺГражданинʺ:

* ID (integer, primary key)
* Фамилия (varchar (50))
* Имя (varchar (50))
* Отчество (varchar (50))
* Дата\_рождения (date)
* Образование (varchar (50))
* Адрес (varchar (100))
* Статус (varchar (20))

2.3.2 Таблица ʺВоенная\_службаʺ:

* ID (integer, primary key)
* ID\_Гражданина (integer, внешний ключ на таблицу Гражданин)
* Начало\_службы (date)
* Окончание\_службы (date)
* Подразделение(varchar (100))
* Звание (varchar (30))
* Статус\_службы (varchar (20))

2.3.3 Таблица ʺОтсрочкаʺ:

* ID (integer, primary key)
* ID\_Гражданина (integer, внешний ключ на таблицу Гражданин)
* ID\_Основание\_отсрочки (integer, внешний ключ на таблицу Основание\_отсрочки)
* Дата\_начала (date)
* Дата\_окончания (date)
* Статус (varchar (30))

2.3.4 Таблица ʺМедосмотрʺ:

* ID (integer, primary key)
* ID\_Гражданина (integer, внешний ключ на таблицу Гражданин)
* ID\_Вид\_осмотра (integer, внешний ключ на таблицу Вид\_осмотра)
* ID\_Врач (integer, внешний ключ на таблицу Врач)
* Дата\_осмотра (date)
* Результат (varchar (120))

2.3.5 Таблица ʺВид\_осмотраʺ:

* ID (integer, primary key)
* Наименование (varchar (50))

2.3.6 Таблица ʺОснование\_отсрочекʺ:

* ID (integer, primary key)
* Название (varchar (50))

2.3.7 Таблица ʺВрачʺ:

* ID (integer, primary key)
* ФИО (varchar (150))

3. Разработка и реализация АИС

3.1 SQL запросы

SQL запросы являются основным механизмом взаимодействия с реляционными базами данных. Они позволяют эффективно управлять данными, обеспечивая их хранение, извлечение, модификацию и удаление. [4]

Результатом запроса является представление – виртуальная таблица, представляющая собой поименованный запрос, который будет подставлен как подзапрос при использовании представления. В отличии от обычных таблиц реляционных баз данных, представление не является самостоятельной частью набора данных, хранящегося в базе. [5]

3.1.1 Запрос создания таблицы (CREATE TABLE):

Запрос на создание таблицы ʺГражданинʺ

CREATE TABLE Медосмотр (ID INTEGER PRIMARY KEY, ID\_Гражданина INTEGER REFERENCES Гражданин (ID), [Вид осмотра] INTEGER REFERENCES [Вид осмотра] (ID), [Дата осмотра] TEXT, Результат TEXT, Врач INTEGER REFERENCES Врач (ID));

Этот запрос выполняет следующие действия:

- CREATE TABLE Медосмотр – команда создаёт новую таблицу с названием ʺМедосмотрʺ.

- ID INTEGER PRIMARY KEY – создаёт столбец ʺIDʺ с типом данных INTEGER, который будет уникальным идентификатором каждой записи в таблице ʺМедосмотрʺ. Этот столбец является первичным ключом и будет использоваться для уникальной идентификации осмотра.

- ID\_Гражданина INTEGER REFERENCES Гражданин (ID) – создаёт столбец ʺID\_Гражданинаʺ, который хранит идентификатор гражданина, прошедшего медицинский осмотр. Этот столбец является ключевым внешним ключом, ссылающимся на столбец ʺIDʺ в таблице ʺГражданинʺ. Это связывает каждый осмотр с конкретным гражданином.

- [Вид осмотра] INTEGER REFERENCES [Вид осмотра] (ID) – создаёт столбец ʺВид осмотраʺ, который хранить идентификатор типа осмотра. Этот столбец является внешним ключом, ссылающимся на столбец ʺIDʺ в таблице ʺВид осмотраʺ. Это позволяет связать каждый осмотр с определённым типом медицинского осмотра.

- [Дата осмотра] TEXT – создаёт столбец ʺДата осмотраʺ с типом данных TEXT для хранения даты осмотра. Он указывает, когда был проведён медицинский осмотр.

- Результат TEXT – создаёт столбец ʺРезультатʺ, в котором будет храниться текстовое описание результатов медицинского осмотра. Это поле может содержать информацию о состоянии здоровья гражданина по результату осмотра.

- Врач INTEGER REFERENCES Врач (ID) – создаёт столбец ʺВрачʺ, который хранит идентификатор врача, проводившего осмотр. Этот столбец является внешним ключом, ссылающимся на столбец ʺIDʺ в таблице ʺВрачʺ. Это связывает каждый осмотр с конкретным врачом, который его проводил.

3.1.2 Запрос выборки из таблиц (SELECT)

1) Запрос для извлечения данных о службе гражданина по его ID

SELECT Звание, [Начало службы], [Окончание службы], Подразделение

FROM [Запись о службе]

WHERE ID\_Гражданина = @CitizenId

Этот запрос выполняет следующие действия:

- SELECT Звание, [Начало службы], [Окончание службы], Подразделение – выбираются столбцы, содержащие информацию о звании, дате начала и окончания службы, а также подразделении, в котором служит гражданин.

- FROM [Запись о службе] – данные извлекаются из таблицы ʺЗапись о службеʺ.

- WHERE ID\_Гражданина = @CitizenId – фильтрует записи, выбирая только те, у которых значение в столбце ʺID\_Гражданинаʺ соответствует значению переменной @CitizenId.

2) Запрос для извлечения данных по определённым критериям

SELECT ID, Фамилия, Имя, Отчество, [Дата рождения], Образование, Статус

FROM Гражданин

WHERE (Фамилия LIKE @SearchTerm0

OR Имя LIKE @SearchTerm0

OR Отчество LIKE @SearchTerm0

OR Образование LIKE @SearchTerm0

OR Статус LIKE @SearchTerm0)

Этот запрос выполняет следующие действия:

- SELECT ID, Фамилия, Имя, Отчество, [Дата рождения], Образование, Статус – выбираются столбцы из таблицы ʺГражданинʺ.

- FROM Гражданин – данные извлекаются из таблицы ʺГражданинʺ.

- WHERE (Фамилия LIKE @SearchTerm0 OR ...) – фильтрует записи, выбирая только те, у которых хотя бы одно из полей (Фамилия, Имя, Отчество, Образование или Статус) содержит подстроку, соответствующую значению переменной @SearchTerm0.

3) Запрос для извлечения отсрочек по гражданину

SELECT [Основание отсрочки], [Дата начала], [Дата окончания]

FROM Отсрочка

WHERE ID\_Гражданина = @CitizenId

Этот запрос выполняет следующие действия:

- SELECT [Основание отсрочки], [Дата начала], [Дата окончания] – выбираются столбцы ʺОснование отсрочкиʺ, ʺДата началаʺ и ʺДата окончанияʺ из таблицы ʺОтсрочкаʺ.

- FROM Отсрочка – данные извлекаются из таблицы ʺОтсрочкаʺ.

- WHERE ID\_Гражданина = @CitizenId – фильтрует записи, выбирая только те, у которых значение в столбце ʺID\_Гражданинаʺ соответствует значению переменной @CitizenId.

3.1.3 Запрос вставки в таблицу (INSERT)

1) Запрос для добавления новой записи о службе

INSERT INTO [Запись о службе] (ID\_Гражданина, Звание, [Начало службы], [Окончание службы], Подразделение)

VALUES (@CitizenId, @Rank, @StartDate, @EndDate, @Place)

Этот запрос выполняет следующие действия:

- INSERT INTO [Запись о службе] (ID\_Гражданина, Звание, [Начало службы], [Окончание службы], Подразделение) – вставляем новые данные в таблицу ʺЗапись о службеʺ, перечисляя столбцы, в которые будут вставлены значения.

- VALUES (@CitizenId, @Rank, @StartDate, @EndDate, @Place) – задаём значения для столбцов. Переменные @CitizenId, @Rank, @StartDate, @EndDate, @Place будут заменены при выполнении запроса.

2) Запрос для добавления новой записи об отсрочке

INSERT INTO Отсрочка (ID\_Гражданина, [Основание отсрочки], [Дата начала], [Дата окончания])

VALUES (@CitizenId, @DefermentReason, @StartDate, @EndDate)

Этот запрос выполняет следующие действия:

- INSERT INTO Отсрочка (ID\_Гражданина, [Основание отсрочки], [Дата начала], [Дата окончания]) – вставляем новые данные в таблицу "Отсрочка", перечисляя столбцы для вставки значений.

- VALUES (@CitizenId, @DefermentReason, @StartDate, @EndDate) – задаём значение для столбцов. Переменные @CitizenId, @DefermentReason, @StartDate, @EndDate будут заменены при выполнении запроса.

3.1.4 Запросы обновления записей в таблицах (UPDATE)

1) Запрос для обновления существующей записи о гражданине

UPDATE Гражданин

SET Фамилия = @LastName, Имя = @FirstName, Отчество = @MiddleName, [Дата рождения] = @BirthDate,

Образование = @Education, Статус = @Status

WHERE ID = @CitizenId

Этот запрос выполняет следующие действия:

- UPDATR Гражданин SET – указывает, что обновляются данные в таблице ʺГражданинʺ.

- Фамилия = = @LastName, Имя = @FirstName, Отчество = @MiddleName – устанавливаются новые значения для столбцов Фамилия, Имя и Отчество с использованием значений переменных @LastName, @FirstName и @MiddleName.

- [Дата рождения] = @BirthDate, Образование = @Education, Статус = @Status – аналогично обновляются другие столбцы.

- WHERE ID = @CitizenId – обновление будет выполнено только для записи с соответствующим значением ID.

2) Запрос для обновления записи о военной службе

UPDATE [Военная служба]

SET Подразделение = @subdivision,

Звание = @rank,

[Статус службы] = @status

WHERE ID = @recordId

Этот запрос выполняет следующие действия:

- UPDATE [Военная служба]: указывает таблицу, где будут обновляться данные.

- SET: устанавливает новые значения для столбцов Подразделение, Звание и Статус службы из переменных.

- WHERE ID = @recordId: обновляет запись, соответствующую указанному идентификатору ID.

3.1.5 Запросы для удаления записи из таблицы (DELETE)

1) Запрос для удаления записи о службе

DELETE FROM [Военная служба] WHERE ID = @recordId

Этот запрос выполняет следующие действия:

- DELETE FROM [Военная служба]: удаляет запись из таблицы Военная служба.

- WHERE ID = @recordId: удаляется запись с заданным идентификатором.

2) Запрос для удаления записи о отсрочке

DELETE FROM Отсрочка WHERE ID = @defermentId

Этот запрос выполняет следующие действия:

- DELETE FROM Отсрочка: удаляет запись из таблицы Отсрочка.

-WHERE ID = @defermentId: удаляется запись с указанным идентификатором.

3.2 Руководство пользователя

При открытии приложения первым шагом является окно авторизации, показанное на рисунке 1, которое служит для идентификации и аутентификации пользователей.

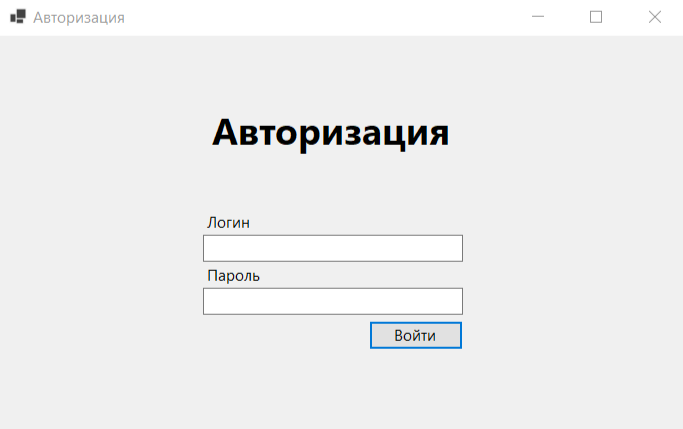


Рисунок 1 – окно авторизации

В окне авторизации пользователь вводит логин и пароль от своей учётной записи, что показано на рисунках 2, 3, 4 и соответственно если логин и пароль правильные, то он попадает в рабочее окно, что показано на рисунках 5, 6, 7.

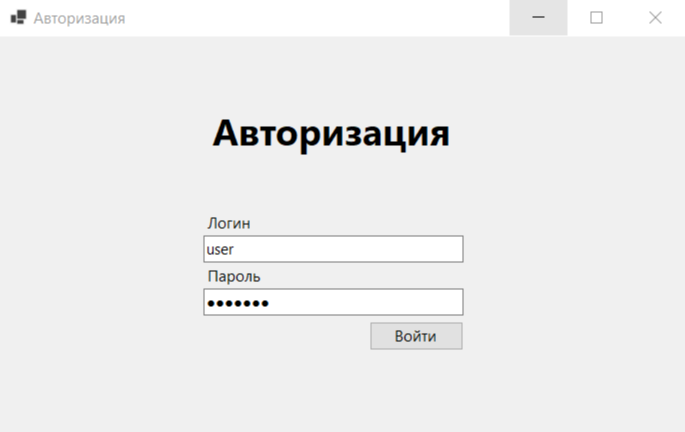


Рисунок 2 – ввод данных от учётной записи сотрудника

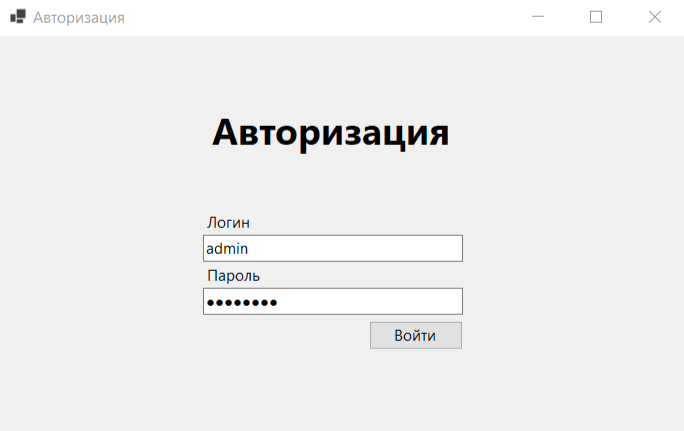


Рисунок 3 – ввод данных от учётной записи администратора

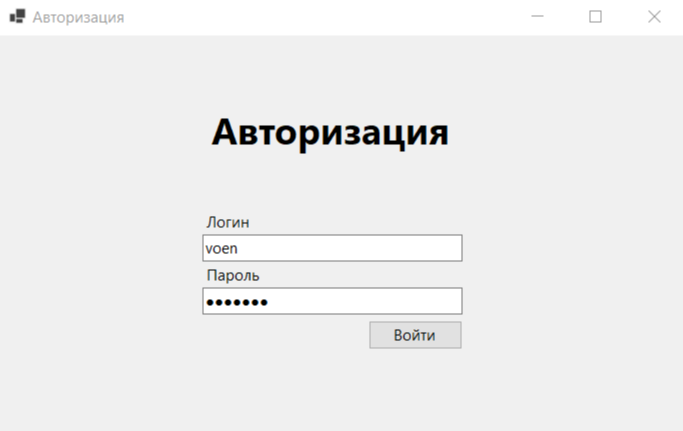


Рисунок 4 – ввод данных от учётной записи комиссара

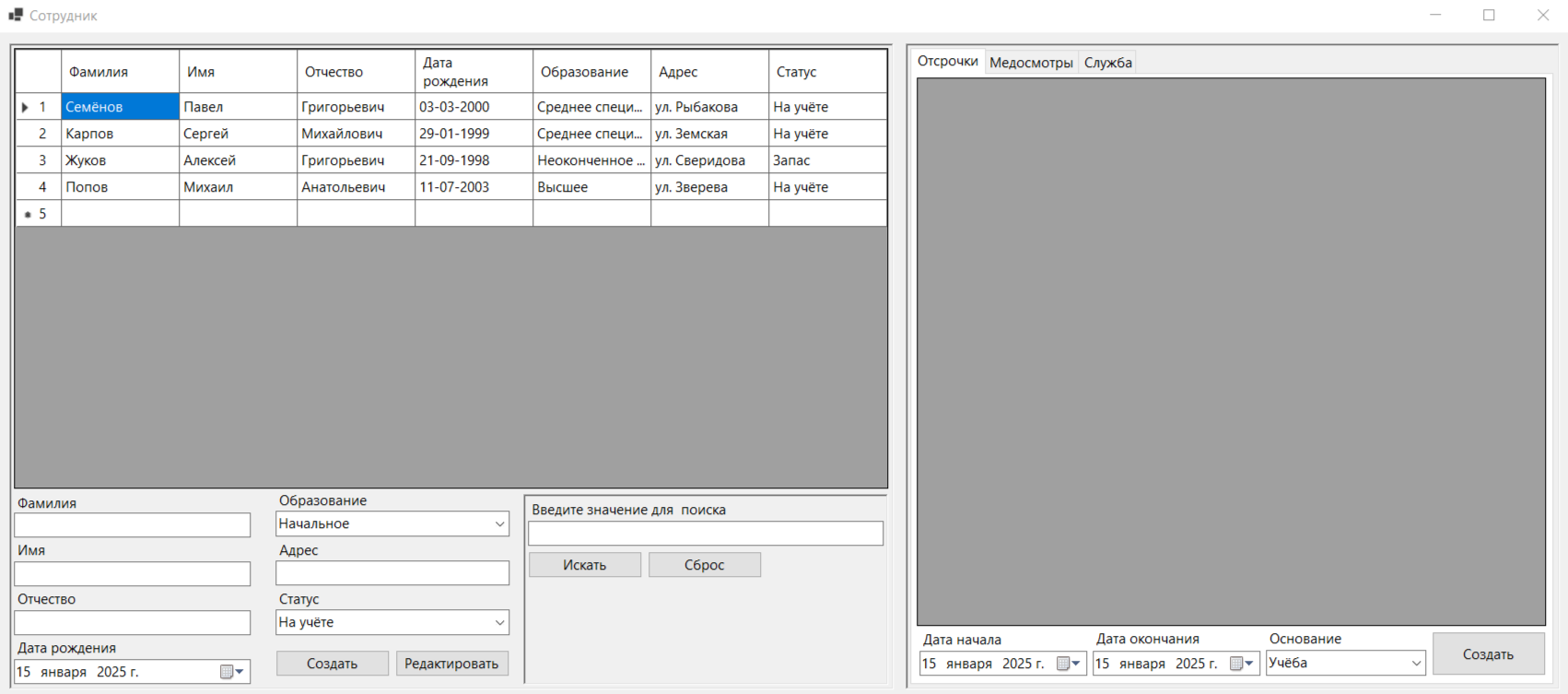


Рисунок 5 – рабочее окно сотрудника

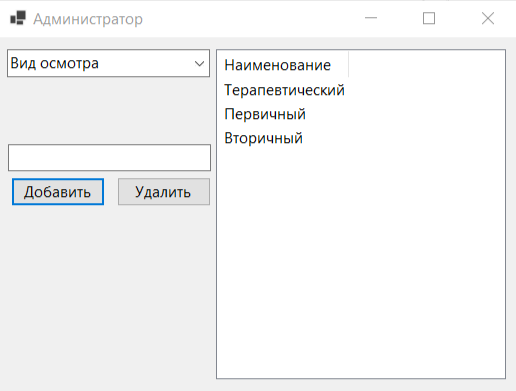


Рисунок 6 – рабочее окно администратора

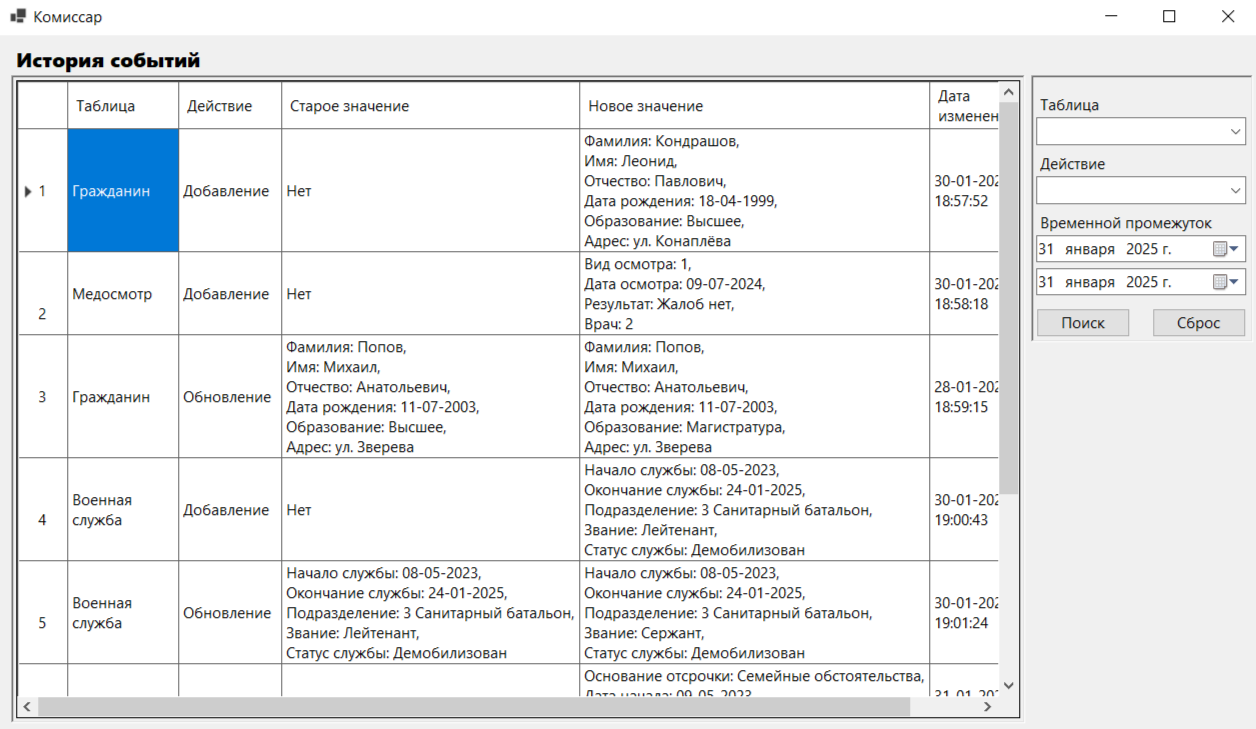


Рисунок 7 – рабочее окно комиссара

Форма сотрудника, показанная на рисунке 5, имеет следующие функции:

1) Добавление нового гражданина через поля ввода. Для этого после заполнения полей и нажатия на кнопку ʺСоздатьʺ, данные будут записаны. Результат показан на рисунках 8 – 9.

2) Поиск гражданина по различным полям, таким как фамилия, имя, отчество, образование и статус. Результат показан на рисунке 10.

3) Отображение на вкладках отсрочки, медосмотры и служба после выбора записи гражданина дополнительных данных. Результат показан на рисунках 11 – 13.

4) Создание, редактирование и удаление записей на дополнительных вкладках. Результат показан на рисунке 14 – 19.

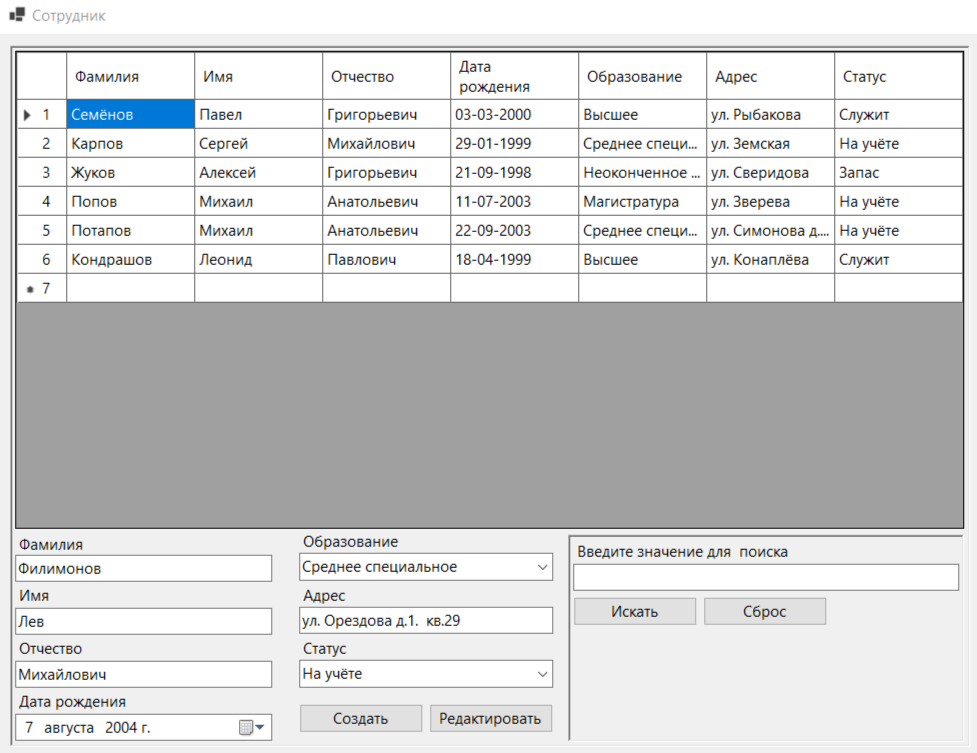


Рисунок 8 – ввод данных гражданина

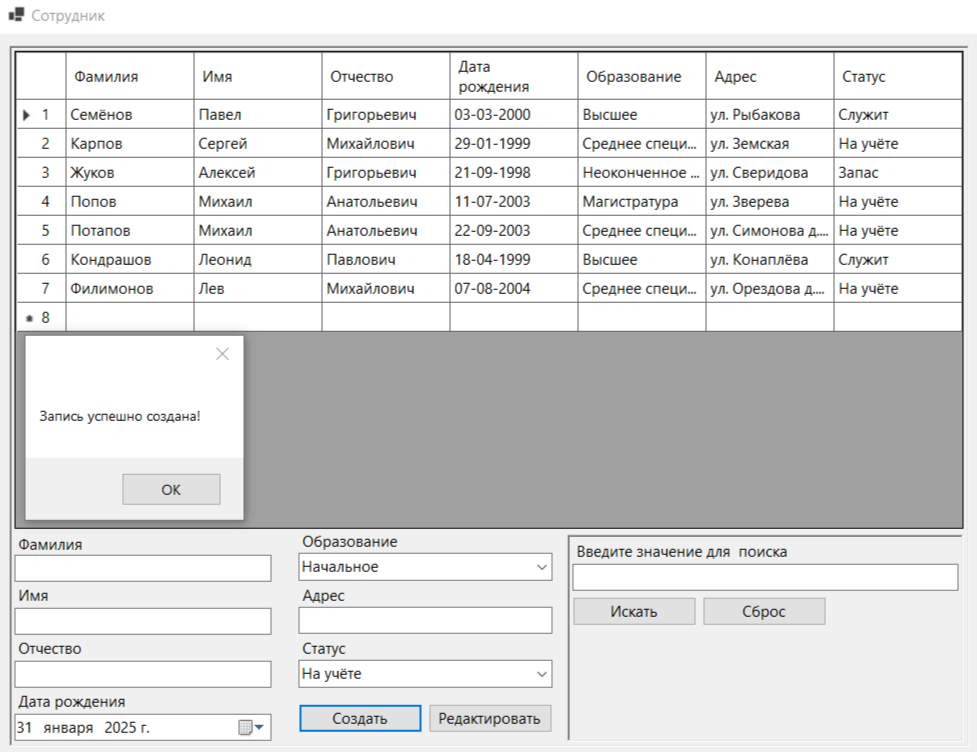


Рисунок 9 – успешное создание записи

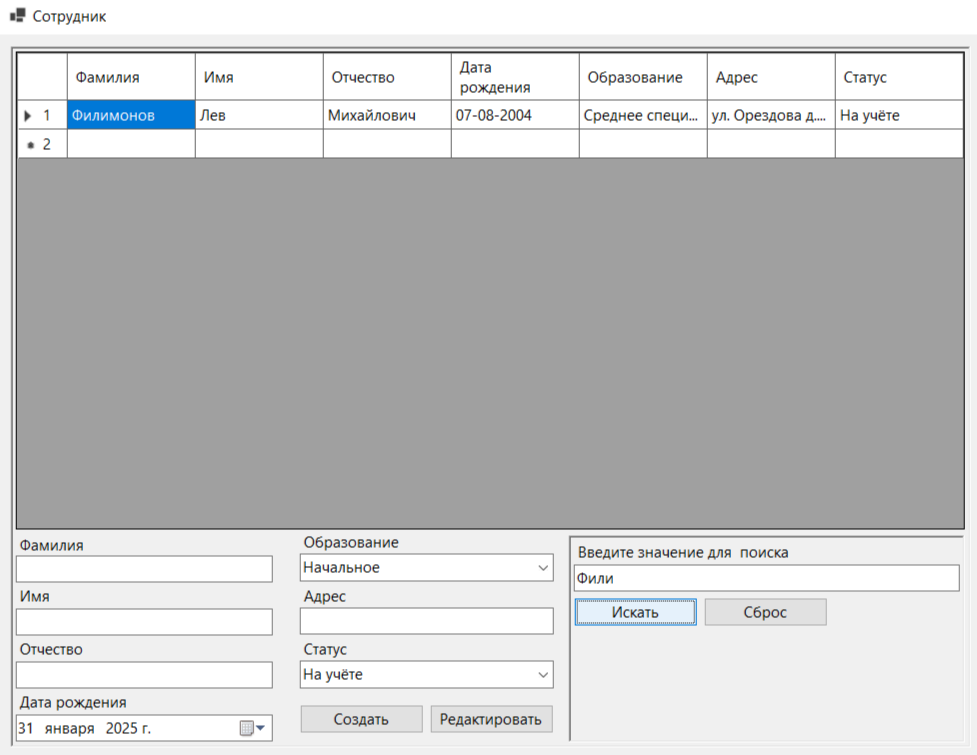


Рисунок 10 – поиск записи по полю

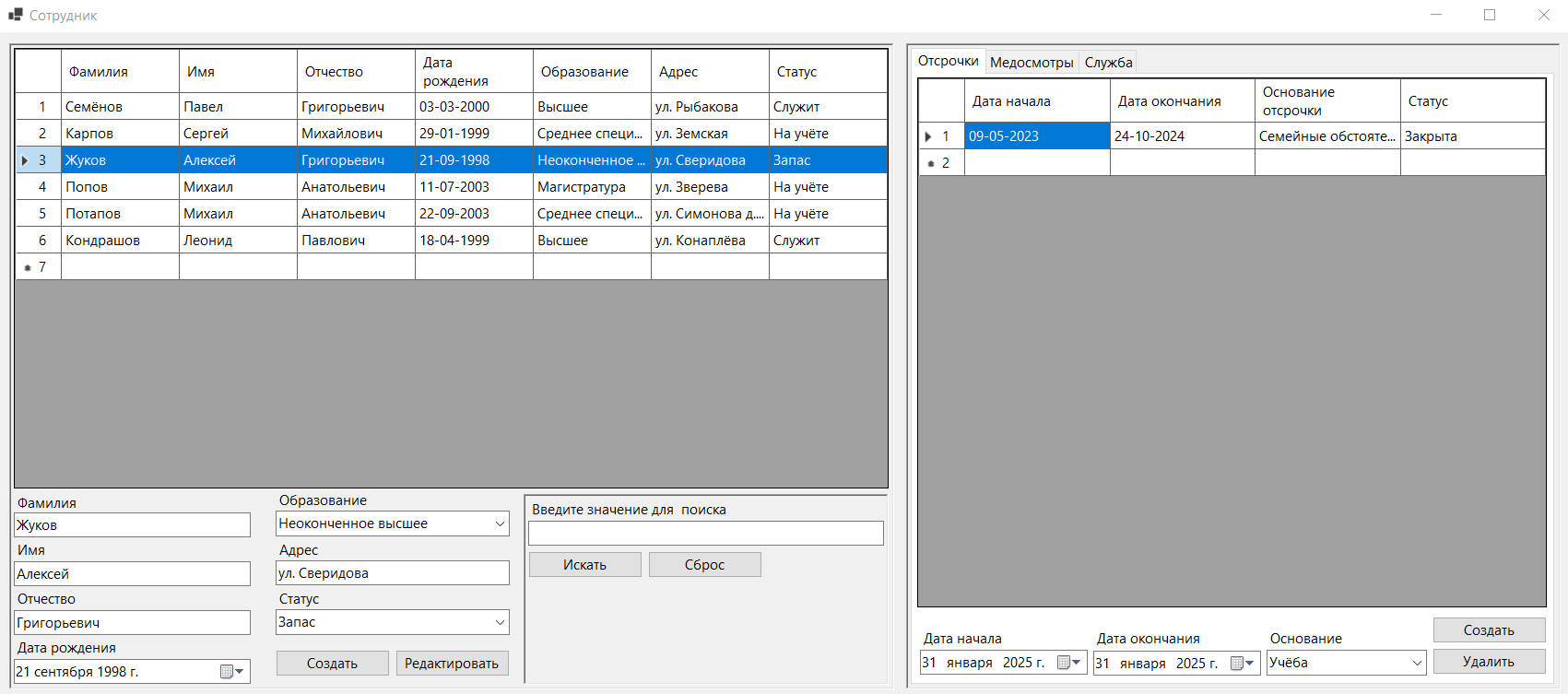


Рисунок 11 – страница с отсрочками

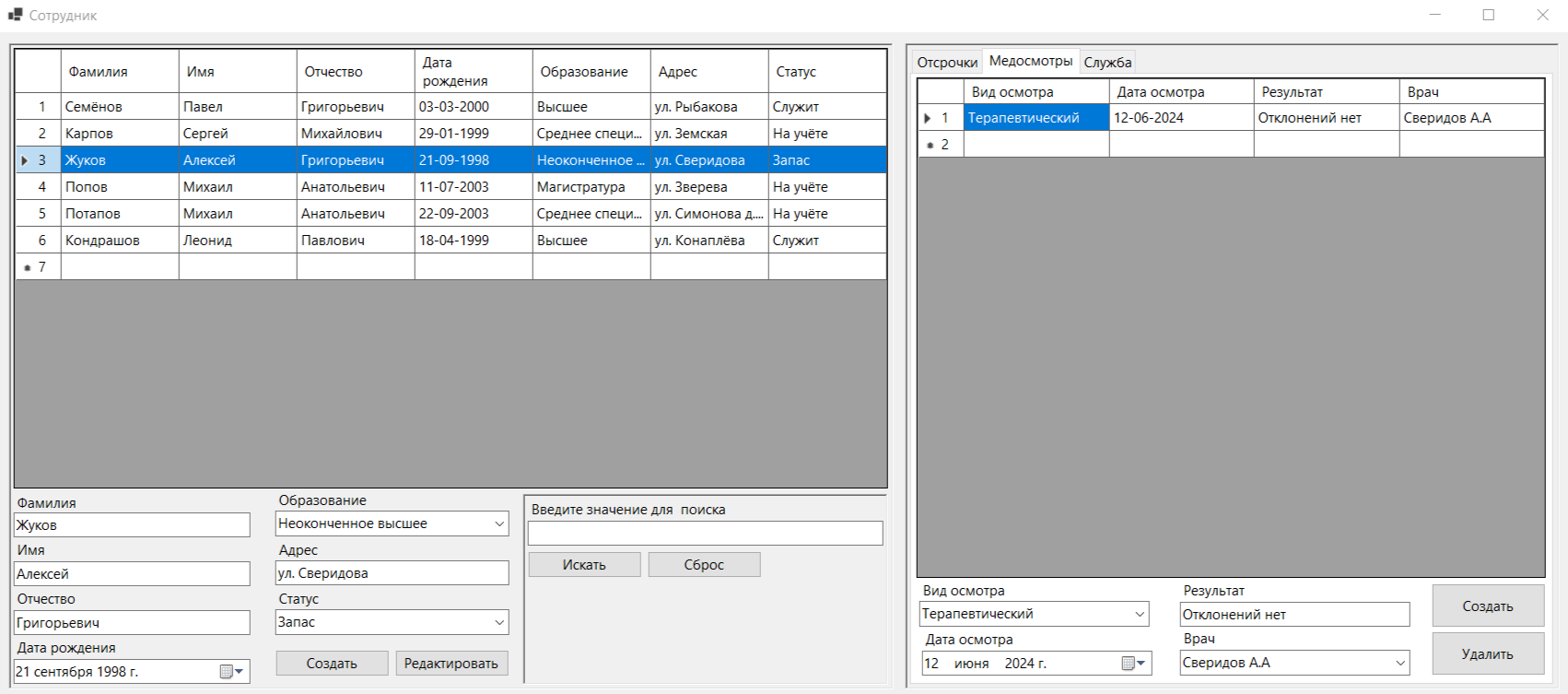


Рисунок 12 – страница с медосмотрами

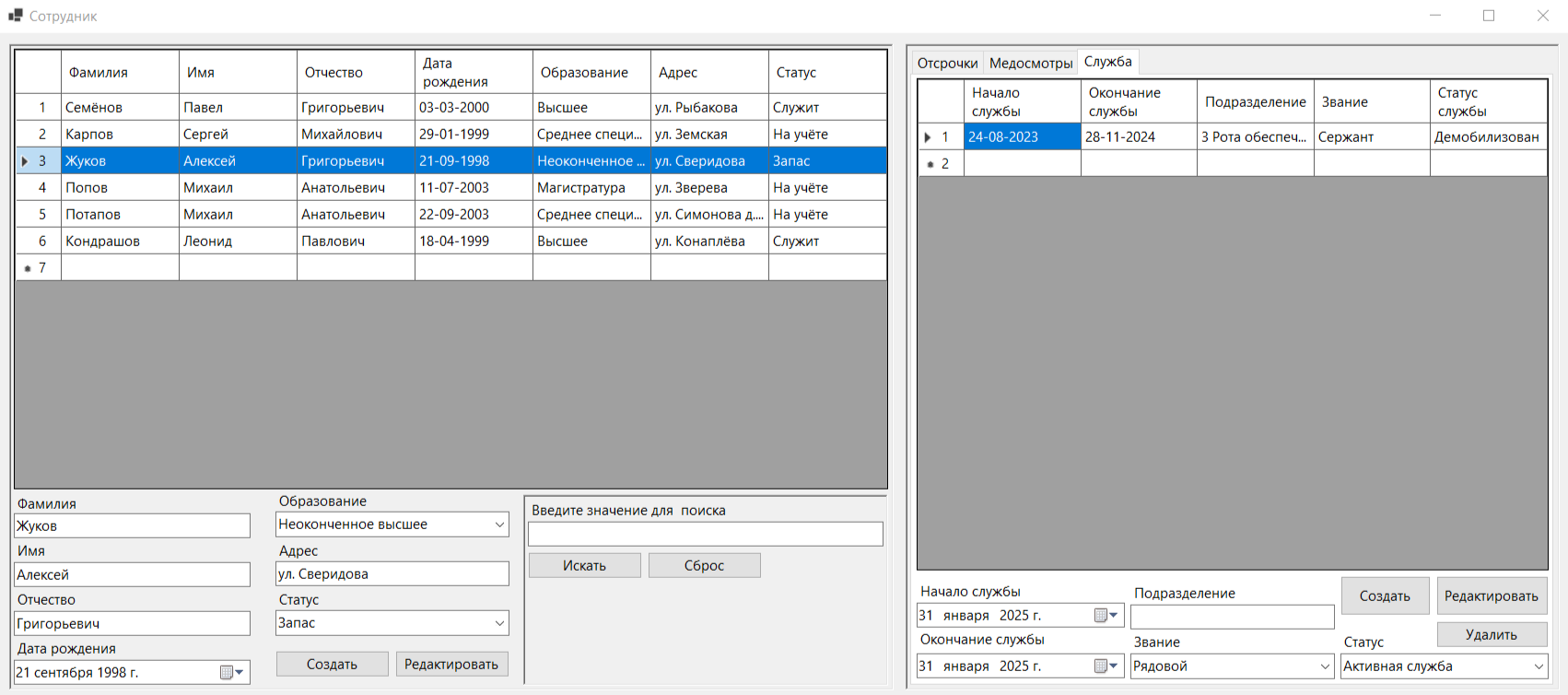


Рисунок 13 – страница с периодами службы

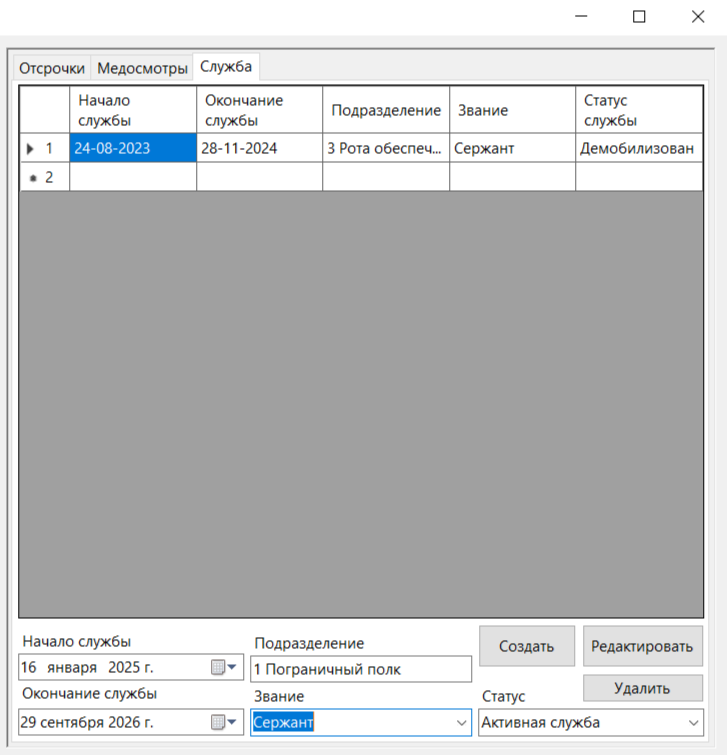


Рисунок 14 – ввод данных службы

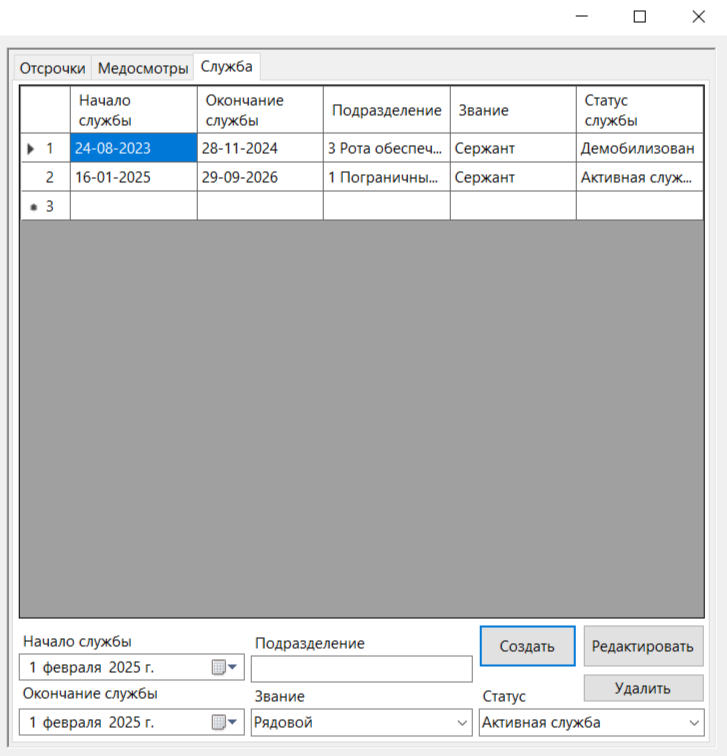


Рисунок 15 – успешное создание записи

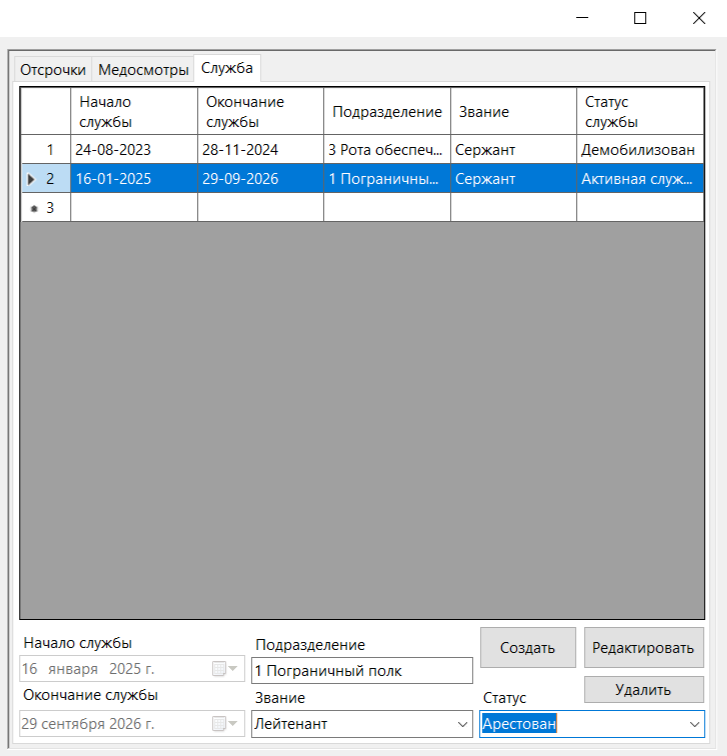


Рисунок 16 – редактирование данных

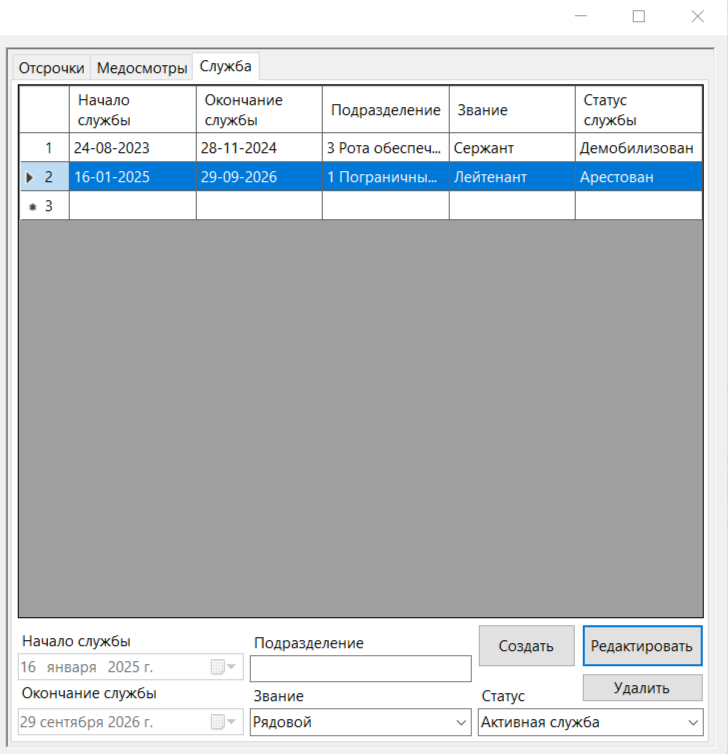


Рисунок 17 – подтверждение редактирования

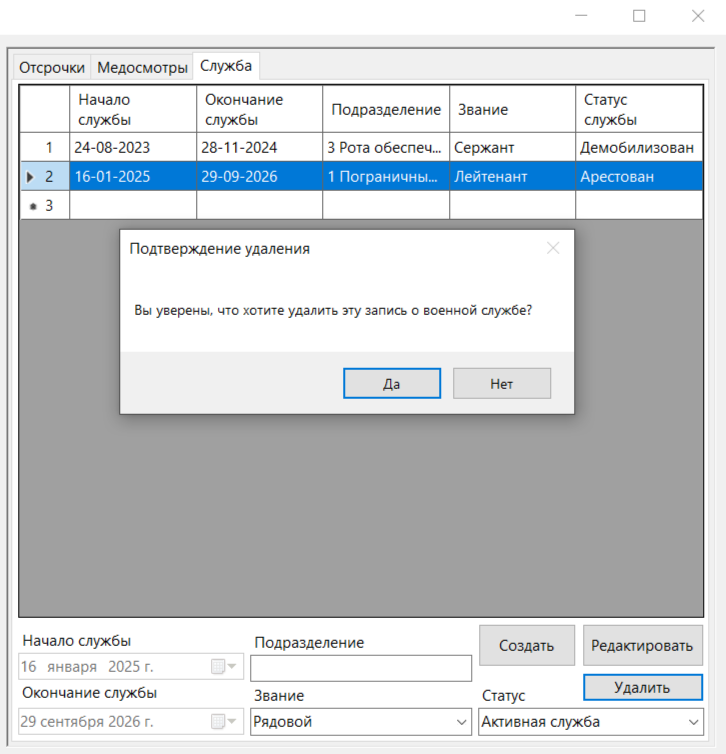


Рисунок 18 – удаление записи

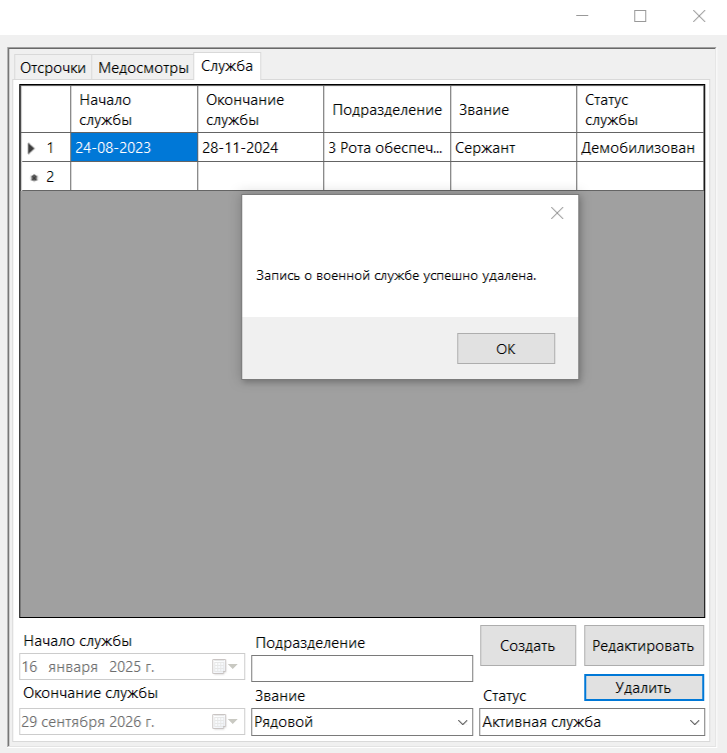


Рисунок 19 – успешное удаление записи

Форма администратора, показанная на рисунке 6 имеет следующие функции:

1) Вывод имеющихся в базе видов осмотра и оснований осмотра. Результат показан на рисунке 20 – 21.

2) Добавление и удаление записей видов осмотра и оснований осмотра. Результат показан на рисунках 22 – 24.

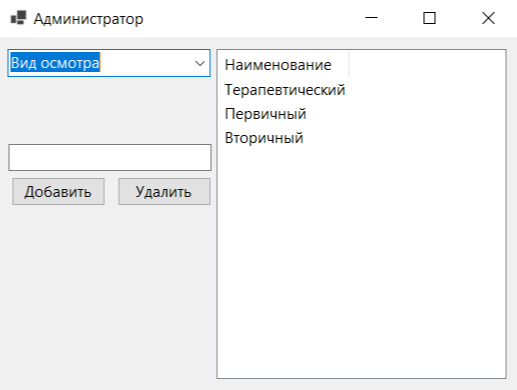


Рисунок 20 – виды осмотров

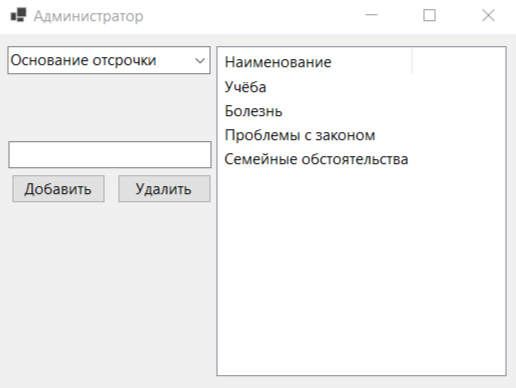


Рисунок 21 – основания отсрочки

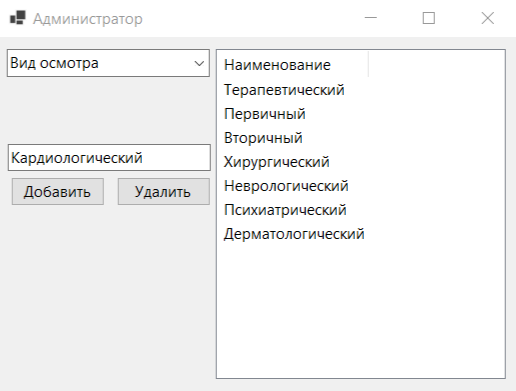


Рисунок 22 – ввод нового вида

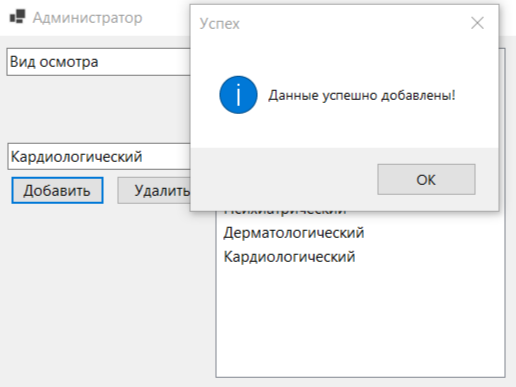


Рисунок 23 – успешное добавление

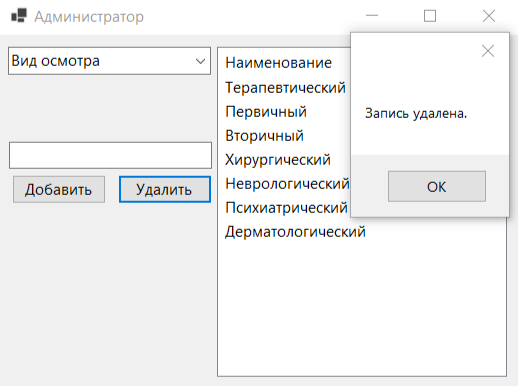


Рисунок 24 – удаление записи

Форма комиссара, показанная на рисунке 7 имеет следующие функции:

1) Выведение истории событий, происходящих при использовании программы.

2) Фильтрация записей по таблицам, происходящим действиям и временному периоду. Результат показан на рисунках 25 - 26.

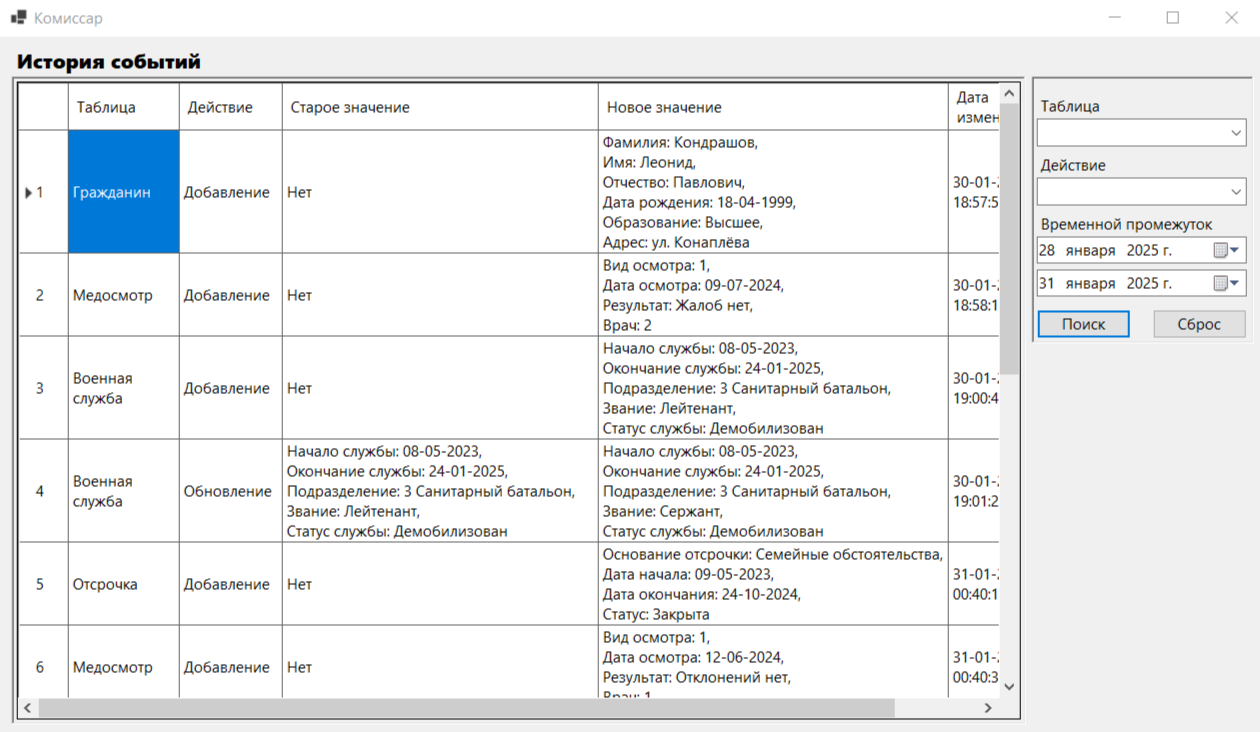


Рисунок 25 – фильтрация записей

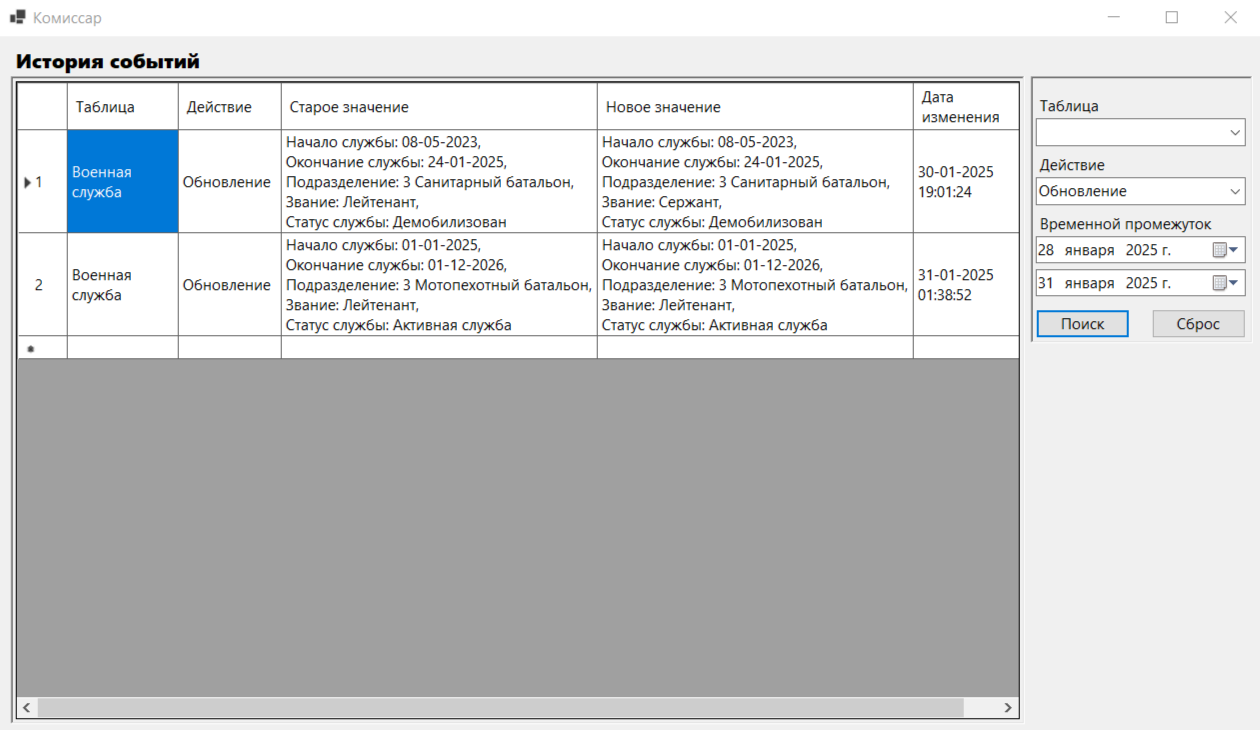


Рисунок 26 – фильтрация записей

3.3 Руководство программиста

3.3.1 Введение

Проект написан на языке программирования C# в среде Visual Studio с использованием базы данных SQLite. Проект включает в себя формы для различных типов пользователей, таких как сотрудники, граждане, администраторы и комиссары.

3.3.2 Установка и настройка системы

Требования к системе:

* Visual Studio 2019 или новее
* .NET Framework 4.7.2 или новее [6]
* Библиотека SQLite

Установка необходимых инструментов и библиотек:

* Откройте Visual Studio
* Создайте или откройте проект
* Установите пакет SQLite через NuGet Package Manager: Install-Package System.Data.SQLite

3.3.3 Структура проекта

Проект состоит из следующих файлов и директорий:

* AdminForm.cs: Форма для администратора.
* UserForm.cs: Форма для сотрудника.
* VoenForm.cs: Форма для военкома.
* Form1.cs: Форма входа в приложение.

3.3.4 Описание форм

1) AdminForm.cs:

* Конструктор AdminForm: инициализирует форму, вызывает методы для настройки элементов интерфейса, таких как comboBox и listViewData.
* Метод ConnectToDatabase: устанавливает соединение с базой данных SQLite. В случае ошибки подключения выводит сообщение об ошибке.
* Метод InitializeComboBox: настраивает комбинированный список comboBox, добавляя в него два элемента: "Вид осмотра" и "Основание отсрочки". Также подписывается на событие изменения выбора в comboBox.
* Метод InitializeListView: настраивает listViewData, устанавливая отображение в виде подробностей, разрешая полное выделение строк и запрещая множественный выбор строк. Подписывается на событие выбора строки в ListView и обработчик события кнопки удаления.
* Метод ListViewData\_ItemSelectionChanged: обрабатывает изменение выбранной строки в listViewData.
* Метод ComboBox\_SelectedexChanged: обрабатывает изменение выбранного элемента в comboBox. В зависимости от выбранного значения загружает соответствующие данные из базы данных.
* Метод LoadDataFromDatabase: загружает данные из выбранной таблицы базы данных в listViewData. Очистит ListView, выполняет запрос и добавляет данные в ListView. Также скрывает колонку с ID, чтобы отображались только значения.
* Метод DeleteFromDatabase: выполняет удаление записи из базы данных по переданному ID. Выполняет SQL-запрос на удаление и выводит сообщение об успешном удалении или ошибке.
* Обработчик buttonDelete\_Click: если строка в listViewData выбрана, она удаляется из ListView, и затем выполняется запрос на удаление из базы данных.
* Обработчик buttonAdd\_Click: получает введенное значение в textBoxInput, проверяет его и выполняет запрос на добавление в выбранную таблицу базы данных. После добавления данных обновляет отображение в ListView и очищает текстовое поле для ввода.

2) UserForm.cs:

* Конструктор UserForm: инициализирует форму, подключает базу данных, загружает данные и настраивает обработчики событий для отображения номеров строк в таблицах и заполнения выпадающих списков.
* Метод ConnectToDatabase: подключает к базе данных SQLite и обновляет статус отсрочек в зависимости от текущей даты.
* Метод LoadData: загружает данные о гражданах из базы данных в DataGridView1.
* Метод FillComboBoxes: заполняет ComboBox значениями для образования и статуса гражданина.
* Обработчик buttonRecord\_Click: добавляет нового гражданина в базу данных с проверкой возраста и обязательных полей, затем обновляет данные в таблице.
* Метод SubscribeToRowPostPaint: подписывает событие для отображения номеров строк в таблицах.
* Метод DataGridView\_RowPostPaint: рисует номер в заголовке строки таблицы.
* Обработчик buttonSearch\_Click: запускает поиск граждан по введённому запросу, вызывая SearchData.
* Метод SearchData: выполняет поиск граждан по полям (Фамилия, Имя, Образование и т.д.)
* Обработчик buttonClearFilter\_Click: очищает фильтры поиска и перезагружает все данные.
* Обработчик dataGridView1\_CellClick: загружает данные о выбранном гражданине в поля для редактирования и отображает связанные данные (отсрочки, медосмотры, военная служба).
* Метод LoadMilitaryService: загружает информацию о военной службе выбранного гражданина в DataGridView4.
* Обработчик buttonAddMilitaryService\_Click: добавляет запись о военной службе для выбранного гражданина с проверкой на активные отсрочки и корректность данных.
* Метод CheckActiveDeferments: проверяет наличие активных отсрочек для гражданина на текущую дату. Возвращает true, если есть активные отсрочки.
* Метод LoadRankComboBox: загружает список званий в ComboBox, устанавливая первое значение по умолчанию.
* Метод LoadStatusComboBox: загружает список статусов службы в ComboBox, устанавливая первое значение по умолчанию.
* Метод dataGridView4\_SelectionChanged: при выборе строки в таблице загружает данные службы в поля и сохраняет ID для дальнейших операций.
* Обработчик buttonSaveMilitaryService\_Click: сохраняет изменения по службе, проверяет корректность данных и обновляет соответствующую запись в базе данных.
* Метод UpdateRowInDataGridView: обновляет строку в DataGridView на основе изменения данных.
* Метод LoadDeferments: загружает данные отсрочек для выбранного гражданина и отображения их в таблице.
* Обработчик buttonSaveChanges\_Click: сохраняет изменения гражданина, проверяя обязательные поля и обновляя данные в базе.
* Обработчик buttonAddDeferment\_Click: добавляет отсрочку для выбранного гражданина, проверяя корректность дат и обязательных полей.
* Метод FillDefermentComboBox: заполняет ComboBox основаниями отсрочек, загружая их из базы данных.
* Метод LoadMedicalExams: загружает данные медицинских осмотров для выбранного гражданина на основе его ID. Выполняет SQL-запрос, чтобы извлечь информацию о медосмотрах из базы данных и отображает результаты в DataGridView3.
* Метод LoadComboBoxes: загружает данные для двух комбинированных списков: врачей и видов осмотров.
* Обработчик buttonAddExam\_Click: собирает информацию из комбинированных списков и текстовых полей, проверяет данные и добавляет новый медосмотр для выбранного гражданина в базу данных.
* Обработчик buttonDeleteDeferment\_Click: удаляет выбранную отсрочку для гражданина. Проверяет, что строка с отсрочкой выбрана в dataGridView2, подтверждает удаление и затем выполняет SQL-запрос для удаления записи из базы данных.
* Обработчик buttonDeleteExam\_Click: удаляет выбранный медосмотр. Проверяет, что строка с медосмотром выбрана в dataGridView3, подтверждает удаление и выполняет запрос на удаление записи из базы данных.
* Обработчик buttonDeleteMilitaryService\_Click: удаляет запись о военной службе для выбранного гражданина. Проверяет выбор строки в dataGridView4, подтверждает удаление записи и выполняет SQL-запрос на удаление записи о военной службе.

3) VoenForm.cs:

* Конструктор VoenForm: инициализирует форму, устанавливает соединение с базой данных, загружает историю изменений, а также настраивает обработчики событий для кнопок поиска и сброса фильтров.
* Метод ConnectToDatabase: подключает к базе данных SQLite с использованием строки подключения. Если подключение не удалось, выводит сообщение об ошибке.
* Метод LoadHistoryData: загружает данные из таблицы в базу данных и отображает их в dataGridView1. Применяет форматирование для длинных текстов и настраивает отображение многострочного текста в таблице.
* Метод FormatLongTextColumns: проходит по всем строкам таблицы и форматирует значения в столбцах "Старое значение" и "Новое значение", если они содержат длинные строки (добавляет переносы строк).
* Метод FormatLongString: добавляет переносы строк после каждого запятой и пробела в длинных строках.
* Метод EnableMultiLineDisplay: настраивает отображение многострочного текста в столбцах DataGridView, автоматически подстраивает строки и столбцы под размер содержимого.
* Метод SearchHistoryData: выполняет поиск в таблице "История" по фильтрам: категория, действие, и диапазон дат. Запрос динамически строится в зависимости от выбранных фильтров, используя параметры для защиты от SQL инъекций.
* Обработчик buttonSearch\_Click: вызывает метод поиска данных (SearchHistoryData) при нажатии на кнопку поиска.
* Обработчик buttonReset\_Click: сбрасывает фильтры и загружает все данные без фильтрации, возвращая отображение к начальному состоянию.
* Метод DataGridView1\_RowPostPaint: номерует строки в заголовке строк dataGridView1.

4) Form1.cs:

* Конструктор Form1: инициализирует форму и выполняет настройку компонентов управления. После этого форма готова к обработке пользовательских действий, таких как ввод логина и пароля.
* Метод button1\_Click: этот метод обрабатывает нажатие кнопки на форме. Далее выполняется проверка введённых пользователем логина и пароля.

4. Тестирование АИС

Тестирование автоматизированной информационной системы – ключевой этап разработки, обеспечивающий её корректность, надёжность и безопасность. Первым шагом взаимодействия пользователя с приложением является окно авторизации (рисунок 27), предназначенное для идентификации и аутентификации. Этот элемент играет важную роль в защите данных и управления доступом к функционалу системы, что делает его приоритетным объектом тестирования.

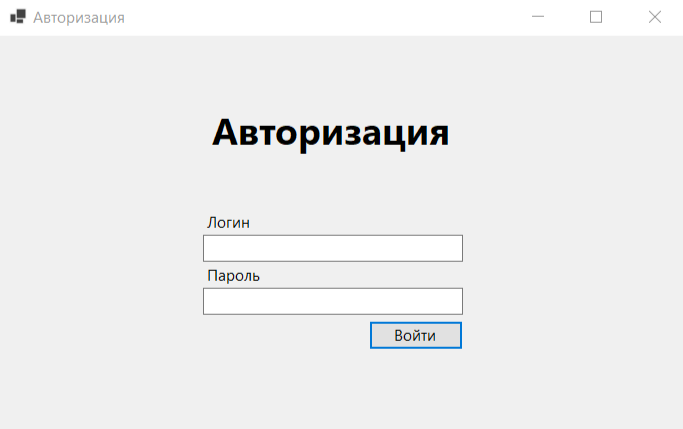


Рисунок 27 – окно авторизации

В окне авторизации пользователь вводит логин и пароль своей учётной записи. Если введённые данные корректны, открывается рабочее окно. В случае некорректного логина или пароля приложение выводит сообщение об ошибке. Иллюстрация этого процесса представлена на рисунке 28.

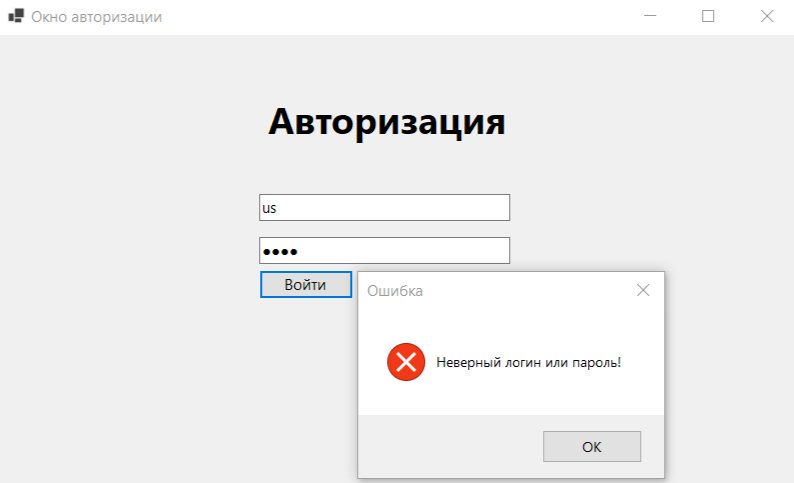


Рисунок 28 – уведомление об ошибке входа при неправильном логине или пароле

Когда пользователь добавляет гражданина, ему необходимо удостоверится что гражданин является совершеннолетним. Результат показан на рисунке 29.

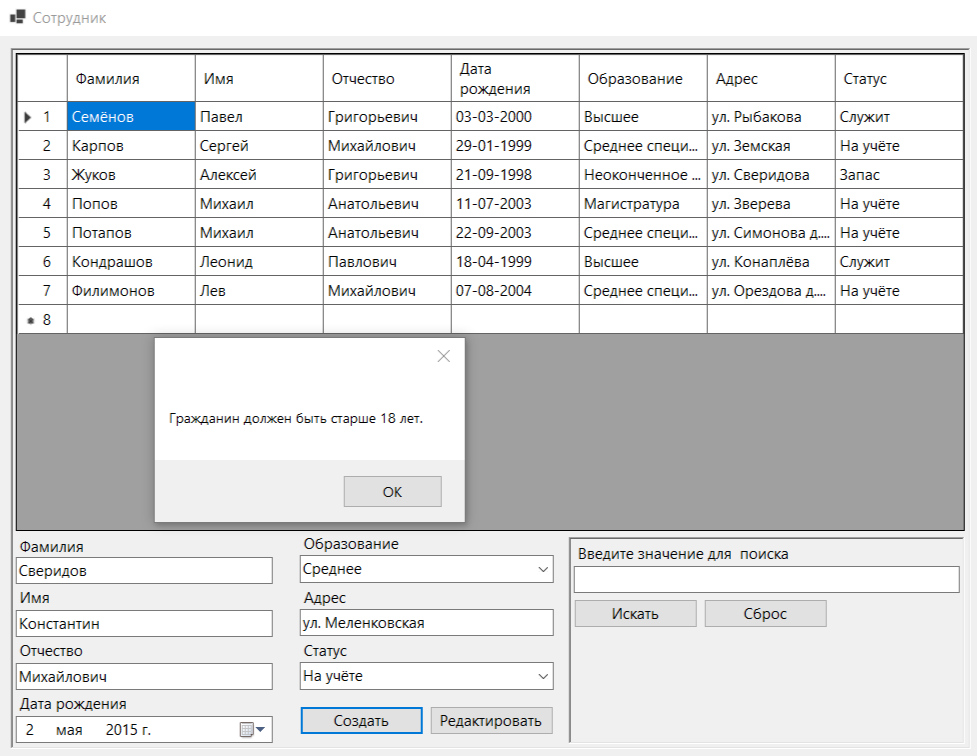


Рисунок 29 – уведомление о несоответствии возраста

Если в таблицах «Отсрочки», «Медосмотры», «Служба» пользователь попытается создать запись при этом не выбрав гражданина, то получить следующие сообщения, показанные на рисунках 30 – 32.

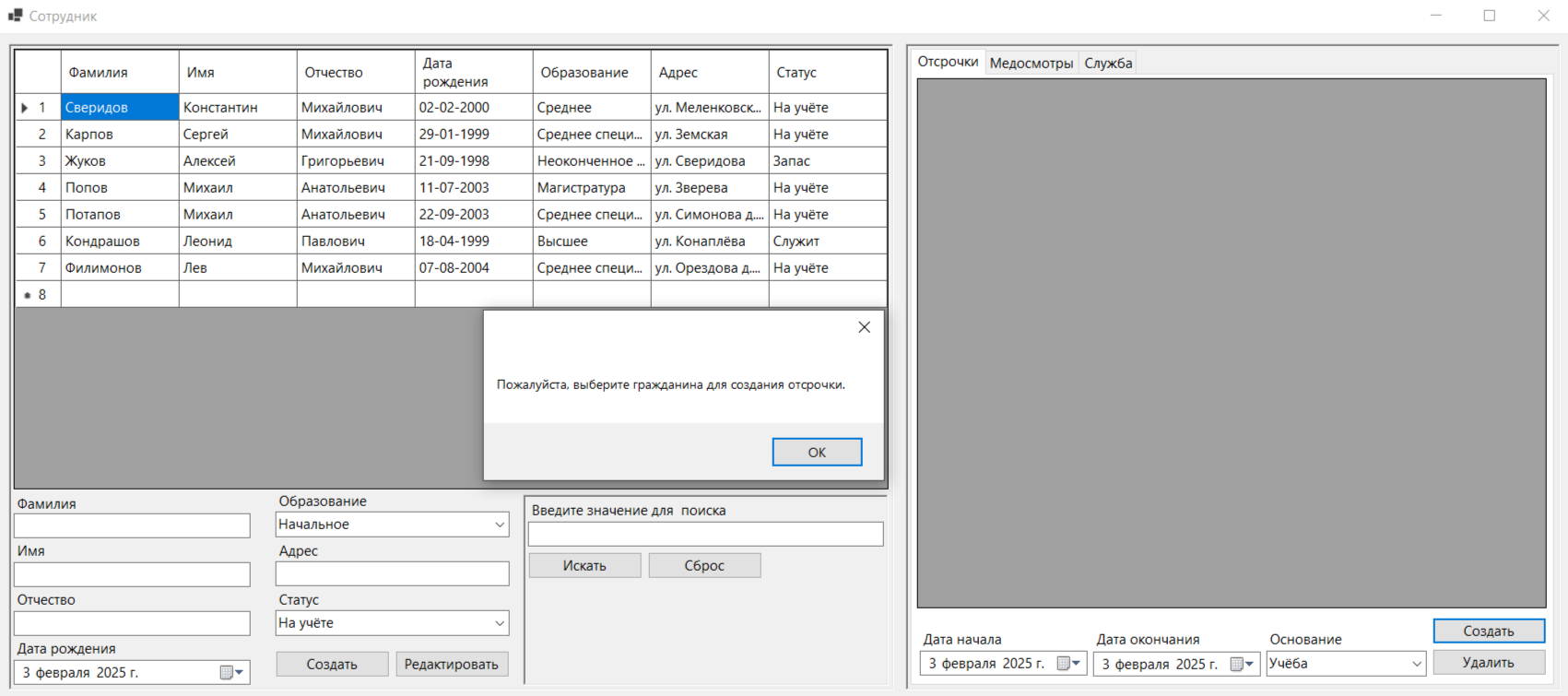


Рисунок 30 – уведомление о невыбранном гражданине для создания отсрочки

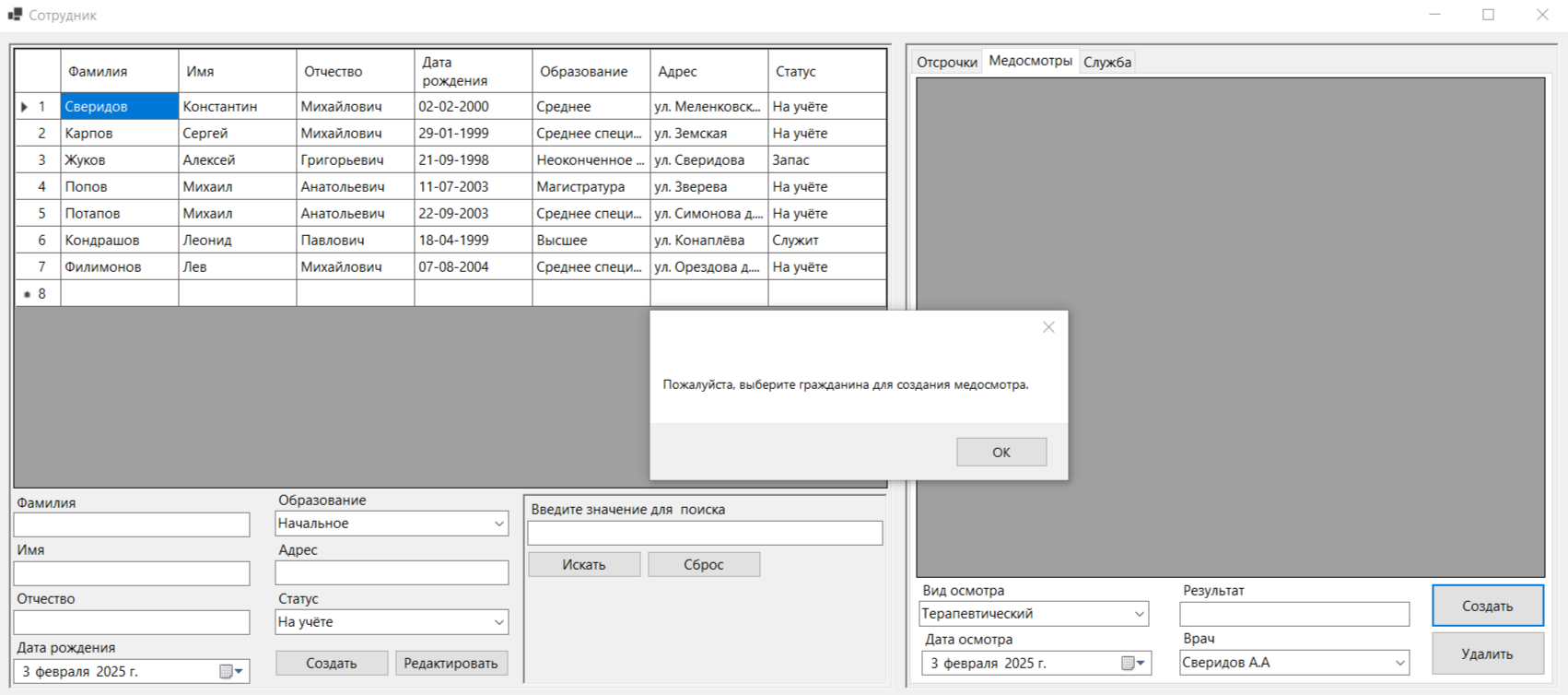


Рисунок 31 – уведомление о невыбранном гражданине для создания медосмотра

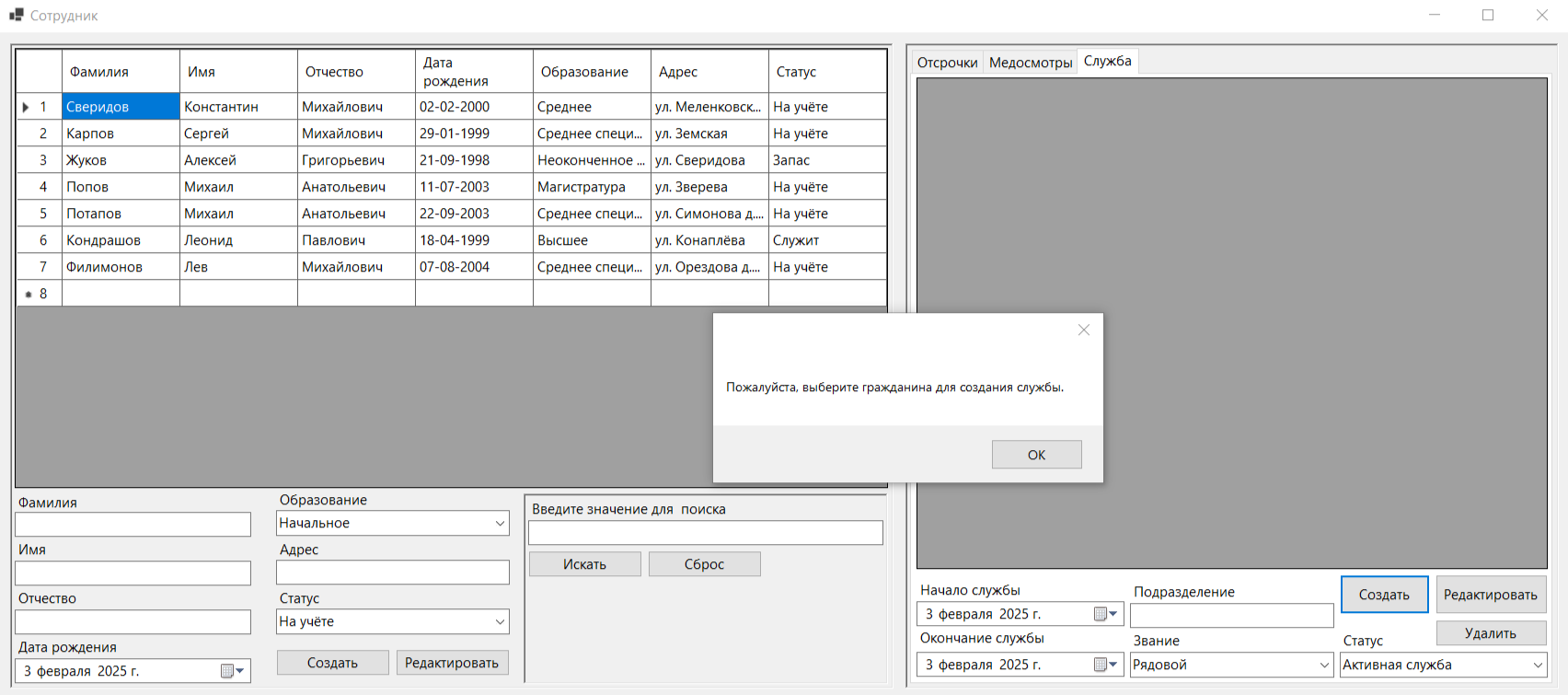


Рисунок 32 – уведомление о невыбранном гражданине для создания службы

Когда пользователю необходимо удалить запись в таблицах «Отсрочки», «Медосмотры», «Служба» и этом он не выделяет запись, которую необходимо удалить, то выводятся следующие сообщения 33 – 35.

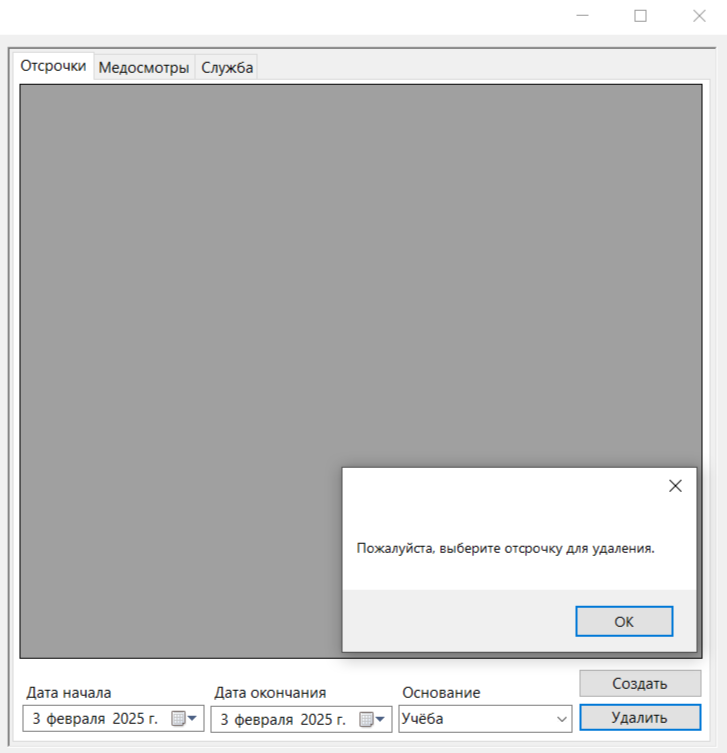


Рисунок 33 – уведомление о невыбранной отсрочки для удаления

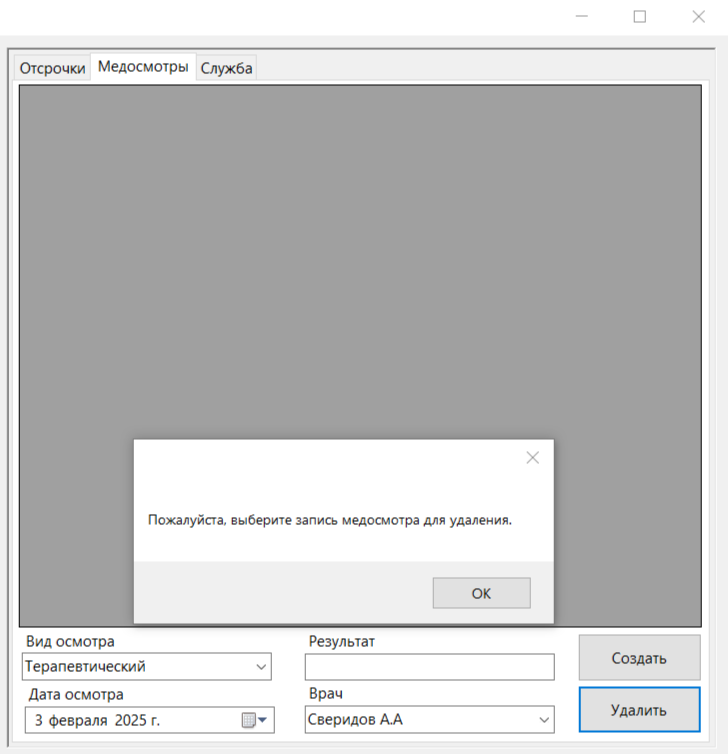


Рисунок 34 – уведомление о невыбранном медосмотре для удаления

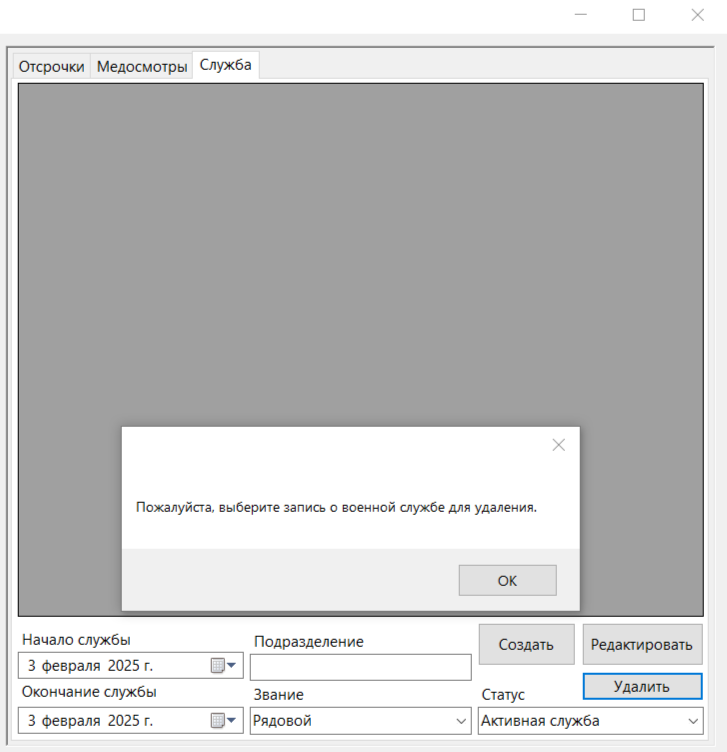


Рисунок 35 – уведомление о невыбранной службе для удаления

Когда пользователем создаётся запись о военной службе программой проверяется корректность дат, а также наличие активных отсрочек у гражданина в случае несоответствия пользователь получает следующие сообщения показанные на рисунках 36 – 37.

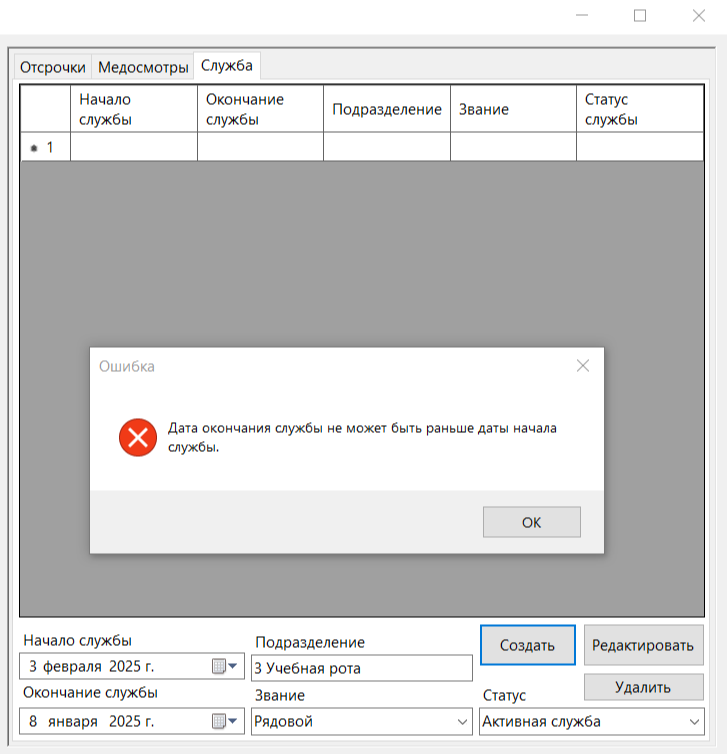


Рисунок 36 – уведомление о некорректности даты

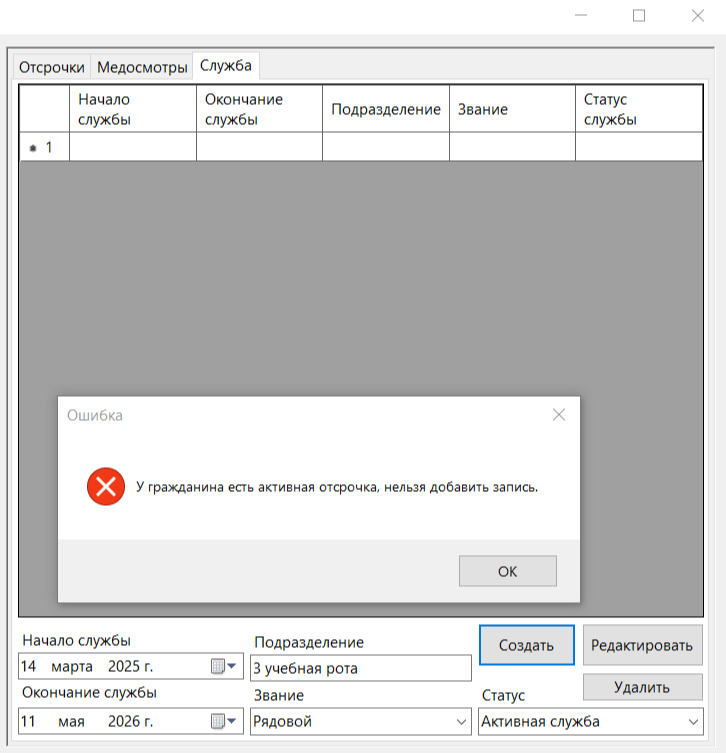


Рисунок 37 – уведомление о наличии отсрочки

При удаление пользователем записей из таблиц ʺВид осмотраʺ, ʺОснование отсрочкиʺ при этом не выбрав значение, которое следует удалить пользователь получает следующее сообщение, показанное на рисунке 38.

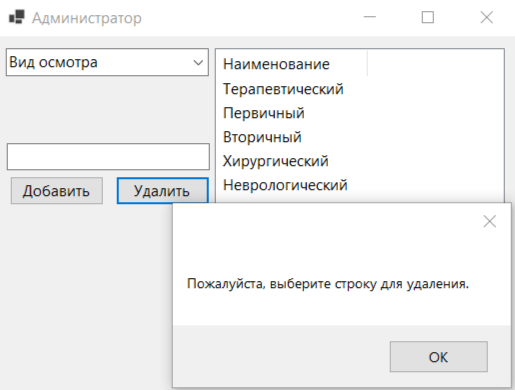


Рисунок 38 – уведомление о невыбранной записи

Когда в таблице «Вид осмотра» или «Основание отсрочки», необходимо добавить новую запись, и пользователь не введёт значение в текстовое поле, то при попытке добавления записи будет выведено следующее сообщение, показанное на рисунке 39.

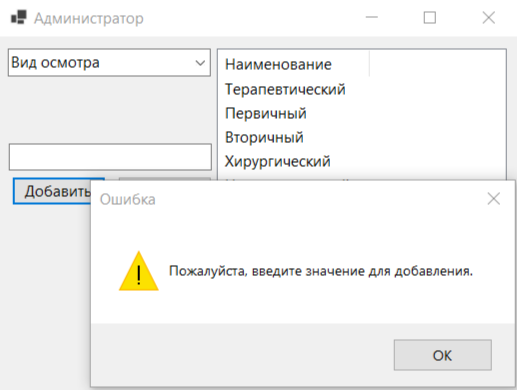


Рисунок 39 – уведомление об отсутствии значения

Если в процессе поиска данных в таблице по заданным фильтрам не будет найдено ни одной записи, пользователю будет выведено следующее сообщение, показанное на рисунке 40.

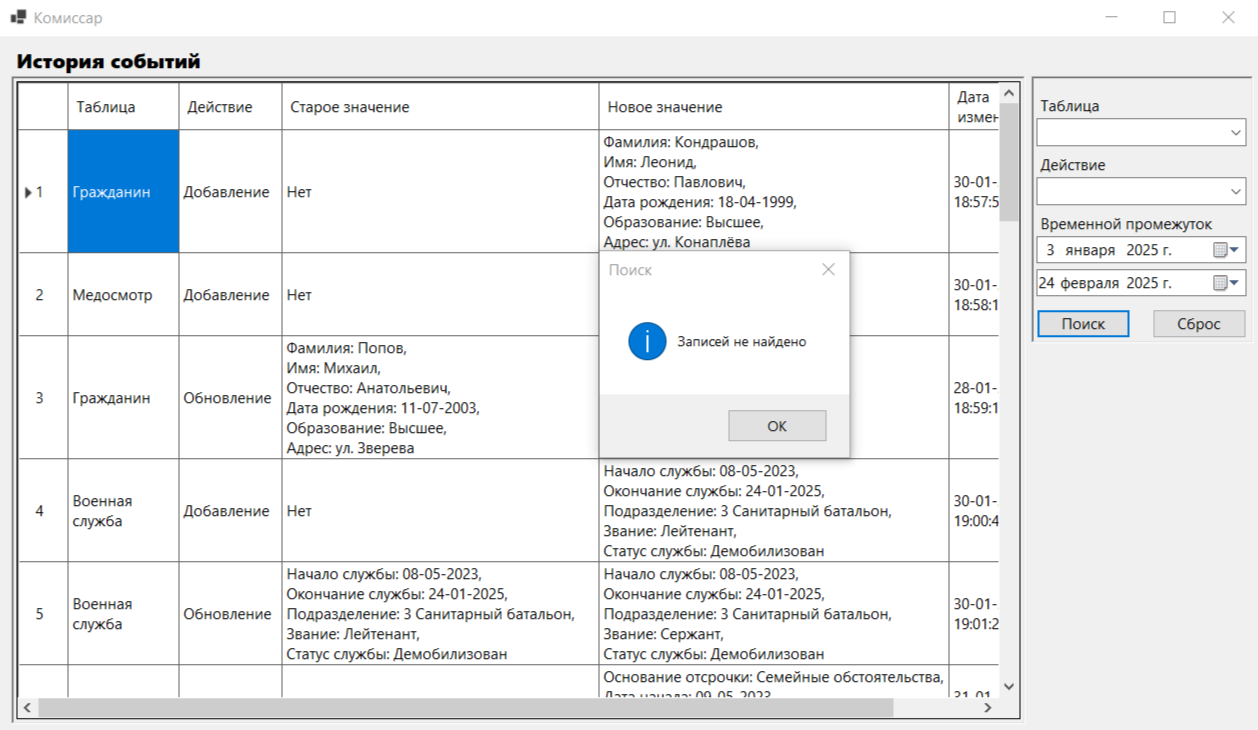


Рисунок 40 – уведомление об отсутствии найденных записей

Заключение

В процессе выполнения данной курсовой работы была разработана автоматизированная информационная система для военкомата, использующая реляционную базу данных для эффективного хранения и обработки данных.

Процесс разработки включал в себя анализ требований, проектирование базы данных и реализацию пользовательского интерфейса с использованием современных технологий. При этом особое внимание было уделено ключевым аспектам, таким как безопасность данных, целостность информации и удобство использования системы.

Разработанная АИС позволила автоматизировать ключевые процессы, что способствовало повышению эффективности работы военкомата и снижению вероятности ошибок. Также система обеспечивает высокий уровень безопасности данных и ускоряет принятие решений в различных ситуациях.

В рамках тестирования система показала высокую стабильность и успешность обработки всех операций, что подтверждает ее готовность к эксплуатации.

Таким образом, выполненная курсовая работа демонстрирует возможность создания надежной, эффективной и масштабируемой системы.

Список литературы

1. Гарсия-Молина, Х., Ульман, Дж. Д., Видом, Дж. **"Основы систем баз данных."** – Москва: Вильямс, 2020. – 256с.

2. Кауфман, О. **"SQLite для профессионалов."** – СПб.: Питер, 2020. – 476с.

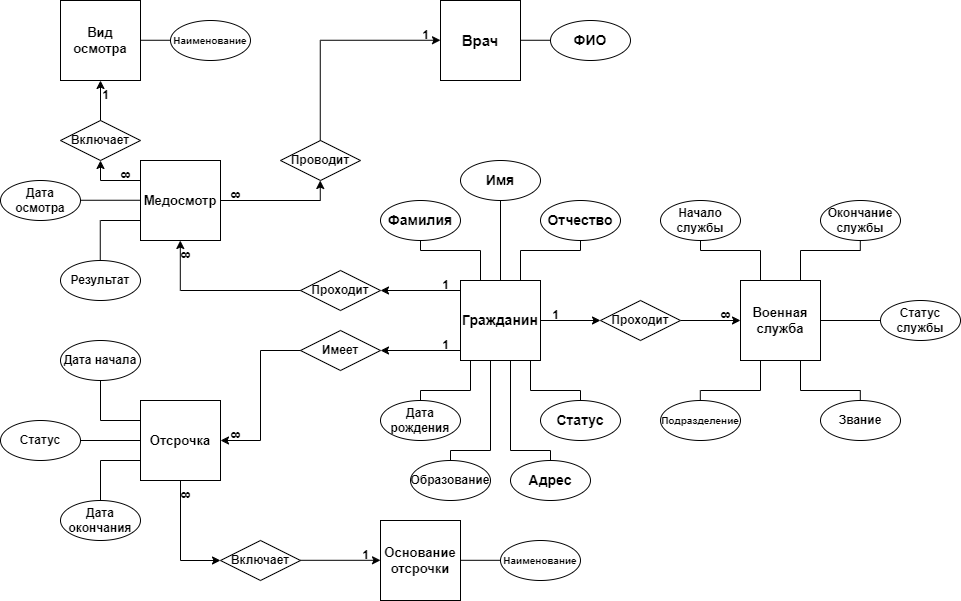
3. Мотев А. А. М85 Уроки MySQL. Самоучитель. — СПб.: БХВ-Петербург, 2006. — 208 с.: ил.

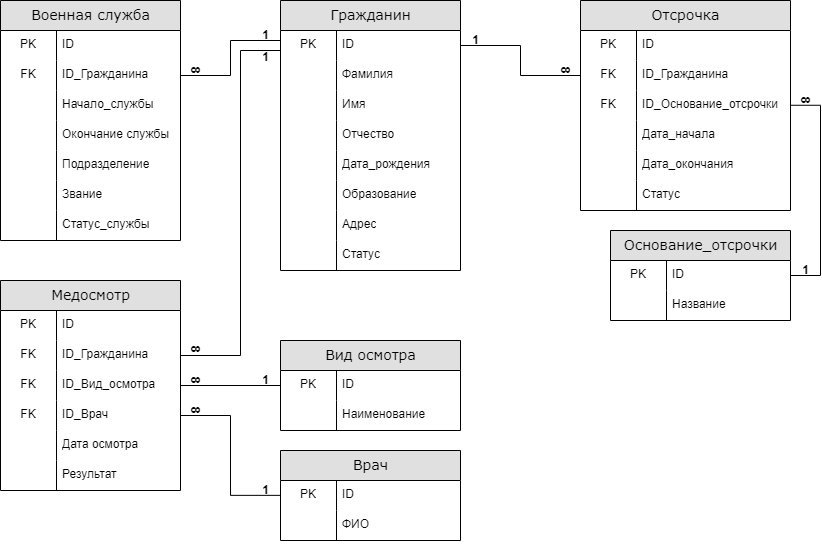
4. Разработка Windows Forms приложений на языке программирования C#: учебно-методическое пособие / Сост.: А.Ю. Са паров, Ижевск, 2020. 57 с.

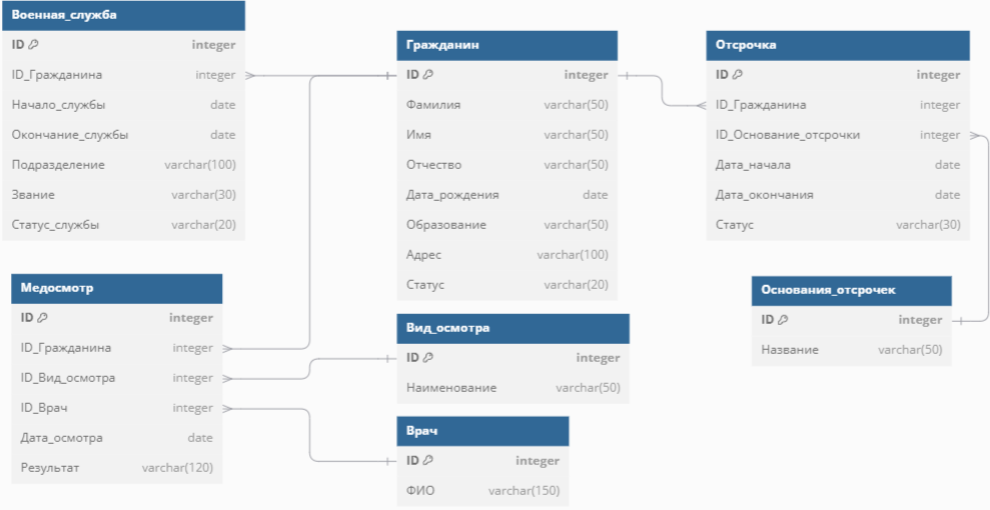
5. Сергеева Т.И. Базы данных: модели данных, проектирование, язык SQL: учеб. пособие / Т.И. Сергеева, М.Ю. Сергеев. Воронеж: ФГБОУ ВПО «Воронежский государственный технический университет», 2012. 233 с.

6. Хортон, М. "C# и .NET: руководство для начинающих." – СПб.: Символ-Плюс, 2019. – 272с.

Приложение 1. Модели данных







Приложение 2. Текст программы

Вся программа расположена на Git:

<https://github.com/Danila-coder/Voenkomat2.git>