**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

федеральное государственное автономное образовательное учреждение высшего образования

«Санкт-Петербургский политехнический университет Петра Великого» (ФГАОУ ВО «СПБПУ»)

**Институт среднего профессионального образования**

Отделение информационных технологий

Специальность: 09.02.07 Информационные системы и программирование

Дисциплина: МДК.11.01 Технология разработки и защиты баз данных

Группа: 22919/12

**КУРСОВОЙ ПРОЕКТ**

**на тему:**

**«Разработка информационной системы «Аэропорт»»**

Студент \_\_\_\_\_\_\_Лелюков М.А.\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(ФИО)

(подпись)

Руководитель \_\_\_\_\_\_\_Сынкова А.Д.\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

(ФИО)

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_ \_Сынкова А.Д.\_

(ФИО)

(подпись)

(оценка)

Санкт-Петербург

2025

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

федеральное государственное автономное образовательное учреждение высшего образования

«Санкт-Петербургский политехнический университет Петра Великого» (ФГАОУ ВО «СПБПУ»)

**Институт среднего профессионального образования**

Отделение информационных технологий

**Задание на курсовое проектирование**

**по МДК.11.01 «Технология разработки и защиты баз данных»**

Специальность: 09.02.07 Информационные системы и программирование

**студенту(ке) \_Лелюков** М.А.группа 22919/12

(фамилия, инициалы)

**Тема: Разработка базы данных информационной системы «Аэропорт»**

* + - * + Программа должна разграничивать уровни доступа: «Пассажиры», «Сотрудники аэропорта», «Главный администратор»
        + Программа должна обеспечивать для администратора возможность просматривать и управлять набором сотрудников, через таблицы «Пассажиры», «Сотрудники»;
        + Программа должна обеспечивать просмотр записей (возможность просматривать записи в виде таблиц);
        + Программа должна обеспечивать изменение записей (возможность редактировать уже существующие данные);
        + Программа должна обеспечивать удаление записей (возможность удаления существующих записей);
        + Программа должна обеспечивать поиск записей (возможность осуществлять поиск записей по выделенному полю);
        + Программа должна обеспечивать фильтрацию записей (Возможность фильтровать данные по различным параметрам: имя, дата и т.д.);
        + Программа должна обеспечивать создание отчетов (возможность выгружать данные таблиц в оформленные документы в формате .xls, .xlsx за определенный период);

**Руководитель**  Сынкова А.Д.

(подпись, дата) (инициалы, фамилия)

**Задание принял к выполнению:**

**Студент**  Лелюков М.А.

(подпись, дата) (инициалы, фамилия)

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 4](#_Toc201602217)

[1 Общий раздел 5](#_Toc201602218)

[1.1 Описание предметной области 5](#_Toc201602219)

[1.2 Постановка задачи 5](#_Toc201602220)

[1.3 Функции проектируемого приложения 6](#_Toc201602221)

[1.4 Анализ рынка существующих решений 7](#_Toc201602222)

[1.5 Обоснование и выбор технологии и инструментальных средств проектирования и разработки 8](#_Toc201602223)

[1.5.1 Технологии разработки клиента 8](#_Toc201602224)

[1.5.2 Технологии разработки сервера 9](#_Toc201602225)

[2 СПЕЦИАЛЬНЫЙ РАЗДЕЛ 11](#_Toc201602226)

[2.1 Диаграмма прецедентов использования 11](#_Toc201602227)

[2.2 Инфологическая модель базы данных 12](#_Toc201602228)

[2.3 Инфологическая модель базы данных 12](#_Toc201602229)

[2.4 Описание структуры базы данных 13](#_Toc201602230)

[2.4.1 Связи 13](#_Toc201602231)

[2.4.2 Ограничения 14](#_Toc201602232)

[2.5 Разработка пользовательского интерфейса 14](#_Toc201602233)

[2.6 Разграничение прав доступа 18](#_Toc201602234)

[2.7 Тестирование информационной системы 21](#_Toc201602235)

[2.8 Справочная информация 21](#_Toc201602236)

[ЗАКЛЮЧЕНИЕ 23](#_Toc201602237)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧИНКОВ 24](#_Toc201602238)

[ПРИЛОЖЕНИЕ А 25](#_Toc201602239)

[ПРИЛОЖЕНИЕ Б 30](#_Toc201602240)

[ПРИЛОЖЕНИЕ В 32](#_Toc201602241)

# **ВВЕДЕНИЕ**

Информационные системы аэропортов — это ключевая часть глобальной индустрии, обеспечивающая эффективное управление внутренними процессами для обслуживания пассажиров, авиакомпаний, наземных служб и административного персонала. Большинство операций, включая координацию рейсов, управление багажом, контроль расписания и взаимодействие между службами, осуществляется через специализированные цифровые платформы. Эти решения доступны на различных устройствах — от компьютеров и терминалов до мобильных устройств персонала, что гарантирует оперативность, точность и безопасность в управлении аэропортом.

Данный проект создается в учебных целях и не предназначен для конкуренции с ведущими системами, такими как «Авиасейлс» или «S7». Однако популярность подобных решений открывает перспективы для разработки «улучшенной» версии с дополнительным функционалом, которая могла бы использоваться, например, в небольших аэропортах.

Актуальность таких систем обусловлена масштабами этой отрасли. По данным Международной ассоциации воздушного транспорта (IATA), в 2024 году авиакомпании перевезли около 4,5 миллиарда пассажиров по всему миру, а объем мирового рынка авиаперевозок достиг $800 миллиардов. В 2025 году ожидается рост пассажиропотока на 8%, что подчеркивает необходимость надежных и современных информационных систем для управления увеличивающимся числом рейсов и пассажиров.

# **Общий раздел**

## **Описание предметной области**

Информационные системы играю невероятно важную роль в управлении сложными процессами в аэропортах, обеспечивая эффективную обработку данных о рейсах, пассажирах, обслуживании рейсов и сотрудниках.

Разрабатываемая информационная система для аэропорта предназначена для полной автоматизации процессов управления рейсами. Регистрации пассажиров, обработкой обслуживания и координации служб аэропорта. Система ориентирована как на сотрудников аэропорта, обеспечивая контроль над данными так и на пассажиров, предоставляя инструменты для просмотра расписания рейсов, регистрации на эти рейсы. Реализованная с использованием Windows Forms и SQL Server, она обеспечивает надежное хранение данных, разграничение прав доступа и удобный интерфейс для работы с информацией о рейсах, пассажирах п прочих операционных процессов аэропорта.

## **Постановка задачи**

Разработка информационной системы «Аэропорт» обусловлена необходимостью упростить управление процессами, улучшить качество обслуживания пассажиров и обеспечения стабильной работы информационной системы в условии роста пассажиропотока.

Информационная система «Аэропорт» создает единую платформу для управления рейсами, регистрации пассажиров, и общей координации служб и сотрудников. Реализованная с использованием Windows Forms и SQL Server она обеспечивает производительность, надежность и защиту данных. Основной функционал включает в себя, управлению расписанием, регистрацию на рейсы, управлению обслуживанием и т.д.

Особое внимание уделено разделению ролей: пассажиры через специализированную форму могут смотреть расписание рейсов и регистрироваться на удобный. Пассажиры вводят свои данные через специальное окно, после чего, если данные корректны, пассажир считается зарегистрированным, соответствующая запись внесется в базу данных. В свою очередь сотрудники могут войти под своим именем и просмотреть свои данные, а именно: дату выплаты зарплаты, дата выхода в отпуск, текущие рейсы. Через эту же форму администратор может попасть в свою панель, где может видеть и управлять всеми данными.

Дополнительно, система поддерживает формирование отчетов, в программе Excel, для удобного переноса и анализа данных таблиц. Система также способствует улучшению администрирования за счет функции поиска по конкретному значения поля, что особенно важно при огромном количестве данных в БД.

## **Функции проектируемого приложения**

Система обладает следующей базовой функциональностью, обеспечивающей автоматизацию процессов управления аэропорта:

* + - * + Вход в систему с использованием логина и пароля для администратора, с проверкой учетных данных через базу данных SQL Server;
        + Просмотр записей (возможность просматривать записи в виде таблиц);
        + Удаление записей (возможность удаления существующих записей);
        + Поиск записей (возможность осуществлять поиск записей по выделенному полю);
        + Фильтрация записей (Возможность фильтровать данные по различным параметрам: имя, дата и т.д.);
        + Создание отчетов (возможность выгружать данные таблиц в оформленные документы в формате .xls, .xlsx за определенный период);

## **Анализ рынка существующих решений**

Одним из аналогов разрабатываемого курсового проекта является система IS-AODB, представляющая собой мощное информационное ядро для комплексного управления операционной деятельностью аэропорта. Эта система, схожая по своим возможностям с широко известной платформой Amadeus Airport Operational Database, активно применяется в авиационной отрасли для централизованного управления данными о рейсах, ресурсах и операциях. IS-AODB обеспечивает интеграцию с различными системами аэропорта, такими как FIDS (информационные табло), DCS (системы регистрации пассажиров) и ERP, что делает ее универсальным решением для крупных авиаузлов.

В отличии от данной курсовой работы, IS-AODB обладает более широким функционалом, включая поддержку сложных расписаний с периодичностью (например, «каждый второй понедельник»), автоматизацию сообщений в формате IATA (MVT, DIV, LDM) и возможность мониторинга операций через мобильное приложение. Это приложение позволяет сотрудникам аэропорта в реальном времени отслеживать статус рейсов, координировать действия служб и оперативно реагировать на изменения (Рисунок 1).

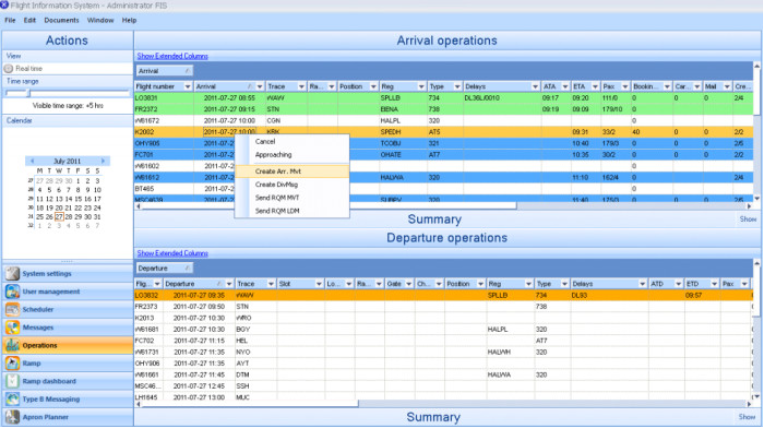


Рисунок 1 – система IS-AODB

Разрабатываемая система ориентирована на малые и средние аэропорты, предлагая более компактное и простое в использовании решение. Реализованная на базе Windows Forms с интеграцией SQL Server, она обеспечивает надежное хранение данных, удобный интерфейс с пастельной цветовой палитрой и закругленными элементами, а также ролевой доступ для пассажиров, сотрудников и администраторов. Основной функционал включает автоматизацию процессов регистрации на рейсы, формирования аналитических отчетов. В сравнении с IS-AODB, система курсового проекта не включает мобильное приложение и имеет менее сложные возможности для управления расписаниями, но выигрывает в простоте внедрения и адаптации под нужды небольших аэропортов.

## **Обоснование и выбор технологии и инструментальных средств проектирования и разработки**

### Технологии разработки клиента

В качестве клиента была выбрана приложение с графическим интерфейсом. Было принципиально важно, чтобы программа работала быстро и компилировалась в исходный код. Мой выбор стоял между технологиями, которые мы изучали, именно: С++ (библиотека GTK) и C# (.NET framework) Сравнение проводилось по критериям: Быстрая разработка интерфейсов, и удобство работы с данными.

C++ в связке с библиотекой GTK, предоставляет инструменты для создания графических интерфейсов, но процесс разработки интерфейсов в C++ с GTK требует значительных усилий. GTK не имеет встроенных визуальных редакторов, сравнимых с современными IDE, и очень часто приходится писать код для размещения элементов интерфейса вручную. Это увеличивает время разработки, особенно для сложных интерфейсов, таких как панели управления рейсами или информационные киоски, где требуется множество элементов управления (кнопки, таблицы, графики).

C# в связке с .NET Framework же (предоставляет мощные визуальные редакторы в Visual Studio, позволяющие быстро проектировать интерфейсы с помощью drag-and-drop, а также готовые компоненты (например, DataGrid, Charts) ускоряют разработку сложных интерфейсов, таких как панели управления рейсами или информационные киоски.

Исходя из вышесказанного, а также полагаясь на собственный опыт работы с этими технологами, я выбрал С# с использованием .NET Framework

### Технологии разработки сервера

При выборе системы управления базами данных для курсового проекта, где данные будут храниться на сервере, пришлось сравнить две популярные системы: Microsoft SQL Server и PostgreSQL. Основным фактором, влияющим на выбор, стал уровень интеграции каждой из этих систем с выбранным языком программирования, который в данном случае ориентирован на экосистему С# .NET.

Microsoft SQL Server, благодаря своему инструменту управления SQL Server Management Studio (SSMS), предлагает встроенную нативную интеграцию с платформой .NET через библиотеку «System.Data.SqlClient». Это обеспечивает удобство разработки, высокую производительность и оптимальное взаимодействие между приложением и базой данных, минимизируя необходимость в дополнительных настройках или использовании сторонних решений. Такая нативная поддержка делает SQL Server оптимальным выбором для проектов, где важна глубокая интеграция с экосистемой Microsoft.

С другой стороны, PostgreSQL, будучи мощной и гибкой системой с открытым исходным кодом, для работы с .NET требует использования стороннего драйвера, такого как «Npgsql». Хотя этот драйвер хорошо зарекомендовал себя и активно поддерживается разработчиками, он всё же не является полноценной частью экосистемы .NET, что может привести к дополнительной настройке и трудности с использованием, отладке или оптимизации.

Учитывая основной критерий выбора, а также мой личный опыт использования этих систем, Microsoft SQL Server был выбран как наиболее подходящая СУБД для реализации серверной части курсового проекта. Это решение позволяет максимально эффективно использовать преимущества нативной поддержки, упрощая процесс разработки и обеспечивая более четкое взаимодействие между клиентом и сервером.

# **СПЕЦИАЛЬНЫЙ РАЗДЕЛ**

## **Диаграмма прецедентов использования**

Диаграмма прецедентов (Use Case) отражает взаимодействие пользователей с информационной системой. На ней представлены основные функции, доступные для каждой роли (Рисунок 2).



Рисунок 2 – Диаграмма прецедентов

## **Инфологическая модель базы данных**

Инфологическая модель (ER-диаграмма) отображает сущности системы и связи между ними (Рисунок 3).

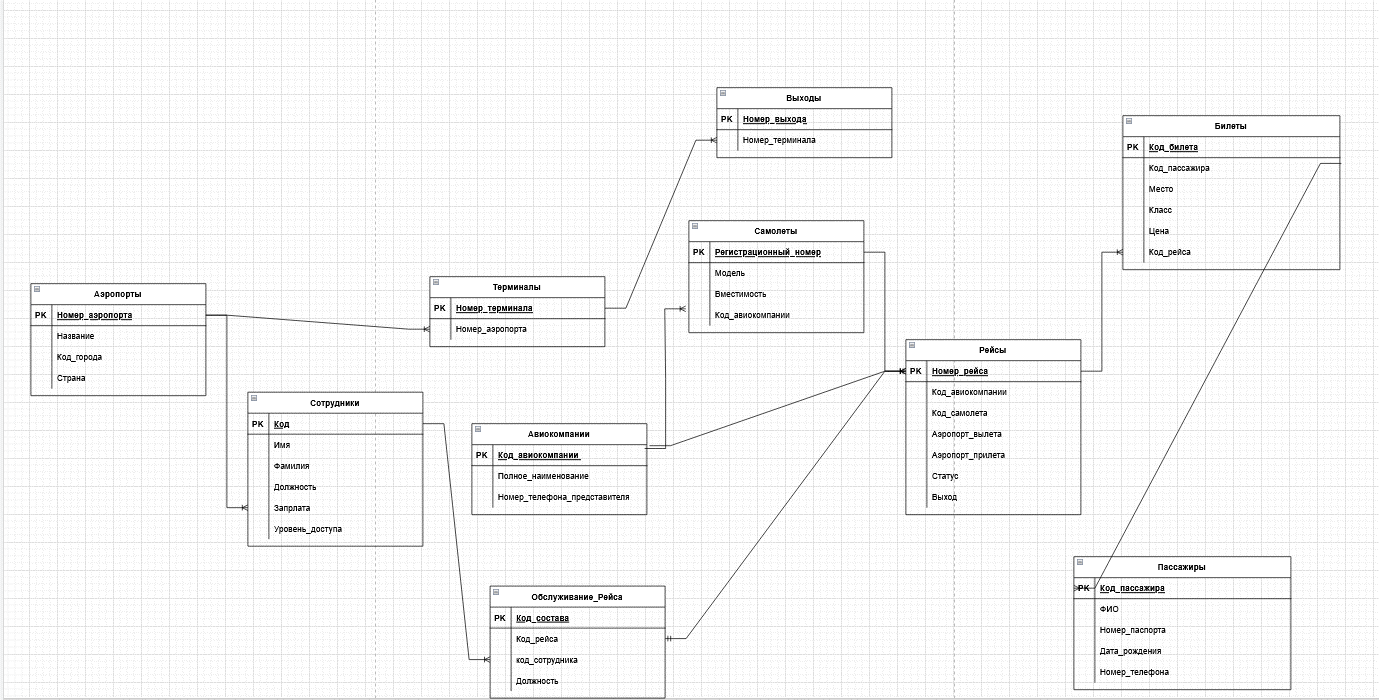


Рисунок 3 – Инфологическая модель

## **Инфологическая модель базы данных**

Даталогическая модель представляет собой детализированную инфологическую модель. Указываются типы данных полей базы данных (Рисунок 4).

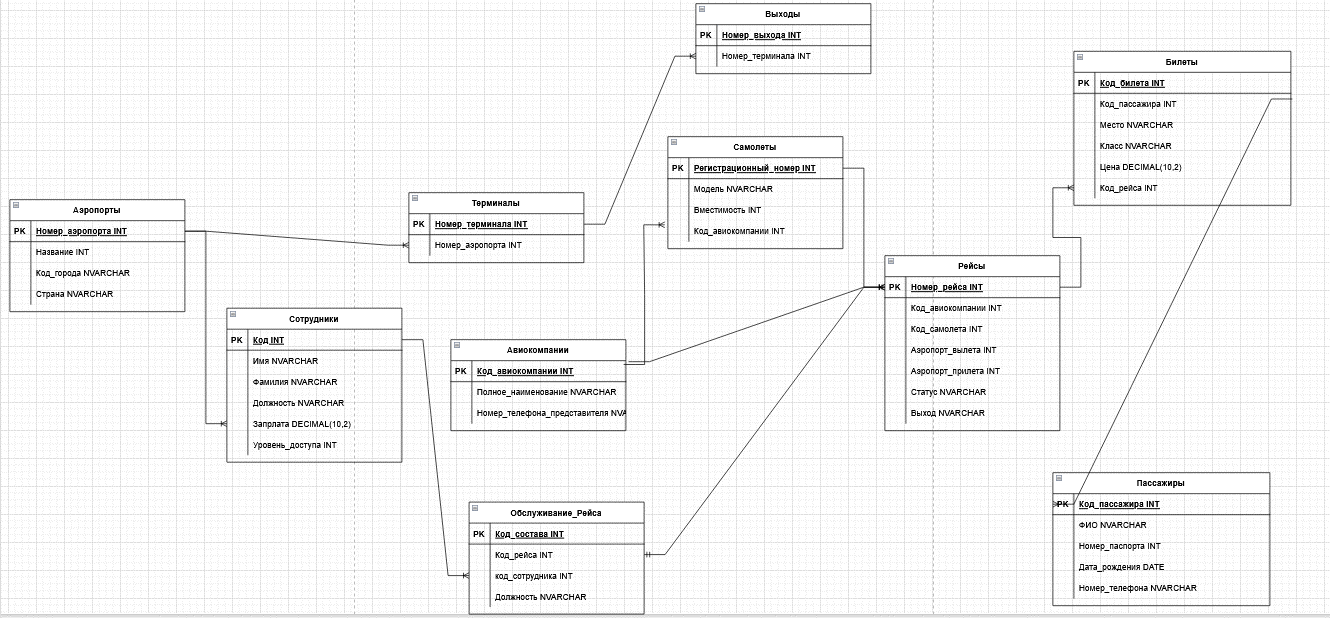


Рисунок 4 – Даталогическая модель

## **Описание структуры базы данных**

## 2.4.1 **Связи**

В базе денных представлены следующие связи:

**«Аэропорты» - «Терминалы»**: один ко многим. Один аэропорт может иметь несколько терминалов, каждый терминал принадлежит одному аэропорту.

**«Терминалы» - «Выходы»**: один ко многим. Один терминал содержит несколько выходов, каждый выход относится к одному терминалу.

**«Авиакомпании» - «Самолёты»**: один ко многим. Одна авиакомпания владеет несколькими самолётами, каждый самолёт принадлежит одной авиакомпании.

**«Самолёты» - «Рейсы»**: один ко многим. Один самолёт выполняет несколько рейсов, каждый рейс привязан к одному самолёту.

**«Выходы» - «Рейсы»**: один ко многим. Один выход используется для нескольких рейсов, каждый рейс привязан к одному выходу.

**«Рейсы» - «Билеты»**: один ко многим. На один рейс продаётся много билетов, каждый билет относится к одному рейсу.

**«Пассажиры» - «Билеты»**: один ко многим. Один пассажир покупает несколько билетов, каждый билет принадлежит одному пассажиру.

**«Рейсы» - «Обслуживание\_рейсов»**: один ко многим. Один рейс обслуживается несколькими сотрудниками, каждое обслуживание связано с одним рейсом.

**«Сотрудники» - «Обслуживание\_рейсов»**: один ко многим. Один сотрудник обслуживает несколько рейсов, каждое обслуживание связано с одним сотрудником.

## 2.4.2 **Ограничения**

Для обеспечения целостности базе данных были следующие ограничения (Приложение А)

## **Разработка пользовательского интерфейса**

Для отображения таблиц базы данных «Аэропорт» был реализован следующий пользовательский интерфейс (Рисунок 5-8). Пользователь может переключаться между категориями таблиц, и выбирать нужные ему таблицы.

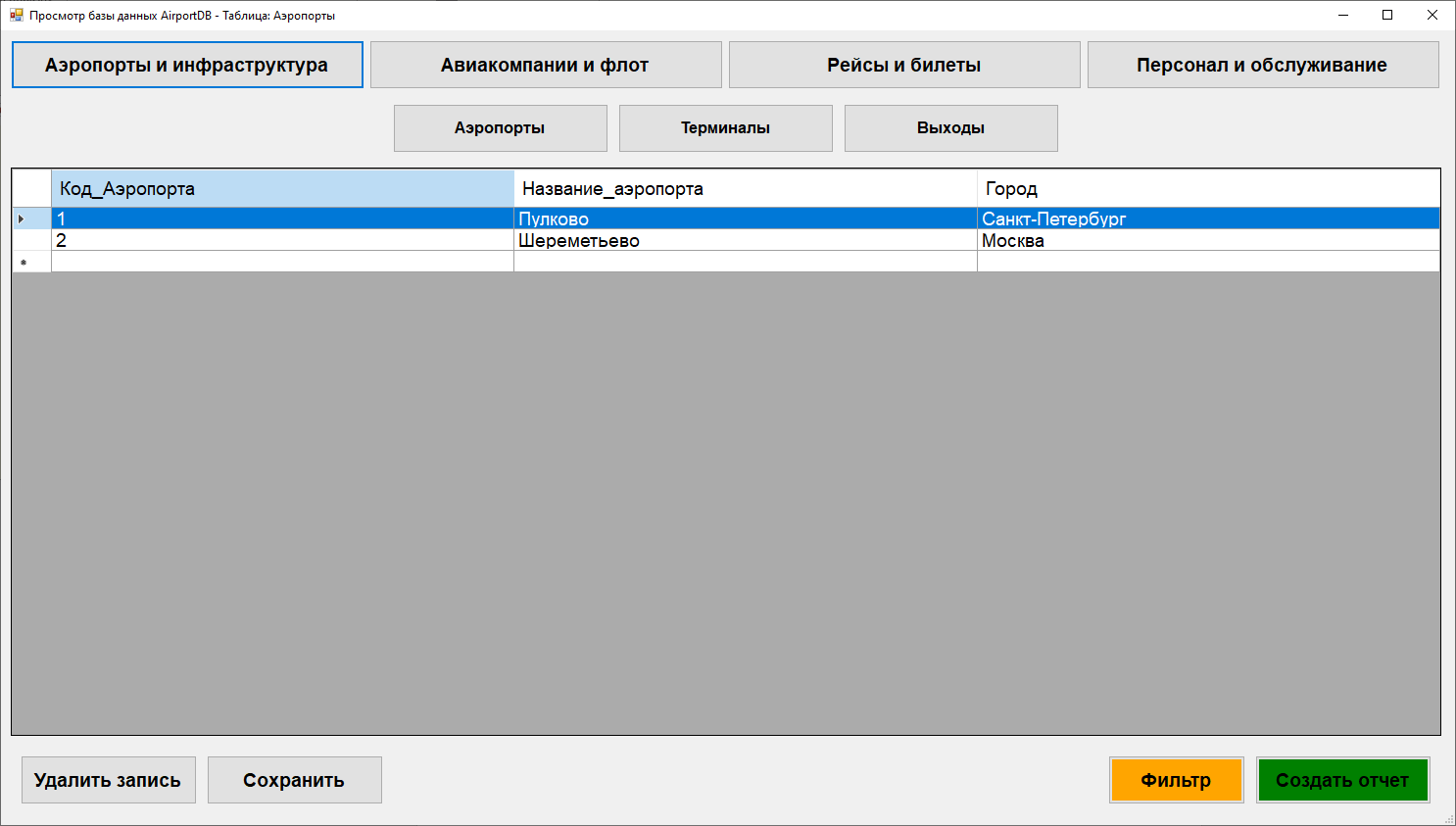


Рисунок 5 – Таблица «Аэропорты»

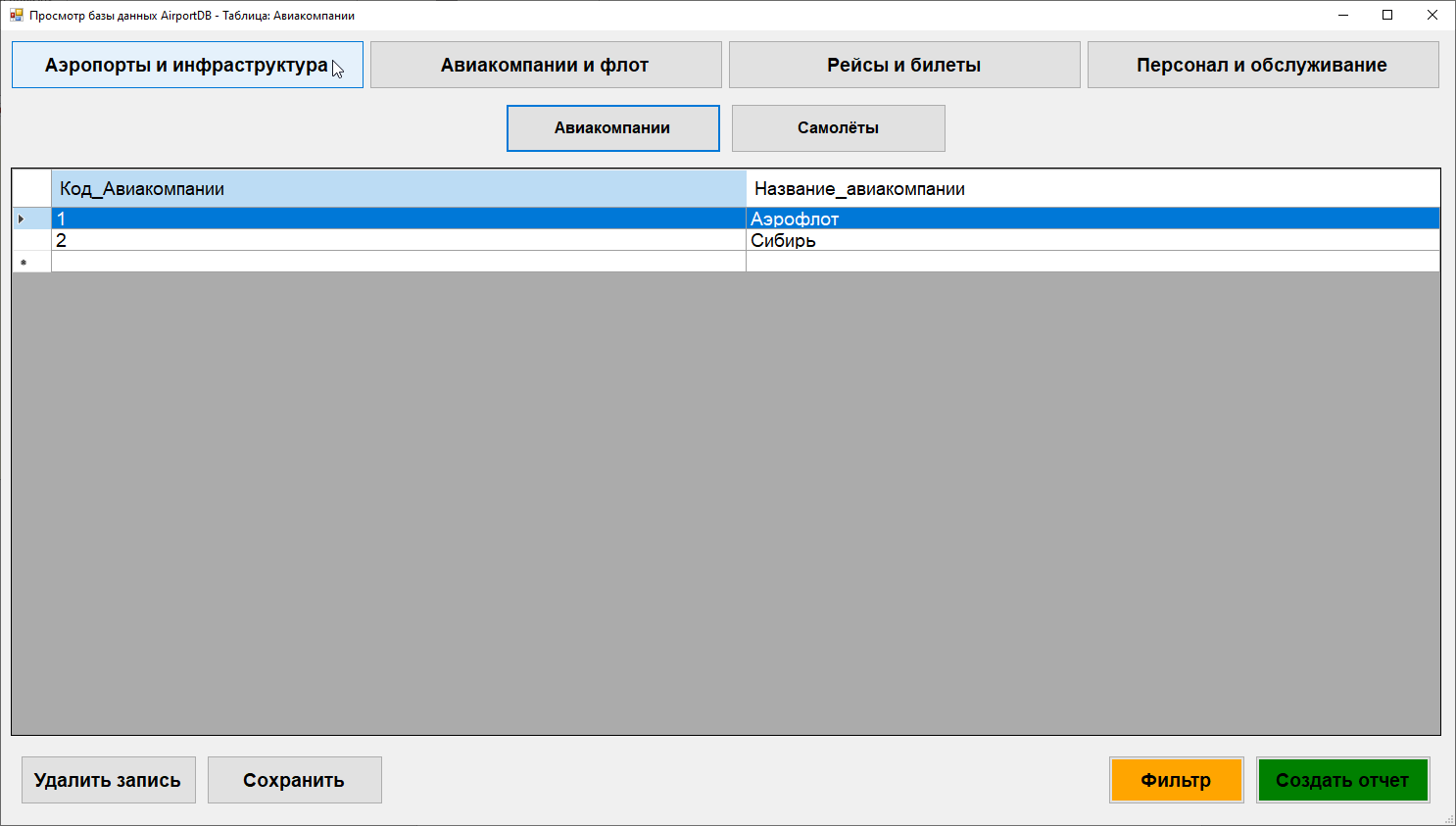


Рисунок 6 – Таблица «Авиокомпании»

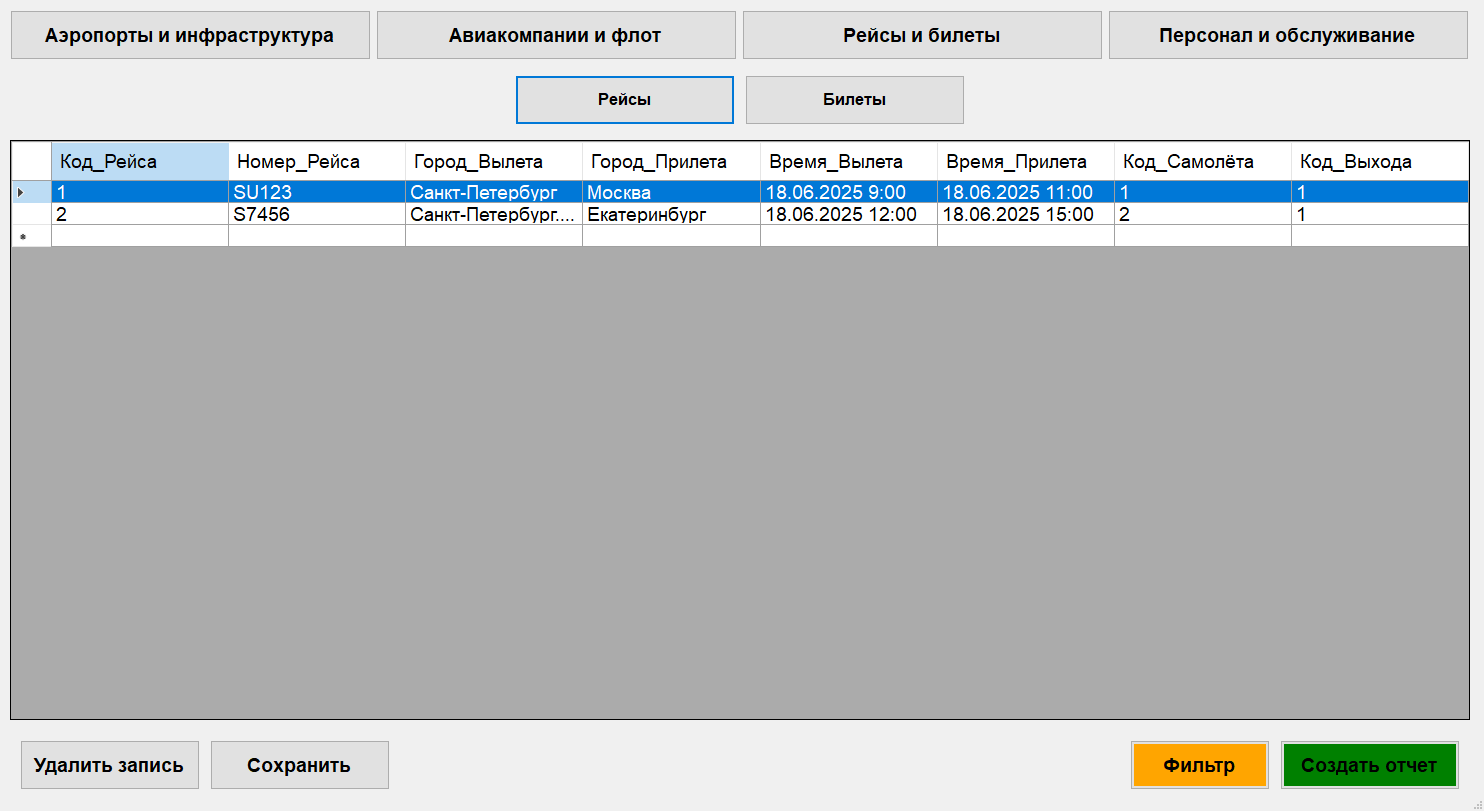


Рисунок 7 – Таблица «Рейсы»

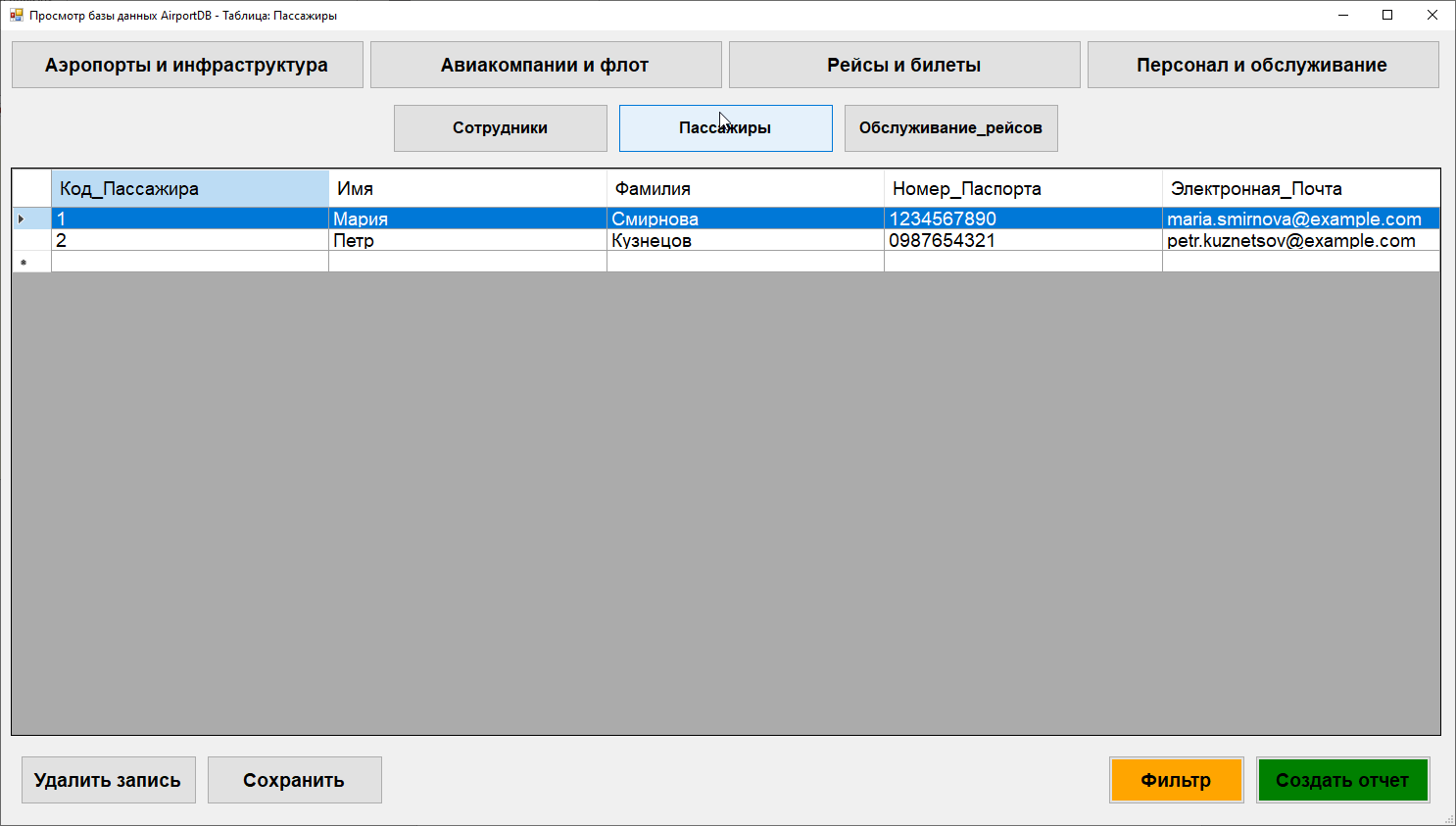


Рисунок 8 – таблица «Пассажиры»

Помимо полного взаимодействия с таблицами (добавление, удаление, изменение), пользователю также доступна опция фильтра записей по конкретному полю (Рисунок 9-10).

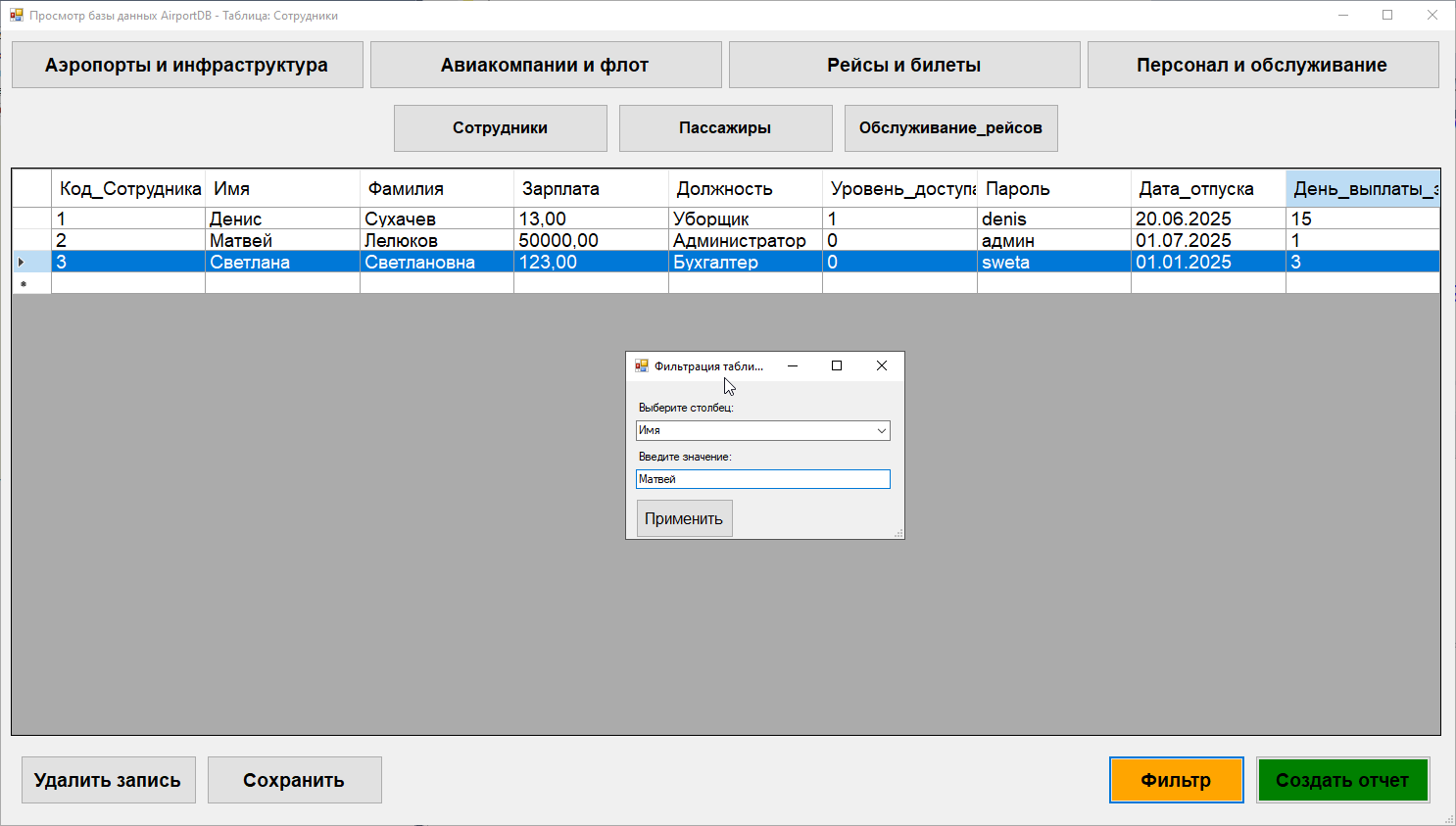


Рисунок 9 – окно фильтра записей

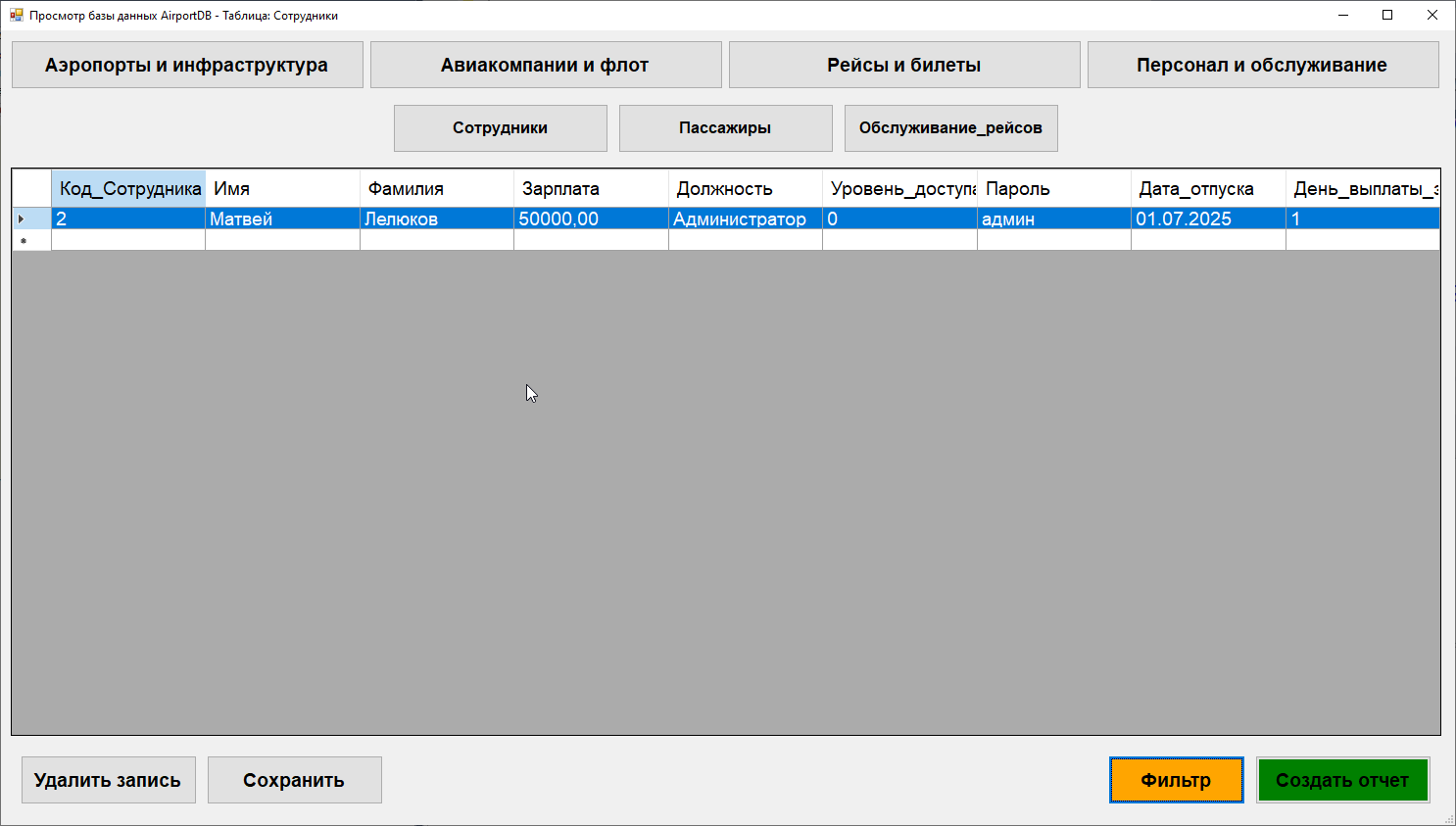


Рисунок 10 – отфильтрованные записи

Также у пользователя есть возможность перенести содержимое всех таблиц в формат .xls, нажав соответствующую кнопку (Рисунок 11)

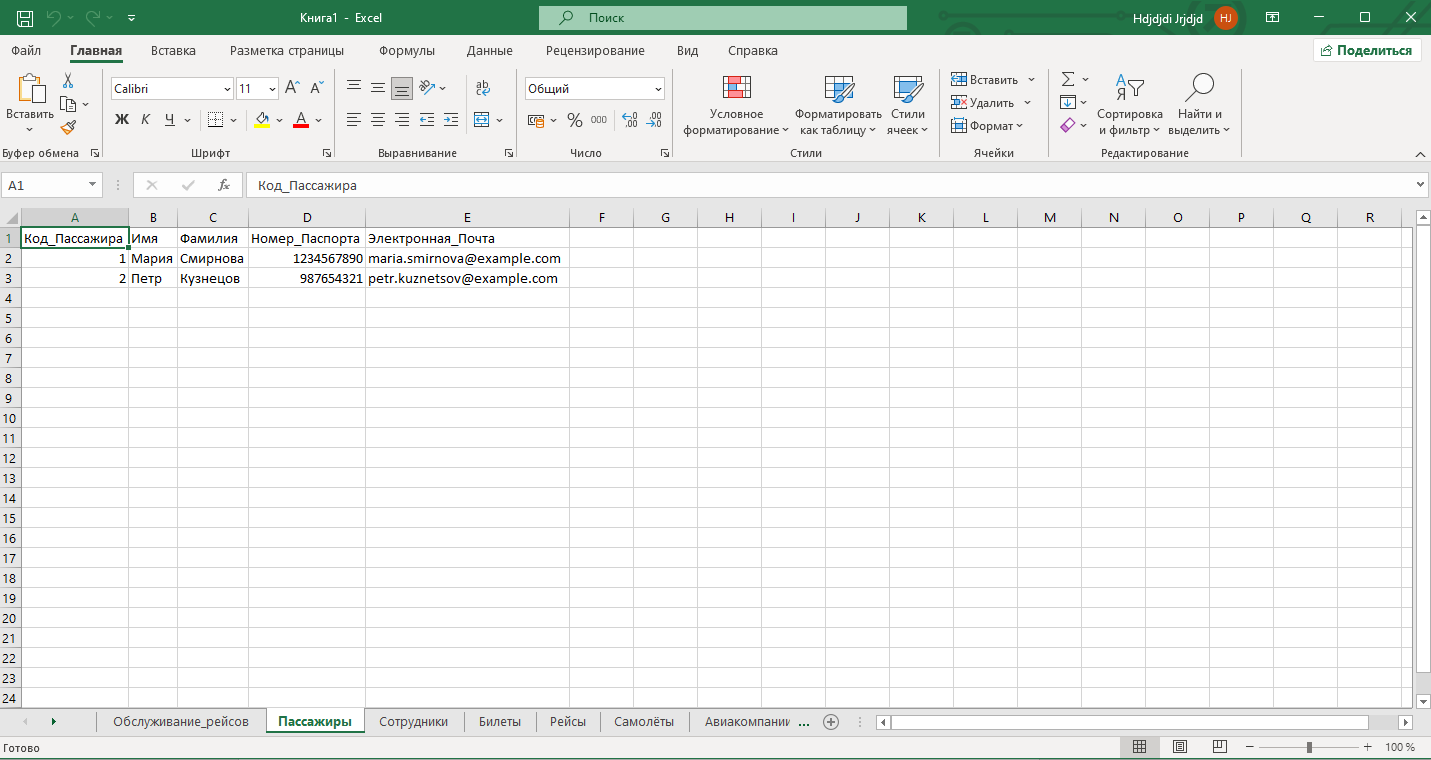


Рисунок 11 – выгруженные таблицы

## **Разграничение прав доступа**

Пользователи в системе имеют различный уровень доступа к данным в зависимости от назначенной роли. Роли «Администратор» доступны для чтения и изменения все таблицы. В роли «Пассажир» доступна совсем другая форма выбора и регистрации на рейс (Рисунок 12-15). В свою очередь для сотрудников, с ограниченном уровнем доступа доступна форма вывода их персональных данных (Рисунок 16-18).

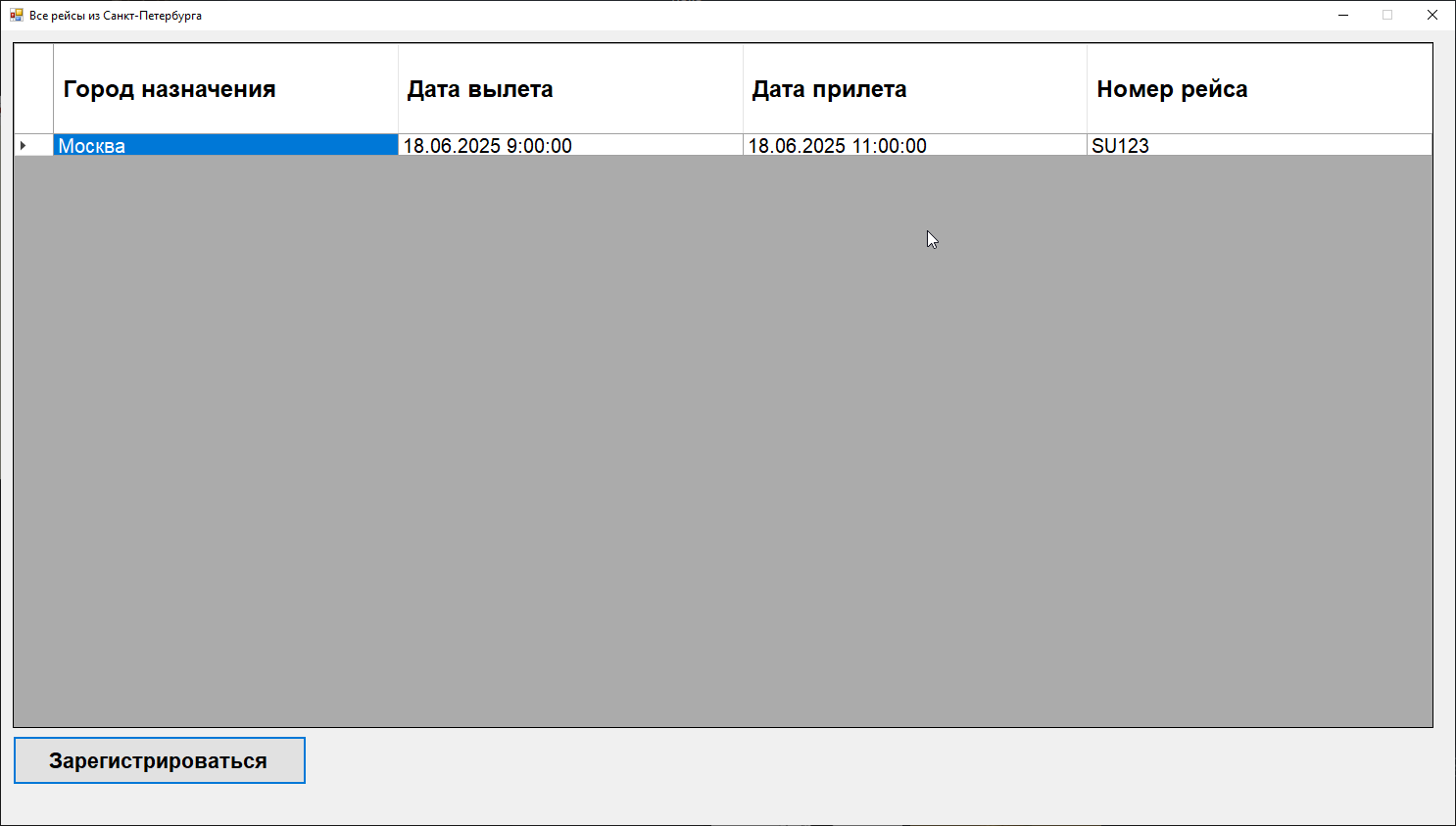


Рисунок 12 – форма выбора рейса пассажиром

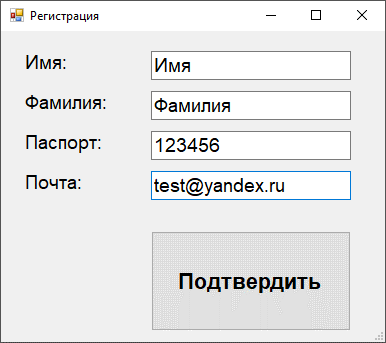


Рисунок 13 – заполнение окна регистрации

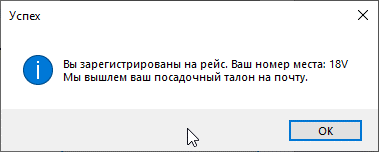


Рисунок 14 – Успешная регистрация

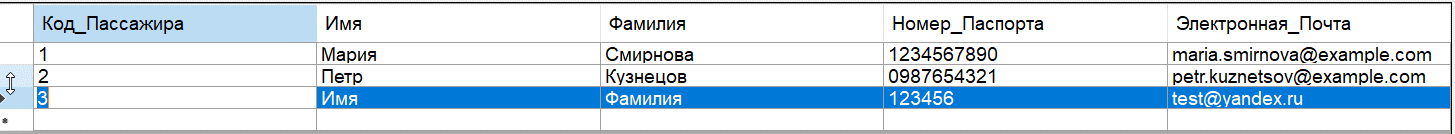


Рисунок 15 – новая запись в БД

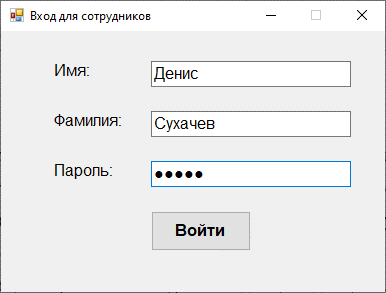


Рисунок 16 – форма авторизации сотрудника

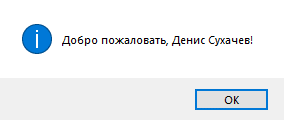


Рисунок 17 – успешный вход

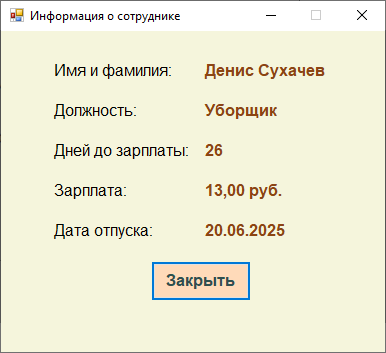


Рисунок 18 – данные сотрудника

## **Тестирование информационной системы**

Таблица 1 – Протокол испытаний

| **№** | **Проверяемые требования** | **Сообщения программы и вводимые значения** | **Ожидаемые результаты** | **Результат** |
| --- | --- | --- | --- | --- |
| 1 | Способность программы регистрировать нового пассажира на рейс | * Запустить программу; * Нажать кнопку «Купить билет» * Выбрать рейс и нажать «Зарегистрироваться» * Заполнить поля Имя: Светлана, Фамилия: Семьянова, Паспорт:123456, Электронная почта: [swetlana@yandex.ru](mailto:swetlana@yandex.ru) * Нажать кнопку «ОК» | Появится окно с успешной регистрации на рейс | Пройден  Рис. А.1 – А.4 |
| 2 | Способность программы обеспечивать авторизацию сотрудника аэропорта | * Запустить программу * Нажать «Вход для сотрудников» * Заполнить поля: Имя: Денис, Фамилия: Сухачев, Пароль denis * Нажать кнопку «Войти» | Откроется форма с данными о сотруднике | Пройден  Рис. А.4 – А.8 |
| 3 | Способность программы обеспечивать авторизацию администратора | * Запустить программу * Нажать «Вход для сотрудников» * Заполнить поля: Имя: Матвей, Фамилия: Лелюков, Пароль админ * Нажать кнопку «Войти» | Откроется форма администратора | Пройден  Рис. А.8 – А.11 |

## **Справочная информация**

Данная программа предназначена для полноценного управления аэропортом и рассчитана как на пассажиров, так и на сотрудников с административными, и не только, полномочиями. В приложении предусмотрена полноценное разделение пользователей на различные роли, поскольку система ориентирована на полноценное использованием всех групп. Такая архитектура обеспечивает полный контроль и над всеми функциями системы и использованием всеми группами пользователей.

Пользователи имеют полный доступ к функционалу программы, включая просмотр рейсов и регистрация на них.

Для обеспечения безопасности доступа реализована система авторизации, в которой каждый сотрудник вводит свое полное ФИО и пароль. Аутентификация позволяет ограничить доступ к программе посторонним лицам и сохранить целостность данных, включая информацию о пассажирах и их паспортных данных.

# **ЗАКЛЮЧЕНИЕ**

В ходе выполнения курсового проекта была разработана информационная система, способная повысить эффективность работы сотрудников аэропорта.

Разработанная система создает единую платформу для управления рейсами, регистрации пассажиров, и общей координации служб и сотрудников. Реализованная с использованием Windows Forms и SQL Server для надёжного хранения информации, система обеспечивает удобный ввод данных за счёт валидации, возможность создания отчетов, контроль доступа на основе ролевой модели (администратор, сотрудник, пассажир), быстрый поиск и фильтрацию данных по ключевым параметрам.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧИНКОВ**

1. Microsoft SQL Server Documentation [https://docs.microsoft.com/en-us/sql/sql-server, 18.06.2025]
2. Дейт, К. Дж. Д27 Введение в системы баз данных, 8-е издание.: Пер. с англ. — М.: Издательский дом "Вильяме", 2005. — 1328 с.: ил. — Парал. тит. англ.
3. Техническая документация по SQL Server [Электронный ресурс] — URL: https://learn.microsoft.com/ru-ru/sql/sql-server/?view=sql-server-ver17 (дата обращения: 03.05.2025)
4. Разработка с помощью Entity Framework Core [Электронный ресурс] — URL: https://learn.microsoft.com/ru-ru/ef/ (дата обращения: 03.05.2025)
5. А. Хейлсберг, М. Торгерсен, С. Вилтамут, П. Голд ., Язык программирования C#, 4-е издание, СПб:-ПИТЕР, 2022
6. Фленов М., Библия C#, 3-е издание, СПб:-БХВ, 2022
7. Описание языка C# [Электронный ресурс] — URL: https://techrocks.ru/2019/02/16/c-sharp-programming-language-overview/ (дата обращения: 03.12.2023)
8. Эванс, Эрик. Э 14 Предметно-ориентированное проектирование (DDD): структуризация сложных программных систем. : Пер. с англ. - М.: 000 "И.Д. Вильямс", 2011. - 448 с. : ил. - Парал. тит. Англ
9. Документация по Visual Studio [Электронный ресурс] — URL: https://learn.microsoft.com/ru-ru/visualstudio/windows/?view=vs-2022 (дата обращения: 03.05.2025)

# **ПРИЛОЖЕНИЕ А**

(справочное)

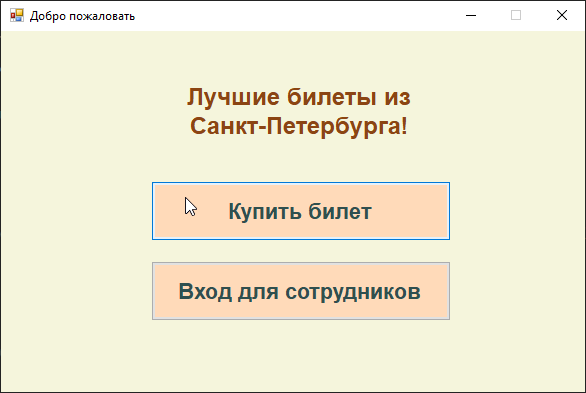


Рисунок А.1 – форма входа

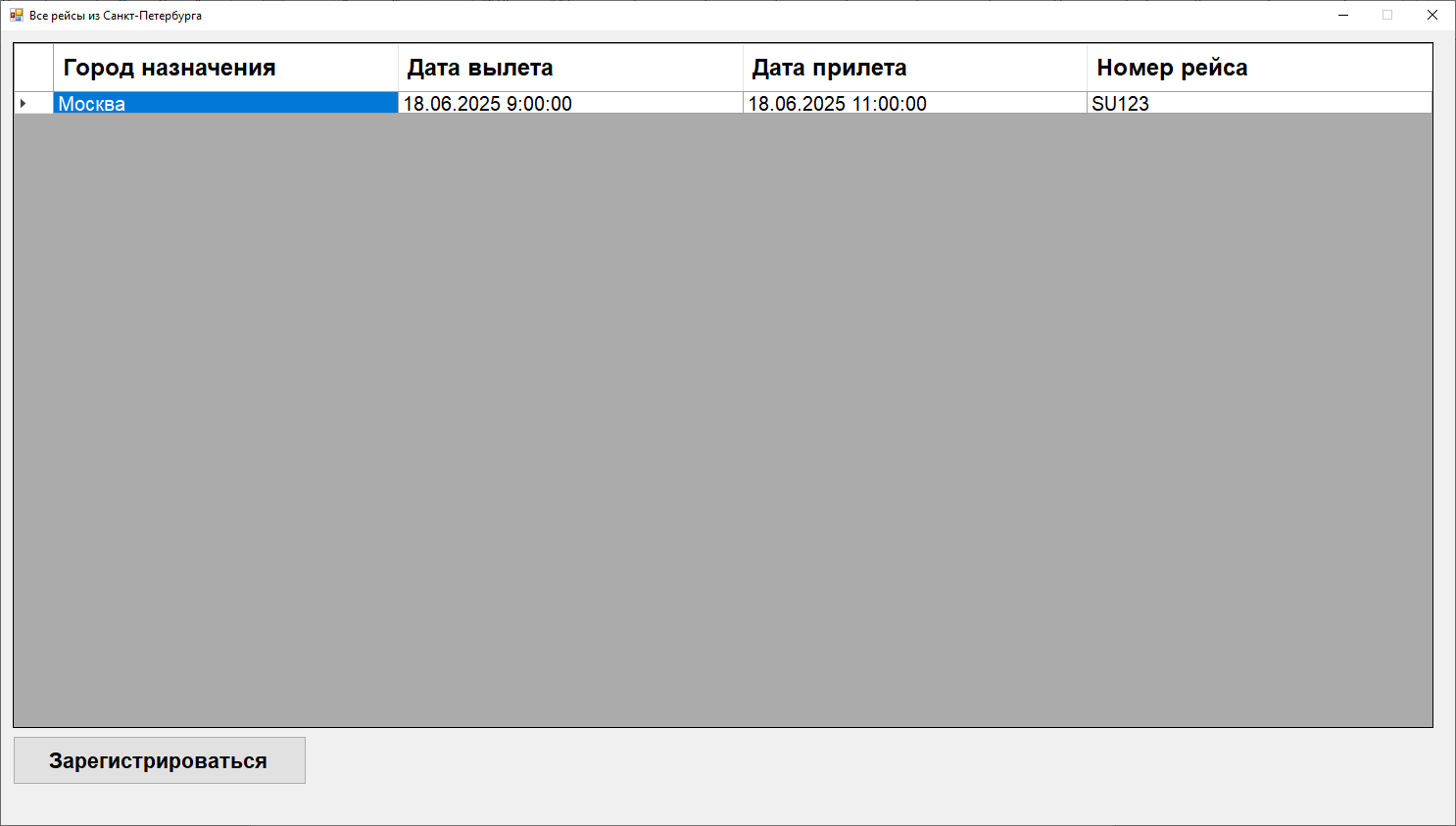


Рисунок А.2 – окно просмотра рейсов

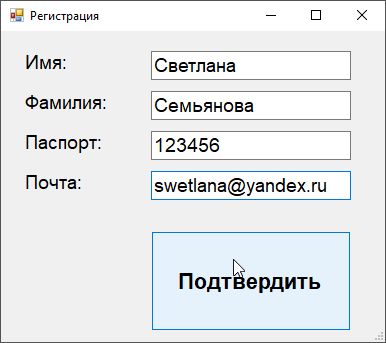


Рисунок А.3 – Окно регистрации на рейс

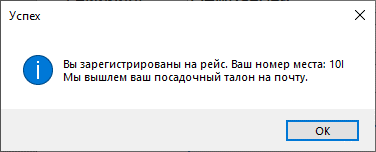


Рисунок А.4 – успешная регистрация

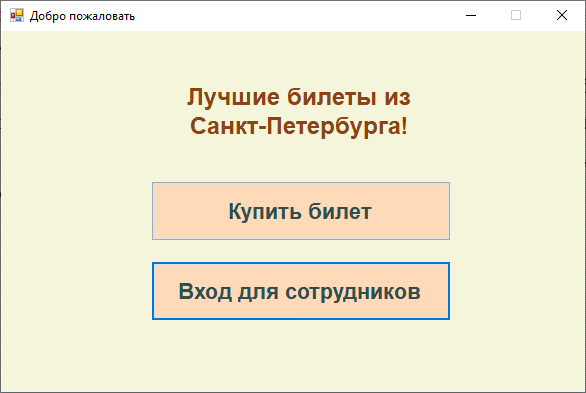


Рисунок А.5 – форма входа

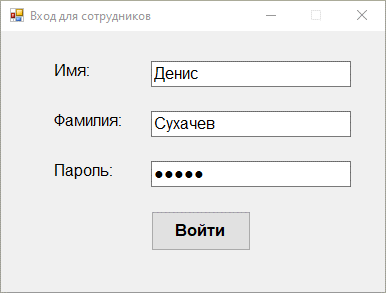


Рисунок А.6 – форма авторизации сотрудника

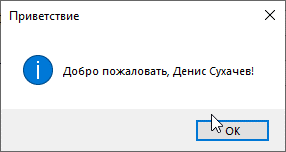


Рисунок А.7 – успешный вход

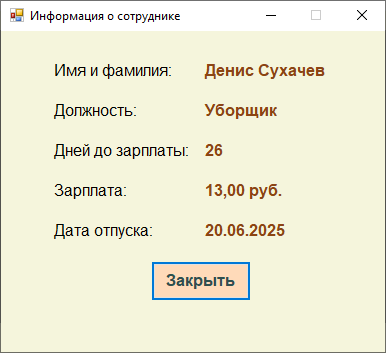


Рисунок А.8 – форма данных сотрудника

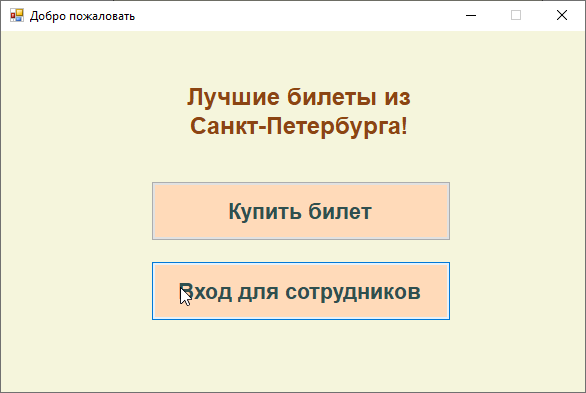


Рисунок А.9 – форма входа

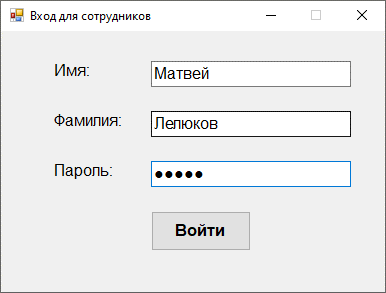


Рисунок А.9 – форма входа

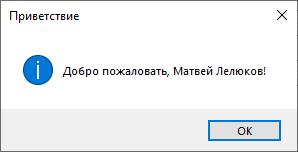


Рисунок А.10 – успешный вход

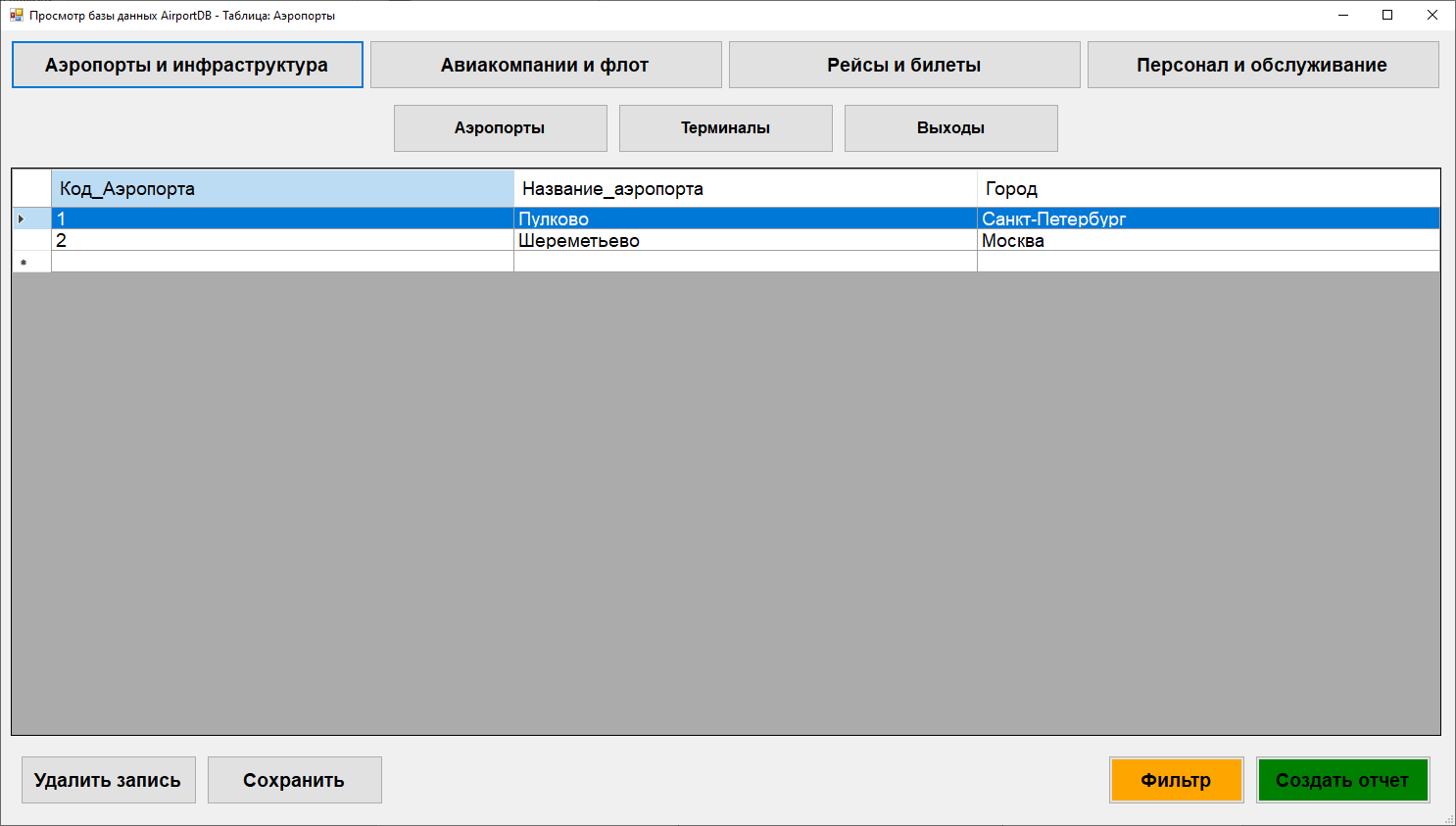


Рисунок А.11 – панель администрования

# **ПРИЛОЖЕНИЕ Б**

(справочное)

CREATE TABLE Аэропорты (

Код\_Аэропорта INT PRIMARY KEY IDENTITY(1,1),

Название\_аэропорта NVARCHAR(100) NOT NULL,

Город NVARCHAR(100) NOT NULL

);

CREATE TABLE Терминалы (

Код\_Терминала INT PRIMARY KEY IDENTITY(1,1),

Код\_Аэропорта INT FOREIGN KEY REFERENCES Аэропорты(Код\_Аэропорта),

Название\_терминала NVARCHAR(100) NOT NULL

);

CREATE TABLE Выходы (

Код\_Выхода INT PRIMARY KEY IDENTITY(1,1),

Код\_Терминала INT FOREIGN KEY REFERENCES Терминалы(Код\_Терминала),

Номер\_выхода NVARCHAR(10) NOT NULL

);

CREATE TABLE Авиакомпании (

Код\_Авиакомпании INT PRIMARY KEY IDENTITY(1,1),

Название\_авиакомпании NVARCHAR(100) NOT NULL

);

CREATE TABLE Самолёты (

Код\_Самолёта INT PRIMARY KEY IDENTITY(1,1),

Код\_Авиакомпании INT FOREIGN KEY REFERENCES Авиакомпании(Код\_Авиакомпании),

Вместимость INT NOT NULL

);

CREATE TABLE Рейсы (

Код\_Рейса INT PRIMARY KEY IDENTITY(1,1),

Номер\_Рейса NVARCHAR(20) NOT NULL,

Город\_Вылета NVARCHAR(100) NOT NULL,

Город\_Прилета NVARCHAR(100) NOT NULL,

Время\_Вылета DATETIME NOT NULL,

Время\_Прилета DATETIME NOT NULL,

Код\_Самолёта INT FOREIGN KEY REFERENCES Самолёты(Код\_Самолёта),

Код\_Выхода INT FOREIGN KEY REFERENCES Выходы(Код\_Выхода)

);

CREATE TABLE Пассажиры (

Код\_Пассажира INT PRIMARY KEY IDENTITY(1,1),

Имя NVARCHAR(100) NOT NULL,

Национальность NVARCHAR(100) NOT NULL

);

CREATE TABLE Сотрудники (

Код\_Сотрудника INT PRIMARY KEY IDENTITY(1,1),

Имя NVARCHAR(100) NOT NULL,

Фамилия NVARCHAR(100) NOT NULL,

Зарплата DECIMAL(10,2) NOT NULL,

Должность NVARCHAR(50) NOT NULL,

Уровень\_доступа INT NOT NULL DEFAULT 1,

Пароль NVARCHAR(50) NOT NULL DEFAULT 'defaultpassword',

Дата\_отпуска DATETIME,

День\_выплаты\_зарплаты INT

);

CREATE TABLE Билеты (

Код\_Билета INT PRIMARY KEY IDENTITY(1,1),

Код\_Рейса INT FOREIGN KEY REFERENCES Рейсы(Код\_Рейса),

Код\_Пассажира INT FOREIGN KEY REFERENCES Пассажиры(Код\_Пассажира),

Номер\_Места NVARCHAR(10) NOT NULL,

Цена DECIMAL(10,2) NOT NULL

);

CREATE TABLE Обслуживание\_рейсов (

Код\_Обслуживания INT PRIMARY KEY IDENTITY(1,1),

Код\_Рейса INT FOREIGN KEY REFERENCES Рейсы(Код\_Рейса),

Код\_Сотрудника INT FOREIGN KEY REFERENCES Сотрудники(Код\_Сотрудника)

);

# **ПРИЛОЖЕНИЕ В**

(справочное)

namespace Airport

{

public partial class Form2 : Form

{

private string connectionString = @"Data Source=DESKTOP-KDS9A4G;Initial Catalog=ПОРТ;Integrated Security=True";

private DataGridView dataGridViewTickets;

private Button btnRegister;

private Random random = new Random();

private int passengerCodeCounter = 1;

public Form2()

{

InitializeComponent();

InitializeControls();

LoadTickets();

}

private void InitializeControls()

{

this.Text = "Все рейсы из Санкт-Петербурга";

this.Size = new Size(1500, 850);

this.FormBorderStyle = FormBorderStyle.FixedSingle;

this.MaximizeBox = false;

// Initialize DataGridView

dataGridViewTickets = new DataGridView

{

Location = new Point(12, 12),

Size = new Size(1450, 700),

AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill,

ReadOnly = true,

AllowUserToAddRows = false,

AllowUserToResizeRows = false,

ColumnHeadersDefaultCellStyle = new DataGridViewCellStyle

{

Font = new Font("Arial", 18, FontStyle.Bold),

BackColor = Color.LightGray

},

DefaultCellStyle = new DataGridViewCellStyle

{

Font = new Font("Arial", 15)

}

};

// Установка заголовков столбцов

dataGridViewTickets.Columns.Add("City", "Город назначения");

dataGridViewTickets.Columns.Add("DepartureDate", "Дата вылета");

dataGridViewTickets.Columns.Add("ArrivalDate", "Дата прилета");

dataGridViewTickets.Columns.Add("FlightNumber", "Номер рейса");

// Initialize Register Button

btnRegister = new Button

{

Text = "Зарегистрироваться",

Location = new Point(12, 720),

Size = new Size(300, 50),

Font = new Font("Arial", 16, FontStyle.Bold)

};

btnRegister.Click += BtnRegister\_Click;

// Add controls to form

this.Controls.Add(dataGridViewTickets);

this.Controls.Add(btnRegister);

}

private void LoadTickets()

{

try

{

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

string query = @"

SELECT

Р.Город\_Прилета AS [Город назначения],

Р.Время\_Вылета AS [Дата вылета],

Р.Время\_Прилета AS [Дата прилета],

Р.Номер\_Рейса AS [Номер рейса]

FROM Рейсы Р

WHERE Р.Город\_Вылета = N'Санкт-Петербург'";

using (SqlDataAdapter adapter = new SqlDataAdapter(query, connection))

{

DataTable ticketsTable = new DataTable();

adapter.Fill(ticketsTable);

dataGridViewTickets.Rows.Clear();

foreach (DataRow row in ticketsTable.Rows)

{

dataGridViewTickets.Rows.Add(

row["Город назначения"],

row["Дата вылета"],

row["Дата прилета"],

row["Номер рейса"]

);

}

}

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при загрузке данных: {ex.Message}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

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

{

Form registerForm = new Form

{

Text = "Регистрация",

Size = new Size(400, 350),

StartPosition = FormStartPosition.CenterParent

};

Label lblName = new Label { Text = "Имя:", Location = new Point(20, 20), Font = new Font("Arial", 14) };

TextBox txtName = new TextBox { Location = new Point(150, 20), Width = 200, Font = new Font("Arial", 14) };

Label lblFamily = new Label { Text = "Фамилия:", Location = new Point(20, 60), Font = new Font("Arial", 14) };

TextBox txtFamily = new TextBox { Location = new Point(150, 60), Width = 200, Font = new Font("Arial", 14) };

Label lblPassport = new Label { Text = "Паспорт:", Location = new Point(20, 100), Font = new Font("Arial", 14) };

TextBox txtPassport = new TextBox { Location = new Point(150, 100), Width = 200, Font = new Font("Arial", 14) };

Label lblEmail = new Label { Text = "Электронная почта:", Location = new Point(20, 140), Font = new Font("Arial", 14) };

TextBox txtEmail = new TextBox { Location = new Point(150, 140), Width = 200, Font = new Font("Arial", 14) };

Button btnSubmit = new Button { Text = "Подтвердить", Location = new Point(150, 200), Width = 100, Font = new Font("Arial", 16, FontStyle.Bold) };

btnSubmit.Click += (s, ev) =>

{

// Валидация паспорта (например, формат 4 цифры пробел 6 цифр)

string passportPattern = @"^\d{4}\s\d{6}$";

if (!Regex.IsMatch(txtPassport.Text, passportPattern))

{

MessageBox.Show("Неверный формат паспорта! Ожидается: 1234 567890.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

// Валидация email (простая проверка)

string emailPattern = @"^[^@\s]+@[^@\s]+\.[^@\s]+$";

if (!Regex.IsMatch(txtEmail.Text, emailPattern))

{

MessageBox.Show("Неверный формат электронной почты!", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

string seatNumber = $"{random.Next(10, 20)}{Convert.ToChar(random.Next(65, 91))}"; // Случайное место, например "12A", "15B"

string flightNumber = dataGridViewTickets.Rows.Count > 0 ? dataGridViewTickets.Rows[0].Cells["FlightNumber"].Value.ToString() : "N/A";

try

{

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

// Получение Код\_Рейса по Номер\_Рейса

int flightCode = GetFlightCode(flightNumber, connection);

// Вставка в таблицу Пассажиры

string passengerInsertQuery = @"

INSERT INTO Пассажиры (Имя, Фамилия, Паспорт, Электронная\_Почта)

VALUES (@Name, @Family, @Passport, @Email)";

using (SqlCommand cmdPassenger = new SqlCommand(passengerInsertQuery, connection))

{

cmdPassenger.Parameters.AddWithValue("@Name", txtName.Text);

cmdPassenger.Parameters.AddWithValue("@Family", txtFamily.Text);

cmdPassenger.Parameters.AddWithValue("@Passport", txtPassport.Text);

cmdPassenger.Parameters.AddWithValue("@Email", txtEmail.Text);

cmdPassenger.ExecuteNonQuery();

}

// Получение последнего вставленного Код\_Пассажира

int passengerCode = GetLastPassengerCode(connection);

// Вставка в таблицу Билеты

string ticketInsertQuery = @"

INSERT INTO Билеты (Код\_Рейса, Код\_Пассажира, Номер\_Места, Цена)

VALUES (@FlightCode, @PassengerCode, @Seat, 7500.00)"; // Цена задана как фиксированная для примера

using (SqlCommand cmdTicket = new SqlCommand(ticketInsertQuery, connection))

{

cmdTicket.Parameters.AddWithValue("@FlightCode", flightCode);

cmdTicket.Parameters.AddWithValue("@PassengerCode", passengerCode);

cmdTicket.Parameters.AddWithValue("@Seat", seatNumber);

cmdTicket.ExecuteNonQuery();

}

MessageBox.Show($"Вы зарегистрированы на рейс. Ваш номер места: {seatNumber}\nМы вышлем ваш посадочный талон на почту.", "Успех", MessageBoxButtons.OK, MessageBoxIcon.Information);

registerForm.Close();

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при сохранении данных: {ex.Message}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

};

registerForm.Controls.Add(lblName);

registerForm.Controls.Add(txtName);

registerForm.Controls.Add(lblFamily);

registerForm.Controls.Add(txtFamily);

registerForm.Controls.Add(lblPassport);

registerForm.Controls.Add(txtPassport);

registerForm.Controls.Add(lblEmail);

registerForm.Controls.Add(txtEmail);

registerForm.Controls.Add(btnSubmit);

registerForm.ShowDialog(this);

}

private int GetFlightCode(string flightNumber, SqlConnection connection)

{

string query = "SELECT Код\_Рейса FROM Рейсы WHERE Номер\_Рейса = @FlightNumber";

using (SqlCommand cmd = new SqlCommand(query, connection))

{

cmd.Parameters.AddWithValue("@FlightNumber", flightNumber);

object result = cmd.ExecuteScalar();

return result != null ? (int)result : -1;

}

}

private int GetLastPassengerCode(SqlConnection connection)

{

string query = "SELECT MAX(Код\_Пассажира) FROM Пассажиры";

using (SqlCommand cmd = new SqlCommand(query, connection))

{

object result = cmd.ExecuteScalar();

return result != null ? (int)result : 0;

}

}

}

}

namespace Airport

{

public partial class Form2 : Form

{

private string connectionString = @"Data Source=DESKTOP-KDS9A4G;Initial Catalog=ПОРТ;Integrated Security=True";

private DataGridView dataGridViewTickets;

private Button btnRegister;

private Random random = new Random();

private int passengerCodeCounter = 1;

public Form2()

{

InitializeComponent();

InitializeControls();

LoadTickets();

}

private void InitializeControls()

{

this.Text = "Все рейсы из Санкт-Петербурга";

this.Size = new Size(1500, 850);

this.FormBorderStyle = FormBorderStyle.FixedSingle;

this.MaximizeBox = false;

// Initialize DataGridView

dataGridViewTickets = new DataGridView

{

Location = new Point(12, 12),

Size = new Size(1450, 700),

AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill,

ReadOnly = true,

AllowUserToAddRows = false,

AllowUserToResizeRows = false,

ColumnHeadersDefaultCellStyle = new DataGridViewCellStyle

{

Font = new Font("Arial", 18, FontStyle.Bold),

BackColor = Color.LightGray

},

DefaultCellStyle = new DataGridViewCellStyle

{

Font = new Font("Arial", 15)

}

};

// Установка заголовков столбцов

dataGridViewTickets.Columns.Add("City", "Город назначения");

dataGridViewTickets.Columns.Add("DepartureDate", "Дата вылета");

dataGridViewTickets.Columns.Add("ArrivalDate", "Дата прилета");

dataGridViewTickets.Columns.Add("FlightNumber", "Номер рейса");

// Initialize Register Button

btnRegister = new Button

{

Text = "Зарегистрироваться",

Location = new Point(12, 720),

Size = new Size(300, 50),

Font = new Font("Arial", 16, FontStyle.Bold)

};

btnRegister.Click += BtnRegister\_Click;

// Add controls to form

this.Controls.Add(dataGridViewTickets);

this.Controls.Add(btnRegister);

}

private void LoadTickets()

{

try

{

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

string query = @"

SELECT

Р.Город\_Прилета AS [Город назначения],

Р.Время\_Вылета AS [Дата вылета],

Р.Время\_Прилета AS [Дата прилета],

Р.Номер\_Рейса AS [Номер рейса]

FROM Рейсы Р

WHERE Р.Город\_Вылета = N'Санкт-Петербург'";

using (SqlDataAdapter adapter = new SqlDataAdapter(query, connection))

{

DataTable ticketsTable = new DataTable();

adapter.Fill(ticketsTable);

dataGridViewTickets.Rows.Clear();

foreach (DataRow row in ticketsTable.Rows)

{

dataGridViewTickets.Rows.Add(

row["Город назначения"],

row["Дата вылета"],

row["Дата прилета"],

row["Номер рейса"]

);

}

}

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при загрузке данных: {ex.Message}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

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

{

Form registerForm = new Form

{

Text = "Регистрация",

Size = new Size(400, 350),

StartPosition = FormStartPosition.CenterParent

};

Label lblName = new Label { Text = "Имя:", Location = new Point(20, 20), Font = new Font("Arial", 14) };

TextBox txtName = new TextBox { Location = new Point(150, 20), Width = 200, Font = new Font("Arial", 14) };

Label lblFamily = new Label { Text = "Фамилия:", Location = new Point(20, 60), Font = new Font("Arial", 14) };

TextBox txtFamily = new TextBox { Location = new Point(150, 60), Width = 200, Font = new Font("Arial", 14) };

Label lblPassport = new Label { Text = "Паспорт:", Location = new Point(20, 100), Font = new Font("Arial", 14) };

TextBox txtPassport = new TextBox { Location = new Point(150, 100), Width = 200, Font = new Font("Arial", 14) };

Label lblEmail = new Label { Text = "Электронная почта:", Location = new Point(20, 140), Font = new Font("Arial", 14) };

TextBox txtEmail = new TextBox { Location = new Point(150, 140), Width = 200, Font = new Font("Arial", 14) };

Button btnSubmit = new Button { Text = "Подтвердить", Location = new Point(150, 200), Width = 100, Font = new Font("Arial", 16, FontStyle.Bold) };

btnSubmit.Click += (s, ev) =>

{

// Валидация паспорта (например, формат 4 цифры пробел 6 цифр)

string passportPattern = @"^\d{4}\s\d{6}$";

if (!Regex.IsMatch(txtPassport.Text, passportPattern))

{

MessageBox.Show("Неверный формат паспорта! Ожидается: 1234 567890.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

// Валидация email (простая проверка)

string emailPattern = @"^[^@\s]+@[^@\s]+\.[^@\s]+$";

if (!Regex.IsMatch(txtEmail.Text, emailPattern))

{

MessageBox.Show("Неверный формат электронной почты!", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

string seatNumber = $"{random.Next(10, 20)}{Convert.ToChar(random.Next(65, 91))}"; // Случайное место, например "12A", "15B"

string flightNumber = dataGridViewTickets.Rows.Count > 0 ? dataGridViewTickets.Rows[0].Cells["FlightNumber"].Value.ToString() : "N/A";

try

{

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

// Получение Код\_Рейса по Номер\_Рейса

int flightCode = GetFlightCode(flightNumber, connection);

// Вставка в таблицу Пассажиры

string passengerInsertQuery = @"

INSERT INTO Пассажиры (Имя, Фамилия, Паспорт, Электронная\_Почта)

VALUES (@Name, @Family, @Passport, @Email)";

using (SqlCommand cmdPassenger = new SqlCommand(passengerInsertQuery, connection))

{

cmdPassenger.Parameters.AddWithValue("@Name", txtName.Text);

cmdPassenger.Parameters.AddWithValue("@Family", txtFamily.Text);

cmdPassenger.Parameters.AddWithValue("@Passport", txtPassport.Text);

cmdPassenger.Parameters.AddWithValue("@Email", txtEmail.Text);

cmdPassenger.ExecuteNonQuery();

}

// Получение последнего вставленного Код\_Пассажира

int passengerCode = GetLastPassengerCode(connection);

// Вставка в таблицу Билеты

string ticketInsertQuery = @"

INSERT INTO Билеты (Код\_Рейса, Код\_Пассажира, Номер\_Места, Цена)

VALUES (@FlightCode, @PassengerCode, @Seat, 7500.00)"; // Цена задана как фиксированная для примера

using (SqlCommand cmdTicket = new SqlCommand(ticketInsertQuery, connection))

{

cmdTicket.Parameters.AddWithValue("@FlightCode", flightCode);

cmdTicket.Parameters.AddWithValue("@PassengerCode", passengerCode);

cmdTicket.Parameters.AddWithValue("@Seat", seatNumber);

cmdTicket.ExecuteNonQuery();

}

MessageBox.Show($"Вы зарегистрированы на рейс. Ваш номер места: {seatNumber}\nМы вышлем ваш посадочный талон на почту.", "Успех", MessageBoxButtons.OK, MessageBoxIcon.Information);

registerForm.Close();

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при сохранении данных: {ex.Message}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

};

registerForm.Controls.Add(lblName);

registerForm.Controls.Add(txtName);

registerForm.Controls.Add(lblFamily);

registerForm.Controls.Add(txtFamily);

registerForm.Controls.Add(lblPassport);

registerForm.Controls.Add(txtPassport);

registerForm.Controls.Add(lblEmail);

registerForm.Controls.Add(txtEmail);

registerForm.Controls.Add(btnSubmit);

registerForm.ShowDialog(this);

}

private int GetFlightCode(string flightNumber, SqlConnection connection)

{

string query = "SELECT Код\_Рейса FROM Рейсы WHERE Номер\_Рейса = @FlightNumber";

using (SqlCommand cmd = new SqlCommand(query, connection))

{

cmd.Parameters.AddWithValue("@FlightNumber", flightNumber);

object result = cmd.ExecuteScalar();

return result != null ? (int)result : -1;

}

}

private int GetLastPassengerCode(SqlConnection connection)

{

string query = "SELECT MAX(Код\_Пассажира) FROM Пассажиры";

using (SqlCommand cmd = new SqlCommand(query, connection))

{

object result = cmd.ExecuteScalar();

return result != null ? (int)result : 0;

}

}

}

}

**WorkerPanel.cs**

using System;

using System.Drawing;

using System.Windows.Forms;

namespace Airport

{

public partial class Form4 : Form

{

public Form4(string name, string surname, string position, int? salaryDay, DateTime? vacationDate, decimal? salary)

{

InitializeComponent();

InitializeFormControls(name, surname, position, salaryDay, vacationDate, salary);

}

private void InitializeFormControls(string name, string surname, string position, int? salaryDay, DateTime? vacationDate, decimal? salary)

{

this.Text = "Рабочая панель сотрудника";

this.Size = new Size(450, 450);

this.BackColor = Color.Beige;

this.FormBorderStyle = FormBorderStyle.FixedSingle;

this.MaximizeBox = false;

this.StartPosition = FormStartPosition.CenterScreen;

int yOffset = 30;

int labelWidth = 150;

int valueWidth = 250;

Label lblFullName = new Label

{

Text = "Имя и фамилия:",

Location = new Point(50, yOffset),

Size = new Size(labelWidth, 30),

Font = new Font("Arial", 12)

};

Label lblFullNameValue = new Label

{

Text = $"{name} {surname}",

Location = new Point(200, yOffset),

Size = new Size(valueWidth, 30),

Font = new Font("Arial", 12, FontStyle.Bold),

ForeColor = Color.SaddleBrown

};

yOffset += 50;

Label lblPosition = new Label

{

Text = "Должность:",

Location = new Point(50, yOffset),

Size = new Size(labelWidth, 30),

Font = new Font("Arial", 12)

};

Label lblPositionValue = new Label

{

Text = position ?? "Не указана",

Location = new Point(200, yOffset),

Size = new Size(valueWidth, 30),

Font = new Font("Arial", 12, FontStyle.Bold),

ForeColor = Color.SaddleBrown

};

yOffset += 50;

Label lblDaysToSalary = new Label

{

Text = "Дней до зарплаты:",

Location = new Point(50, yOffset),

Size = new Size(labelWidth, 30),

Font = new Font("Arial", 12)

};

Label lblDaysToSalaryValue = new Label

{

Text = CalculateDaysToSalary(salaryDay),

Location = new Point(200, yOffset),

Size = new Size(valueWidth, 30),

Font = new Font("Arial", 12, FontStyle.Bold),

ForeColor = Color.SaddleBrown

};

yOffset += 50;

Label lblSalary = new Label

{

Text = "Зарплата:",

Location = new Point(50, yOffset),

Size = new Size(labelWidth, 30),

Font = new Font("Arial", 12)

};

Label lblSalaryValue = new Label

{

Text = salary.HasValue ? $"{salary.Value:F2} руб." : "Не указана",

Location = new Point(200, yOffset),

Size = new Size(valueWidth, 30),

Font = new Font("Arial", 12, FontStyle.Bold),

ForeColor = Color.SaddleBrown

};

yOffset += 50;

Label lblDaysToVacation = new Label

{

Text = "Дней до отпуска:",

Location = new Point(50, yOffset),

Size = new Size(labelWidth, 30),

Font = new Font("Arial", 12)

};

Label lblDaysToVacationValue = new Label

{

Text = CalculateDaysToVacation(vacationDate),

Location = new Point(200, yOffset),

Size = new Size(valueWidth, 30),

Font = new Font("Arial", 12, FontStyle.Bold),

ForeColor = Color.SaddleBrown

};

yOffset += 50;

Button btnClose = new Button

{

Text = "Закрыть",

Location = new Point(175, yOffset),

Size = new Size(100, 40),

Font = new Font("Arial", 12, FontStyle.Bold),

BackColor = Color.PeachPuff,

ForeColor = Color.DarkSlateGray

};

btnClose.Click += (s, e) => this.Close();

this.Controls.Add(lblFullName);

this.Controls.Add(lblFullNameValue);

this.Controls.Add(lblPosition);

this.Controls.Add(lblPositionValue);

this.Controls.Add(lblDaysToSalary);

this.Controls.Add(lblDaysToSalaryValue);

this.Controls.Add(lblSalary);

this.Controls.Add(lblSalaryValue);

this.Controls.Add(lblDaysToVacation);

this.Controls.Add(lblDaysToVacationValue);

this.Controls.Add(btnClose);

}

private string CalculateDaysToSalary(int? salaryDay)

{

if (!salaryDay.HasValue)

return "Не указан";

int day = salaryDay.Value;

if (day < 1 || day > 31)

return "Некорректный день";

DateTime today = DateTime.Today;

DateTime nextSalaryDate;

if (today.Day <= day)

{

nextSalaryDate = new DateTime(today.Year, today.Month, Math.Min(day, DateTime.DaysInMonth(today.Year, today.Month)));

}

else

{

DateTime nextMonth = today.AddMonths(1);

nextSalaryDate = new DateTime(nextMonth.Year, nextMonth.Month, Math.Min(day, DateTime.DaysInMonth(nextMonth.Year, nextMonth.Month)));

}

int daysUntilSalary = (nextSalaryDate - today).Days;

return daysUntilSalary.ToString();

}

private string CalculateDaysToVacation(DateTime? vacationDate)

{

if (!vacationDate.HasValue)

return "Не указана";

DateTime today = DateTime.Today;

DateTime vacation = vacationDate.Value.Date;

if (vacation < today)

return "Отпуск прошел";

int daysUntilVacation = (vacation - today).Days;

return daysUntilVacation.ToString();

}

}

}

AdminPanel.cs

namespace Airport

{

public partial class Form1 : Form

{

private string connectionString = @"Data Source=DESKTOP-KDS9A4G;Initial Catalog=ПОРТ;Integrated Security=True";

private DataGridView dataGridView;

private Button[] categoryButtons;

private Button[] tableButtons;

private Panel tableButtonsPanel;

private Button deleteButton;

private Button filterButton;

private Button reportButton;

private Button saveButton;

private Panel actionButtonsPanel;

private string currentTable; // To track the currently displayed table

public Form1()

{

InitializeComponent();

InitializeControls();

}

private void InitializeControls()

{

this.Text = "Просмотр базы данных Аэропорт";

this.Size = new Size(1500, 850); // Увеличен размер формы для свободного места внизу

// Инициализация кнопок категорий

string[] categories = {

"Аэропорты и инфраструктура",

"Авиакомпании и флот",

"Рейсы и билеты",

"Персонал и обслуживание"

};

categoryButtons = new Button[categories.Length];

int buttonWidth = (this.ClientSize.Width - 20) / categories.Length; // Равномерное распределение по ширине

int buttonX = 10;

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

{

categoryButtons[i] = new Button

{

Text = categories[i],

Location = new Point(buttonX, 10),

Size = new Size(buttonWidth - 5, 50),

Tag = categories[i],

Font = new Font("Arial", 14, FontStyle.Bold)

};

categoryButtons[i].Click += CategoryButton\_Click;

this.Controls.Add(categoryButtons[i]);

buttonX += buttonWidth;

}

// Панель для кнопок таблиц

tableButtonsPanel = new Panel

{

Location = new Point(10, 70),

Size = new Size(1460, 60),

AutoScroll = true

};

this.Controls.Add(tableButtonsPanel);

// Инициализация кнопок для таблиц

string[] tableNames = {

"Аэропорты", "Терминалы", "Выходы",

"Авиакомпании", "Самолёты",

"Рейсы", "Билеты",

"Сотрудники", "Пассажиры", "Обслуживание\_рейсов"

};

tableButtons = new Button[tableNames.Length];

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

{

tableButtons[i] = new Button

{

Text = tableNames[i],

Size = new Size(220, 50), // Увеличена ширина до 220 пикселей

Tag = tableNames[i],

Visible = false,

Font = new Font("Arial", 12, FontStyle.Bold)

};

tableButtons[i].Click += TableButton\_Click;

tableButtonsPanel.Controls.Add(tableButtons[i]);

}

// Инициализация DataGridView

dataGridView = new DataGridView

{

Location = new Point(10, 140),

Size = new Size(1460, 580),

AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill,

AllowUserToAddRows = true, // Разрешено добавление строк

ReadOnly = false, // Включено редактирование

EditMode = DataGridViewEditMode.EditOnKeystrokeOrF2, // Редактирование по клавише или F2

SelectionMode = DataGridViewSelectionMode.FullRowSelect,

Font = new Font("Arial", 14), // Шрифт для ячеек

ColumnHeadersHeightSizeMode = DataGridViewColumnHeadersHeightSizeMode.EnableResizing,

ColumnHeadersHeight = 40 // Увеличенная высота заголовков

};

this.Controls.Add(dataGridView);

// Панель для кнопок действий

actionButtonsPanel = new Panel

{

Location = new Point(10, 730),

Size = new Size(1460, 60)

};

this.Controls.Add(actionButtonsPanel);

// Инициализация кнопок действий

buttonX = 10;

deleteButton = new Button

{

Text = "Удалить запись",

Location = new Point(buttonX, 10),

Size = new Size(180, 50),

Font = new Font("Arial", 14, FontStyle.Bold)

};

deleteButton.Click += DeleteButton\_Click;

actionButtonsPanel.Controls.Add(deleteButton);

buttonX += 190;

saveButton = new Button

{

Text = "Сохранить",

Location = new Point(buttonX, 10),

Size = new Size(180, 50),

Font = new Font("Arial", 14, FontStyle.Bold)

};

saveButton.Click += SaveButton\_Click;

actionButtonsPanel.Controls.Add(saveButton);

buttonX += 190;

// Размещаем кнопки "Фильтр" и "Создать отчет" справа

buttonX = actionButtonsPanel.Width - (140 + 180 + 20);

filterButton = new Button

{

Text = "Фильтр",

Location = new Point(buttonX, 10),

Size = new Size(140, 50),

Font = new Font("Arial", 14, FontStyle.Bold),

BackColor = Color.Orange

};

filterButton.Click += FilterButton\_Click;

actionButtonsPanel.Controls.Add(filterButton);

buttonX += 150;

reportButton = new Button

{

Text = "Создать отчет",

Location = new Point(buttonX, 10),

Size = new Size(180, 50),

Font = new Font("Arial", 14, FontStyle.Bold),

BackColor = Color.Green

};

reportButton.Click += ReportButton\_Click;

actionButtonsPanel.Controls.Add(reportButton);

// Показать таблицы первой категории по умолчанию

ShowTablesForCategory("Аэропорты и инфраструктура");

// Изменено для явного вызова таблицы Аэропорты при загрузке

LoadTableData("Аэропорты");

}

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

{

Button button = sender as Button;

if (button != null)

{

string category = button.Tag.ToString();

ShowTablesForCategory(category);

// Загружаем первую таблицу категории при выборе

string[] tables = GetTablesForCategory(category);

if (tables.Length > 0)

{

LoadTableData(tables[0]);

}

}

}

private void ShowTablesForCategory(string category)

{

// Скрыть все кнопки таблиц

foreach (var button in tableButtons)

{

button.Visible = false;

}

// Определение таблиц для каждой категории

string[] tables = GetTablesForCategory(category);

int visibleButtonCount = tables.Length;

if (visibleButtonCount == 0) return; // Если нет таблиц, выходим

// Параметры панели и кнопок

int panelWidth = tableButtonsPanel.Width; // Ширина панели (1460 пикселей)

int buttonWidth = 220; // Ширина каждой кнопки

int spacing = 10; // Небольшое расстояние между кнопками

int totalWidth = visibleButtonCount \* buttonWidth + (visibleButtonCount - 1) \* spacing; // Общая ширина с учетом промежутков

int startX = (panelWidth - totalWidth) / 2; // Центрируем кнопки

// Показать кнопки соответствующих таблиц

int buttonX = startX;

foreach (var table in tables)

{

var button = tableButtons.FirstOrDefault(b => b.Tag.ToString() == table);

if (button != null)

{

button.Location = new Point(buttonX, 5);

button.Visible = true;

buttonX += buttonWidth + spacing; // Следующая кнопка с учетом шага

}

}

}

private string[] GetTablesForCategory(string category)

{

switch (category)

{

case "Аэропорты и инфраструктура":

return new[] { "Аэропорты", "Терминалы", "Выходы" };

case "Авиакомпании и флот":

return new[] { "Авиакомпании", "Самолёты" };

case "Рейсы и билеты":

return new[] { "Рейсы", "Билеты" };

case "Персонал и обслуживание":

return new[] { "Сотрудники", "Пассажиры", "Обслуживание\_рейсов" };

default:

return Array.Empty<string>();

}

}

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

{

Button button = sender as Button;

if (button != null)

{

string tableName = button.Tag.ToString();

Console.WriteLine($"Нажата кнопка для таблицы: {tableName} at {DateTime.Now}");

LoadTableData(tableName); // Явно загружаем данные для выбранной таблицы

}

}

private void LoadTableData(string tableName)

{

Console.WriteLine($"Попытка загрузки таблицы: {tableName} at {DateTime.Now}");

string[] validTables = { "Аэропорты", "Терминалы", "Выходы", "Авиакомпании", "Самолёты", "Рейсы", "Билеты", "Пассажиры", "Сотрудники", "Обслуживание\_рейсов" };

if (!validTables.Contains(tableName))

{

MessageBox.Show($"Недопустимое имя таблицы: '{tableName}'. Доступные таблицы: {string.Join(", ", validTables)}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

try

{

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

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

Console.WriteLine($"Выполняется запрос: {query}");

using (SqlDataAdapter adapter = new SqlDataAdapter(query, connection))

{

DataTable dataTable = new DataTable();

adapter.Fill(dataTable);

if (dataTable.Rows.Count == 0)

{

MessageBox.Show($"Таблица '{tableName}' пуста.", "Информация", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

dataGridView.DataSource = dataTable;

currentTable = tableName;

}

}

this.Text = $"Просмотр базы данных AirportDB - Таблица: {tableName}";

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при загрузке данных: {ex.Message}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

dataGridView.DataSource = null;

currentTable = null;

Console.WriteLine($"Исключение: {ex.Message}");

}

}

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

{

if (dataGridView.SelectedRows.Count == 0)

{

MessageBox.Show("Пожалуйста, сначала выделите запись для удаления.", "Предупреждение", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return;

}

if (dataGridView.DataSource is DataTable dataTable && !string.IsNullOrEmpty(currentTable))

{

try

{

// Получаем выделенную строку

DataRowView rowView = (DataRowView)dataGridView.SelectedRows[0].DataBoundItem;

if (rowView != null)

{

DataRow row = rowView.Row;

if (row != null)

{

// Предполагаем, что первичный ключ — первая колонка (нужно уточнить структуру)

string primaryKeyColumn = dataTable.Columns[0].ColumnName;

object primaryKeyValue = row[primaryKeyColumn];

// Рекурсивное удаление связанных записей

DeleteRelatedRecords(currentTable, primaryKeyColumn, primaryKeyValue);

// Удаляем строку из текущей таблицы

row.Delete();

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

string query = $"SELECT \* FROM [{currentTable}]";

using (SqlDataAdapter adapter = new SqlDataAdapter(query, connection))

{

using (SqlCommandBuilder builder = new SqlCommandBuilder(adapter))

{

adapter.Update(dataTable); // Выполняем удаление в базе данных

}

}

}

MessageBox.Show($"Запись удалена.\nВозможные связи таблицы '{currentTable}' уже обработаны.", "Успех", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при удалении записи: {ex.Message}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

if (dataTable != null) dataTable.RejectChanges(); // Откат изменений при ошибке

}

}

else

{

MessageBox.Show("Данные таблицы не загружены или произошла ошибка.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void DeleteRelatedRecords(string tableName, string primaryKeyColumn, object primaryKeyValue)

{

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

// Определяем связанные таблицы на основе предположений

switch (tableName)

{

case "Аэропорты":

ExecuteDeleteQuery(connection, "DELETE FROM [Терминалы] WHERE [ID\_Аэропорта] = @Key", primaryKeyValue);

ExecuteDeleteQuery(connection, "DELETE FROM [Выходы] WHERE [ID\_Аэропорта] = @Key", primaryKeyValue);

break;

case "Терминалы":

ExecuteDeleteQuery(connection, "DELETE FROM [Выходы] WHERE [ID\_Терминала] = @Key", primaryKeyValue);

break;

case "Авиакомпании":

ExecuteDeleteQuery(connection, "DELETE FROM [Самолёты] WHERE [ID\_Авиакомпании] = @Key", primaryKeyValue);

break;

case "Рейсы":

ExecuteDeleteQuery(connection, "DELETE FROM [Билеты] WHERE [ID\_Рейса] = @Key", primaryKeyValue);

ExecuteDeleteQuery(connection, "DELETE FROM [Обслуживание\_рейсов] WHERE [ID\_Рейса] = @Key", primaryKeyValue);

break;

case "Сотрудники":

ExecuteDeleteQuery(connection, "DELETE FROM [Обслуживание\_рейсов] WHERE [ID\_Сотрудника] = @Key", primaryKeyValue);

break;

case "Пассажиры":

ExecuteDeleteQuery(connection, "DELETE FROM [Билеты] WHERE [ID\_Пассажира] = @Key", primaryKeyValue);

break;

}

// Удаляем из текущей таблицы (если есть зависимости, они уже обработаны выше)

ExecuteDeleteQuery(connection, $"DELETE FROM [{tableName}] WHERE [{primaryKeyColumn}] = @Key", primaryKeyValue);

}

}

private void ExecuteDeleteQuery(SqlConnection connection, string query, object keyValue = null)

{

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

{

if (keyValue != null)

{

command.Parameters.AddWithValue("@Key", keyValue);

}

command.ExecuteNonQuery();

}

}

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

{

if (dataGridView.DataSource is DataTable dataTable && !string.IsNullOrEmpty(currentTable))

{

FilterGrid(dataGridView, currentTable);

}

else

{

MessageBox.Show("Данные таблицы не загружены или произошла ошибка.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void FilterGrid(DataGridView grid, string tableName)

{

try

{

Form filterForm = new Form();

filterForm.Text = $"Фильтрация таблицы {tableName}";

filterForm.Size = new Size(300, 200);

filterForm.StartPosition = FormStartPosition.CenterParent;

Label columnLabel = new Label();

columnLabel.Text = "Выберите столбец:";

columnLabel.Location = new Point(10, 20);

columnLabel.AutoSize = true;

filterForm.Controls.Add(columnLabel);

ComboBox columnComboBox = new ComboBox();

columnComboBox.Location = new Point(10, 40);

columnComboBox.Size = new Size(260, 20);

if (grid.DataSource is DataTable dataTable)

{

foreach (DataColumn column in dataTable.Columns)

{

columnComboBox.Items.Add(column.ColumnName);

}

}

columnComboBox.SelectedIndex = 0; // По умолчанию первый столбец

filterForm.Controls.Add(columnComboBox);

Label valueLabel = new Label();

valueLabel.Text = "Введите значение:";

valueLabel.Location = new Point(10, 70);

valueLabel.AutoSize = true;

filterForm.Controls.Add(valueLabel);

TextBox valueTextBox = new TextBox();

valueTextBox.Location = new Point(10, 90);

valueTextBox.Size = new Size(260, 20);

filterForm.Controls.Add(valueTextBox);

Button confirmButton = new Button();

confirmButton.Text = "Применить";

confirmButton.Font = new Font("Arial", 12);

confirmButton.Location = new Point(10, 120);

confirmButton.Size = new Size(100, 40);

confirmButton.Click += (s, e) =>

{

if (grid.DataSource is DataTable filteredDataTable)

{

string selectedColumn = columnComboBox.SelectedItem.ToString();

string filterValue = valueTextBox.Text.Trim();

DataView dataView = filteredDataTable.DefaultView;

if (string.IsNullOrEmpty(filterValue))

{

dataView.RowFilter = ""; // Сброс фильтра

}

else

{

// Проверяем тип данных столбца

if (filteredDataTable.Columns[selectedColumn].DataType == typeof(string))

{

dataView.RowFilter = $"{selectedColumn} LIKE '%{filterValue}%'";

}

else

{

dataView.RowFilter = $"{selectedColumn} = '{filterValue}'";

}

}

grid.DataSource = dataView;

}

filterForm.Close();

};

filterForm.Controls.Add(confirmButton);

filterForm.ShowDialog();

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при фильтрации: {ex.Message}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

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

{

DataSet dataSet = new DataSet();

try

{

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

// Экспорт всех таблиц

string[] tableNames = {

"Аэропорты", "Терминалы", "Выходы",

"Авиакомпании", "Самолёты",

"Рейсы", "Билеты",

"Сотрудники", "Пассажиры", "Обслуживание\_рейсов"

};

foreach (string tableName in tableNames)

{

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

using (SqlDataAdapter adapter = new SqlDataAdapter(query, connection))

{

DataTable table = new DataTable(tableName);

adapter.Fill(table);

dataSet.Tables.Add(table);

}

}

}

// Экспорт в Excel с добавлением диаграммы

ExportToExcel(dataSet, currentTable);

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при создании отчета: {ex.Message}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void ExportToExcel(DataSet dataSet, string currentTable)

{

Excel.Application excelApp = null;

Excel.Workbook workBook = null;

try

{

// Создание нового Excel-приложения

excelApp = new Excel.Application();

excelApp.Visible = true; // Excel будет виден пользователю

workBook = excelApp.Workbooks.Add();

// Создаем лист "Дополнительно" как первый лист

Excel.Worksheet workSheetAdditional = (Excel.Worksheet)workBook.Sheets[1];

workSheetAdditional.Name = "Дополнительно";

// Перебираем все таблицы в dataSet и добавляем их в Excel

int sheetIndex = 2;

foreach (DataTable table in dataSet.Tables)

{

// Создаем новый лист с именем таблицы

Excel.Worksheet workSheet = (Excel.Worksheet)workBook.Sheets.Add();

workSheet.Name = table.TableName.Length > 31 ? table.TableName.Substring(0, 31) : table.TableName; // Ограничение длины имени листа в Excel

// Заголовки колонок

for (int i = 0; i < table.Columns.Count; i++)

{

workSheet.Cells[1, i + 1] = table.Columns[i].ColumnName;

}

// Данные таблицы

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

{

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

{

workSheet.Cells[i + 2, j + 1] = table.Rows[i][j]?.ToString(); // Приведение к строке для избежания проблем с типами

}

}

// Автоматически растягиваем колонки и строки

workSheet.Columns.AutoFit();

workSheet.Rows.AutoFit();

sheetIndex++;

}

// Добавляем диаграмму на лист "Дополнительно" для текущей таблицы

if (!string.IsNullOrEmpty(currentTable))

{

DataTable currentDataTable = dataSet.Tables[currentTable];

if (currentDataTable != null)

{

CreateChart(workSheetAdditional, currentDataTable, currentTable);

}

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при экспорте в Excel: {ex.Message}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

finally

{

// Освобождаем ресурсы

if (workBook != null)

System.Runtime.InteropServices.Marshal.ReleaseComObject(workBook);

if (excelApp != null)

{

excelApp.Quit();

System.Runtime.InteropServices.Marshal.ReleaseComObject(excelApp);

}

}

}

private void CreateChart(Excel.Worksheet workSheet, DataTable dataTable, string tableName)

{

try

{

// Определяем данные для диаграммы в зависимости от таблицы

string categoryColumn = "";

string valueColumn = "";

string chartTitle = "";

bool useCount = true; // Флаг для подсчета записей (используется, если нет подходящего числового столбца)

switch (tableName)

{

case "Аэропорты":

categoryColumn = "Название\_аэропорта"; // Соответствует вашей структуре

valueColumn = null; // Подсчитываем количество терминалов

chartTitle = "Количество терминалов по аэропортам";

break;

case "Терминалы":

categoryColumn = "ID\_Аэропорта";

valueColumn = null; // Подсчитываем количество выходов

chartTitle = "Количество терминалов по аэропортам";

break;

case "Выходы":

categoryColumn = "ID\_Терминала";

valueColumn = null; // Подсчитываем количество выходов

chartTitle = "Количество выходов по терминалам";

break;

case "Авиакомпании":

categoryColumn = "Название\_авиакомпании";

valueColumn = null; // Подсчитываем количество самолетов

chartTitle = "Количество самолетов по авиакомпаниям";

break;

case "Самолёты":

categoryColumn = "ID\_Авиакомпании";

valueColumn = "Вместимость";

useCount = false; // Используем вместимость как числовое значение

chartTitle = "Вместимость самолетов по авиакомпаниям";

break;

case "Рейсы":

categoryColumn = "ID\_Авиакомпании";

valueColumn = null; // Подсчитываем количество рейсов

chartTitle = "Количество рейсов по авиакомпаниям";

break;

case "Билеты":

categoryColumn = "ID\_Рейса";

valueColumn = null; // Подсчитываем количество билетов

chartTitle = "Количество билетов по рейсам";

break;

case "Сотрудники":

categoryColumn = "Должность";

valueColumn = null; // Подсчитываем количество сотрудников по должности

chartTitle = "Количество сотрудников по должностям";

break;

case "Пассажиры":

categoryColumn = "Национальность";

valueColumn = null; // Подсчитываем количество пассажиров по национальности

chartTitle = "Количество пассажиров по национальности";

break;

case "Обслуживание\_рейсов":

categoryColumn = "ID\_Рейса";

valueColumn = null; // Подсчитываем количество обслуживаний по рейсам

chartTitle = "Количество обслуживаний по рейсам";

break;

default:

return; // Если таблица не определена, пропускаем

}

// Проверяем наличие столбца категорий

if (!dataTable.Columns.Contains(categoryColumn))

{

MessageBox.Show($"Столбец '{categoryColumn}' не найден в таблице '{tableName}'. Проверьте структуру таблицы.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

// Подготовка данных для диаграммы

var groupedData = dataTable.AsEnumerable()

.GroupBy(r => r.Field<object>(categoryColumn)?.ToString())

.Select(g => new

{

Category = g.Key,

Value = useCount

? (double)g.Count() // Преобразуем int в double

: g.Sum(row => row.Field<object>(valueColumn) != null ? Convert.ToDouble(row.Field<object>(valueColumn)) : 0)

})

.OrderBy(g => g.Category)

.ToList();

if (groupedData.Count == 0)

{

MessageBox.Show($"Нет данных для построения диаграммы для таблицы '{tableName}'.", "Информация", MessageBoxButtons.OK, MessageBoxIcon.Information);

return;

}

// Записываем данные в лист "Дополнительно"

workSheet.Cells[1, 1] = "Категория";

workSheet.Cells[1, 2] = useCount ? "Количество" : valueColumn;

int excelRow = 2; // Переименовано из row, чтобы избежать конфликта

foreach (var item in groupedData)

{

workSheet.Cells[excelRow, 1] = item.Category ?? "Не указано";

workSheet.Cells[excelRow, 2] = item.Value;

excelRow++;

}

// Создаем диапазон данных для диаграммы

Excel.Range chartRange = workSheet.Range[workSheet.Cells[1, 1], workSheet.Cells[excelRow - 1, 2]];

// Создаем диаграмму

Excel.ChartObjects chartObjects = (Excel.ChartObjects)workSheet.ChartObjects();

Excel.ChartObject chartObject = chartObjects.Add(100, 100, 600, 400); // Позиция и размер диаграммы

Excel.Chart chart = chartObject.Chart;

chart.SetSourceData(chartRange);

chart.ChartType = Excel.XlChartType.xlColumnClustered; // Тип диаграммы: столбчатая

chart.HasTitle = true;

chart.ChartTitle.Text = chartTitle;

chart.Axes(Excel.XlAxisType.xlCategory).HasTitle = true;

chart.Axes(Excel.XlAxisType.xlCategory).AxisTitle.Text = categoryColumn;

chart.Axes(Excel.XlAxisType.xlValue).HasTitle = true;

chart.Axes(Excel.XlAxisType.xlValue).AxisTitle.Text = useCount ? "Количество" : valueColumn;

// Форматирование диаграммы

chart.Legend.Delete(); // Удаляем легенду, так как она не нужна для одной серии данных

chartRange.Columns.AutoFit();

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при создании диаграммы: {ex.Message}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

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

{

SaveChanges();

}

private void SaveChanges()

{

try

{

if (dataGridView.DataSource is DataTable dataTable && !string.IsNullOrEmpty(currentTable))

{

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

string query = $"SELECT \* FROM [{currentTable}]";

using (SqlDataAdapter adapter = new SqlDataAdapter(query, connection))

{

using (SqlCommandBuilder builder = new SqlCommandBuilder(adapter))

{

adapter.Update(dataTable); // Сохраняет новые, изменённые и удалённые строки

}

}

MessageBox.Show("Изменения успешно сохранены!", "Успех", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

else

{

MessageBox.Show("Данные таблицы не загружены или произошла ошибка.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при сохранении изменений: {ex.Message}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

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

{

// Автоматически сохраняем изменения после завершения редактирования ячейки

SaveChanges();

}

}

}