**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.

1. **ОБЩИЙ РАЗДЕЛ**
   1. **Описание предметной области**

Страховые компании играют важную роль в обеспечении финансовой защиты клиентов от различных рисков, таких как ущерб имуществу, здоровью или гражданской ответственности. Обработка большого объема данных о полисах, клиентах и страховых случаях требует эффективных информационных систем.

Разрабатываемая информационная система для страховой компании предназначена для автоматизации процессов управления страховыми полисами, заявками и рисками. Система ориентирована на сотрудников страховой компании (менеджеров, администраторов) и клиентов, предоставляя инструменты для создания и поиска полисов, подачи заявок, управления дополнительными рисками и формирования отчетов. Реализованная с использованием Windows Forms и SQL Server, она обеспечивает надежное хранение данных, разграничение прав доступа и удобный интерфейс для работы с информацией о страховых продуктах, включая ОСАГО, КАСКО, страхование имущества, медицинское страхование и страхование путешественников.

* 1. **Постановка задачи**

Разработка информационной системы «Страховая компания» обусловлена необходимостью повышения эффективности управления страховыми процессами, улучшения качества обслуживания клиентов и укрепления конкурентных позиций компании на динамично развивающемся страховом рынке. В условиях стремительного роста объемов данных, включающих информацию о страховых полисах, клиентах, дополнительных рисках и заявках, страховые компании сталкиваются с вызовами, связанными с обработкой больших массивов информации при строгом соблюдении требований законодательства, таких как Закон РФ «Об организации страхового дела» и положения Гражданского кодекса РФ [1]. Традиционные подходы к управлению данными, основанные на ручной обработке или устаревших программных решениях, часто приводят к задержкам, ошибкам и снижению качества клиентского сервиса. Современные реалии требуют внедрения автоматизированных систем, способных оптимизировать рутинные операции, обеспечить надежное хранение данных и повысить прозрачность всех процессов для участников — от клиентов, оформляющих полисы, до администраторов, управляющих системой.

Информационная система «Страховая компания» разработана для создания единой цифровой платформы, которая интегрирует управление страховыми продуктами, обработку клиентских заявок и анализ данных в реальном времени. Система реализована с использованием технологии Windows Forms и базы данных SQL Server, что обеспечивает высокую производительность, надежность и защиту информации. Она предоставляет пользователям удобный интерфейс с пастельной цветовой палитрой и закругленными элементами, соответствующий современным стандартам usability, что упрощает взаимодействие с системой для всех категорий пользователей. Основной функционал включает автоматизацию процессов создания страховых полисов, их поиска по номеру или типу страхования, а также расчета страховых премий на основе базовых ставок (например, 2.5% для ОСАГО или 4.0% для КАСКО) и дополнительных рисков, таких как «Угон» (0.7%) или «Залив квартиры» (0.5%).

Особое внимание уделено поддержке ролевого доступа: клиенты через специализированную форму могут подавать заявки на новые полисы или просматривать свои действующие договоры, не имея возможности редактировать или удалять данные; менеджеры получают доступ к управлению полисами, обработке заявок и настройке дополнительных рисков; администраторы контролируют конфигурацию системы. Система позволяет гибко настраивать страховые продукты, включая ОСАГО, КАСКО, страхование имущества, медицинское страхование и страхование путешественников, с учетом специфических рисков для каждого типа.

Дополнительно система поддерживает формирование аналитических отчетов, которые помогают выявлять тенденции в страховых выплатах, оценивать популярность продуктов и прогнозировать финансовые показатели. Интеграция с SQL Server обеспечивает проверку данных на уровне базы, предотвращая ошибки, что повышает надежность работы. Система также способствует улучшению клиентского опыта за счет упрощения подачи заявок и быстрого доступа к информации о полисах, что особенно важно для удержания клиентов в условиях высокой конкуренции. В перспективе система может быть расширена за счет внедрения модуля анализа страховых случаев, интеграции с внешними API для проверки данных клиентов или использования машинного обучения для оценки рисков, что позволит компании адаптироваться к новым вызовам страхового рынка.

* + 1. **Аналоги**

Одним из аналогом данного курсового проекта является Система IS-AODB, является информационным ядром управления аэропортом. Ее можно описать как аналог системы «Amadus Airport Operational Database», которая широко используется в аэропортах для централизованного управления данными.

В отличие от моего курсового проекта, IS-AODB имеет мобильное приложение для мониторинга операций, а также расширенный функционал.

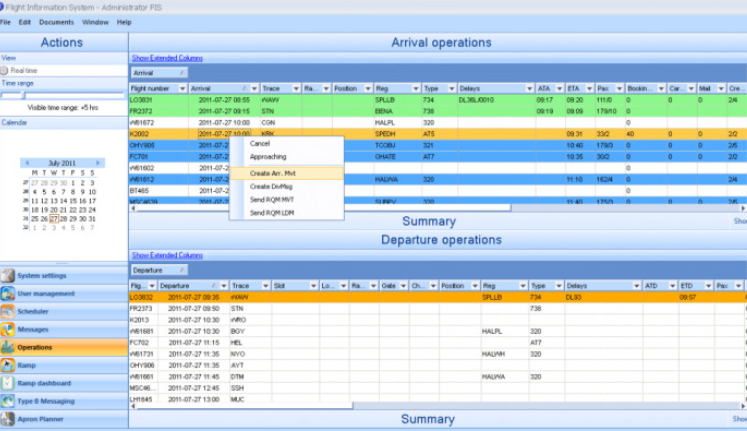


Рисунок 1 - IS-AODB

* 1. **Сравнение технологий**
     1. **Технологии для разработки интерфейса**

В качестве клиента была выбрана приложение с графическим интерфейсом. Было важно, чтобы программа работала быстро и компилировалась в исходный код.

Мой выбор стоял между С++ (библиотека GTK) и C# (.NET framework)

Буду сравнивать их по критериям: Быстрая разработка интерфейсов, удобство работы с данными, экосистема

С++ (GTK):

C++ не имеет встроенных высокоуровневых инструментов для работы с базами данных. Для интеграции с SQL или NoSQL базами данных (например, PostgreSQL, MySQL) нужно использовать сторонние библиотеки, такие как ODBC, что требует дополнительной настройки.

Работа с данными требует ручного написания запросов и обработки результатов, что увеличивает объем кода и вероятность ошибок.

C# (.NET framework)

NET Framework (особенно с Windows Forms или WPF) предоставляет мощные визуальные редакторы в Visual Studio, позволяющие быстро проектировать интерфейсы с помощью drag-and-drop.

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

* + 1. **Технологии разработки сервера**

Данные будут хранится на сервере в БД. Стоял выбор между самыми популярными СУБД: Microsoft SQL Server и PostreSQL.

Исходя из выбранного языка программирования, основным критерием выбора СУБД является интеграция с ним:

SQL Server (SSMS) - Нативная интеграция с .NET через System.Data.SqlClient.

P ostgreSQL - Интеграция через драйвер Npgsql,

Поскольку SSMS подключается нативное, именно она была выбрана в качестве технологии для сервера.

* 1. **Требования**

От сервера потребуется обработка следующих запросов:

- Регистрация новых пассажиров на рейсы;

- Рассчет времени до отпуска и выплаты зарплаты сотрудника;

- получение всех списков;

Также, сервер должен реализовывать следующие команды администратора:

- Создание бэкапа БД;

- Выгрузка файлов с данными БД;

- Обработка SQL команд;

1. **СПЕЦИАЛЬНЫЙ РАЗДЕЛ**
   1. **Диаграмма прецендентов использования**



Рисунок 1 – UML диаграмма

* 1. **Инфологическая модель**

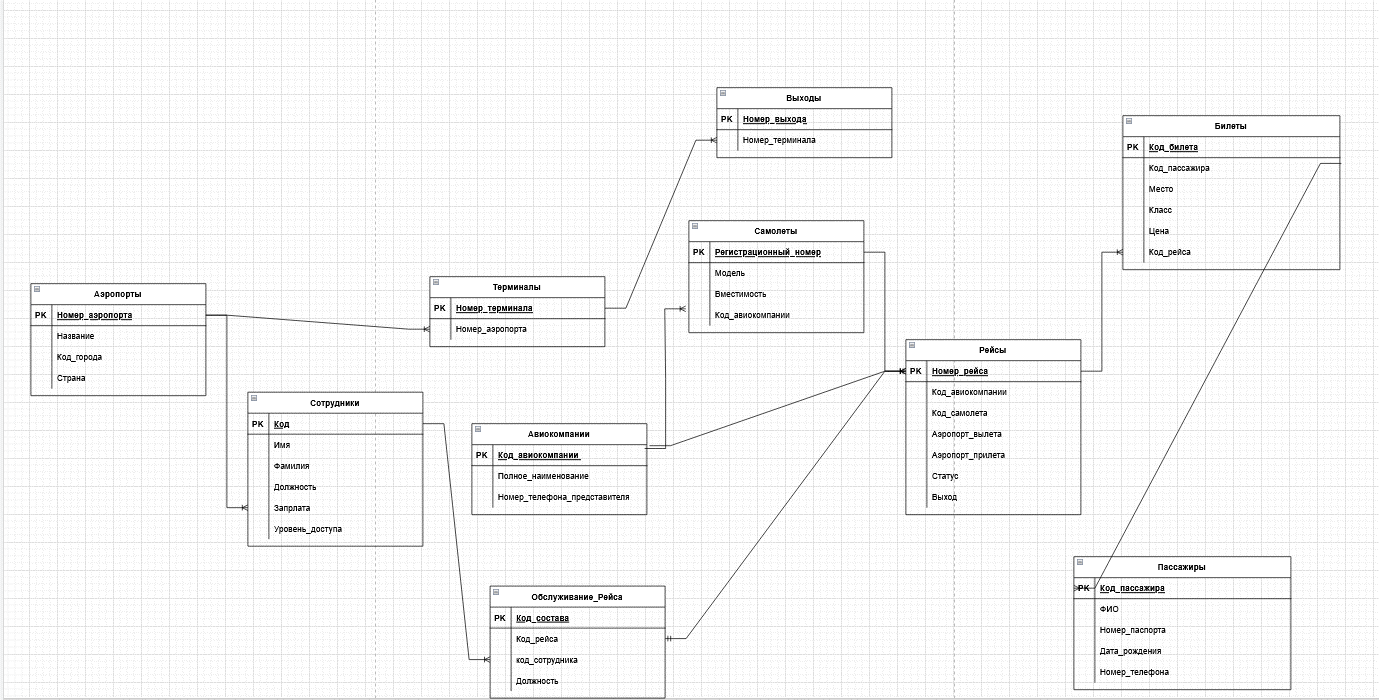


Рис 2 – Инфологическая модель БД

* 1. **Проект основных дисплейных фрагментов**

При запуске программы появляется окно

**ЗАКЛЮЧЕНИЕ**

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧИНКОВ**

1. <https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/>
2. <https://learn.microsoft.com/ru-ru/dotnet/csharp/advanced-topics/interop/how-to-access-office-interop-objects>
3. <https://learn.microsoft.com/ru-ru/troubleshoot/developer/visualstudio/csharp/language-compilers/create-sql-server-database-programmatically>

**ПРИЛОЖЕНИЕ А**

**Листинг SQL-скрипта**

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 Б**

**Исходный код**

**LOGINPANEL.cs**

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;

}

}

}

}

**PassagesForm.cs**

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();

}

}

}