

Министерство науки и высшего образования Российской Федерации
Муромский институт (филиал)
федерального государственного бюджетного образовательного учреждения высшего образования
«Владимирский государственный университет
Имени Александра Григорьевича и Николая Григорьевича Столетовых»
(МИВлГУ)

Факультет ИТ
Кафедра ПИн

КУРСОВАЯ РАБОТА

по Системы управления базами данных

Тема Автоматизированная информационная система Страхового агентства

(оценка)

Руководитель

Быков А.А.
(фамилия, инициалы)

(подпись) (дата)

Члены комиссии

(подпись) (Ф.И.О.)

Студент ПИн-119
(группа)

(подпись) (Ф.И.О.)

Лямина И.А.
(фамилия, инициалы)

(подпись) (дата)

Муром 2021

Муромский институт (филиал) федерального государственного бюджетного
образовательного учреждения высшего образования
«Владимирский государственный университет имени Александра Григорьевича и
Николая Григорьевича Столетовых»

Факультет информационных технологий

«УТВЕРЖДАЮ»

Зав. кафедрой ПИН А.Л. Жизняков

(подпись)

« » 2021 г.

ЗАДАНИЕ

На курсовую работу по курсу Системы управления базами данных

Студент Лямина И.А. гр. ПИН-119

1. Тема работы АИС Страхового агентства

2. Сроки сдачи студентом законченного проекта «10» декабря 2021 г.

3. Исходные данные к проекту

предоставляет услуги по страхованию автомобилей ОСАГО, КАСКО. Клиент заключает с компанией договор страхования сроком (6 или 12 месяцев). В договоре указывается сумма страховки, получаемой страховщиком при наступлении страхового случая. В полисе указываются данные автомобиля, лиц, допущенных к управлению, сроки страхования и т.п.

Если в период действия договора наступает страховой случай, компания (после проверки обстоятельств) производит выплату клиенту страховой суммы. Если страховые обстоятельства не наступают до окончания срока действия договора, он утрачивает силу. В таком случае компания обычно предупреждает клиента о скором истечении срока действия договора, предлагая ему перезаключить этот договор.

Руководство компании интересует спрос на виды страхования, поэтому оно периодически анализирует количество и суммы заключенных договоров по каждому из видов, а также оценивает риски, подсчитывая суммы страховых выплат по каждому виду договоров. Кроме того, составляется финансовый отчет деятельности компании за заданный период времени.

В БД предусмотреть хранение изображений (минимум в одном поле) в соответствии с тематикой курсовой работы.

4. Содержание расчетно–пояснительной записки (перечень подлежащих разработке вопросов).

Аннотация (на двух языках)

Содержание

Введение

1. Анализ технического задания

2. Разработка моделей данных

3. Разработка и реализация АИС

4. Тестирование АИС

Заключение

Список используемой литературы

Приложение 1. Модели данных

Приложение 2. Текст программы

Приложение 3. Снимки окон программы (скриншоты программы)

5. Перечень графического материала (с точным указанием обязательных чертежей и графиков)

*Концептуальная, логическая и физическая модели данных,
текст программы с комментариями,
скриншоты окон программы*

6. Рекомендуемая литература

1. Медведкова, И. Е. Базы данных : учебное пособие / И. Е. Медведкова, Ю. В. Бугаев, С. В. Чикунов. — Воронеж : Воронежский государственный университет инженерных технологий, 2014. — 104 с. — ISBN 978-5-00032-060-0. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. — URL: <https://www.iprbookshop.ru/47418.html>

2. Панова, Н. Ф. FireBird. Установка, разработка баз данных, реализация запросов : методические указания / Н. Ф. Панова. — Оренбург : Оренбургский государственный университет, ЭБС АСВ, 2014. — 45 с. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. — URL: <https://www.iprbookshop.ru/50015.html>

3. Шацков, В. В. Программирование приложений баз данных с использованием СУБД MS SQL Server : учебное пособие / В. В. Шацков. — Санкт-Петербург : Санкт-Петербургский государственный архитектурно-строительный университет, ЭБС АСВ, 2015. — 80 с. — ISBN 978-5-9227-0607-0. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. — URL: <https://www.iprbookshop.ru/63638.html>

7. Дата выдачи задания 07.09.2021

8. Календарный график работы над проектом (на весь период проектирования, с указанием сроков выполнения и трудоемкости отдельных этапов)

Введение, анализ ТЗ, описание БД 10%, 2 нед

Описание бизнес-логики , разработка концептуальной модели данных 20%, 3 нед.

Разработка логической модели данных, нормализация таблиц 30%, 5 нед.

Разработка физической модели данных, разработка основных

запросов к базе данных 40%, 6 нед

Программирование основных функций АИС 60%, 12 нед

Завершение программирования, реализация контроля действий

пользователя, обработка исключительных ситуаций 70%, 13 нед

Тестирование, отладка 80%, 14 нед.

Оформление пояснительной записи 100%, 15 нед.

Руководитель _____
(подпись)

Задание принял к исполнению (дата) _____

Подпись студента _____

Примечание. Это задание прилагается к законченному проекту.

В данной курсовой работе была разработана БД и АИС Страхового агентства. В ходе выполнения курсовой работы выявлены требования к программе, разработаны модели данных. На основе разработанных моделей создана БД и приложение, работающие с ней. На основе разработанных моделей реализован набор классов и разработано приложение на языке программирования C# в среде разработки Microsoft Visual Studio 2019. На заключительном этапе работы произведено тестирование разработанного продукта.

In this course work there was a budget database and an automated information system of the Insurance Agency. In the course of the course work, the requirements for the program were identified, data models were developed. On the basis of the developed models, a database and an application working with it have been created. On the basis of the developed models, a set of classes was implemented and an application was developed in the C # programming language in the Microsoft Visual Studio 2019 development environment. At the final stage of the product operation, the developed product was tested.

Содержание

Введение	6
1 Анализ технического задания.....	8
1.1 Аналоги.....	8
1.2 Обоснование выбора средств реализации	11
1.3 Функциональные возможности.....	12
1.4 Системные требования.....	13
2 Разработка моделей данных	14
2.1 Концептуальная модель данных.....	15
2.2 Функциональные зависимости.....	18
2.3 Логическая модель данных	20
2.4 Соответствие сущностей логической и физической моделей	21
2.5 Физическая модель данных.....	26
3 Разработка и реализация АИС	28
4 Руководство администратора	40
5 Тестирование АИС	41
Заключение.....	68
Список используемой литературы.....	69
Приложение 1 Создание БД	70
Приложение 2 Снимки окон программы (скриншоты программы).....	74
Приложение 3 Текст программы	83

Введение

Сегодня, большинство пользовательских приложений рассчитаны на работу с базами данных, в которых может храниться не только текстовая информация, но и графика, музыка или любой другой тип данных. Так как потребность в использовании БД растет, появилось большое количество СУБД.

Система управления базами данных (СУБД) – это комплекс программно-языковых средств, позволяющих создать базы данных и управлять данными. Иными словами, СУБД — это набор программ, позволяющий организовывать, контролировать и администрировать базы данных.

Эффективное управление предприятием в современных условиях невозможно без использования компьютерных технологий. Разработка автоматизированной информационной системы (АИС) страхового агентства важна, так как будет разработано приложение для автоматизации процесса страхования, которое поможет страховым агентам сократить время на работу с документацией.

Внедрение информационных технологий в процесс планирования и управления деятельностью страховых компаний предусматривает не только обработку больших и взаимосвязанных массивов данных, но может использоваться также для их анализа и обоснований вариантов управленческих решений.

Объемы информации, высокие требования к точности и достоверности, необходимость эффективного анализа финансового состояния клиентуры и страховой фирмы — вот основные причины, предопределяющие автоматизацию страхового бизнеса.

Функции, которые будут автоматизированы:

– Функция поиска. Если приходил страхователь, то сотруднику придется искать вручную его бумажные договора, возможно спрашивая

Изм.	Лист	№ докум.	Подпись	Дата

самого клиента. В АИС достаточно написать номер телефона или паспорт страхователя и сотрудник получит всю информацию как о самом клиенте, так и о всех заключенных договорах;

– Функция продления договоров. Чтобы продлить договор приходилось переписывать и перепроверять все данные. В АИС можно просто нажать на кнопку продлить и выбрать срок, на который нужно продлить;

– Функция составления отчётов. Отчёты составляются автоматически. Сотруднику достаточно выбрать вид страхования и период времени, и он получит следующие данные: количество и сумма заключенных договоров, сумма страховых выплат.

Целью курсовой работы является проектирование и разработка автоматизированной информационной системы страхового агентства.

Для достижения поставленной цели были поставлены следующие задачи:

- проанализировать предметную область;
- разработать модели данных;
- реализовать базу данных;
- разработать клиентское приложение;
- протестировать программный продукт.

Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

1 Анализ технического задания

1.1 Аналоги

Для наиболее точного анализа предоставленного технического задания были найдены несколько аналогов разрабатываемого программного средства.

Полисы ОСАГО 1.0.9

Полисы ОСАГО 1.0.9 [1] - программа для заполнения договоров обязательного автострахования. Программа может быть полезна страховым компаниям, страховым агентам, работающим с разными компаниями.

В программе реализовано ведение журнала полисов, хранение информации по страхователям, страховщикам, расчет страховой премии. Скриншот работы программы представлен на рисунке 1.

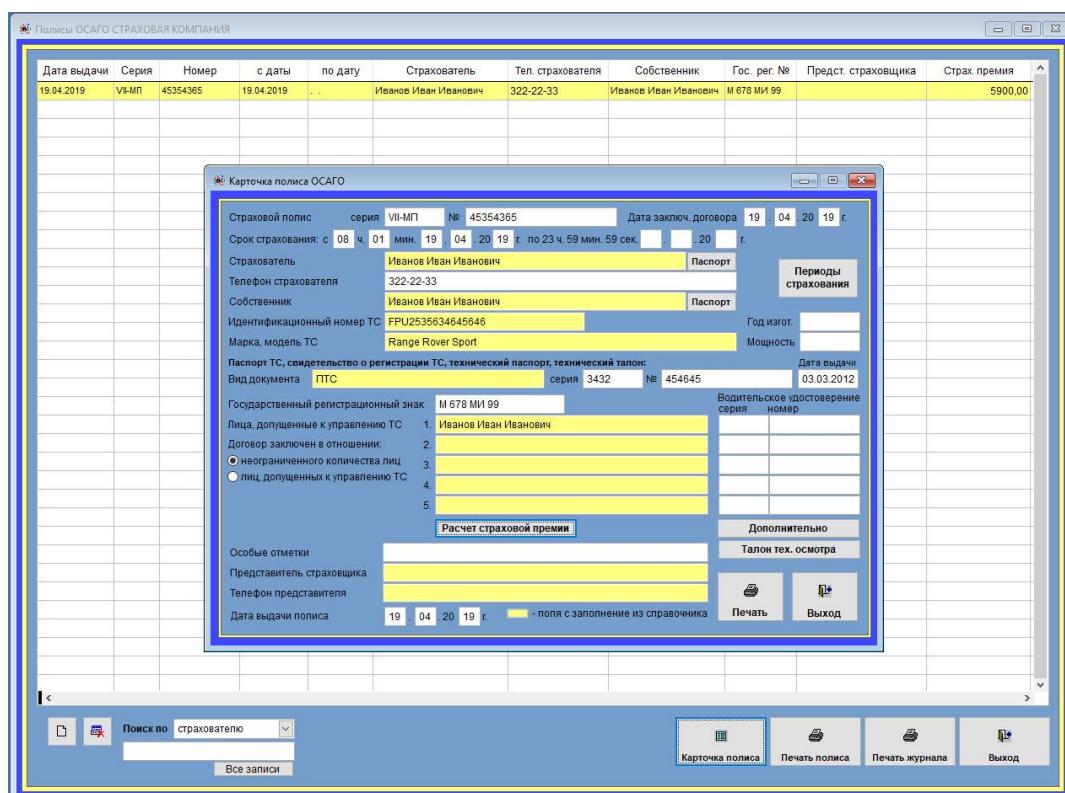


Рисунок 1 - скриншот работы программы

Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

Учёт автострахования ОСАГО

Программа "Учёт автострахования ОСАГО" [2] предназначена для автоматизации учёта транспортных средств в пунктах страхования, регистрации, заполнения и печати бланков заявлений, полисов и квитанций, ведения сопутствующей отчётности в количественном и стоимостном разрезе. Таким образом, один раз заполнив данные на клиента, у Вас всегда есть возможность оперативно поднять всю историю взаимодействия с ним. При том Вы сможете за считанные секунды выписать новый полис на основании предыдущего полиса. Скриншот работы программы представлен на рисунке 2.

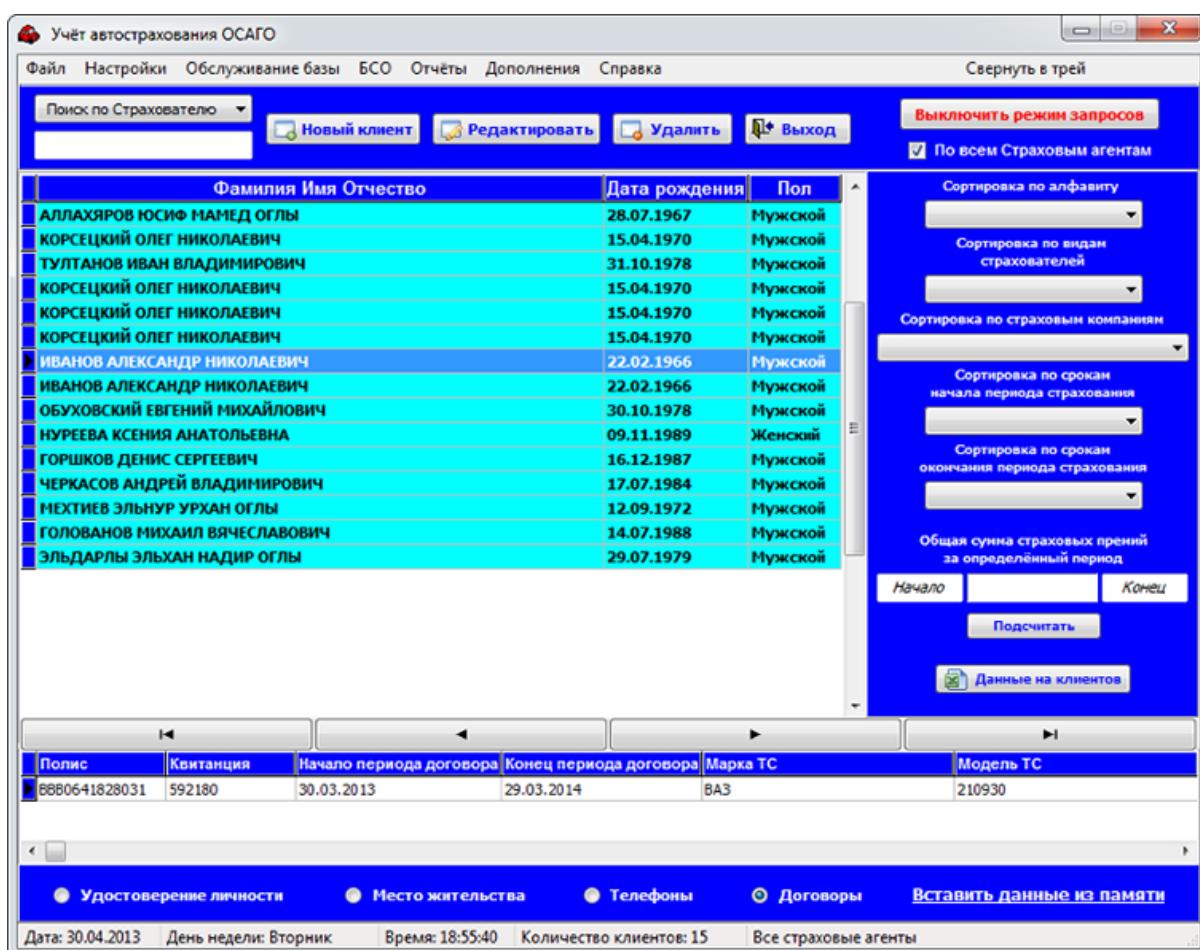


Рисунок 2 - скриншот работы программы

Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

Также программа была разработана на основании действующих полисов ОСАГО и КАСКО. Пример полиса ОСАГО [4] представлен на рисунке 3, а КАСКО [3] на рисунке 4.

Наименование страховщика																
СТРАХОВОЙ ПОЛИС серия ВВВ № 0000000000 МЕСТО ДЛЯ ШТАМПА ОБЯЗАТЕЛЬНОГО СТРАХОВАНИЯ ГРАЖДАНСКОЙ ОТВЕТСТВЕННОСТИ ВЛАДЕЛЬЦА ТРАНСПОРТНОГО СРЕДСТВА СТРАХОВОЙ КОМПАНИИ																
Срок действия договора с 00 ч. 00 мин. 29 06 20 10 г. по 23 ч. 59 мин. 59 сек. 28 06 20 11 г.																
Страхование распространяется на страховые случаи, произошедшие в период использования транспортного средства в течение срока действия договора с 28.06.2010 г. по 28.06.2011 г., с —. —.20— г. по —. —.20— г., с —. —.20— г. по —. —.20— г.																
1. Страхователь (полное наименование юридического лица или фамилия, имя, отчество гражданина) Иванов Иван Петрович																
Собственник транспортного средства (полное наименование юридического лица или фамилия, имя, отчество гражданина) Петров Петр Иванович																
2. Транспортное средство <table border="1"> <tr> <td>Марка, модель транспортного средства ВАЗ 2110</td> <td>Идентификационный номер транспортного средства V3RK87J7890657837</td> <td>Государственный регистрационный знак транспортного средства У777РД</td> </tr> <tr> <td colspan="3">Паспорт транспортного средства, свидетельство о регистрации транспортного средства, технический паспорт, технический талон (либо аналогичный документ) вид документа ПТС</td> </tr> <tr> <td colspan="3">серия 78 КУ номер 322265</td> </tr> </table>		Марка, модель транспортного средства ВАЗ 2110	Идентификационный номер транспортного средства V3RK87J7890657837	Государственный регистрационный знак транспортного средства У777РД	Паспорт транспортного средства, свидетельство о регистрации транспортного средства, технический паспорт, технический талон (либо аналогичный документ) вид документа ПТС			серия 78 КУ номер 322265								
Марка, модель транспортного средства ВАЗ 2110	Идентификационный номер транспортного средства V3RK87J7890657837	Государственный регистрационный знак транспортного средства У777РД														
Паспорт транспортного средства, свидетельство о регистрации транспортного средства, технический паспорт, технический талон (либо аналогичный документ) вид документа ПТС																
серия 78 КУ номер 322265																
3. Договор заключен в отношении: неограниченного количества лиц, допущенных к управлению транспортным средством лиц, допущенных к управлению транспортным средством																
<table border="1"> <thead> <tr> <th>№ п/п</th> <th>Лица, допущенные к управлению транспортным средством (фамилия, имя, отчество)</th> <th>Водительское удостоверение (серия, номер)</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>Иванов Иван Петрович</td> <td>78ВА005245</td> </tr> <tr> <td>2</td> <td>Степанов Олег Николаевич</td> <td>78ВН055641</td> </tr> <tr> <td></td> <td></td> <td></td> </tr> <tr> <td></td> <td></td> <td></td> </tr> </tbody> </table>		№ п/п	Лица, допущенные к управлению транспортным средством (фамилия, имя, отчество)	Водительское удостоверение (серия, номер)	1	Иванов Иван Петрович	78ВА005245	2	Степанов Олег Николаевич	78ВН055641						
№ п/п	Лица, допущенные к управлению транспортным средством (фамилия, имя, отчество)	Водительское удостоверение (серия, номер)														
1	Иванов Иван Петрович	78ВА005245														
2	Степанов Олег Николаевич	78ВН055641														
4. Страховая сумма, в пределах которой страховщик при наступлении каждого страхового случая (независимо от количества страховых случаев в течение срока действия договора обязательного страхования) обязуется возместить потерпевшим причиненный вред, составляет: в части возмещения вреда, причиненного жизни или здоровью каждого потерпевшего, – не более 160 тыс. рублей; в части возмещения вреда, причиненного имуществу нескольких потерпевших, – не более 160 тыс. рублей; в части возмещения вреда, причиненного имуществу одного потерпевшего, – не более 120 тыс. рублей.																
5. Страховой случай – наступление гражданской ответственности владельца транспортного средства за причинение вреда жизни, здоровью или имуществу потерпевших при использовании транспортного средства, влекущее за собой в соответствии с договором обязательного страхования обязанность страховщика осуществлять страховую выплату.																
6. Страховой полис действует на территории Российской Федерации.																
7. Страховая премия 3564 (Три тысячи пятьсот шестьдесят четыре) руб 00 коп.																
8. Особые отметки																
Дата заключения договора 27 июня 2010 г. Страхователю выданы Правила обязательного страхования гражданской ответственности владельцев транспортных средств, перечень представителей страховщика в субъектах Российской Федерации согласно приложению 2 бланка извещения о дорожно-транспортном происшествии.																
 Страхователь Петров А.С. Дата выдачи полиса 27 июня 2010 г. <small>Фамилия, имя, отчество</small>																

Рисунок 3 - пример страхового полиса ОСАГО

Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

МИВУ 09.03.04-8.000 ПЗ

Лист

10

BCK СТРАХОВОЙ ДОМ
военно-страховая компания
MILITARY INSURANCE COMPANY

СТРАХОВОЙ ПОЛИС № 10868C500C INSURANCE POLICY

Настоящим Полисом подтверждается заключение Договора страхования транспортного средства (ТС) и дополнительного оборудования, установленного на нем, на условиях Правил добровольного страхования средств наземного транспорта, гражданской ответственности владельцев транспортных средств и водителя и пассажиров от несчастного случая" ОАО "BCK" от 28.03.2008 г., являющихся неотъемлемой частью Договора.

ВИД ПОЛИСА /Л-Каско	ДОГОВОР ЗАКЛЮЧАЕТСЯ В: V Рублях РФ	В валютом эквиваленте: экв. Долларов США	экв. Евро
СТРАХОВАТЕЛЬ			
Фамилия Имя Отчество			
Адрес места жительства, контактный телефон			
Паспортные данные			
Выгодоприобретатель			
ТРАНСПОРТНОЕ СРЕДСТВО (ТС)			
Марка / модель	MAZDA 6	Паспорт ТС	78 УЕ
Идентификационный № (VIN)		Год выпуска ТС	2008
Кузов (шасси) №		Мощность л. с.	120
Номер двигателя:	20260575	Регистрационный знак	
На момент заполнения заявления ТС:	<input checked="" type="checkbox"/>	Оборудовано:	<input type="checkbox"/>
механическим противоугонным устройством:	(наименование)		
<input checked="" type="checkbox"/>	электронным противоугонным устройством:	(наименование) Штатное	
спутниковой охранный-поисковой системой:	(наименование)		
Лица, допущенные к управлению ТС:	<input checked="" type="checkbox"/>	Водители согласно перечню:	<input type="checkbox"/>
Любое лицо, допущенное к управлению ТС на законных основаниях		Фамилия Имя Отчество	Дата рождения
1.		26.08.1975	г. 2003
2.		-----	-----
3.		-----	-----
4.		-----	-----
5.		-----	-----
ДОПОЛНИТЕЛЬНОЕ ОБОРУДОВАНИЕ (ДО)			
Наименование объекта ДО	Страховая сумма по риску "Автокаско"	Наименование объекта ДО	Страховая сумма по риску "Автокаско"
-----	-----	-----	-----
-----	-----	-----	-----
Страхование транспортного средства и дополнительного оборудования осуществляется по рискам:			
Риск	Объект	Общая страховая сумма	Безусловная франшиза
Автокаско	ТС	650 000,00	не установлена
	ДО	-----	не установлена
Внешнее воздействие	ТС и ДО	32 500,00	не установлена
Возмещение ущерба ТС (ДО) в результате любого внешнего механического воздействия производится без предоставления подтверждающих документов компетентным органов в пределах страховой суммы, установленной по риску "Внешнее воздействие". При этом лимит ответственности Страховщика по каждому объекту ДО устанавливается в размере страховой суммы по риску "Автокаско", установленной для данного объекта, но в любом случае не более страховой суммы по риску "Внешнее воздействие".			
СТРАХОВАНИЕ ГРАЖДАНСКОЙ ОТВЕТСТВЕННОСТИ ВЛАДЕЛЬЦА ТРАНСПОРТНОГО СРЕДСТВА			
Страховая сумма	Страховая премия	В соответствии с п. 8.3.7. Правил, выплата страхового возмещения производится с учетом произведенного страхового возмещения по договору обязательного страхования гражданской ответственности владельца ТС	
-----	-----	-----	
СТРАХОВАНИЕ ВОДИТЕЛЯ ИЛИ ПАССАЖИРОВ ОТ НЕСЧАСТНОГО СЛУЧАЯ			
Лимит ответственности на одно застрахованное место:		Страховая премия	
Установлен в размере:	<input type="checkbox"/> Не установлен (паушальная система)	Временная утрата трудоспособности:	
водителя	-----	-----	
на место водителя	-----	-----	
на место(а) рядом с водителем (кол-во)	-----	-----	
на места пассажиров (кол-во)	-----	-----	
	<input type="checkbox"/> Застрахована		
	<input type="checkbox"/> Не застрахована		

ИИН 770026574 Финанс ОАО «Военно-страховая компания» «BCK-Москва»: 4020260229280010830 //г 300708040000000229 Банк 044523225 в Кийском ОСБ-3278 Сбербанк России ОАО - Москва А/Олимпийская, Москва, 121152, Россия, тел +7(495)785-27-79, fax +7(495)829-34-34, www.bck.ru, e-mail: info@bck.ru, факс: 0956421-77-72/1952 Минск, Республика Беларусь, тел +375 17 32-76-00, факс: +375 17 32-76-01

Рисунок 4 - пример страхового полиса КАСКО

1.2 Обоснование выбора средств реализации

Для разработки базы данных необходимо выбрать СУБД. В настоящее время существует достаточно большое количество различных СУБД, позволяющих создавать и управлять разного рода базами данных.

Чтобы разработать БД Страхового агентства выбрана СУБД SQL Server. SQL Server является надежной базой данных для любых целей, может продолжать расширяться по мере наполнения информацией, без заметного

Изм.	Лист	№ докум.	Подпись	Дата	Лист	
					МИВУ 09.03.04-8.000 ПЗ	
						11

уменьшения быстродействия операций с записями в многопользовательском режиме. Обеспечивается максимальная безопасность. Поскольку безопасность на уровне пользователя, пользователи могут иметь ограниченный доступ к записи данных, тем самым защищая их от модификации или поиска, указав доступ на уровне пользовательских привилегий.

SQL Server обрабатывает запросы от пользователей и только отправляет пользователю результаты запроса. Таким образом, минимальная информация передается по сети. Это улучшает время отклика и устраняет узкие места в сети.

Техническое обслуживание SQL Server очень простое и не требует больших знаний. Возможны изменения в структуре данных, а также резервное копирование во время работы сервера, без остановки.

В качестве среды для разработки прикладной программы для работы с созданной в SQL Server базой данных, была выбрана среда объектно-ориентированного программирования Visual Studio, язык программирования C#. Visual Studio включает в себя полный набор новых и улучшенных функций, упрощающих все этапы процесса разработки приложения.

Визуальная часть приложения разработана на WPF - системе для построения клиентских приложений Windows с визуально привлекательными возможностями взаимодействия с пользователем.

Разрабатываемое программного средства должна обеспечивать получение из базы данных всей необходимой информации в полном объеме, а также возможность её редактирования и удаления. Также программа должна иметь визуальный интерфейс для работы с базой данных.

1.3 Функциональные возможности

В данной курсовой работе необходимо спроектировать и разработать АИС Страхового агентства.

Изм.	Лист	№ докум.	Подпись	Дата

Пользоваться системой страхового агентства будут только сотрудники, которые будут иметь возможность просмотра, добавления, изменения и удаления данных.

Программа должна содержать следующие функциональные возможности:

- добавление данных о страхователях, сотрудниках, автомобилях, лицах, допущенных к управлению;
- добавление полисов;
- возможность работы со страховыми случаями;
- возможность изменения информации;
- возможность удаления выбранной информации;
- предоставление информации на форме в табличном виде;
- возможность анализировать количество и суммы заключенных договоров по каждому из видов, а также оценивать риски, подсчитывая суммы страховых выплат по каждому виду договоров, а также составлять финансовый отчет деятельности компании за заданный период времени.

Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

2 Разработка моделей данных

Модели базы данных разрабатываются, чтобы определить логическую структуру базы данных и каким образом данные могут храниться, организовываться и обрабатываться. Разработка моделей данных является очень важным этапом в разработке АИС, на котором выделяются сущности, атрибуты сущностей и связи между ними. Необходимо разработать концептуальную, логическую и физическую модели данных.

Но прежде, чем разрабатывать модели данных нужно выявить ограничения предметной области. В данной предметной области существуют следующие ограничения:

- страхователь может оформлять несколько полисов;
- полис оформляется одним страхователем;
- полис оформляется на один автомобиль;
- на каждый автомобиль можно оформить несколько полисов;
- одновременно на автомобиль могут быть оформлены полисы ОСАГО и КАСКО. Пока не пройдёт срок действия ОСАГО нельзя оформлять ещё один полис ОСАГО. Аналогично с КАСКО;
- у автомобиля должна быть хотя бы одна фотография, но есть возможность закреплять за каждым автомобилем сразу несколько фотографий;
- каждая фотография может закрепляться только за одним автомобилем;
- полис не перестаёт действовать после наступления страхового случая, поэтому за каждым полисом могут быть закреплены несколько страховых случаев;
- страховой случай закрепляется за одним конкретным полисом;
- при оформлении полиса должен указываться сотрудник, который его оформил;
- сотрудник может оформить несколько полисов;

Изм.	Лист	№ докум.	Подпись	Дата

– в полисе должно указываться хотя бы одно лицо, допущенное к управлению, но есть возможность закреплять за каждым полисом сразу несколько лиц, допущенных к управлению;

– лицо, допущенное к управлению может быть закреплено за несколькими полисами;

– если страхователь оформляет полис, то он не является лицом, допущенным к управлению по умолчанию, поэтому при необходимости его также нужно добавлять в данный список.

2.1 Концептуальная модель данных

Концептуальная модель — это отражение предметной области, для которой разрабатывается база данных. Так, все сущности обозначаются в виде прямоугольника. Атрибуты, характеризующие объект - в виде овала, а связи между объектами - ромбами. Мощность связи обозначаются стрелками (в направлении, где мощность равна многим - двойная стрелка, а со стороны, где она равна единице - одинарная).

Концептуальная модель, разработанная для БД Страхового агентства представлена на рисунке 5. Объектами на разработанной модели являются Страхователи, Автомобили, Изображения, Полисы, Лица, допущенные к управлению, Страховые случаи, Сотрудники. Страхователь может оформить полис, который включает Лица, допущенные к управлению, Страховые случаи и Автомобили, у которых есть изображения. Полис регистрирует Сотрудник.

Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

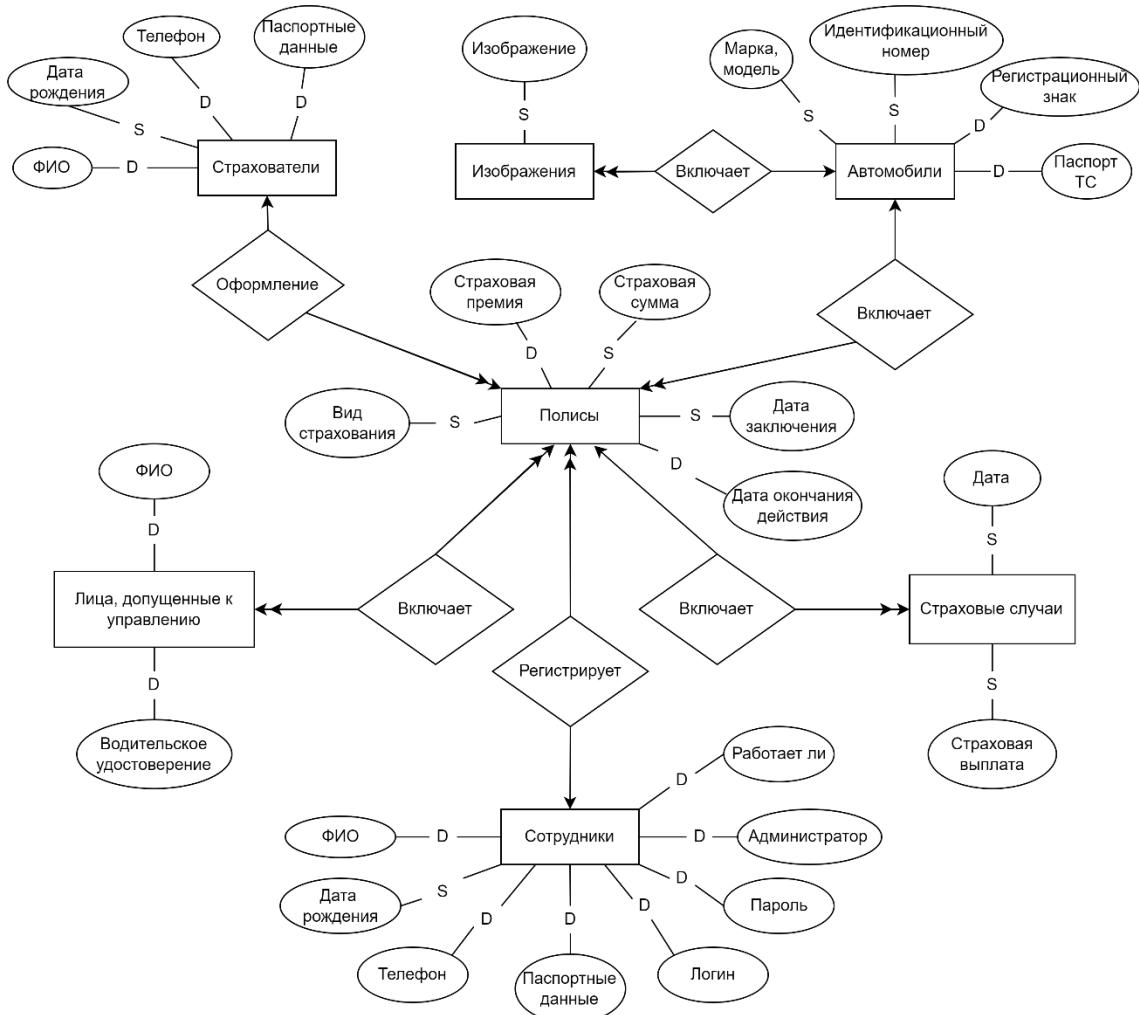


Рисунок 5 - Концептуальная модель данных

Страхователи

$\text{Dom}(\text{ФИО}) = \{\text{строка символов длиной не более } 64\}$

Dom (Дата рождения) = {строка символов длиной 10, символами являются быть цифры или «-»}

$\text{Dom}(\text{Телефон}) = \{\text{строка символов длиной 15, символами являются цифры}\}$

$\text{Dom}(\text{Паспортные данные}) = \{\text{строка символов длиной 10, символами являются цифры}\}$

Автомобили

$\text{Dom}(\text{Марка, модель}) = \{\text{строка символов длиной не более } 50\}$

<i>Изм.</i>	<i>Лист</i>	<i>№ докум.</i>	<i>Подпись</i>	<i>Дата</i>

Dom (Идентификационный номер) = {строка символов длиной 17, символами являются цифры и буквы}

Dom (Регистрационный знак) = {строка символов длиной не более 25, символами являются цифры и буквы}

Dom (Паспорт ТС) = {строка символов длиной 10, символами являются цифры и буквы}

Изображения

Dom (Изображение) = {закодированное изображение}

Полисы

Dom (Вид страхования) = {строка символов длиной 5, символами являются буквы}

Dom (Страховая сумма) = {строка символов длиной не более 10, символами являются цифры}

Dom (Страховая премия) = {строка символов длиной не более 10, символами являются цифры}

Dom (Дата заключения) = {строка символов длиной 10, символами могут быть цифры или «-»}

Dom (Дата окончания действия) = {строка символов длиной 10, символами могут быть цифры или «-»}

Лица, допущенные к управлению

Dom (ФИО) = {строка символов длиной не более 64}

Dom (Водительское удостоверение) = {строка символов длиной не более 10, символами являются цифры}

Страховые случаи

Dom (Дата) = {строка символов длиной 10, символами являются цифры или «.»}

Изм.	Лист	№ докум.	Подпись	Дата

Dom (Телефон) = {строка символов длиной 11, символами являются цифры}

Dom (Страховая выплата) = {строка символов длиной не более 50, символами являются цифры}

Сотрудники

Dom (ФИО) = {строка символов длиной не более 64}

Dom (Дата рождения) = {строка символов длиной 10, символами являются цифры или «.»}

Dom (Телефон) = {строка символов длиной 15, символами являются цифры}

Dom (Паспортные данные) = {строка символов длиной 10, символами являются цифры}

Dom (Логин) = {строка символов длиной не более 50}

Dom (Пароль) = {строка символов длиной не более 50}

Dom (Администратор) = {1 символ, символом является 1 или 0}

Dom (Работает ли) = {1 символ, символом является 1 или 0}

2.2 Функциональные зависимости

Данные, которые будут храниться в одном отношении после объединения должны быть связаны между собой. Эта связь определяется посредством функциональных зависимостей между атрибутами отношения. Это означает, что значения одного атрибута зависят от значений других атрибутов. Получить набор зависимостей возможно путем анализа ограничений предметной области. Таким образом, при функциональной зависимости значения кортежа на одном множестве атрибутов однозначно определяют значения кортежа на другом множестве атрибутов.

Страхователи(ФИО, Дата рождения, Телефон, Паспортные данные)

Функциональные зависимости:

Изм.	Лист	№ докум.	Подпись	Дата

– {Телефон, Паспортные данные} → ФИО

– {Телефон, Паспортные данные} → Дата рождения

Автомобили(Марка, модель, Идентификационный номер, Паспорт ТС, Регистрационный знак)

Функциональные зависимости:

– {Идентификационный номер} → Марка/модель

– {Идентификационный номер} → Паспорт ТС

– {Идентификационный номер} → Регистрационный знак

Изображения(Изображение)

Полисы(Вид страхования, Страховая премия, Страховая сумма, Дата заключения, Дата окончания действия)

Функциональные зависимости:

– {Вид страхования, Страховая премия} → Страховая сумма

– {Дата заключения} → Дата окончания действия

Лица, допущенные к управлению(ФИО, Водительское удостоверение)

Функциональные зависимости:

– {ФИО} → Водительское удостоверение

Сотрудники(ФИО, Дата рождения, Телефон, Паспортные данные, Логин, Пароль, Администратор, Работает ли)

Функциональные зависимости:

– {Телефон, Паспортные данные, Логин} → ФИО

– {Телефон, Паспортные данные, Логин} → Дата рождения

– {Логин} → Пароль

– {Телефон, Паспортные данные, Логин} → Администратор

– {Телефон, Паспортные данные, Логин} → Работает ли

Страховые случаи(Дата, Страховая выплата)

Функциональные зависимости:

– {Дата} → Страховая выплата

Изм.	Лист	№ докум.	Подпись	Дата

2.3 Логическая модель данных

Целью построения логической модели является получение графического представления логической структуры исследуемой предметной области. Логическая модель предметной области иллюстрирует сущности, а также их взаимоотношения между собой.

Логическая модель должна читаться по схеме: <Сущность 1> — <отношение / влияние> — <Сущность 2>. Чтение логической модели, представленной на рисунке 6: Страхователь оформляет полис. Полис включает страховые случаи и автомобиль. Автомобиль включает фотографии. Полис регистрирует сотрудника. Полис включает лица, допущенные к управлению (соединение через дополнительную таблицу, так как связь “многие ко многим”).

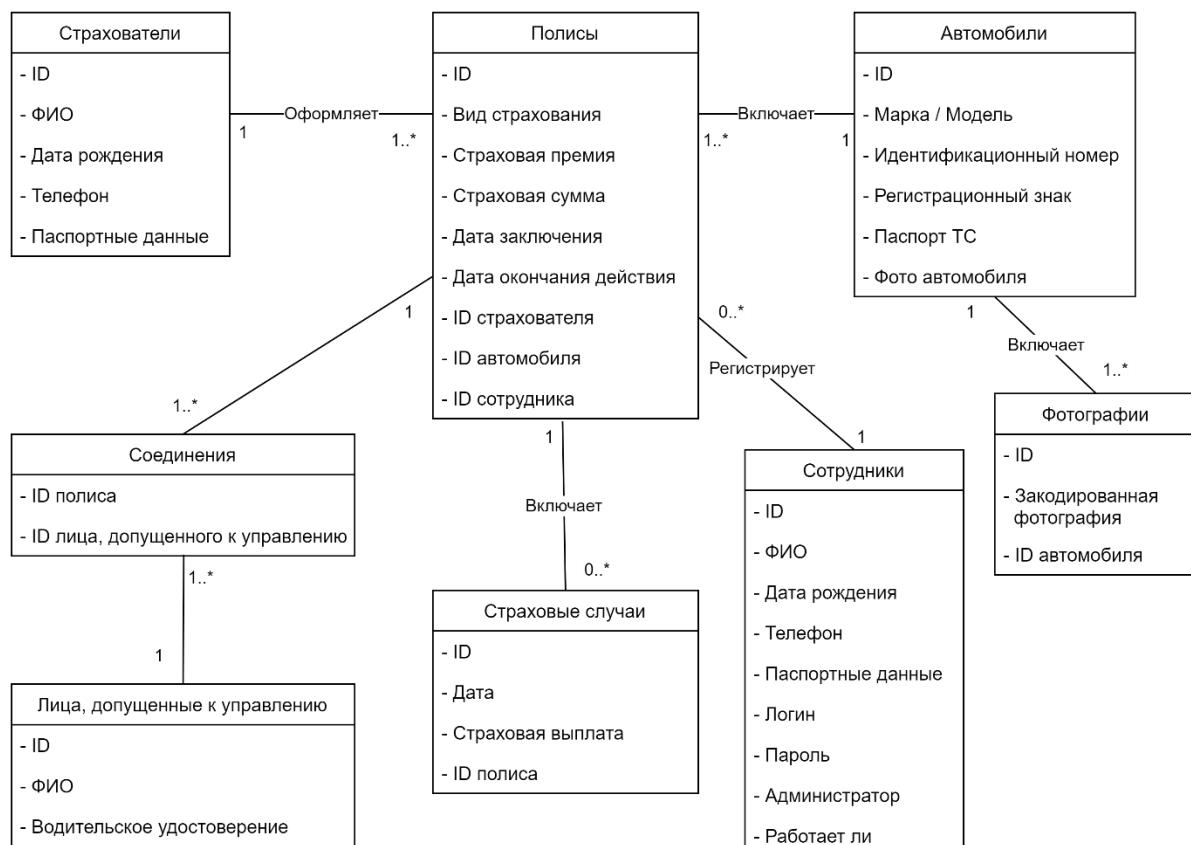


Рисунок 6 - Логическая модель данных

Изм.	Лист	№ докум.	Подпись	Дата

2.4 Соответствие сущностей логической и физической моделей

Построение физической модели БД производится на основе логической модели. В таблицах 1-8 описано соответствие сущностей логической модели и таблиц физической.

Таблица 1 - Таблица Страхователи(Policyholders)

Сущность	Таблица
ID	ID
ФИО	FullName
Дата рождения	Birthday
Телефон	Telephone
Паспортные данные	Passport

Таблица 2 - Таблица Автомобили(Cars)

Сущность	Таблица
ID	ID
Марка, модель	Model
Идентификационный номер	VIN
Регистрационный знак	RegistrationPlate
Паспорт ТС	VehiclePassport

Таблица 3 - Таблица Фотографии(Photos)

Сущность	Таблица
ID	ID
Закодированная фотография	EncodedPhoto
ID автомобиля	Car_ID

Таблица 4 - Таблица Полисы(Policies)

Сущность	Таблица
ID	ID
Вид страхования	InsuranceType
Страховая премия	InsurancePremium
Страховая сумма	InsuranceAmount
Дата заключения	DateOfConclusion
Дата окончания действия	ExpirationDate
ID страхователя	Policyholder_ID
ID автомобиля	Car_ID
ID сотрудника	Employee_ID

Таблица 5 - Таблица Соединения(Connections)

Сущность	Таблица
ID полиса	Policy_ID
ID лица, допущенного к управлению	PersonAllowedToDrive_ID

Таблица 6 - Таблица Лица, допущенные к управлению
(PersonsAllowedToDrive)

Сущность	Таблица
ID	ID
ФИО	FullName
Водительское удостоверение	DrivingLicence

Таблица 7 - Таблица Страховые случаи (InsuranceEvents)

Сущность	Таблица
ID	ID
Дата	Date
Страховая выплата	InsurancePayment
ID полиса	Policy_ID

Таблица 8 - Таблица Сотрудники(Employees)

Сущность	Таблица
ID	ID
ФИО	FullName
Дата рождения	Birthday
Телефон	Telephone
Паспортные данные	Passport
Логин	Login
Пароль	Password
Администратор	Admin
Работает ли	Works

Подробное описание каждой сущности приведено в таблицах 9-16. Также в данных таблица приведён расчет памяти, необходимой для хранения одной записи.

Таблица 9 - Таблица Policyholders

Имя столбца	Тип	Размер(байт)
ID	INT	4
FullName	VARCHAR(64)	64
Birthday	DATE	3
Telephone	VARCHAR(15)	15
Passport	VARCHAR(10)	10
Итого		96

Таблица 10 - Таблица Connections

Имя столбца	Тип	Размер(байт)
Policy_ID	INT	4
PersonAllowedToDrive_ID	INT	4
Итого		8

Таблица 11 - Таблица Cars

Имя столбца	Тип	Размер(байт)
ID	INT	4
Model	VARCHAR(50)	50
VIN	VARCHAR(17)	17
RegistrationPlate	VARCHAR(25)	25
VehiclePassport	VARCHAR(10)	10
Итого		106

Таблица 12 - Таблица Photos

Имя столбца	Тип	Размер(байт)
ID	INT	4
EncodedPhoto	VARCHAR(MAX)	1600000 (среднее значение занимаемого места закодированными фотографиями)
Car_ID	INT	4
Итого		1600008

Таблица 13 - Таблица Policies

Имя столбца	Тип	Размер(байт)
ID	INT	4
InsuranceType	VARCHAR(5)	5
InsurancePremium	INT	4
InsuranceAmount	INT	4
DateOfConclusion	DATE	3
ExpirationDate	DATE	3
Policyholder_ID	INT	4
Car_ID	INT	4
Employee_ID	INT	4
Итого		35

Таблица 14 - Таблица PersonsAllowedToDrive

Имя столбца	Тип	Размер(байт)
ID	INT	4
FullName	VARCHAR(64)	64
DrivingLicence	VARCHAR(10)	10
Итого		78

Таблица 15 - Таблица InsuranceEvents

Имя столбца	Тип	Размер(байт)
ID	INT	4
Date	DATE	3
InsurancePayment	INT	4
Policy_ID	INT	4
Итого		15

Таблица 16 - Таблица Employees

Имя столбца	Тип	Размер(байт)
ID	INT	4
FullName	VARCHAR(64)	64
Birthday	DATE	3
Telephone	VARCHAR(15)	15
Passport	VARCHAR(10)	10
Login	VARCHAR(32)	32
Password	VARCHAR(32)	32
Admin	BIT	1
Works	BIT	
Итого		129

Определим объем внешней памяти, необходимой для размещения данных за год использования. Для того чтобы оценить объем, занимаемый таблицами базы данных, необходимо оценить объем каждой таблицы. Примерный расчет необходимо объема памяти приведен в таблице 17.

Таким образом, при максимальном заполнении БД объем таблиц составит: $V_{\text{данных}} = 3125319,414 \text{ Кб} \approx 3000 \text{ Мб/год.}$

Таблица 17 - Расчет объема ПЗУ для хранения данных

Таблица	Размер записи, байт	Максимальное (оценочное) количество записей	Всего, Кбайт
Policyholders	96	700	65,625
Cars	106	700	72,461
Photos	1600008	2000	3125015,625
Connections	8	1700	13,281
Policies	35	1000	34,180
PersonsAllowedToDrive	78	1500	114,258
InsuranceEvents	15	100	1,465
Employees	129	20	2,520
Итого			3125319,414 (3000 Мбайт)

2.5 Физическая модель данных

Физическая модель данных, зависит от конкретной СУБД, фактически являясь отображением системного каталога. Физическая модель БД определяет способ размещения данных в среде хранения и способы доступа к этим данным, которые поддерживаются на физическом уровне.

Изм.	Лист	№ докум.	Подпись	Дата

Физическая модель, разработанная для БД Страхового агентства представлена на рисунке 7.

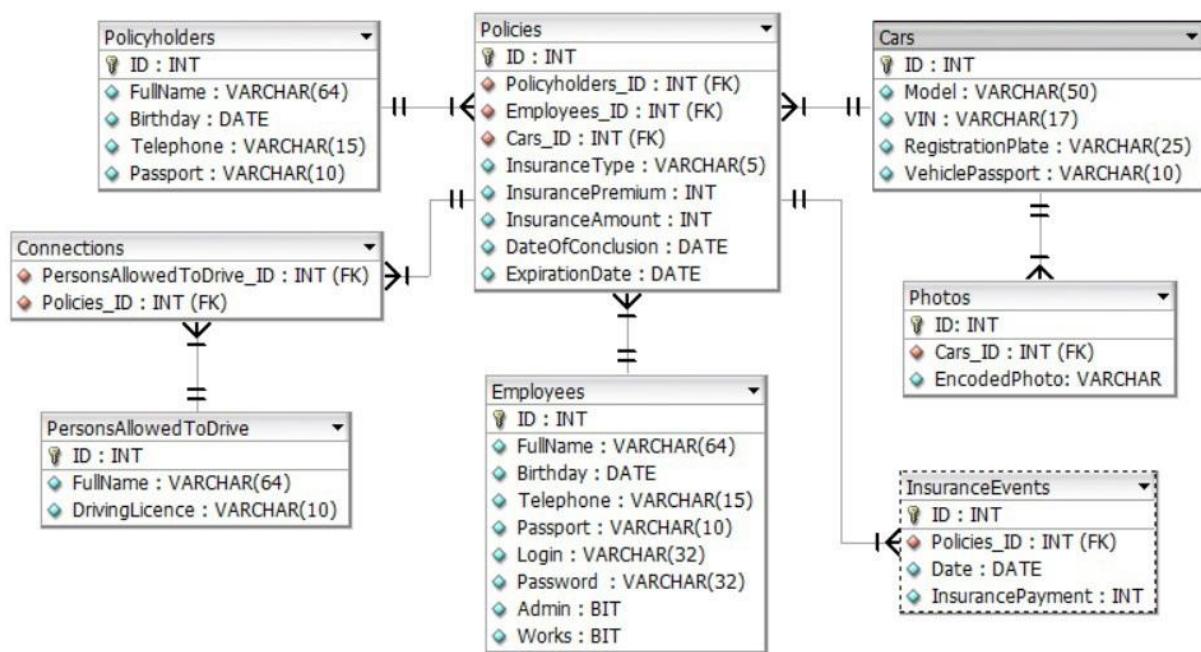


Рисунок 7 - Физическая модель данных

Изм.	Лист	№ докум.	Подпись	Дата

3 Разработка и реализация АИС

После разработки моделей данных была создана сама база данных и соответствующие таблицы. SQL запросы создания таблиц приведены в приложении 1.

Так как основу программы составляет взаимодействия с базой данных, то рассмотрим его на примере некоторых методов из класса Database. (Полный код данного класса будет приведён в приложении 3)

Снимки разработанного приложения приведены в приложение 2.

3.1 Авторизация

Данный метод отправляет запрос к БД, который осуществляет выборку полей Login, Admin, Works, где Логин и Пароль соответствуют тем, что ввёл пользователь. Если таких не существует, то выводится ошибка или если данный сотрудник уже не работает, а информация о нём хранится в БД, то также выводится ошибка. Иначе в классе сохраняются логин пользователя и информация о том, является ли он администратором.

Листинг метода:

```
public static void Authorization(string login, string password)
{
    bool flag = false;
    try
    {
        using (SqlConnection con = new SqlConnection(ConnectionString))
        {
            string query = "SELECT Login, Admin, Works FROM Employees WHERE Login = @login AND Password = @password";
            SqlCommand command = new SqlCommand(query, con);
            command.Parameters.Add(new SqlParameter("@login", login));
            command.Parameters.Add(new SqlParameter("@Password", GetHash(password)));
            con.Open();
            SqlDataReader reader = command.ExecuteReader();
            if (!reader.HasRows)
            {
                flag = true;
            }
        }
    }
}
```

Изм.	Лист	№ докум.	Подпись	Дата

```

        throw new Exception("Неправильно указан логин и/или пароль");
    }
    while (reader.Read())
    {
        if (Convert.ToBoolean(reader["Works"]))
        {
            _login = reader["Login"].ToString();
            _admin = Convert.ToBoolean(reader["Admin"]);
        }
        else
        {
            flag = true;
            throw new Exception("Данный сотрудник больше не работает");
        }
    }
    reader.Close();
    con.Close();
}
catch (Exception exp)
{
    if (flag)
    {
        throw exp;
    }
    else
    {
        throw new Exception("Ошибка в работе БД");
    }
}
}

```

3.2 Добавление

Рассмотрим метод добавления Страхователя. Примерно такой же синтаксис имеют методы добавления Сотрудника и Страхового случая.

В данном методе сначала отправляются два запроса, с помощью которых происходит проверка на наличие в БД, уже существующих Телефона и Паспорта. При наличии таковых выводится ошибка.

Если таковых нет, то происходит добавление Страхователя со значениями, которые ввёл/выбрал пользователь.

Листинг метода:

```

public static void AddPolicyholder(Policyholder policyholder)
{
    bool flag = false;
    try
    {

```

Изм.	Лист	№ докум.	Подпись	Дата

```

using (SqlConnection con = new SqlConnection(ConnectionString))
{
    string query1 = "SELECT ID FROM Policyholders WHERE Telephone = @telephone";
    SqlCommand command1 = new SqlCommand(query1, con);
    command1.Parameters.Add(new SqlParameter("@telephone", policyholder.Telephone));

    string query2 = "SELECT ID FROM Policyholders WHERE Passport = @passport";
    SqlCommand command2 = new SqlCommand(query2, con);
    command2.Parameters.Add(new SqlParameter("@passport", policyholder.Passport));

    string query = "INSERT INTO Policyholders (FullName, Birthday, Telephone, Passport) " +
                  "VALUES(@fullName, @birthday, @telephone, @passport)";

    SqlCommand command = new SqlCommand(query, con);
    command.Parameters.Add(new SqlParameter("@fullName", policyholder.FullName));
    command.Parameters.Add(new SqlParameter("@birthday", policyholder.Birthday));
    command.Parameters.Add(new SqlParameter("@telephone", policyholder.Telephone));
    command.Parameters.Add(new SqlParameter("@passport", policyholder.Passport));

    con.Open();
    SqlDataReader reader = command1.ExecuteReader();
    if (reader.HasRows)
    {
        flag = true;
        throw new Exception("Данный телефон уже используется");
    }
    reader.Close();

    reader = command2.ExecuteReader();
    if (reader.HasRows)
    {
        flag = true;
        throw new Exception("Данный паспорт уже используется");
    }
    reader.Close();

    command.ExecuteNonQuery();
    con.Close();
}
}
catch (Exception exp)
{
    if (flag)
    {
        throw exp;
    }
    else
    {
        throw new Exception("Ошибка в работе БД");
    }
}
}

```

3.3 Добавление с транзакцией

Рассмотрим метод добавления Автомобиля.

Изм.	Лист	№ докум.	Подпись	Дата

В целом метод похож на обычное Добавление, но так как при добавление автомобиля необходимо, чтобы добавилась как информация об автомобиле, так и фотографии (а друг без друга они не должны добавляться), то используется транзакция.

То есть сначала мы начинаем транзакцию, затем добавляем автомобиль и получаем последний добавленный ID (то есть ID автомобиля, который добавляем), и добавляем фотографии, которые в качестве внешнего ключа используют ID добавленного автомобиля, после чего завершаем транзакцию, тем самым фиксируя изменения.

Листинг метода:

```
public static void AddCarWithPhotos(Car car, List<string> photos)
{
    bool flag = false;
    try
    {
        using (SqlConnection con = new SqlConnection(ConnectionString))
        {
            string query1 = "SELECT ID FROM Cars WHERE VIN = @vin";
            SqlCommand command1 = new SqlCommand(query1, con);
            command1.Parameters.Add(new SqlParameter("@vin", car.VIN));

            string query = "BEGIN TRANSACTION " +
                "INSERT INTO Cars(Model, VIN, RegistrationPlate, VehiclePassport) " +
                "VALUES(@model, @vin, @registrationPlate, @vehiclePassport); " +
                "DECLARE @id INT; " +
                "SET @id=SCOPE_IDENTITY(); " +
                "INSERT INTO Photos(EncodedPhoto, CarID) VALUES ";
            for (var i = 0; i < photos.Count - 1; i++)
            {
                query += "(@photo" + i + ", @id), ";
            }
            query += "(@photo" + (photos.Count - 1) + ", @id); " +
                "COMMIT TRANSACTION";

            SqlCommand command = new SqlCommand(query, con);
            command.Parameters.Add(new SqlParameter("@model", car.Model));
            command.Parameters.Add(new SqlParameter("@vin", car.VIN));
            command.Parameters.Add(new SqlParameter("@registrationPlate", car.RegistrationPlate));
            command.Parameters.Add(new SqlParameter("@vehiclePassport", car.VehiclePassport));
            for (var i = 0; i < photos.Count; i++)
            {
                command.Parameters.Add(new SqlParameter("@photo" + i, photos[i]));
            }

            con.Open();
        }
    }
}
```

Изм.	Лист	№ докум.	Подпись	Дата

```
SqlDataReader reader = command1.ExecuteReader();
if (reader.HasRows)
{
    flag = true;
    throw new Exception("Данный VIN номер уже используется");
}
reader.Close();

command.ExecuteNonQuery();
con.Close();
}

}

catch (Exception exp)
{
    if (flag)
    {
        throw exp;
    }
    else
    {
        throw new Exception("Ошибка в работе БД");
    }
}
```

По данному принципу добавляется Полис, так как при его добавление необходимо, чтобы добавилась как информация о самом полисе, так и список лиц, допущенных к управлению (а друг без друга они не должны добавляться).

3.4 Изменение

Рассмотрим метод изменения Страхователя. Примерно такой же синтаксис имеют методы изменения Сотрудника.

В данном методе сначала отправляются два запроса, с помощью которых происходит проверка на наличие в БД, уже существующих Телефона и Паспорта. При наличии таковых выводится ошибка.

Если таких нет, то происходит изменение Страхователя со значениями, которые ввёл/выбрал пользователь. При этом невозможно изменить дату рождения Страхователя. Аналогично у Сотрудника невозможно изменить дату рождения.

<i>Изм.</i>	<i>Лист</i>	<i>№ докум.</i>	<i>Подпись</i>	<i>Дата</i>

МИВУ 09.03.04-8.000 ПЗ

Лист

32

Листинг метода:

```
public static void ChangePolicyholder(Policyholder policyholder)
{
    bool flag = false;
    try
    {
        using (SqlConnection con = new SqlConnection(ConnectionString))
        {
            string query1 = "SELECT ID FROM Policyholders WHERE Telephone = @telephone AND ID <> @id";
            SqlCommand command1 = new SqlCommand(query1, con);
            command1.Parameters.Add(new SqlParameter("@telephone", policyholder.Telephone));
            command1.Parameters.Add(new SqlParameter("@id", policyholder.ID));

            string query2 = "SELECT ID FROM Policyholders WHERE Passport = @passport AND ID <> @id";
            SqlCommand command2 = new SqlCommand(query2, con);
            command2.Parameters.Add(new SqlParameter("@passport", policyholder.Passport));
            command2.Parameters.Add(new SqlParameter("@id", policyholder.ID));

            string query = "UPDATE Policyholders " +
                "SET FullName = @fullName, " +
                "Telephone = @telephone, " +
                "Passport = @passport " +
                "WHERE ID = @id";
            SqlCommand command = new SqlCommand(query, con);

            command.Parameters.Add(new SqlParameter("@id", policyholder.ID));
            command.Parameters.Add(new SqlParameter("@fullName", policyholder.FullName));
            command.Parameters.Add(new SqlParameter("@telephone", policyholder.Telephone));
            command.Parameters.Add(new SqlParameter("@passport", policyholder.Passport));

            con.Open();
            SqlDataReader reader = command1.ExecuteReader();
            if (reader.HasRows)
            {
                flag = true;
                throw new Exception("Данный телефон уже используется");
            }
            reader.Close();

            reader = command2.ExecuteReader();
            if (reader.HasRows)
            {
                flag = true;
                throw new Exception("Данный паспорт уже используется");
            }
            reader.Close();

            command.ExecuteNonQuery();
            con.Close();
        }
    }
    catch (Exception exp)
    {
        if (flag)
        {
            throw exp;
        }
        else
        {
            throw new Exception("Ошибка в работе БД");
        }
    }
}
```

Изм.	Лист	№ докум.	Подпись	Дата

```
        }  
    }  
}
```

3.5 Изменение с транзакцией

Рассмотрим метод изменения Автомобиля.

При изменении автомобиля необходимо, чтобы удалялись фотографии, которые пользователь убрал из списка, а также добавлялись новые загруженные фотографии. Эти действия не должны выполняться друг без друга, поэтому используется транзакция.

То есть сначала мы начинаем транзакцию, затем изменяем автомобиль (можно изменять только регистрационный знак и паспорт ТС), добавляем новые фотографии, удаляем ненужные фотографии, после чего завершаем транзакцию, тем самым фиксируя изменения.

Листинг метода:

```
public static void ChangeCarWithPhotos(Car car, List<Photo> listDeletePhotos, List<string> listAddPhotos)  
{  
    try  
    {  
        using (SqlConnection con = new SqlConnection(ConnectionString))  
        {  
            string query = "BEGIN TRANSACTION " +  
                "UPDATE Cars " +  
                "SET RegistrationPlate = @registrationPlate, " +  
                "VehiclePassport = @vehiclePassport " +  
                "WHERE ID = @id ";  
            if (listAddPhotos.Count != 0)  
            {  
                query += "INSERT INTO Photos(EncodedPhoto, CarID) VALUES ";  
                for (int i = 0; i < listAddPhotos.Count - 1; i++)  
                {  
                    query += "(@photo" + i + ", @id), ";  
                }  
                query += "(@photo" + (listAddPhotos.Count - 1) + ", @id); ";  
            }  
            if (listDeletePhotos.Count != 0)  
            {  
                for (int i = 0; i < listDeletePhotos.Count; i++)  
                {  
                    query += "DELETE FROM Photos WHERE ID = @photoID" + i + ";";  
                }  
            }  
            query += "COMMIT TRANSACTION";  
  
            SqlCommand command = new SqlCommand(query, con);
```

Изм.	Лист	№ докум.	Подпись	Дата

```

        command.Parameters.Add(new SqlParameter("@id", car.ID));
        command.Parameters.Add(new SqlParameter("@registrationPlate", car.RegistrationPlate));
        command.Parameters.Add(new SqlParameter("@vehiclePassport", car.VehiclePassport));
        for (int i = 0; i < listAddPhotos.Count; i++)
        {
            command.Parameters.Add(new SqlParameter("@photo" + i, listAddPhotos[i]));
        }
        for (int i = 0; i < listDeletePhotos.Count; i++)
        {
            command.Parameters.Add(new SqlParameter("@photoID" + i, listDeletePhotos[i].ID));
        }

        con.Open();
        command.ExecuteNonQuery();
        con.Close();
    }
}
catch
{
    throw new Exception("Ошибка в работе БД");
}
}

```

По данному принципу изменяется Полис, так как при его изменении необходимо, чтобы удалялись лица, допущенные к управлению, которые пользователь убрал из списка, а также добавлялись новые лица, допущенные к управлению. Эти действия не должны выполняться друг без друга, поэтому используется транзакция.

3.6 Удаление

Рассмотрим метод удаления Страхователя. Примерно такой же синтаксис имеют методы удаления Сотрудника, Автомобиля и Лица, допущенного к управлению.

В данном методе сначала отправляются запроса, с помощью которого происходит проверка оформлял ли Страхователь полис. Если да, то его невозможно удалить и выдаётся ошибка.

Аналогично с Сотрудником. У автомобиля проверяется оформление на него полиса, а у Лица, допущенного к управлению - наличие привязки к какому-либо полису.

Если таких связей нет, то можно произвести удаление. Удаление происходит по выбранному ID.

Изм.	Лист	№ докум.	Подпись	Дата

Листинг метода:

```
public static void DeletePolicyholder(int id)
{
    bool flag = false;
    try
    {
        using (SqlConnection con = new SqlConnection(ConnectionString))
        {
            string query1 = "SELECT ID FROM Policies WHERE PolicyholderID = @policyholderID";
            SqlCommand command1 = new SqlCommand(query1, con);
            command1.Parameters.Add(new SqlParameter("@policyholderID", id));

            string query = "DELETE FROM Policyholders WHERE ID = @id";
            SqlCommand command = new SqlCommand(query, con);

            command.Parameters.Add(new SqlParameter("@id", id));

            con.Open();
            SqlDataReader reader = command1.ExecuteReader();
            if (reader.HasRows)
            {
                flag = true;
                throw new Exception("Вы не можете удалить данного страхователя, так как он оформил полис");
            }
            reader.Close();

            command.ExecuteNonQuery();
            con.Close();
        }
    }
    catch (Exception exp)
    {
        if (flag)
        {
            throw exp;
        }
        else
        {
            throw new Exception("Ошибка в работе БД");
        }
    }
}
```

3.7 Поиск

Рассмотрим метод поиска Страхователя. Данный метод отправляет запрос к БД, который осуществляет выборку всех полей, той строки в которой совпадает ID с выбранным для поиска. Если такого Страхователя не существует, то выводится ошибка. Иначе возвращается информация о найденном Страхователе.

Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

Листинг метода:

```
public static Policyholder SearchPolicyholderID(int id)
{
    bool flag = false;
    try
    {
        using (SqlConnection con = new SqlConnection(ConnectionString))
        {
            string query = "SELECT * FROM Policyholders WHERE ID = @id";
            SqlCommand command = new SqlCommand(query, con);
            command.Parameters.Add(new SqlParameter("@id", id));

            con.Open();
            SqlDataReader reader = command.ExecuteReader();

            Policyholder policyholder;
            if (!reader.HasRows)
            {
                flag = true;
                throw new Exception("Данный страхователь не существует");
            }
            while (reader.Read())
            {
                policyholder = new Policyholder(Convert.ToInt32(reader["ID"].ToString()),
                                                reader["FullName"].ToString(),
                                                Convert.ToDateTime(reader["Birthday"].ToString()),
                                                reader["Telephone"].ToString(),
                                                reader["Passport"].ToString());
                reader.Close();
                con.Close();
                return policyholder;
            }
            return null;
        }
    }
    catch (Exception exp)
    {
        if (flag)
        {
            throw exp;
        }
        else
        {
            throw new Exception("Ошибка в работе БД");
        }
    }
}
```

3.8 Отчёты

Данный метод отправляет запрос к БД, который подсчитывает сумму страховых выплат, количество и сумму заключенных договоров за

Изм.	Лист	№ докум.	Подпись	Дата

определённый период времени и по определённому виду страхования или по обоим сразу.

После выполнения три значения проверяются на пустоту и записываются в кортеж, который возвращается из функции.

Листинг метода:

```
public static (int, int, int) Reports(string insuranceType, DateTime dateStart, DateTime dateEnd)
{
    try
    {
        using (SqlConnection con = new SqlConnection(ConnectionString))
        {
            string query;
            SqlCommand command;
            if (insuranceType == "ОСАГО и КАСКО")
            {
                query = "SELECT (SELECT SUM(InsurancePayment) " +
                    "FROM InsuranceEvents " +
                    "WHERE Date >= @dateStart AND Date <= @dateEnd), " +
                    "COUNT(ID), SUM(InsurancePremium) " +
                    "FROM Policies " +
                    "WHERE DateOfConclusion >= @dateStart AND DateOfConclusion <= @dateEnd";

                command = new SqlCommand(query, con);
                command.Parameters.Add(new SqlParameter("@dateStart", dateStart));
                command.Parameters.Add(new SqlParameter("@dateEnd", dateEnd));
            }
            else
            {
                query = "SELECT (SELECT SUM(InsurancePayment) " +
                    "FROM Policies as p LEFT JOIN InsuranceEvents as ie ON p.ID = ie.PolicyID " +
                    "WHERE p.InsuranceType = @insuranceType AND Date >= @dateStart AND Date <=
                    @dateEnd), " +
                    "COUNT(ID), SUM(InsurancePremium) " +
                    "FROM Policies " +
                    "WHERE InsuranceType = @insuranceType AND DateOfConclusion >= @dateStart AND
                    DateOfConclusion <= @dateEnd";

                command = new SqlCommand(query, con);
                command.Parameters.Add(new SqlParameter("@insuranceType", insuranceType));
                command.Parameters.Add(new SqlParameter("@dateStart", dateStart));
                command.Parameters.Add(new SqlParameter("@dateEnd", dateEnd));
            }
        }

        con.Open();
        SqlDataReader reader = command.ExecuteReader();

        int countContracts = 0, sumContracts = 0, sumInsuranceEvents = 0;
        while (reader.Read())
        {
            string tempSumInsuranceEvents = reader.GetValue(0).ToString();
            if (tempSumInsuranceEvents == String.Empty) tempSumInsuranceEvents = "0";
```

Изм.	Лист	№ докум.	Подпись	Дата

```

        string tempCountContracts = reader.GetValue(1).ToString();
        if (tempCountContracts == String.Empty) tempCountContracts = "0";
        string tempSumContracts = reader.GetValue(2).ToString();
        if (tempSumContracts == String.Empty) tempSumContracts = "0";

        countContracts = Convert.ToInt32(tempCountContracts);
        sumContracts = Convert.ToInt32(tempSumContracts);
        sumInsuranceEvents = Convert.ToInt32(tempSumInsuranceEvents);
    }
    reader.Close();
    var tuple = (countContracts, sumContracts, sumInsuranceEvents);
    con.Close();
    return tuple;
}
}
catch
{
    throw new Exception("Ошибка в работе БД");
}
}

```

Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

4 Руководство администратора

Для установки данной программы на компьютер необходимо:

- С помощью СУБД MS SQL Server Management Studio версии 2018 или позднее создать базу данных или импортировать её с помощью функции «Импорт приложения уровня данных» из файла с расширением .басрас, находящимся в распространяемой папке с приложением.
- Создать в базе данных с помощью СУБД пользователя с правами администратора.
- Установить .NET Framework версии 4.7.2 на компьютер.
- Запустить файл с расширением .exe в папке с приложением.

Для функционирования пользовательского приложения и установки SQL Server ПК пользователей должен удовлетворять следующим системным требованиям:

- Microsoft.NET Framework версии 4.7.2;
- Microsoft SQL Server 2018 или новее;
- разрешение экрана не менее 1366 x 768;
- операционная система Windows 10 TH1 1507 или более поздней версии;
- место на жестком диске: 10 Гб;
- видеоадаптер: любой;
- процессор x64 с тактовой частотой 1,4 ГГц (рекомендуется 2,0 ГГц и выше)
- контроллер: клавиатура, мышь.

Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

5 Тестирование

Одним из важнейших этапов создания приложения является его тестирование и отладка. Тестирование позволяет выявить скрытые и явные недостатки программы, либо убедиться в ее пригодности для применения. Обнаруженные недостатки устраняются в ходе отладки.

Целью тестирования является проверка работоспособности программы, правильности выполнения всех функций, а также правильности обработки всех исключений, возникающих в ходе работы программы.

5.1 Авторизация

В данном окне обрабатываются следующие ошибки:

- незаполненное поле логин;
- незаполненное поле пароль.

В классе Database обрабатываются следующие ошибки для данного окна:

- неправильно указан логин и/или пароль;
- введён логин и пароль сотрудника, который больше не работает;
- произошла ошибка в работе БД.

При отсутствии ошибок пользователь переходит в саму программу.

Изм.	Лист	№ докум.	Подпись	Дата

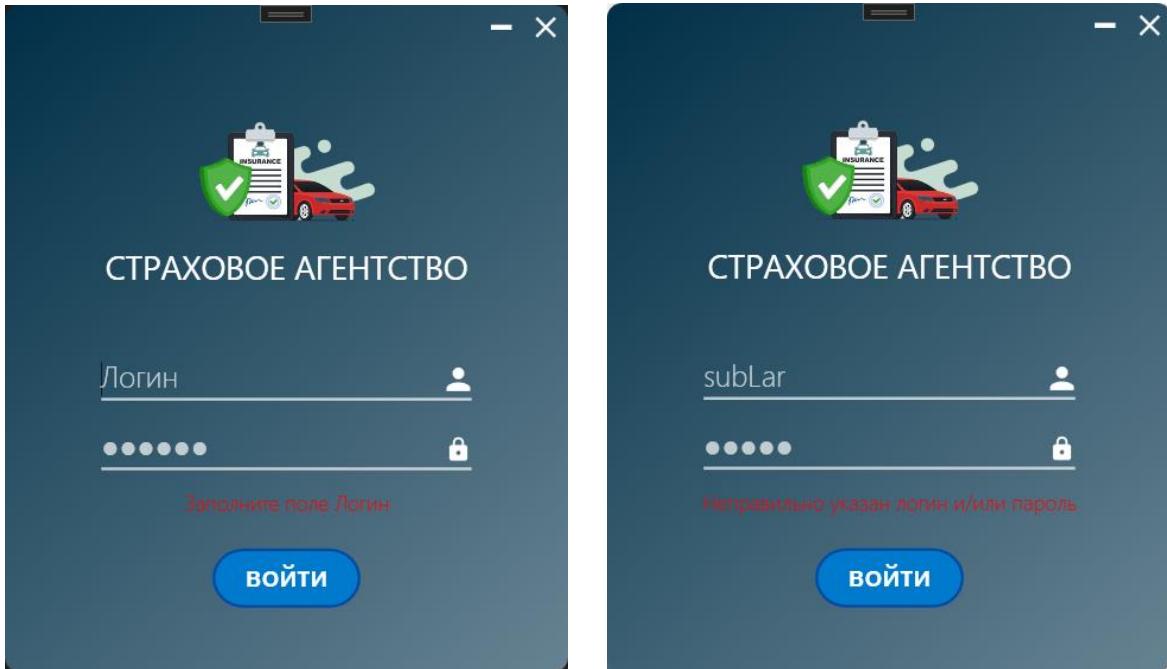


Рисунок 8 - скриншот вывода ошибок

5.2 Добавление сотрудника

На данной странице обрабатываются следующие ошибки:

- незаполненное поле ФИО;
- незаполненное поле Номер телефона;
- введён номер телефона, который больше 15 символов;
- введена серия паспорта, которая не содержит 4 цифры;
- введён номер паспорта, который не содержит 6 цифр;
- введены серия и номер паспорта, которые содержат не только цифры;
- введён логин, длина которого меньше 4 или больше 32 символов;
- введён логин, который содержит пробелы;
- введён пароль, длина которого меньше 4 или больше 32 символов;
- введён пароль, который содержит пробелы
- незаполненное поле Администратор.

В классе Database обрабатываются следующие ошибки для данной страницы:

Изм.	Лист	№ докум.	Подпись	Дата

- введен телефон, который уже используется;
- введен паспорт, который уже используется;
- введен логин, который уже занят;
- произошла ошибка в работе БД.

При отсутствии ошибок страхователь успешно добавляется и выводится сообщение "Сотрудник успешно добавлен".

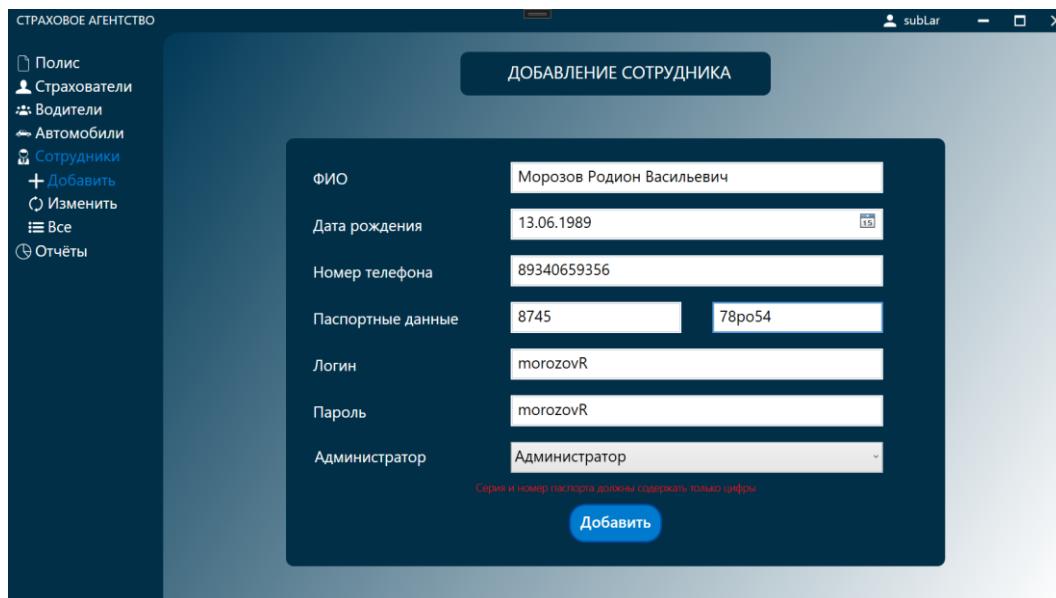


Рисунок 9 - скриншот вывода ошибок

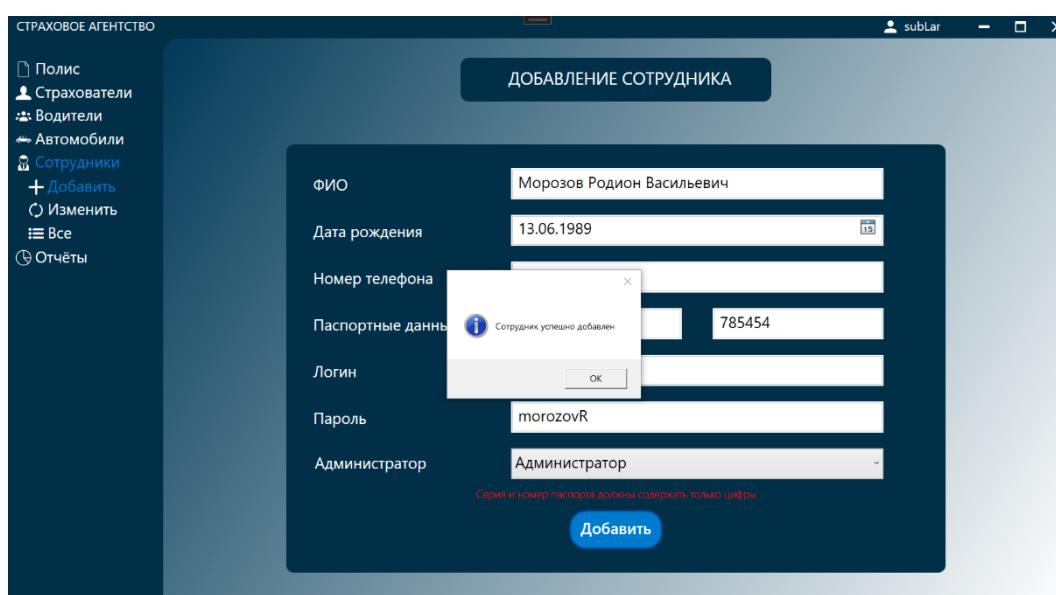


Рисунок 10 - скриншот успешного добавления

Изм.	Лист	№ докум.	Подпись	Дата

5.3 Изменение/удаление сотрудника

На данной странице обрабатываются следующие ошибки:

- незаполненная строка поиска;
- не выбран сотрудник (если пользователь нажал кнопку изменить, но до этого в строке поиска не нашёл нужную информацию);
- незаполненное поле ФИО;
- незаполненное поле Номер телефона;
- введён номер телефона, который содержит больше 15 символов;
- введена серия паспорта, которая не содержит 4 цифры;
- введён номер паспорта, который не содержит 6 цифр;
- введены серия и номер паспорта, которые содержат не только цифры;
- введён логин, длина которого меньше 4 или больше 32 символов;
- введён логин, который содержит пробелы;
- введён пароль, длина которого меньше 4 или больше 32 символов;
- введён пароль, который содержит пробелы
- незаполненное поле Администратор;
- незаполненное поле Работает ли.

В классе Database обрабатываются следующие ошибки для данной страницы:

- введён телефон или паспорт сотрудника, который не существует (для строки поиска);
- введён телефон, который уже используется;
- введён паспорт, который уже используется;
- введён логин, который уже занят;
- попытка удаления сотрудника, который оформил полис;
- произошла ошибка в работе БД.

Также в форме нельзя изменить дату рождения.

Изм.	Лист	№ докум.	Подпись	Дата

При отсутствии ошибок сотрудник успешно изменяется и выводится сообщение "Сотрудник успешно изменён" или удаляется с сообщением "Сотрудник успешно удалён".

Рисунок 11 - скриншот вывода ошибок

Рисунок 12 - скриншот успешного удаления

5.4 Все сотрудники

На данной странице обрабатываются следующие ошибки:

Изм.	Лист	№ докум.	Подпись	Дата

- незаполненная строка поиска.

В классе Database обрабатываются следующие ошибки для данной страницы:

- введён телефон или паспорт сотрудника, который не существует (для строки поиска);
- произошла ошибка в работе БД.

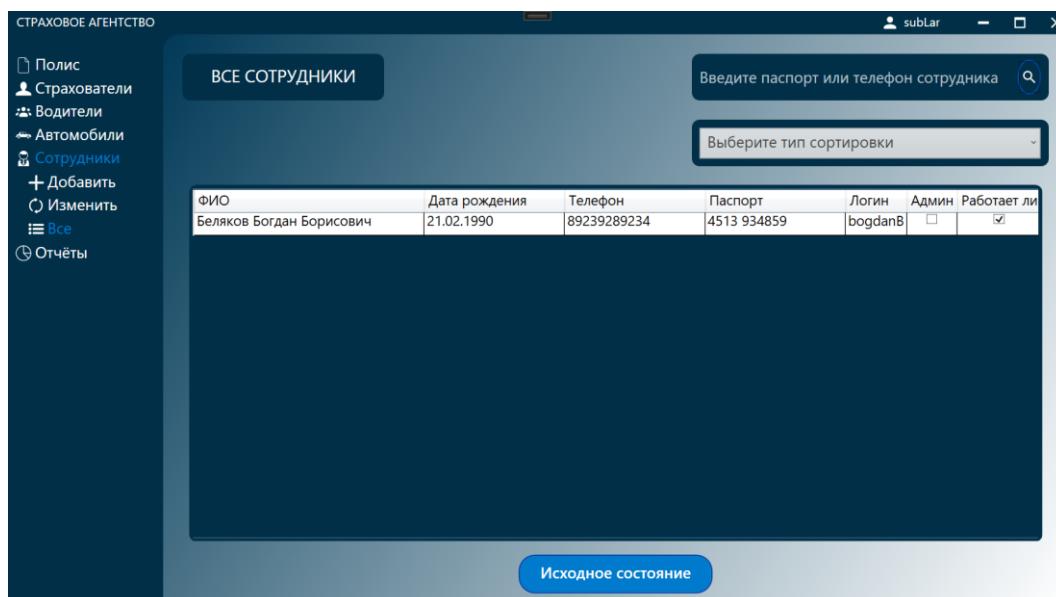


Рисунок 13 - скриншот поиска сотрудника

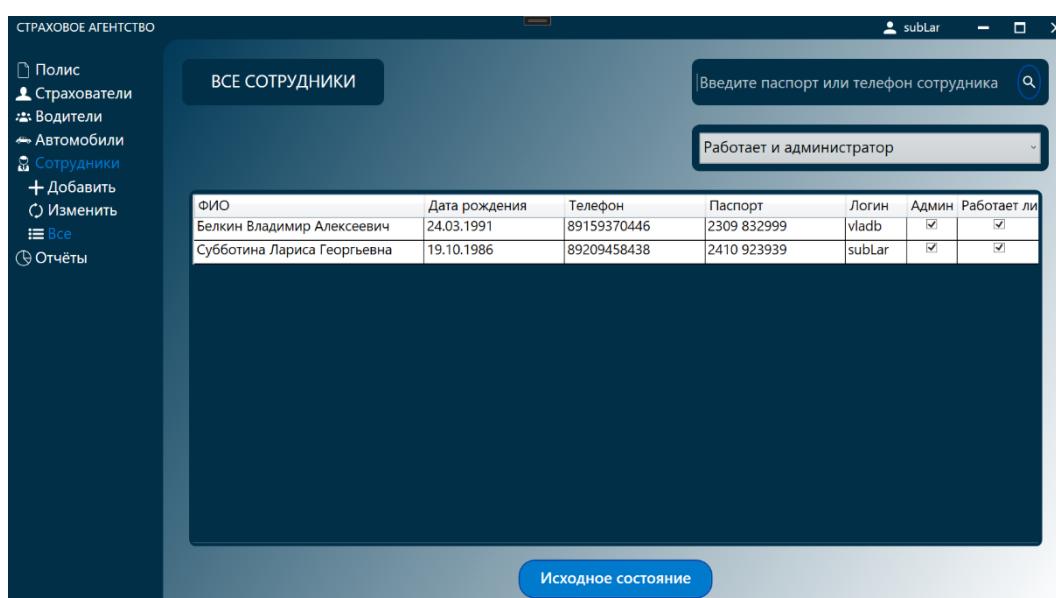


Рисунок 14 - скриншот сортировки

Изм.	Лист	№ докум.	Подпись	Дата

МИВУ 09.03.04-8.000 ПЗ

Лист

46

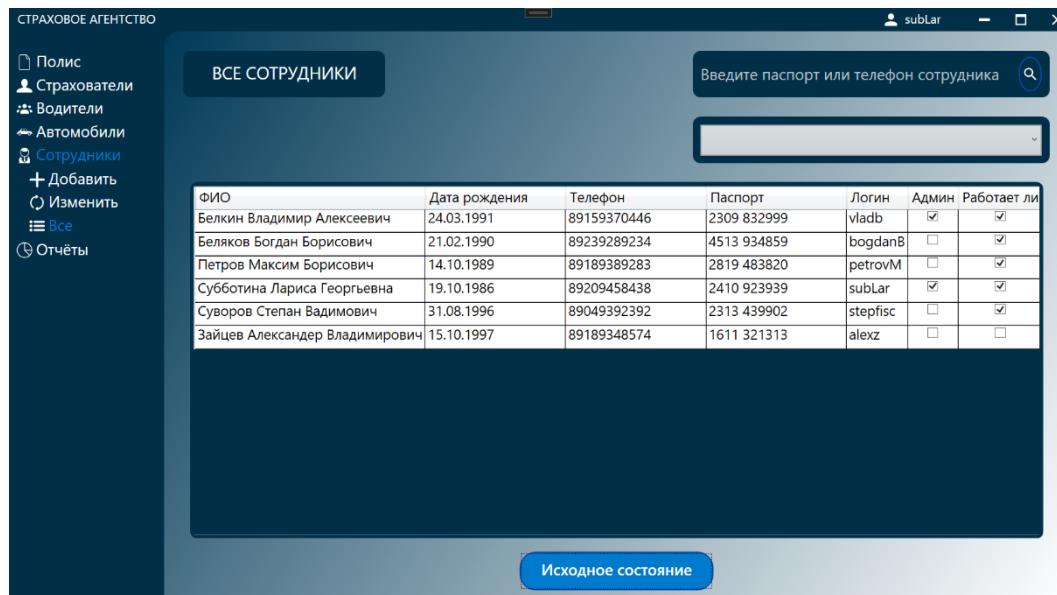


Рисунок 15 - скриншот исходного состояния

5.5 Добавление страхователя

На данной странице обрабатываются следующие ошибки:

- незаполненное поле ФИО;
- незаполненное поле Номер телефона;
- введён номер телефона, который содержит больше 15 символов;
- введена серия паспорта, которая не содержит 4 цифры;
- введён номер паспорта, который не содержит 6 цифр;
- введены серия и номер паспорта, которые содержат не только цифры.

В классе Database обрабатываются следующие ошибки для данной страницы:

- введён телефон, который уже используется;
- введён паспорт, который уже используется;
- произошла ошибка в работе БД.

При отсутствии ошибок страхователь успешно добавляется и выводится сообщение "Страхователь успешно добавлен".

Изм.	Лист	№ докум.	Подпись	Дата

Рисунок 16 - скриншот вывода ошибок

Рисунок 17 - скриншот успешного добавления

5.6 Изменение/удаление страхователя

На данной странице обрабатываются следующие ошибки:

- незаполненная строка поиска;
- не выбран страхователь (если пользователь нажал кнопку изменить, но до этого в строке поиска не нашёл нужную информацию);
- незаполненное поле ФИО;

Изм.	Лист	№ докум.	Подпись	Дата

- незаполненное поле Номер телефона;
- введён номер телефона, который содержит больше 15 символов;
- введена серия паспорта, которая не содержит 4 цифры;
- введён номер паспорта, который не содержит 6 цифр;
- введены серия и номер паспорта, которые содержат не только цифры.

В классе Database обрабатываются следующие ошибки для данной страницы:

- введён телефон или паспорт страхователя, который не существует (для строки поиска);
- введён телефон, который уже используется;
- введён паспорт, который уже используется;
- попытка удаления страхователя, который оформил полис;
- произошла ошибка в работе БД.

Также в форме нельзя изменить дату рождения.

При отсутствии ошибок страхователь успешно изменяется и выводится сообщение "Страхователь успешно изменён" или удаляется с сообщением "Страхователь успешно удалён".

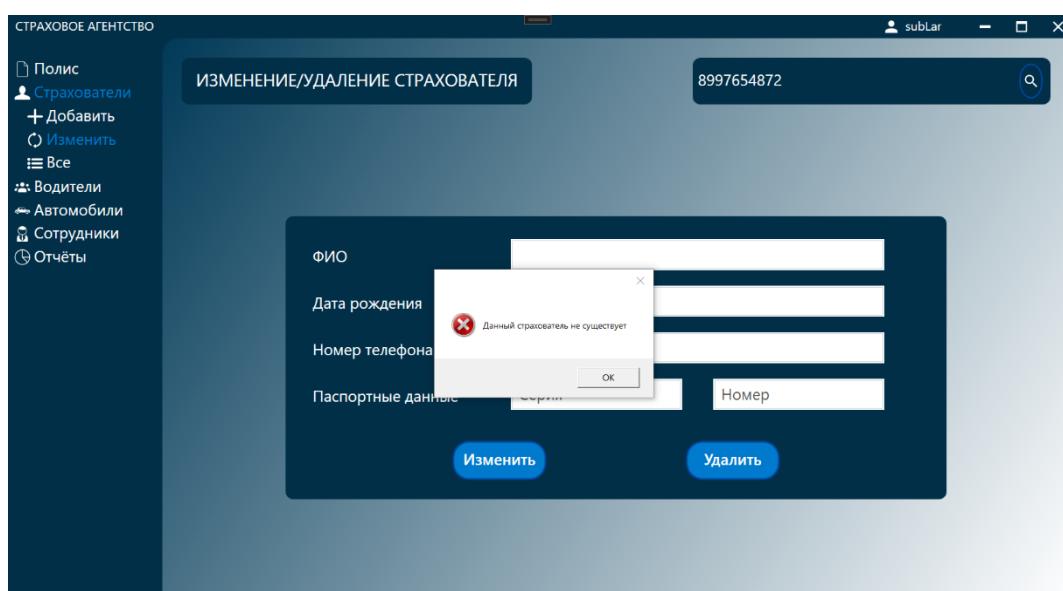


Рисунок 18 - скриншот вывода ошибок

Изм.	Лист	№ докум.	Подпись	Дата

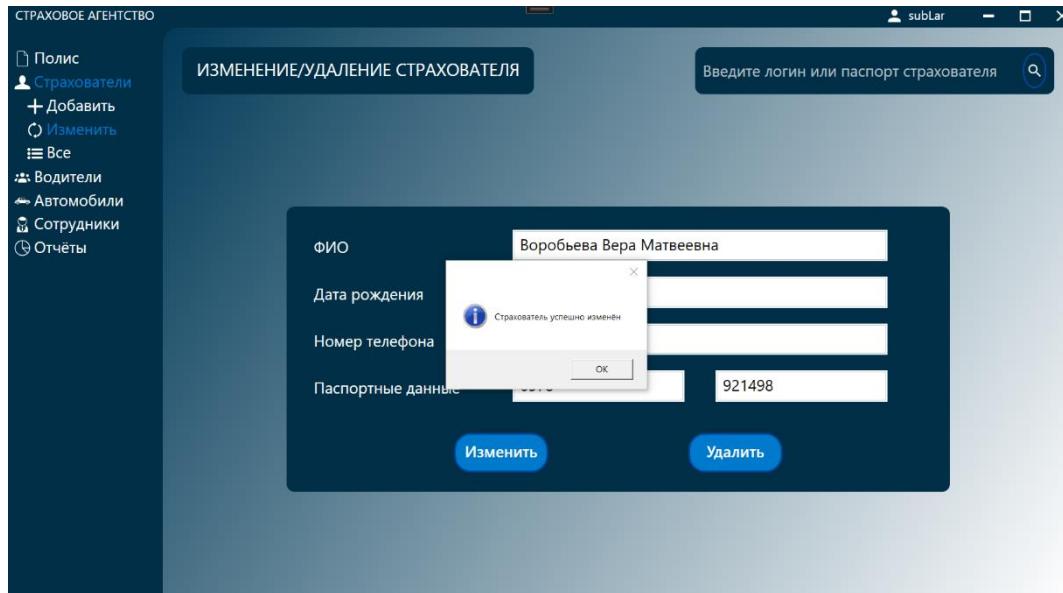


Рисунок 19 - скриншот успешного изменения

5.7 Все страхователи

На данной странице обрабатываются следующие ошибки:

- незаполненная строка поиска.

В классе Database обрабатываются следующие ошибки для данной страницы:

- введён телефон или паспорт страхователя, который не существует (для строки поиска);
- произошла ошибка в работе БД.

Изм.	Лист	№ докум.	Подпись	Дата

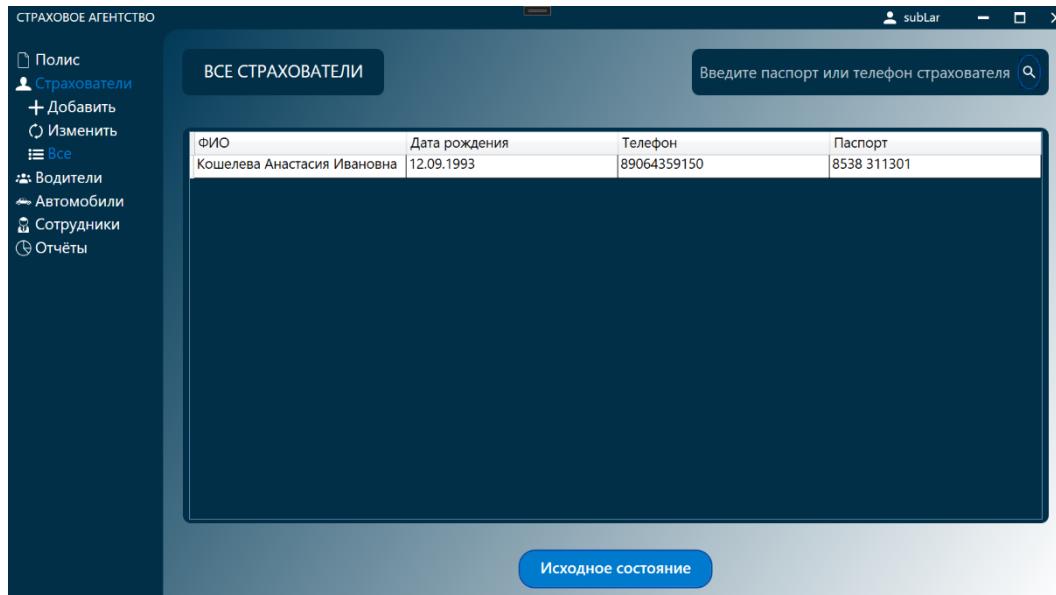


Рисунок 20 - скриншот поиска страхователя

Все Страхователи			
ФИО	Дата рождения	Телефон	Паспорт
Бессонов Иван Савельевич	16.01.1971	89180741287	7330 729340
Васильева Мария Ильинична	08.11.1990	89207652312	2934 701553
Виноградова Полина Павловна	08.11.1990	89207652357	4121 783474
Воробьева Вера Матвеевна	11.11.1960	89976548723	1950 921498
Гаврилов Борис Дмитриевич	15.03.1999	89017654595	7818 864232
Глухов Максим Александрович	09.10.1977	89947652723	6799 921028
Давыдов Михаил Платонович	08.11.1990	89257662355	1621 798477
Киселев Владислав Алексеевич	03.09.1985	89969990555	4371 877670
Колесова Эмилия Тимофеевна	08.11.1991	89217652333	1134 711573
Корнилов Ярослав Максимович	12.01.1976	89200755299	1220 729563
Кошелева Анастасия Ивановна	12.09.1993	89064359150	8538 311301
Маркова Таисия Серафимовна	12.10.1989	89917205634	7317 650936
Морозова Софья Яновна	15.08.1979	89347650895	2218 864204
Наумов Артём Алексеевич	12.09.1984	89679205634	5417 657236
Попов Иван Александрович	20.09.2000	89297623404	8920 902682
Сидоров Максим Максимович	03.09.1983	89969994445	4309 844674

Рисунок 21 - скриншот исходного состояния

5.8 Добавление лица, допущенного к управлению

На данной странице обрабатываются следующие ошибки:

- незаполненное поле ФИО;
- введена серия водительского удостоверения, которая не содержит 4 цифры;

Изм.	Лист	№ докум.	Подпись	Дата

МИВУ 09.03.04-8.000 ПЗ

Лист

51

- введён номер водительского удостоверения, который не содержит 6 цифр;
- введены серия и номер водительского удостоверения, которые содержат не только цифры.

В классе Database обрабатываются следующие ошибки для данной страницы:

- введено водительское удостоверение, которое уже используется;
- произошла ошибка в работе БД.

При отсутствии ошибок страхователь успешно добавляется и выводится сообщение "Страхователь успешно добавлен".

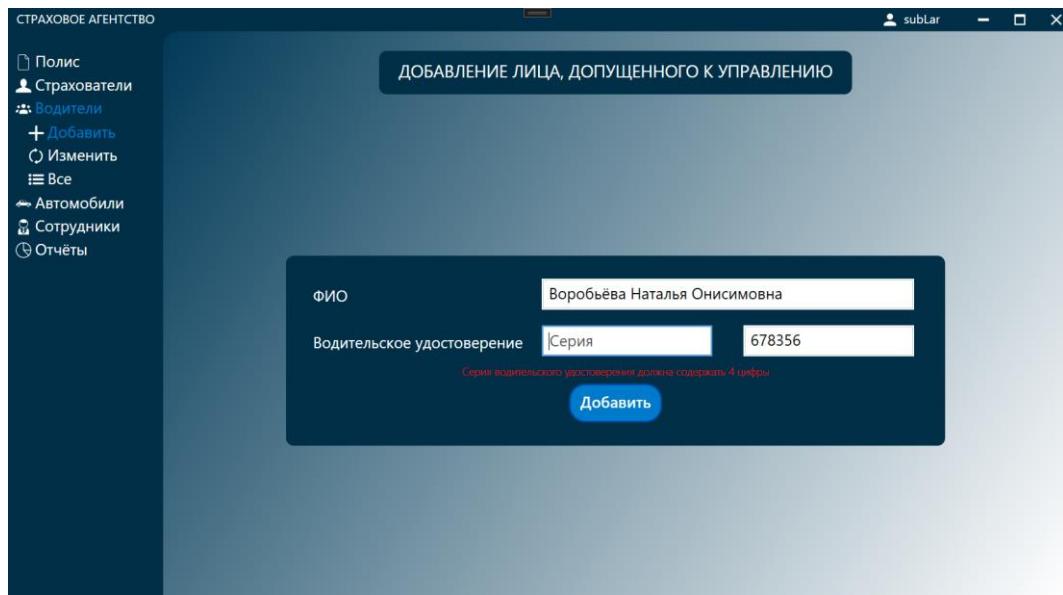


Рисунок 22 - скриншот вывода ошибок

Изм.	Лист	№ докум.	Подпись	Дата

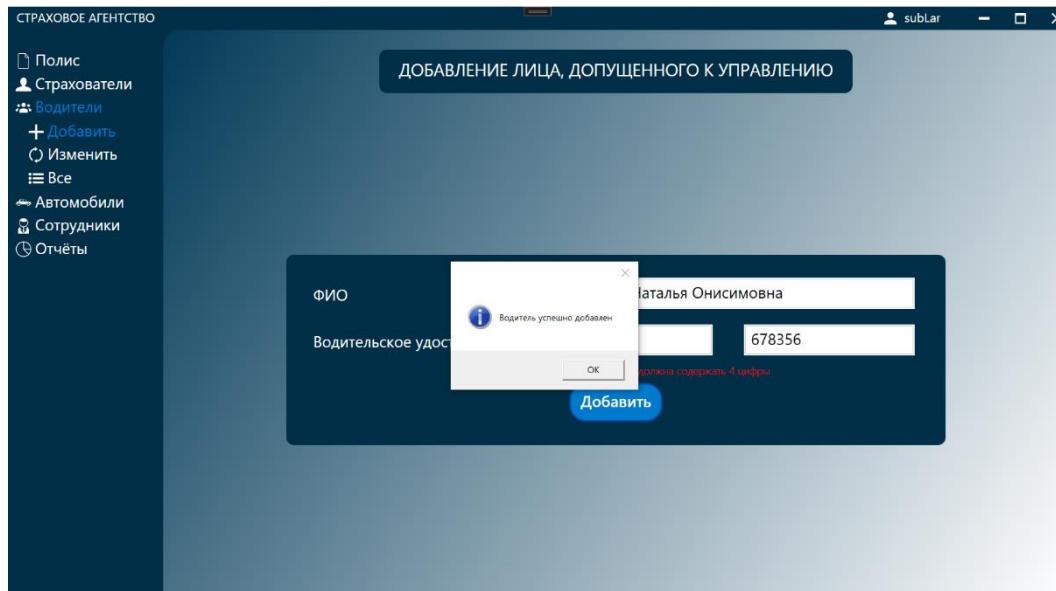


Рисунок 23 - скриншот успешного добавления

5.9 Изменение/удаление лица, допущенного к управлению

На данной странице обрабатываются следующие ошибки:

- незаполненная строка поиска;
- не выбран водитель (если пользователь нажал кнопку изменить, но до этого в строке поиска не нашёл нужную информацию);
- незаполненное поле ФИО;
- введена серия водительского удостоверения, которая не содержит 4 цифры;
- введён номер водительского удостоверения, который не содержит 6 цифр;
- введены серия и номер водительского удостоверения, которые содержат не только цифры.

В классе Database обрабатываются следующие ошибки для данной страницы:

- данный водитель не существует (для строки поиска);
- введено водительское удостоверение, который уже используется;

Изм.	Лист	№ докум.	Подпись	Дата

- попытка удаления лица, допущенного к управлению, на которое оформлен полис;
- произошла ошибка в работе БД.

При отсутствии ошибок лицо, допущенное к управлению успешно изменяется и выводится сообщение "Водитель успешно изменён" или удаляется с сообщением "Водитель успешно удалён".

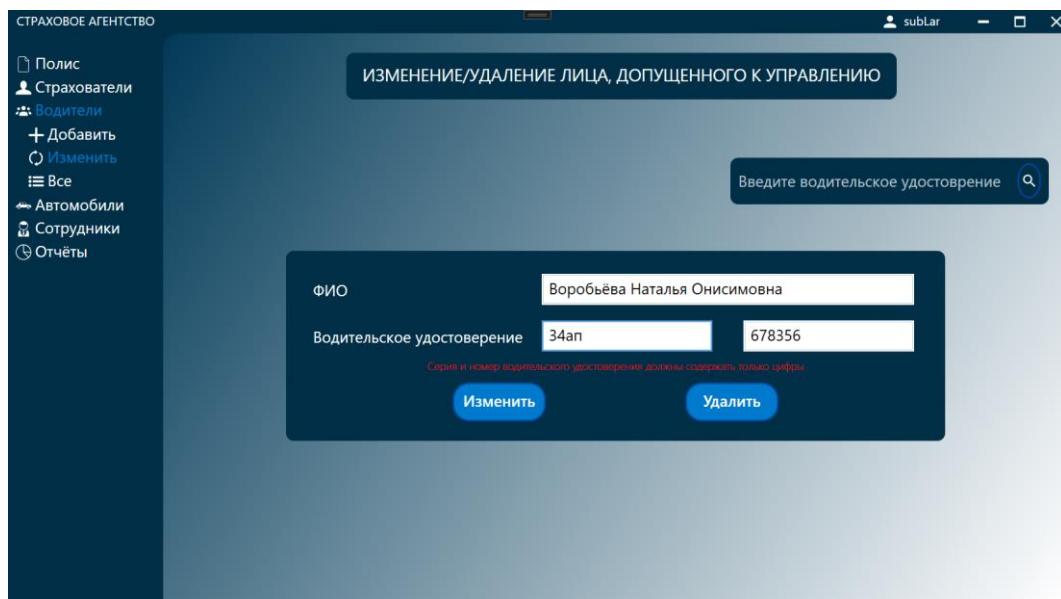


Рисунок 24 - скриншот вывода ошибок

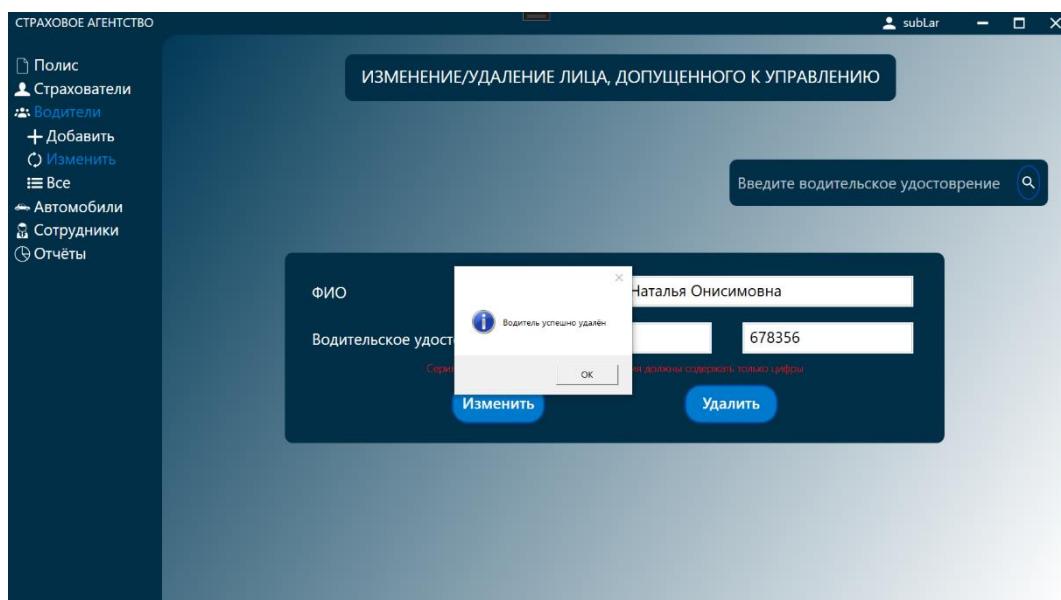


Рисунок 25 - скриншот успешного удаления

Изм.	Лист	№ докум.	Подпись	Дата

5.10 Все лица, допущенные к управлению

На данной странице обрабатываются следующие ошибки:

- незаполненная строка поиска.

В классе Database обрабатываются следующие ошибки для данной страницы:

- введено водительское удостоверение, которого не существует (для строки поиска);
- произошла ошибка в работе БД.

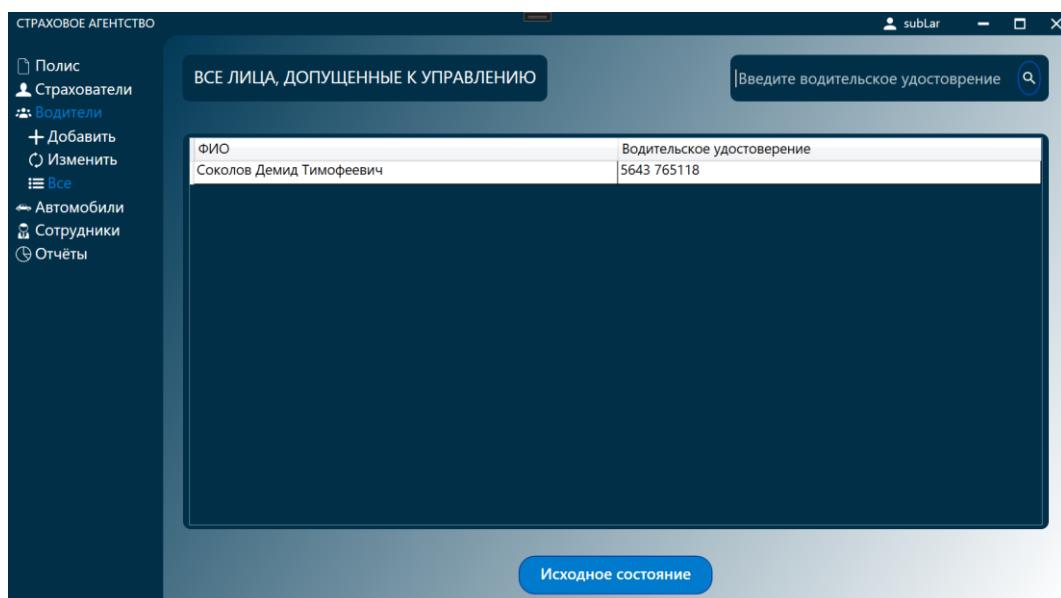


Рисунок 26 - скриншот поиска лица, допущенного к управлению

Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------



Рисунок 27 - скриншот исходного состояния

5.11 Добавление автомобиля

На данной странице обрабатываются следующие ошибки:

- незаполненное поле Модель;
- введённый VIN номер не содержит 17 знаков;
- введённый VIN номер состоит не только из цифр и заглавных латинских букв;
- незаполненное поле Регистрационный знак;
- введённая серия паспорта не содержит 4 знака;
- введённые первые два символа серии паспорта ТС не цифры;
- введённые последние два символа серии паспорта ТС не буквы;
- введённый номер паспорта ТС не содержит 6 цифр;
- введённый номер паспорта ТС содержит не только цифры;
- не была добавлена ни одна фотография автомобиля.

В классе Database обрабатываются следующие ошибки для данной страницы:

- введён VIN номер, который уже используется;
- произошла ошибка в работе БД.

Изм.	Лист	№ докум.	Подпись	Дата

При отсутствии ошибок автомобиль успешно добавляется и выводится сообщение "Автомобиль успешно добавлен".

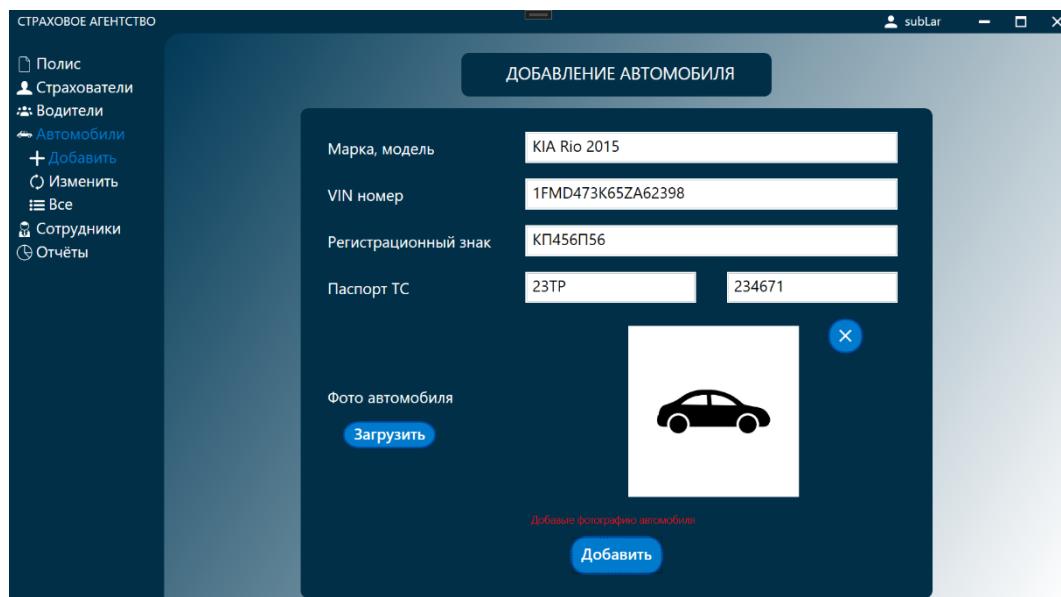


Рисунок 28 - скриншот вывода ошибок

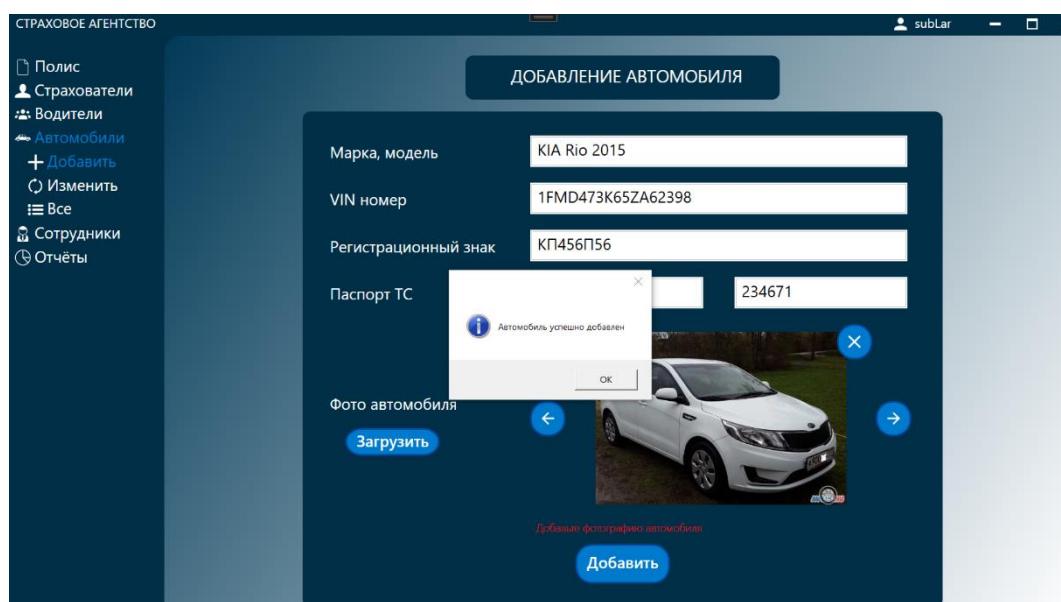


Рисунок 29 - скриншот успешного добавления

5.12 Изменение/удаление автомобиля

На данной странице обрабатываются следующие ошибки:

- незаполненная строка поиска;

Изм.	Лист	№ докум.	Подпись	Дата

- не выбран автомобиль (если пользователь нажал кнопку изменить, но до этого в строке поиска не нашёл нужную информацию);
- незаполненное поле Регистрационный знак;
- введённая серия паспорта не содержит 4 знака;
- введённые первые два символа серии паспорта ТС не цифры;
- введённые последние два символа серии паспорта ТС не буквы;
- введённый номер паспорта ТС не содержит 6 цифр;
- введённый номер паспорта ТС содержит не только цифры;
- в список фотографий не добавлена ни одна фотография.

В классе Database обрабатываются следующие ошибки для данной страницы:

- введён VIN номер автомобиля, которого не существует (для строки поиска);
- попытка удаления автомобиля, на который оформлен полис;
- произошла ошибка в работе БД.

Также в форме нельзя изменить марку/модель и VIN номер автомобиля.

При отсутствии ошибок автомобиль успешно изменяется и выводится сообщение "Автомобиль успешно изменён" или удаляется с сообщением "Автомобиль успешно удалён".

Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

Рисунок 30 - скриншот вывода ошибок

Рисунок 31 - скриншот успешного изменения

5.13 Все автомобили

На данной странице обрабатываются следующие ошибки:

- незаполненная строка поиска.

В классе Database обрабатываются следующие ошибки для данной страницы:

Изм.	Лист	№ докум.	Подпись	Дата

- введён VIN номер автомобиля, которого не существует (для строки поиска);
- произошла ошибка в работе БД.

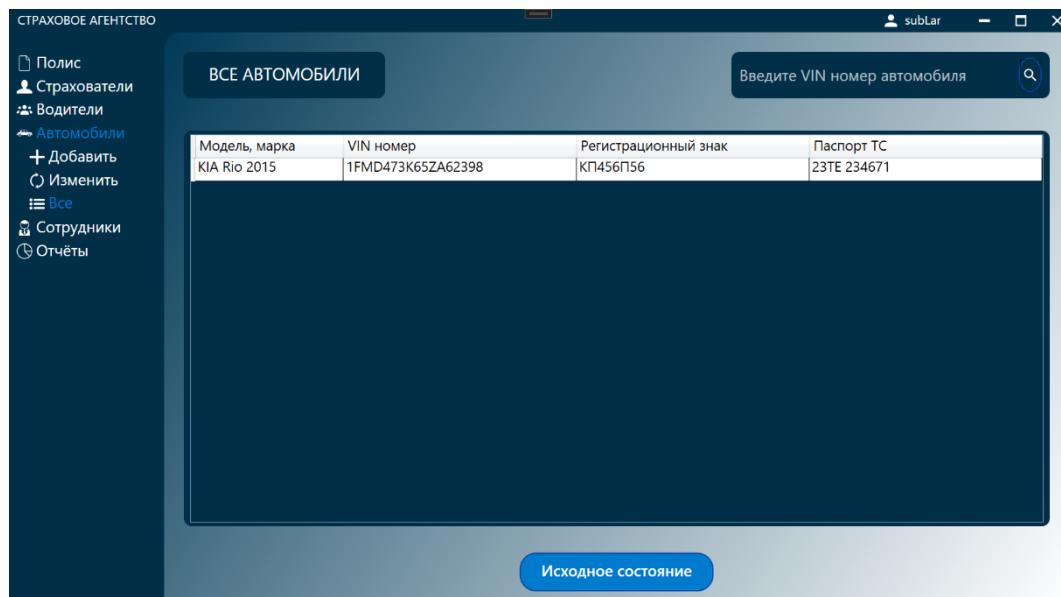


Рисунок 32 - скриншот поиска автомобиля

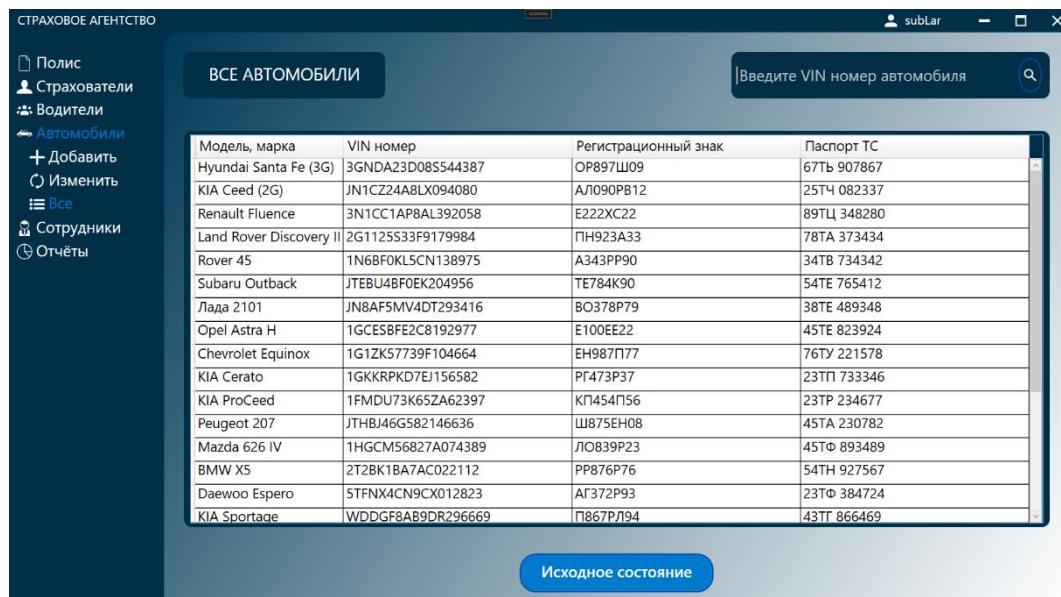


Рисунок 33 - скриншот исходного состояния

5.14 Полис

На данной странице обрабатываются следующие ошибки:

Изм.	Лист	№ докум.	Подпись	Дата

МИВУ 09.03.04-8.000 ПЗ

Лист
60

- незаполненная строка поиска.

В классе Database обрабатываются следующие ошибки для данной страницы:

- введён паспорт или номер телефона страхователя, которого не существует (для строки поиска);
- произошла ошибка в работе БД.

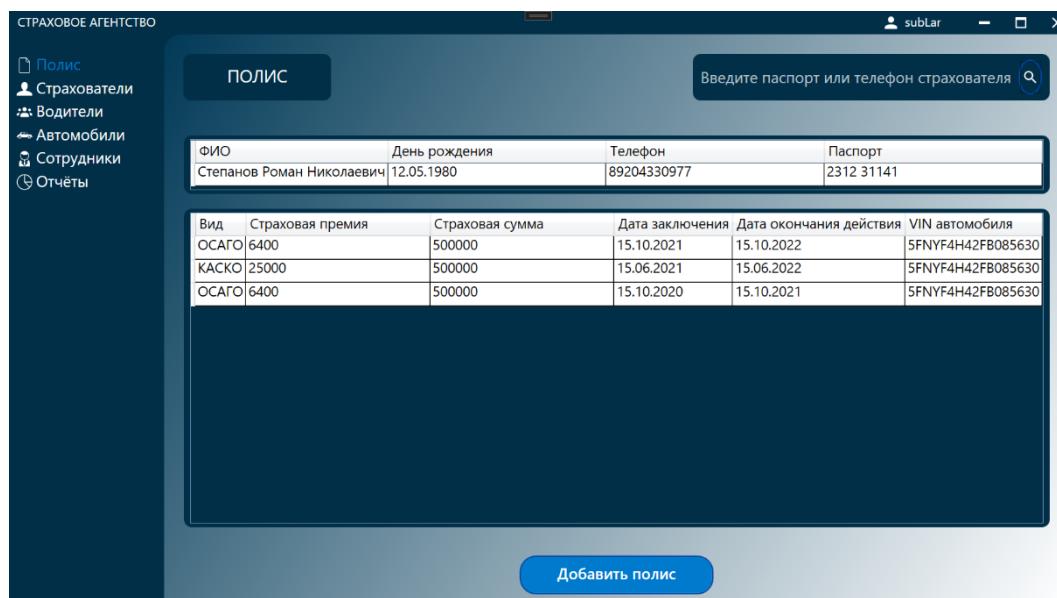


Рисунок 34 - скриншот поиска полисов конкретного страхователя

5.15 Добавление полиса

На данной странице обрабатываются следующие ошибки:

- незаполненное поле Вид страхования;
- незаполненное поле Страховая сумма;
- введена страховая сумма, которая не является целым числом;
- незаполненное поле Страховая премия;
- введена страховая премия, которая не является целым числом;
- незаполненное поле Срок действия;
- введен VIN номер, который не содержит 17 знаков;

Изм.	Лист	№ докум.	Подпись	Дата

- введён VIN номер, который состоит не только из цифр и заглавных латинских букв;
- список лиц, допущенных к управлению пуст;
- введён водитель, который уже добавлен (при добавление нового водителя в список лиц, допущенных к управлению);
- выбран водитель, который не существует в списке добавленных водителей (при удалении водителя из списка лиц, допущенных к управлению).

В классе Database обрабатываются следующие ошибки для данной страницы:

- введён автомобиль, который не существует (добавление автомобиля к полису);
- введён сотрудник, который не существует (добавление сотрудника к полису);
- введён водитель, который не существует (добавление нового водителя в список лиц, допущенных к управлению);
- введён страхователь, которого нет (добавление полиса);
- введён автомобиль, которого нет (добавление полиса);
- введён сотрудник, которого нет (добавление полиса);
- произошла ошибка в работе БД.

При отсутствии ошибок полис успешно добавляется и выводится сообщение "Полис успешно добавлен".

Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

Рисунок 35 - скриншот вывода ошибок

Рисунок 36 - скриншот успешного добавления

5.16 Изменение полиса

На данной странице обрабатываются следующие ошибки:

- незаполненное поле Страховая премия;
- введена страховая премия, которая не является целым числом;
- введена дата окончания действия, которая меньше даты заключения;

Изм.	Лист	№ докум.	Подпись	Дата

- введена дата окончания действия, которая меньше даты последнего страхового случая;
- список лиц, допущенных к управлению пуст;
- введён водитель, который уже добавлен (при добавление нового водителя в список лиц, допущенных к управлению);
- выбран водитель, который не существует в списке добавленных водителей (при удалении водителя из списка лиц, допущенных к управлению).

В классе Database обрабатываются следующие ошибки для данной страницы:

- введён водитель, который не существует (проверка при добавление нового водителя в список лиц, допущенных к управлению);
- произошла ошибка в работе БД.

Также в форме нельзя изменить вид страхования, страховую сумму, дату заключения и VIN номер автомобиля.

При отсутствии ошибок полис успешно изменяется и выводится сообщение "Полис успешно изменён".

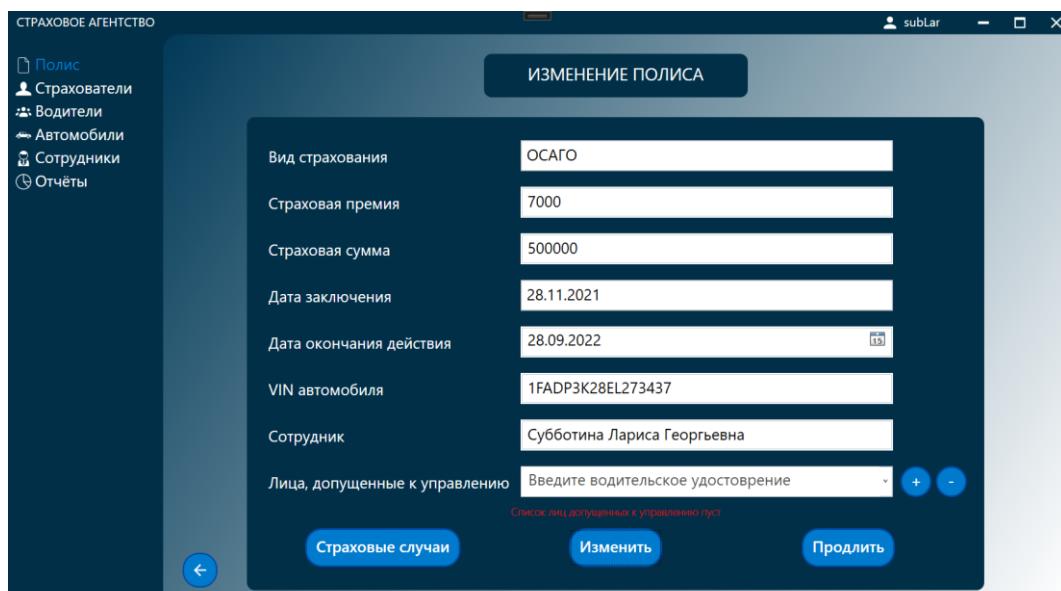


Рисунок 37 - скриншот вывода ошибок

Изм.	Лист	№ докум.	Подпись	Дата

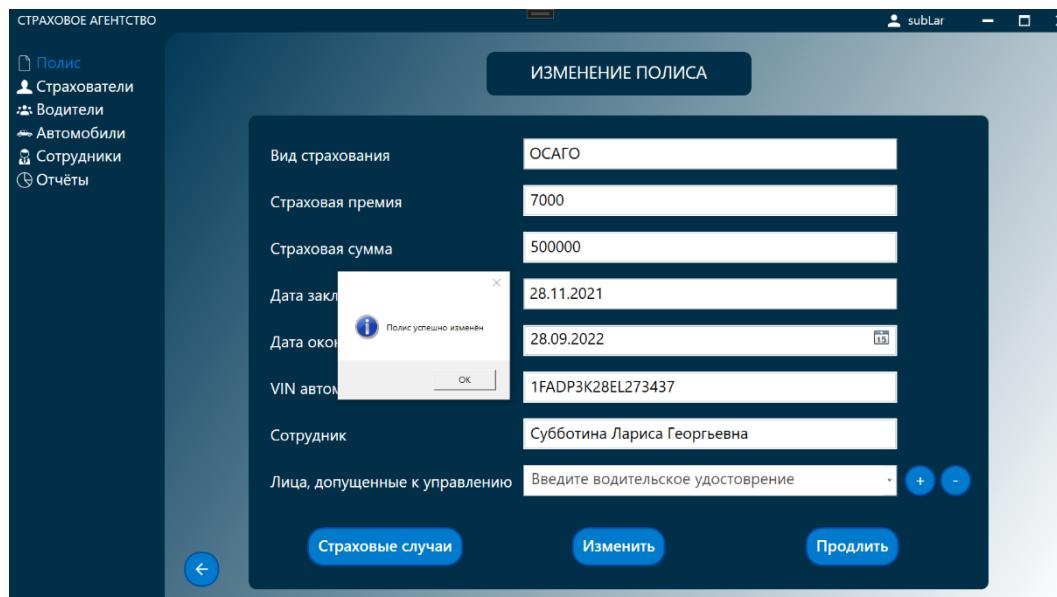


Рисунок 38 - скриншот успешного изменения

5.17 Добавление страхового случая

На данной странице обрабатываются следующие ошибки:

- выбрана дата, которая меньше даты заключения полиса;
- выбрана дата, которая больше даты окончания действия полиса;
- незаполненное поле Страховая выплата;
- введена страховая выплата, которая не является целым числом;
- введена страховая выплата, которая больше Страховой суммы.

В классе Database обрабатываются следующие ошибки для данной страницы:

- произошла ошибка в работе БД.

При отсутствии ошибок страховой случай успешно добавляется и выводится сообщение "Страховой случай успешно добавлен".

Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

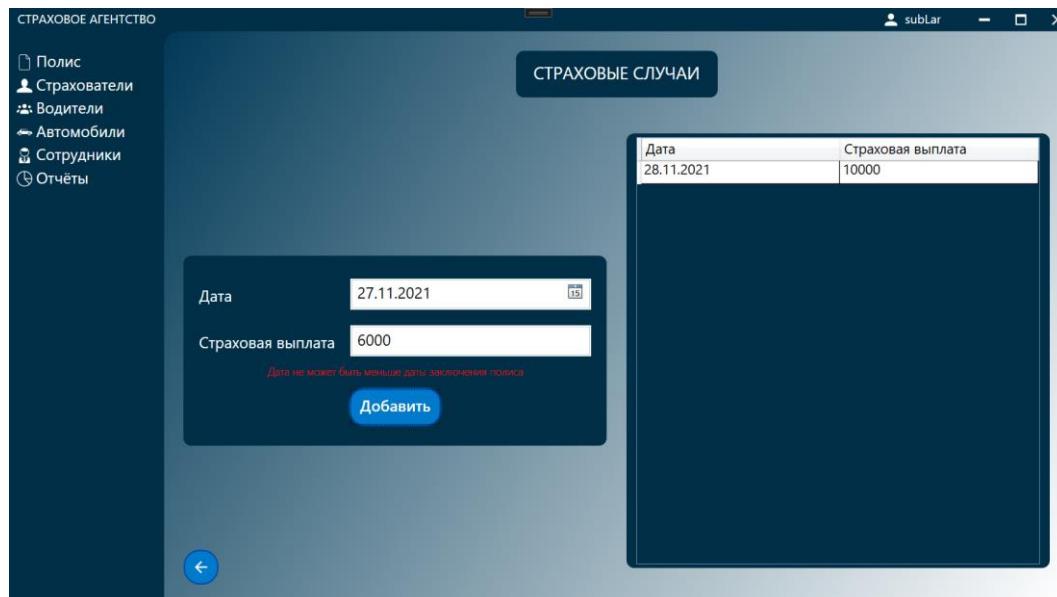


Рисунок 39 - скриншот вывода ошибок

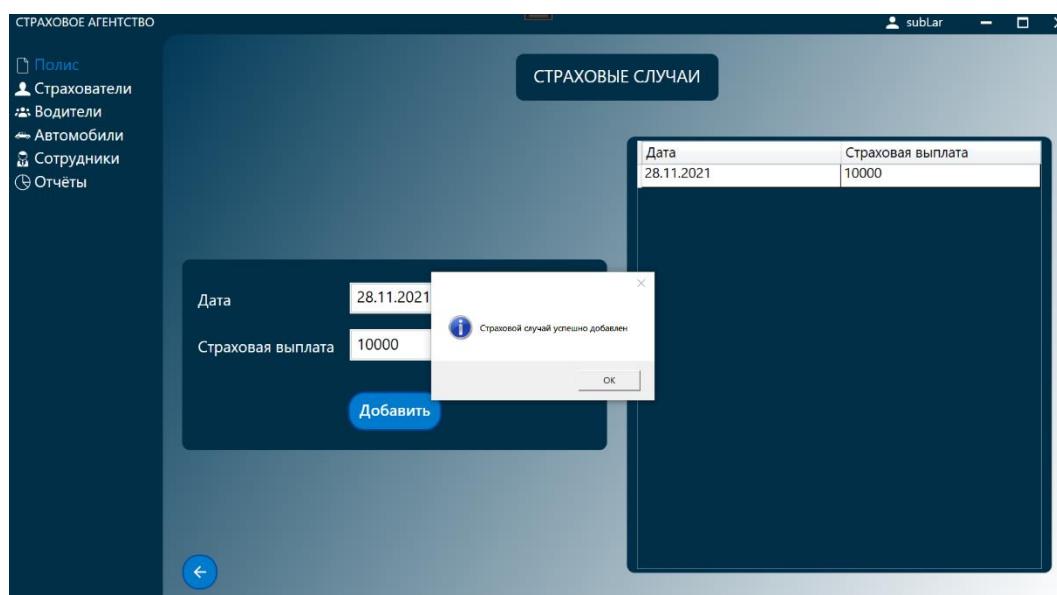


Рисунок 40 - скриншот успешного добавления

5.18 Отчёты

На данной странице обрабатываются следующие ошибки:

- незаполненное поле Вид страхования;
- выбрана дата начала, которая не больше даты окончания.

В классе Database обрабатываются следующие ошибки для данной страницы:

Изм.	Лист	№ докум.	Подпись	Дата

– произошла ошибка в работе БД.

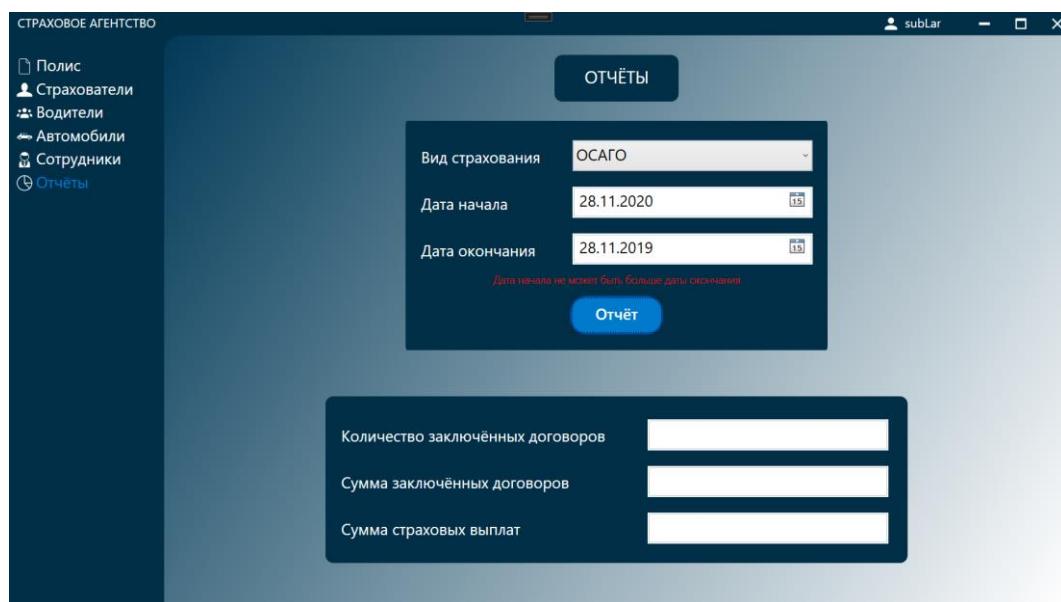


Рисунок 41 - скриншот вывода ошибок

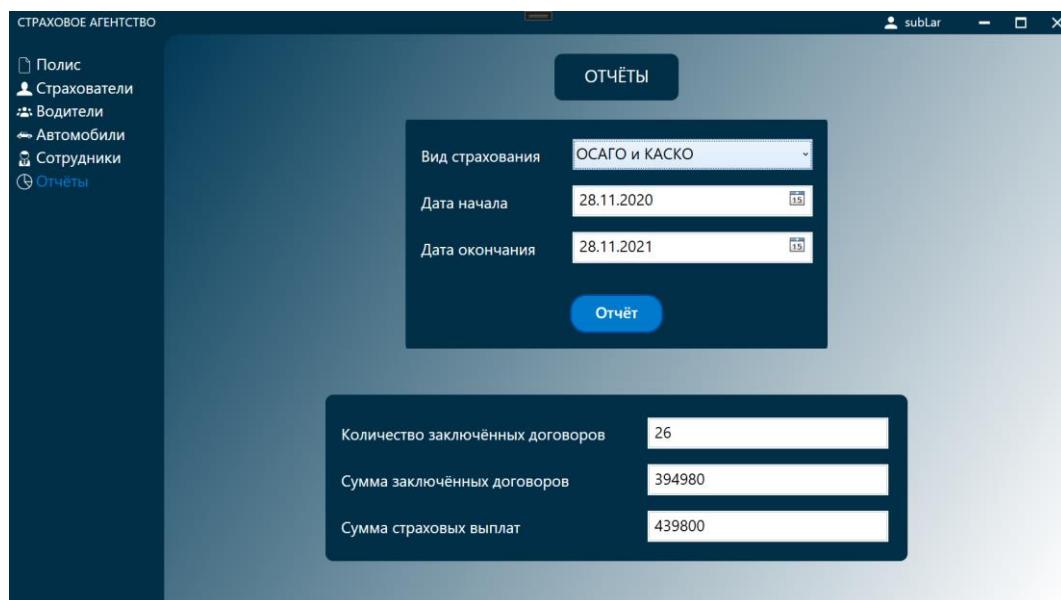


Рисунок 42 - скриншот отчёта

Изм.	Лист	№ докум.	Подпись	Дата

Заключение

В данной курсовой работе в соответствии с заданием была разработана АИС Страхового агентства.

В ходе выполнения курсовой работы были выполнены следующие задачи:

- выявлены требования к программе;
- разработаны модели данных;
- создана база данных;
- разработана программа;
- осуществлено ее тестирование.

Разработанная программа обеспечивает осуществление следующих функций:

- добавление данных о страхователях, сотрудниках, автомобилях, лицах, допущенных к управлению;
- добавление полисов;
- возможность работы со страховыми случаями;
- возможность изменения информации;
- возможность удаления выбранной информации;
- предоставление информации на форме в табличном виде;
- возможность анализировать количество и суммы заключенных договоров по каждому из видов, а также оценивать риски, подсчитывая суммы страховых выплат по каждому виду договоров, а также составлять финансовый отчет деятельности компании за заданный период времени.

Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

Список используемой литературы

1. Аналог Полисы ОСАГО 1.0.9: [Электронный ресурс] // URL: <https://araxgroup.ru/index.php/products/53-other-programs/271-polisi-osago> (Дата обращения – 1.10.2021)

2. Аналог Учёт автострахования ОСАГО: [Электронный ресурс] // URL:

http://bestsoft.moy.su/load/ofis_programmy_dlja_kompjutera_skachat_besplatno/ofis_programmy_dlja_kompjutera_skachat_besplatno/skachat_besplatno_programmu_uchjot_i_zapolnenie_polisov_avtostrakhovanija_osago_5_6_bez_registracii_i_smms_licenzionnyj_kljuch_aktivacii/11-1-0-108 (Дата обращения – 1.10.2021)

3. Изображение страхового полиса КАСКО: [Электронный ресурс] // URL: <https://avtoguru.pro/strahovanie/kasko/vneshnij-vid.html> (Дата обращения – 1.10.2021)

4. Изображение страхового полиса ОСАГО: [Электронный ресурс] // URL: <https://zakon-auto.ru/osago/zapolnenie-osago.php> (Дата обращения – 1.10.2021)

5. Руководство по WPF: [Электронный ресурс] // URL: <https://metanit.com/sharp/wpf/1.php> (Дата обращения – 25.11.2021)

6. Руководство по MS SQL Server: [Электронный ресурс] // URL: <https://metanit.com/sql/sqlserver/> (Дата обращения – 19.11.2021)

7. C# и MS SQL Server: [Электронный ресурс] // URL: <https://metanit.com/sharp/adonet/2.1.php> (Дата обращения – 20.11.2021)

Изм.	Лист	№ докум.	Подпись	Дата

Приложение 1. Создание БД

Таблица страхователей (Policyholders). В таблице присутствуют следующие ограничения:

- ID является первичным ключом;
- Telephone должен быть уникальным;
- Passport должен быть уникальным.

Листинг запроса на создание таблицы страхователей (Policyholders):

```
CREATE TABLE Policyholders(  
    ID INT NOT NULL PRIMARY KEY IDENTITY,  
    FullName VARCHAR(64) NOT NULL,  
    Birthday DATE NOT NULL,  
    Telephone VARCHAR(15) NOT NULL,  
    Passport VARCHAR(10) NOT NULL,  
    CONSTRAINT policyholderTelephoneUnique UNIQUE (Telephone),  
    CONSTRAINT policyholderPassportUnique UNIQUE (Passport)  
);
```

Таблица сотрудников (Employees). В таблице присутствуют следующие ограничения:

- ID является первичным ключом;
- Telephone должен быть уникальным;
- Passport должен быть уникальным;
- Login должен быть уникальным.

Листинг запроса на создание таблицы сотрудников (Employees):

```
CREATE TABLE Employees(  
    ID INT NOT NULL PRIMARY KEY IDENTITY,  
    FullName VARCHAR(64) NOT NULL,  
    Birthday DATE NOT NULL,  
    Telephone VARCHAR(15) NOT NULL,  
    Passport VARCHAR(10) NOT NULL,  
    "Login" VARCHAR(32) NOT NULL,  
    "Password" VARCHAR(32) NOT NULL,  
    "Admin" BIT NOT NULL,  
    Works BIT NOT NULL,  
    CONSTRAINT employeeTelephoneUnique UNIQUE (Telephone),  
    CONSTRAINT employeePassportUnique UNIQUE (Passport),  
    CONSTRAINT loginUnique UNIQUE ("Login")  
);
```

Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

Таблица автомобилей (Cars). В таблице присутствуют следующие ограничения:

- ID является первичным ключом;
- VIN должен быть уникальным.

Листинг запроса на создание таблицы автомобилей (Cars):

```
CREATE TABLE Cars(  
    ID INT NOT NULL PRIMARY KEY IDENTITY,  
    Model VARCHAR(50) NOT NULL,  
    VIN VARCHAR(17) NOT NULL,  
    RegistrationPlate VARCHAR(25) NOT NULL,  
    VehiclePassport VARCHAR(10) NOT NULL,  
    CONSTRAINT VINUnique UNIQUE (VIN)  
)
```

Таблица фотографий (Photos). В таблице присутствуют следующие ограничения:

- ID является первичным ключом;
- CarID является внешним ключом, который хранит ID автомобиля, которому принадлежит данная фотография. Также при удалении автомобиля удаляются все фотографии, которые ссылаются на данный автомобиль.

Листинг запроса на создание таблицы фотографий (Photos):

```
CREATE TABLE Photos(  
    ID INT NOT NULL PRIMARY KEY IDENTITY,  
    EncodedPhoto NVARCHAR(MAX) NOT NULL,  
    CarID INT NOT NULL,  
    FOREIGN KEY (CarID) REFERENCES Cars (ID) ON DELETE CASCADE  
)
```

Таблица полисов (Policies). В таблице присутствуют следующие ограничения:

- ID является первичным ключом;
- PolicyholderID является внешним ключом на ID страхователя, который оформил данный полис;
- CarID является внешним ключом на ID автомобиля, на который оформлен данный полис;

Изм.	Лист	№ докум.	Подпись	Дата

- PolicyholderID является внешним ключом на ID страхователя, который оформил данный полис.

Листинг запроса на создание таблицы полисов (Policies):

```
CREATE TABLE Policies(
    ID INT NOT NULL PRIMARY KEY IDENTITY,
    InsuranceType VARCHAR(5) NOT NULL,
    InsurancePremium INT NOT NULL,
    InsuranceAmount INT NOT NULL,
    DateOfConclusion DATE NOT NULL,
    ExpirationDate DATE NOT NULL,
    PolicyholderID INT NOT NULL,
    CarID INT NOT NULL,
    EmployeeID INT NOT NULL,
    FOREIGN KEY (PolicyholderID) REFERENCES Policyholders (ID),
    FOREIGN KEY (CarID) REFERENCES Cars (ID),
    FOREIGN KEY (EmployeeID) REFERENCES Employees (ID)
);
```

Таблица страховых случаев (InsuranceEvents). В таблице присутствуют следующие ограничения:

- ID является первичным ключом;
- PolicyID является внешним ключом, который хранит ID полиса, к которому привязан данный страховой случай. Также при удалении полиса удаляются все страховые случаи, которые ссылаются на данный полис.

Листинг запроса на создание таблицы страховых случаев (InsuranceEvents):

```
CREATE TABLE InsuranceEvents(
    ID INT NOT NULL PRIMARY KEY IDENTITY,
    "Date" DATE NOT NULL,
    InsurancePayment INT NOT NULL,
    PolicyID INT NOT NULL,
    FOREIGN KEY (PolicyID) REFERENCES Policies (ID) ON DELETE CASCADE
);
```

Таблица лиц, допущенных к управлению (PersonsAllowedToDrive). В таблице присутствуют следующие ограничения:

- ID является первичным ключом;
- DrivingLicence должен быть уникальным.

Изм.	Лист	№ докум.	Подпись	Дата

Листинг запроса на создание таблицы лиц, допущенных к управлению (PersonsAllowedToDrive):

```
CREATE TABLE PersonsAllowedToDrive(
    ID INT NOT NULL PRIMARY KEY IDENTITY,
    FullName VARCHAR(64) NOT NULL,
    DrivingLicence VARCHAR(10) NOT NULL,
    CONSTRAINT drivingLicenceUnique UNIQUE (DrivingLicence)
);
```

Таблица связей (Connections). В таблице присутствуют следующие ограничения:

- связка PolicyID и PersonAllowedToDriveID должна быть уникальной;
- PolicyID является внешним ключом на ID полиса, к которому относится данная связь;
- PersonAllowedToDriveID является внешним ключом на ID лица, допущенного к управлению, к которому относится данная связь.

Листинг запроса на создание таблицы связей (Connections):

```
CREATE TABLE Connections(
    PolicyID INT NOT NULL,
    PersonAllowedToDriveID INT NOT NULL,
    CONSTRAINT connectionUnique UNIQUE (PolicyID, PersonAllowedToDriveID),
    FOREIGN KEY (PolicyID) REFERENCES Policies (ID),
    FOREIGN KEY (PersonAllowedToDriveID) REFERENCES PersonsAllowedToDrive (ID)
);
```

Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

Приложение 2. Снимки окон программы (скриншоты программы)

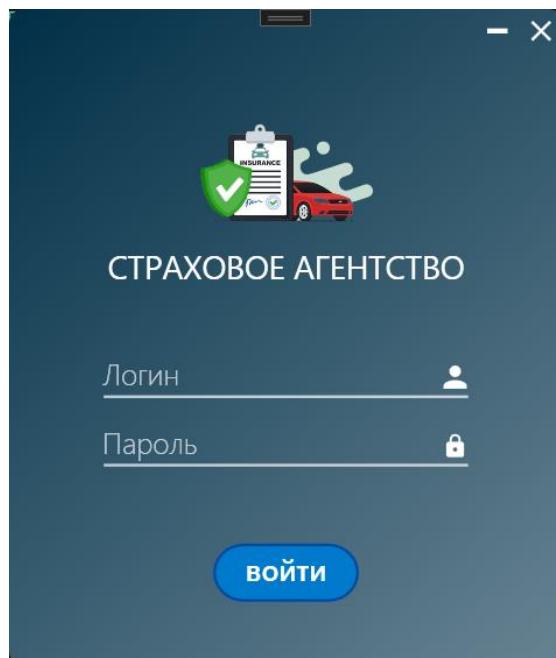


Рисунок 43 - снимок окна авторизации

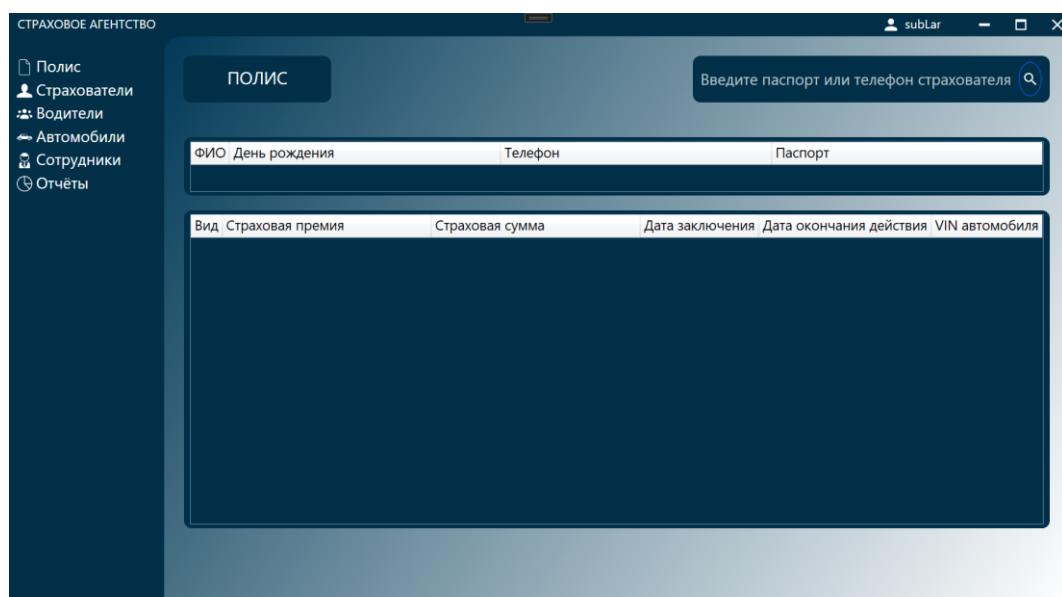


Рисунок 44 - снимок страницы полиса

						Лист
Изм.	Лист	№ докум.	Подпись	Дата	МИВУ 09.03.04-8.000 ПЗ	74

СТРАХОВОЕ АГЕНТСТВО

subLar

Полис
Страхователи
Водители
Автомобили
Сотрудники
Отчёты

ДОБАВЛЕНИЕ ПОЛИСА

Вид страхования	
Страховая премия	
Страховая сумма	
Дата заключения	28.11.2021
Срок действия	
VIN автомобиля	
Лица, допущенные к управлению	Введите водительское удостоверение

Добавить

←

Рисунок 45 - снимок страницы добавления полиса

СТРАХОВОЕ АГЕНТСТВО

subLar

Полис
Страхователи
Водители
Автомобили
Сотрудники
Отчёты

ИЗМЕНЕНИЕ ПОЛИСА

Вид страхования	ОСАГО
Страховая премия	7000
Страховая сумма	500000
Дата заключения	28.11.2021
Дата окончания действия	28.09.2022
VIN автомобиля	1FADP3K28EL273437
Сотрудник	Субботина Лариса Георгьевна
Лица, допущенные к управлению	Введите водительское удостоверение

Страховые случаи Изменить Продлить

←

Рисунок 46 - снимок страницы изменение полиса

Изм.	Лист	№ докум.	Подпись	Дата	Лист	МИВУ 09.03.04-8.000 ПЗ	75

СТРАХОВОЕ АГЕНТСТВО

Полис
Страхователи
Водители
Автомобили
Сотрудники
Отчёты

СТРАХОВЫЕ СЛУЧАИ

Дата 28.11.2021 [15]
Страховая выплата
Добавить

Дата	Страховая выплата
28.11.2021	10000

←

This screenshot shows the 'Insurance Cases' section of a software interface. On the left, there's a sidebar with navigation links: Полис (Policy), Страхователи (Insurees), Водители (Drivers), Автомобили (Vehicles), Сотрудники (Employees), and Отчёты (Reports). The main area is titled 'СТРАХОВЫЕ СЛУЧАИ' (Insurance Cases). It contains a form for adding a new case: 'Дата' (Date) is set to '28.11.2021' and 'Страховая выплата' (Insurance Payment) is empty. A blue 'Добавить' (Add) button is at the bottom of the form. To the right is a table showing one record: 'Дата' (Date) '28.11.2021' and 'Страховая выплата' (Insurance Payment) '10000'. A back arrow is at the bottom left.

Рисунок 47 - снимок страницы страховых случаев

СТРАХОВОЕ АГЕНТСТВО

Полис
Страхователи
+ Добавить
Изменить
Все
Водители
Автомобили
Сотрудники
Отчёты

ДОБАВЛЕНИЕ СТРАХОВАТЕЛЯ

ФИО
Дата рождения 01.01.1990 [15]
Номер телефона
Паспортные данные Серия Номер
Добавить

This screenshot shows the 'Adding Insurer' page. The sidebar includes links for Полис (Policy), Страхователи (Insurees) with a '+ Добавить' (Add) option highlighted, Изменить (Edit), Все (All), Водители (Drivers), Автомобили (Vehicles), Сотрудники (Employees), and Отчёты (Reports). The main title is 'ДОБАВЛЕНИЕ СТРАХОВАТЕЛЯ' (Adding Insurer). The form requires 'ФИО' (Name) and 'Дата рождения' (Date of Birth) set to '01.01.1990'. There are also fields for 'Номер телефона' (Phone Number) and 'Паспортные данные' (Passport Details), which include 'Серия' (Series) and 'Номер' (Number) inputs. A blue 'Добавить' (Add) button is at the bottom.

Рисунок 48 - снимок страницы добавления страхователя

Изм.	Лист	№ докум.	Подпись	Дата	Лист
					МИВУ 09.03.04-8.000 ПЗ

76

Рисунок 49 - снимок страницы изменения/удаления страхователя

Рисунок 50 - снимок страницы всех страхователей

						Лист
						МИВУ 09.03.04-8.000 ПЗ
Изм.	Лист	№ докум.	Подпись	Дата		77

СТРАХОВОЕ АГЕНТСТВО

subLar

Полис
Страхователи
Водители
Добавить
Изменить
Все
Автомобили
Сотрудники
Отчеты

ДОБАВЛЕНИЕ ЛИЦА, ДОПУЩЕННОГО К УПРАВЛЕНИЮ

ФИО []

Водительское удостоверение Серия [] Номер []

Добавить

Рисунок 51 - снимок страницы добавления лица, допущенного к управлению

СТРАХОВОЕ АГЕНТСТВО

subLar

Полис
Страхователи
Водители
Добавить
Изменить
Все
Автомобили
Сотрудники
Отчеты

ИЗМЕНЕНИЕ/УДАЛЕНИЕ ЛИЦА, ДОПУЩЕННОГО К УПРАВЛЕНИЮ

Введите водительское удостоверение

ФИО []

Водительское удостоверение Серия [] Номер []

Изменить Удалить

Рисунок 52 - снимок страницы изменения/удаления лица, допущенного к управлению

Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

МИВУ 09.03.04-8.000 ПЗ

Лист

78

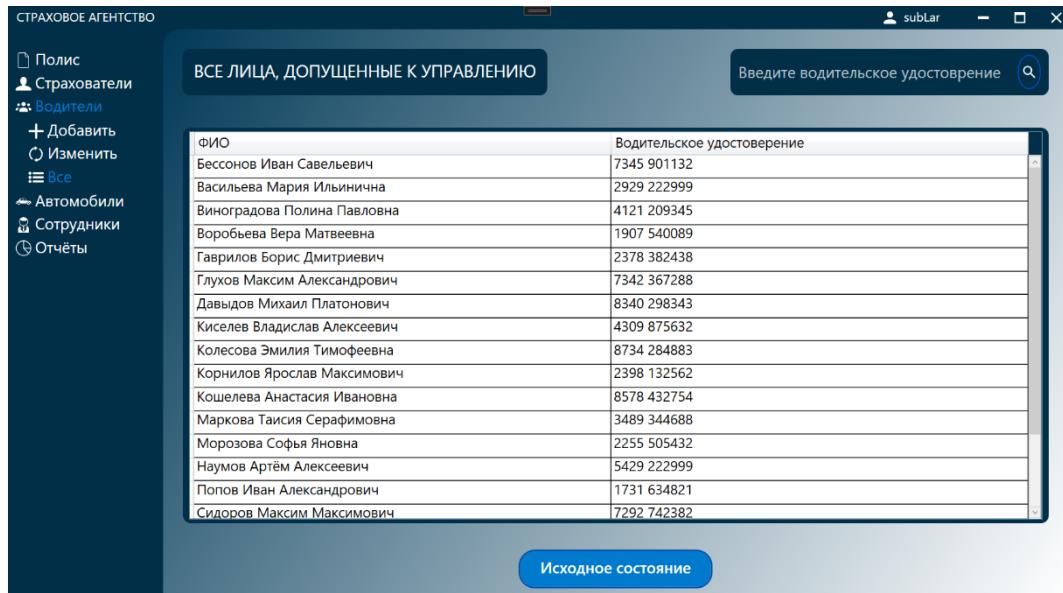


Рисунок 53 - снимок страницы всех лиц, допущенных к управлению

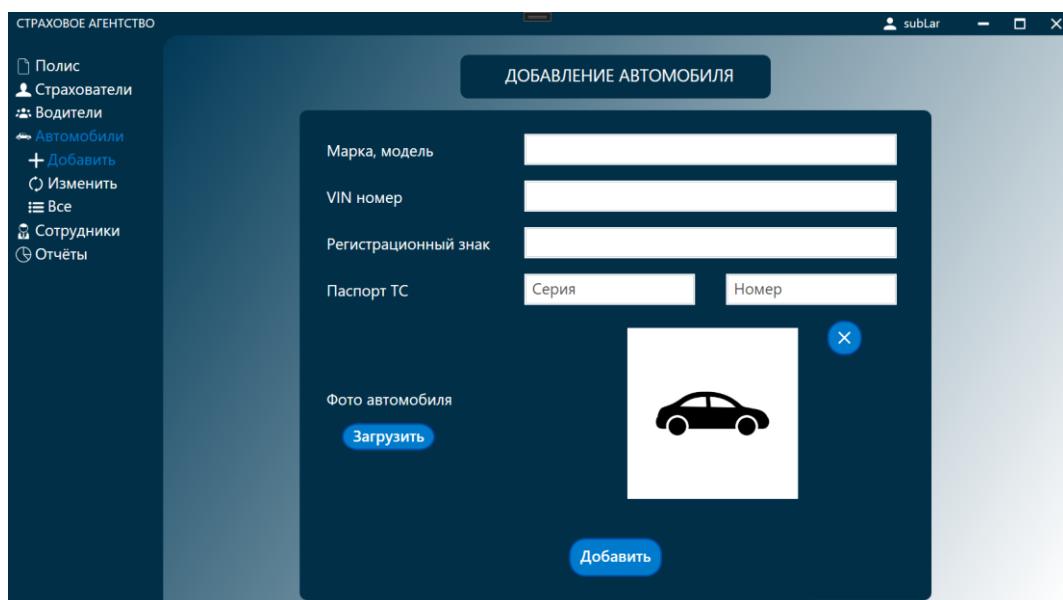


Рисунок 54 - снимок страницы добавления автомобиля

Изм.	Лист	№ докум.	Подпись	Дата	Лист
					МИВУ 09.03.04-8.000 ПЗ

СТРАХОВОЕ АГЕНТСТВО

subLar

Полис
Страхователи
Водители
Автомобили
Добавить
Изменить
Все
Сотрудники
Отчёты

ИЗМЕНЕНИЕ/УДАЛЕНИЕ АВТОМОБИЛЯ

Введите VIN номер автомобиля

Марка, модель

VIN номер

Регистрационный знак

Паспорт ТС

Серия Номер

Фото автомобиля

Загрузить

Изменить Удалить

Рисунок 55 - снимок страницы изменения/удаления автомобиля

СТРАХОВОЕ АГЕНТСТВО

subLar

Полис
Страхователи
Водители
Автомобили
Добавить
Изменить
Все
Сотрудники
Отчёты

Все Автомобили

Введите VIN номер автомобиля

Модель, марка	VIN номер	Регистрационный знак	Паспорт ТС
Hyundai Santa Fe (3G)	3GNDA23D08S544387	ОР897Ш09	67ТЬ 907867
KIA Ceed (2G)	JN1CZ24A8LX094080	АЛ090РВ12	25ТЧ 082337
Renault Fluence	3N1CC1AP8AL392058	Е222ХС22	89ТЦ 348280
Land Rover Discovery II	2G1125533F9179984	ЛН923А33	78ТА 373434
Rover 45	1N6BF0KL5CN138975	А343РР90	34ТВ 734342
Subaru Outback	JTEBU4BF0EK204956	ТЕ784К90	54ТЕ 765412
Лада 2101	JN8AF5MV4DT293416	ВО37РР79	38ТЕ 489348
Opel Astra H	1GCESBFE2C8192977	Е100ЕЕ22	45ТЕ 823924
Chevrolet Equinox	1G1ZK57739F104664	ЕН987П77	76ТУ 221578
KIA Cerato	1GKRRPKD7EJ156582	Р473Р37	23ТП 733346
KIA ProCeed	1FMDU73K65ZA62397	КТ454П56	23ТУ 234677
Peugeot 207	JTHB46G582146636	Ш875ЕН08	45ТА 230782
Mazda 626 IV	1HGCM56827A074389	ЛО83РР23	45ТФ 893489
BMW X5	2T2BK1BA7AC022112	РР876Р76	54ТН 927567
Daewoo Espero	5TFNX4CN9CX012823	АГ372Р93	23ТФ 384724
KIA Sportage	WDDGF8AB9DR296669	П867РЛ94	43ТР 866469

Исходное состояние

Рисунок 56 - снимок страницы всех автомобилей

Лист

МИВУ 09.03.04-8.000 ПЗ

80

Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

СТРАХОВОЕ АГЕНТСТВО

subLar

Полис
Страхователи
Водители
Автомобили
Сотрудники
Добавить
Изменить
Все
Отчеты

ДОБАВЛЕНИЕ СОТРУДНИКА

ФИО
Дата рождения
Номер телефона
Паспортные данные
Логин
Пароль
Администратор

Добавить

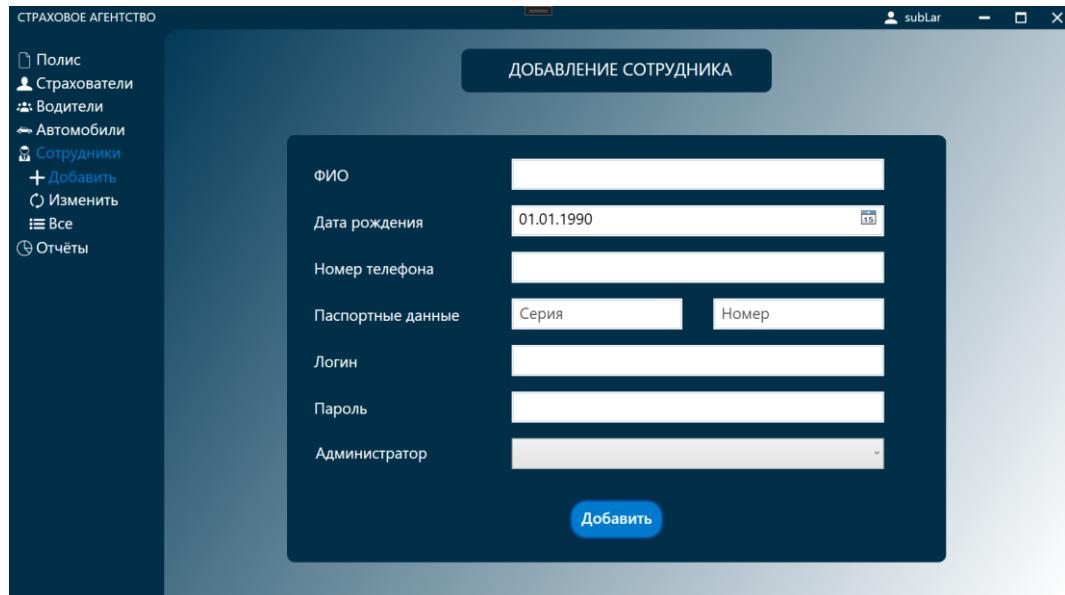


Рисунок 57 - снимок страницы добавления сотрудника

СТРАХОВОЕ АГЕНТСТВО

subLar

Полис
Страхователи
Водители
Автомобили
Сотрудники
Добавить
Изменить
Все
Отчеты

ИЗМЕНЕНИЕ/УДАЛЕНИЕ СОТРУДНИКА

Введите паспорт или телефон сотрудника

ФИО
Дата рождения
Номер телефона
Паспортные данные
Логин
Пароль
Администратор
Работает ли

Изменить Удалить

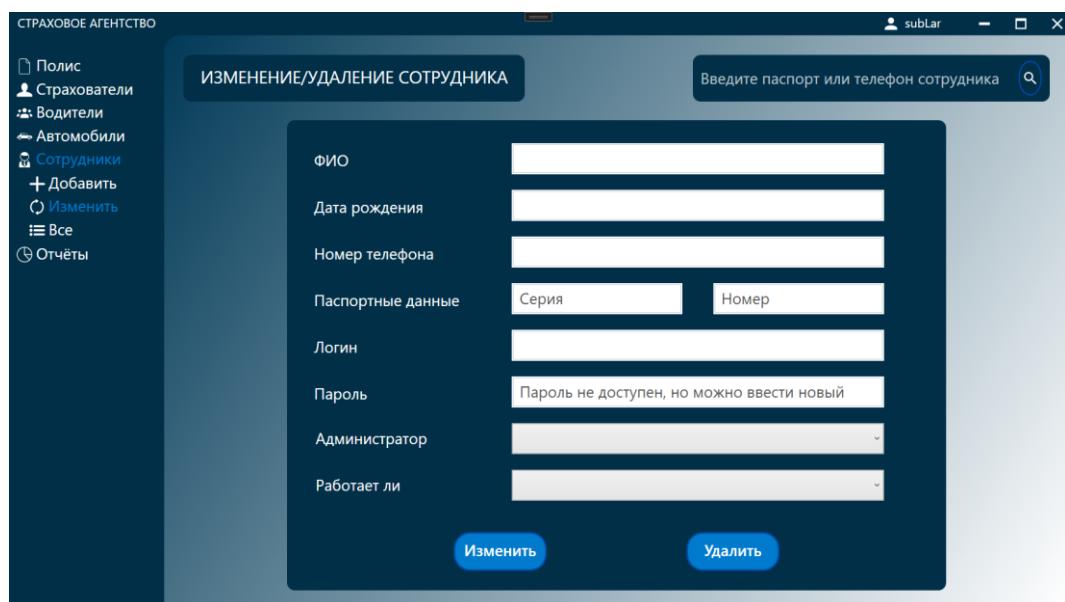


Рисунок 58 - снимок страницы изменения/добавления сотрудника

					Лист МИВУ 09.03.04-8.000 ПЗ
Изм.	Лист	№ докум.	Подпись	Дата	

81

СТРАХОВОЕ АГЕНТСТВО

subLar

Все СОТРУДНИКИ

Введите паспорт или телефон сотрудника

Выберите тип сортировки

ФИО	Дата рождения	Телефон	Паспорт	Логин	Админ	Работает ли
Белкин Владимир Алексеевич	24.03.1991	89159370446	2309 832999	vladb	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Беляков Богдан Борисович	21.02.1990	89239289234	4513 934859	bogdanB	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Петров Максим Борисович	14.10.1989	89189389283	2819 483820	petrovM	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Субботина Лариса Георгиевна	19.10.1986	89209458438	2410 923939	subLar	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Суворов Степан Вадимович	31.08.1996	89049392392	2313 439902	stepfisc	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Зайцев Александр Владимирович	15.10.1997	89189348574	1611 321313	alexz	<input type="checkbox"/>	<input type="checkbox"/>

Исходное состояние

Рисунок 59 - снимок страницы всех сотрудников

СТРАХОВОЕ АГЕНТСТВО

subLar

Отчёты

Вид страхования

Дата начала: 28.11.2020

Дата окончания: 28.11.2021

Отчёт

Количество заключённых договоров

Сумма заключённых договоров

Сумма страховых выплат

Рисунок 60 - снимок страницы отчётов

Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

МИВУ 09.03.04-8.000 ПЗ

Лист

82

Приложение 3. Код программы

Листинг класса Database:

```
namespace InsuranceAgency
{
    public static class Database
    {
        private static string ConnectionString = "Server=localhost;" + "database=DBInsuranceAgency;" + "Integrated Security=True";

        private static string _login;
        private static bool _admin = true;

        public static string Login { get { return _login; } private set { _login = value; } }
        public static bool Admin { get { return _admin; } private set { _admin = value; } }

        public static string GetHash(string input)
        {
            var md5 = MD5.Create();
            var hash = md5.ComputeHash(Encoding.UTF8.GetBytes(input));

            return Convert.ToBase64String(hash);
        }

        public static void Authorization(string login, string password)
        {
            bool flag = false;
            try
            {
                using (SqlConnection con = new SqlConnection(ConnectionString))
                {
                    string query = "SELECT Login, Admin, Works FROM Employees WHERE Login = @login AND Password = @password";
                    SqlCommand command = new SqlCommand(query, con);

                    command.Parameters.Add(new SqlParameter("@login", login));
                    command.Parameters.Add(new SqlParameter("@Password", GetHash(password)));

                    con.Open();
                    SqlDataReader reader = command.ExecuteReader();

                    if (!reader.HasRows)
                    {
                        flag = true;
                        throw new Exception("Неправильно указан логин и/или пароль");
                    }
                    while (reader.Read())
                    {
                        if (Convert.ToBoolean(reader["Works"]))
                        {
                            _login = reader["Login"].ToString();
                            _admin = Convert.ToBoolean(reader["Admin"]);
                        }
                        else
                        {
                            flag = true;
                        }
                    }
                }
            }
        }
    }
}
```

Изм.	Лист	№ докум.	Подпись	Дата

```

        throw new Exception("Данный сотрудник больше не работает");
    }
}
reader.Close();
con.Close();
}
}
catch (Exception exp)
{
    if (flag)
    {
        throw exp;
    }
    else
    {
        throw new Exception("Ошибка в работе БД");
    }
}
}

```

```

//Функции добавления
public static void AddCarWithPhotos(Car car, List<string> photos)
{
    bool flag = false;
    try
    {
        using (SqlConnection con = new SqlConnection(ConnectionString))
        {
            string query1 = "SELECT ID FROM Cars WHERE VIN = @vin";
            SqlCommand command1 = new SqlCommand(query1, con);
            command1.Parameters.Add(new SqlParameter("@vin", car.VIN));

            string query = "BEGIN TRANSACTION " +
                "INSERT INTO Cars(Model, VIN, RegistrationPlate, VehiclePassport) " +
                "VALUES(@model, @vin, @registrationPlate, @vehiclePassport); " +

                "DECLARE @id INT; " +
                "SET @id=SCOPE_IDENTITY(); " +

                "INSERT INTO Photos(EncodedPhoto, CarID) VALUES ";
            for (var i = 0; i < photos.Count - 1; i++)
            {
                query += "(@photo" + i + ", @id), ";
            }
            query += "(@photo" + (photos.Count - 1) + ", @id); " +
                "COMMIT TRANSACTION";

            SqlCommand command = new SqlCommand(query, con);
            command.Parameters.Add(new SqlParameter("@model", car.Model));
            command.Parameters.Add(new SqlParameter("@vin", car.VIN));
            command.Parameters.Add(new SqlParameter("@registrationPlate", car.RegistrationPlate));
            command.Parameters.Add(new SqlParameter("@vehiclePassport", car.VehiclePassport));
            for (var i = 0; i < photos.Count; i++)
            {
                command.Parameters.Add(new SqlParameter("@photo" + i, photos[i]));
            }

            con.Open();
            SqlDataReader reader = command1.ExecuteReader();
            if (reader.HasRows)

```

Изм.	Лист	№ докум.	Подпись	Дата

```

    {
        flag = true;
        throw new Exception("Данный VIN номер уже используется");
    }
    reader.Close();

    command.ExecuteNonQuery();
    con.Close();
}
}
catch (Exception exp)
{
    if (flag)
    {
        throw exp;
    }
    else
    {
        throw new Exception("Ошибка в работе БД");
    }
}
}

public static void AddEmployee(Employee employee)
{
    bool flag = false;
    try
    {
        using (SqlConnection con = new SqlConnection(ConnectionString))
        {
            string query1 = "SELECT ID FROM Employees WHERE Telephone = @telephone";
            SqlCommand command1 = new SqlCommand(query1, con);
            command1.Parameters.Add(new SqlParameter("@telephone", employee.Telephone));

            string query2 = "SELECT ID FROM Employees WHERE Passport = @passport";
            SqlCommand command2 = new SqlCommand(query2, con);
            command2.Parameters.Add(new SqlParameter("@passport", employee.Passport));

            string query3 = "SELECT ID FROM Employees WHERE Login = @login";
            SqlCommand command3 = new SqlCommand(query3, con);
            command3.Parameters.Add(new SqlParameter("@login", employee.Login));

            string query = "INSERT INTO Employees (FullName, Birthday, Telephone, Passport, Login, Password, Admin, Works) " +
                "VALUES(@fullName, @birthday, @telephone, @passport, @login, @password, @admin, @works)";
            SqlCommand command = new SqlCommand(query, con);

            command.Parameters.Add(new SqlParameter("@fullName", employee.FullName));
            command.Parameters.Add(new SqlParameter("@birthday", employee.Birthday));
            command.Parameters.Add(new SqlParameter("@telephone", employee.Telephone));
            command.Parameters.Add(new SqlParameter("@passport", employee.Passport));
            command.Parameters.Add(new SqlParameter("@login", employee.Login));
            command.Parameters.Add(new SqlParameter("@password", GetHash(employee.Password)));
            command.Parameters.Add(new SqlParameter("@admin", employee.Admin));
            command.Parameters.Add(new SqlParameter("@works", employee.Works));

            con.Open();
            SqlDataReader reader = command1.ExecuteReader();
            if (reader.HasRows)
            {
                flag = true;
                throw new Exception("Данный телефон уже используется");
            }
        }
    }
}

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        }

        reader.Close();

        reader = command2.ExecuteReader();
        if (reader.HasRows)
        {
            flag = true;
            throw new Exception("Данный паспорт уже используется");
        }
        reader.Close();

        reader = command3.ExecuteReader();
        if (reader.HasRows)
        {
            flag = true;
            throw new Exception("Данный логин уже занят");
        }
        reader.Close();

        command.ExecuteNonQuery();
        con.Close();
    }

}

catch (Exception exp)
{
    if (flag)
    {
        throw exp;
    }
    else
    {
        throw new Exception("Ошибка в работе БД");
    }
}
}

public static void AddInsuranceEvent(InsuranceEvent insuranceEvent)
{
    try
    {
        using (SqlConnection con = new SqlConnection(ConnectionString))
        {
            string query = "INSERT INTO InsuranceEvents(Date, InsurancePayment, PolicyID) " +
                "VALUES(@date, @insurancePayment, @policyID)";
            SqlCommand command = new SqlCommand(query, con);

            command.Parameters.Add(new SqlParameter("@date", insuranceEvent.Date));
            command.Parameters.Add(new SqlParameter("@insurancePayment",
insuranceEvent.InsurancePayment));
            command.Parameters.Add(new SqlParameter("@policyID", insuranceEvent.PolicyID));

            con.Open();
            command.ExecuteNonQuery();
            con.Close();
        }
    }
    catch
    {
        throw new Exception("Ошибка в работе БД");
    }
}

public static void AddPersonAllowedToDrive(PersonAllowedToDrive personAllowedToDrive)

```

Изм.	Лист	№ докум.	Подпись	Дата

```

{
    bool flag = false;
    try
    {
        using (SqlConnection con = new SqlConnection(ConnectionString))
        {
            string query1 = "SELECT ID FROM PersonsAllowedToDrive WHERE DrivingLicence = @drivingLicence";
            SqlCommand command1 = new SqlCommand(query1, con);
            command1.Parameters.Add(new SqlParameter("@drivingLicence",
personAllowedToDrive.DrivingLicence));

            string query = "INSERT INTO PersonsAllowedToDrive(FullName, DrivingLicence) " +
                "VALUES(@fullName, @drivingLicence)";
            SqlCommand command = new SqlCommand(query, con);

            command.Parameters.Add(new SqlParameter("@fullName", personAllowedToDrive.FullName));
            command.Parameters.Add(new SqlParameter("@drivingLicence",
personAllowedToDrive.DrivingLicence));

            con.Open();
            SqlDataReader reader = command1.ExecuteReader();
            if (reader.HasRows)
            {
                flag = true;
                throw new Exception("Данное водительское удостоверение уже используется");
            }
            reader.Close();

            command.ExecuteNonQuery();
            con.Close();
        }
    }
    catch (Exception exp)
    {
        if (flag)
        {
            throw exp;
        }
        else
        {
            throw new Exception("Ошибка в работе БД");
        }
    }
}

public static void AddPolicyWithConnections(Policy policy, List<PersonAllowedToDrive>
listPersonAllowedToDrive)
{
    bool flag = false;
    try
    {
        using (SqlConnection con = new SqlConnection(ConnectionString))
        {
            string query1 = "SELECT ID FROM Policyholders WHERE ID = @id";
            SqlCommand command1 = new SqlCommand(query1, con);
            command1.Parameters.Add(new SqlParameter("@id", policy.PolicyholderID));

            string query2 = "SELECT ID FROM Cars WHERE ID = @id";
            SqlCommand command2 = new SqlCommand(query2, con);
            command2.Parameters.Add(new SqlParameter("@id", policy.CarID));

            string query3 = "SELECT ID FROM Employees WHERE ID = @id";

```

Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

```

SqlCommand command3 = new SqlCommand(query3, con);
command3.Parameters.Add(new SqlParameter("@id", policy.EmployeeID));

string query = "BEGIN TRANSACTION " +
    "INSERT INTO Policies(InsuranceType, InsurancePremium, InsuranceAmount,
DateOfConclusion, ExpirationDate, PolicyholderID, CarID, EmployeeID) " +
    "VALUES(@insuranceType, " +
        "@insurancePremium, " +
        "@insuranceAmount, " +
        "@dateOfConclusion, " +
        "@expirationDate, " +
        "@policyholderID, " +
        "@carID, " +
        "@employeeID" +
    "); " +
    "DECLARE @id INT; " +
    "SET @id=SCOPE_IDENTITY(); " +
    "INSERT INTO Connections(PolicyID, PersonAllowedToDriveID) VALUES ";
for (var i = 0; i < listPersonAllowedToDrive.Count - 1; i++)
{
    query += "(@id, @personAllowedToDriveID" + i + "), ";
}
query += "(@id, @personAllowedToDriveID" + (listPersonAllowedToDrive.Count - 1) + "); " +
    "COMMIT TRANSACTION";

SqlCommand command = new SqlCommand(query, con);
command.Parameters.Add(new SqlParameter("@insuranceType", policy.InsuranceType));
command.Parameters.Add(new SqlParameter("@insurancePremium", policy.InsurancePremium));
command.Parameters.Add(new SqlParameter("@insuranceAmount", policy.InsuranceAmount));
command.Parameters.Add(new SqlParameter("@dateOfConclusion", policy.DateOfConclusion));
command.Parameters.Add(new SqlParameter("@expirationDate", policy.ExpirationDate));
command.Parameters.Add(new SqlParameter("@policyholderID", policy.PolicyholderID));
command.Parameters.Add(new SqlParameter("@carID", policy.CarID));
command.Parameters.Add(new SqlParameter("@employeeID", policy.EmployeeID));
for (var i = 0; i < listPersonAllowedToDrive.Count; i++)
{
    command.Parameters.Add(new SqlParameter("@personAllowedToDriveID" + i,
listPersonAllowedToDrive[i].ID));
}

con.Open();
SqlDataReader reader = command1.ExecuteReader();
if (!reader.HasRows)
{
    flag = true;
    throw new Exception("Данного страхователя нет");
}
reader.Close();

reader = command2.ExecuteReader();
if (!reader.HasRows)
{
    flag = true;
    throw new Exception("Данного автомобиля нет");
}
reader.Close();

reader = command3.ExecuteReader();
if (!reader.HasRows)
{
    flag = true;
}

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        throw new Exception("Данного сотрудника нет");
    }
    reader.Close();

    command.ExecuteNonQuery();
    con.Close();
}
}

catch (Exception exp)
{
    if (flag)
    {
        throw exp;
    }
    else
    {
        throw new Exception("Ошибка в работе БД");
    }
}
}

public static void AddPolicyholder(Policyholder policyholder)
{
    bool flag = false;
    try
    {
        using (SqlConnection con = new SqlConnection(ConnectionString))
        {
            string query1 = "SELECT ID FROM Policyholders WHERE Telephone = @telephone";
            SqlCommand command1 = new SqlCommand(query1, con);
            command1.Parameters.Add(new SqlParameter("@telephone", policyholder.Telephone));

            string query2 = "SELECT ID FROM Policyholders WHERE Passport = @passport";
            SqlCommand command2 = new SqlCommand(query2, con);
            command2.Parameters.Add(new SqlParameter("@passport", policyholder.Passport));

            string query = "INSERT INTO Policyholders (FullName, Birthday, Telephone, Passport) " +
                "VALUES(@fullName, @birthday, @telephone, @passport)";
            SqlCommand command = new SqlCommand(query, con);

            command.Parameters.Add(new SqlParameter("@fullName", policyholder.FullName));
            command.Parameters.Add(new SqlParameter("@birthday", policyholder.Birthday));
            command.Parameters.Add(new SqlParameter("@telephone", policyholder.Telephone));
            command.Parameters.Add(new SqlParameter("@passport", policyholder.Passport));

            con.Open();
            SqlDataReader reader = command1.ExecuteReader();
            if (reader.HasRows)
            {
                flag = true;
                throw new Exception("Данный телефон уже используется");
            }
            reader.Close();

            reader = command2.ExecuteReader();
            if (reader.HasRows)
            {
                flag = true;
                throw new Exception("Данный паспорт уже используется");
            }
            reader.Close();

            command.ExecuteNonQuery();
        }
    }
}

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        con.Close();
    }
}
catch (Exception exp)
{
    if (flag)
    {
        throw exp;
    }
    else
    {
        throw new Exception("Ошибка в работе БД");
    }
}
}

//Функции изменения
public static void ChangeCarWithPhotos(Car car, List<Photo> listDeletePhotos, List<string> listAddPhotos)
{
    try
    {
        using (SqlConnection con = new SqlConnection(ConnectionString))
        {
            string query = "BEGIN TRANSACTION " +
                "UPDATE Cars " +
                "SET RegistrationPlate = @registrationPlate, " +
                "VehiclePassport = @vehiclePassport " +
                "WHERE ID = @id ";
            if (listAddPhotos.Count != 0)
            {
                query += "INSERT INTO Photos(EncodedPhoto, CarID) VALUES ";
                for (int i = 0; i < listAddPhotos.Count - 1; i++)
                {
                    query += "(@photo" + i + ", @id), ";
                }
                query += "(@photo" + (listAddPhotos.Count - 1) + ", @id); ";
            }
            if (listDeletePhotos.Count != 0)
            {
                for (int i = 0; i < listDeletePhotos.Count; i++)
                {
                    query += "DELETE FROM Photos WHERE ID = @photoID" + i + ";";
                }
            }
            query += "COMMIT TRANSACTION";
            SqlCommand command = new SqlCommand(query, con);
            command.Parameters.Add(new SqlParameter("@id", car.ID));
            command.Parameters.Add(new SqlParameter("@registrationPlate", car.RegistrationPlate));
            command.Parameters.Add(new SqlParameter("@vehiclePassport", car.VehiclePassport));
            for (int i = 0; i < listAddPhotos.Count; i++)
            {
                command.Parameters.Add(new SqlParameter("@photo" + i, listAddPhotos[i]));
            }
            for (int i = 0; i < listDeletePhotos.Count; i++)
            {
                command.Parameters.Add(new SqlParameter("@photoID" + i, listDeletePhotos[i].ID));
            }
            con.Open();
            command.ExecuteNonQuery();
        }
    }
}

```

<i>Изм.</i>	<i>Лист</i>	<i>№ докум.</i>	<i>Подпись</i>	<i>Дата</i>

```

        con.Close();
    }
}
catch
{
    throw new Exception("Ошибка в работе БД");
}
}

public static void ChangeEmployee(Employee employee, bool changePassword)
{
    bool flag = false;
    try
    {
        using (SqlConnection con = new SqlConnection(ConnectionString))
        {
            string query1 = "SELECT ID FROM Employees WHERE Telephone = @telephone AND ID <> @id";
            SqlCommand command1 = new SqlCommand(query1, con);
            command1.Parameters.Add(new SqlParameter("@telephone", employee.Telephone));
            command1.Parameters.Add(new SqlParameter("@id", employee.ID));

            string query2 = "SELECT ID FROM Employees WHERE Passport = @passport AND ID <> @id";
            SqlCommand command2 = new SqlCommand(query2, con);
            command2.Parameters.Add(new SqlParameter("@passport", employee.Passport));
            command2.Parameters.Add(new SqlParameter("@id", employee.ID));

            string query3 = "SELECT ID FROM Employees WHERE Login = @login AND ID <> @id";
            SqlCommand command3 = new SqlCommand(query3, con);
            command3.Parameters.Add(new SqlParameter("@login", employee.Login));
            command3.Parameters.Add(new SqlParameter("@id", employee.ID));

            string query = "UPDATE Employees " +
                "SET FullName = @fullName, " +
                "Birthday = @birthday, " +
                "Telephone = @telephone, " +
                "Passport = @passport, " +
                "Login = @login, " +
                "Password = @password, " +
                "Admin = @admin, " +
                "Works = @works " +
                "WHERE ID = @id";

            SqlCommand command = new SqlCommand(query, con);

            command.Parameters.Add(new SqlParameter("@id", employee.ID));
            command.Parameters.Add(new SqlParameter("@fullName", employee.FullName));
            command.Parameters.Add(new SqlParameter("@birthday", employee.Birthday));
            command.Parameters.Add(new SqlParameter("@telephone", employee.Telephone));
            command.Parameters.Add(new SqlParameter("@passport", employee.Passport));
            command.Parameters.Add(new SqlParameter("@login", employee.Login));
            if (changePassword) command.Parameters.Add(new SqlParameter("@password",
GetHash(employee.Password)));
            else command.Parameters.Add(new SqlParameter("@password", employee.Password));
            command.Parameters.Add(new SqlParameter("@admin", employee.Admin));
            command.Parameters.Add(new SqlParameter("@works", employee.Works));

            con.Open();
            SqlDataReader reader = command1.ExecuteReader();
            if (reader.HasRows)
            {
                flag = true;
                throw new Exception("Данный телефон уже используется");
            }
        }
    }
}

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        reader.Close();

        reader = command2.ExecuteReader();
        if (reader.HasRows)
        {
            flag = true;
            throw new Exception("Данный паспорт уже используется");
        }
        reader.Close();

        reader = command3.ExecuteReader();
        if (reader.HasRows)
        {
            flag = true;
            throw new Exception("Данный логин уже занят");
        }
        reader.Close();

        command.ExecuteNonQuery();
        con.Close();
    }
}

catch (Exception exp)
{
    if (flag)
    {
        throw exp;
    }
    else
    {
        throw new Exception("Ошибка в работе БД");
    }
}
}

public static void ChangePersonAllowedToDrive(PersonAllowedToDrive personAllowedToDrive)
{
    bool flag = false;
    try
    {
        using (SqlConnection con = new SqlConnection(ConnectionString))
        {
            string query1 = "SELECT ID FROM PersonsAllowedToDrive WHERE DrivingLicence =
@drivingLicence AND ID <> @id";
            SqlCommand command1 = new SqlCommand(query1, con);
            command1.Parameters.Add(new SqlParameter("@drivingLicence",
personAllowedToDrive.DrivingLicence));
            command1.Parameters.Add(new SqlParameter("@id", personAllowedToDrive.ID));

            string query = "UPDATE PersonsAllowedToDrive " +
                "SET FullName = @fullName, " +
                "DrivingLicence = @drivingLicence " +
                "WHERE ID = @id";
            SqlCommand command = new SqlCommand(query, con);

            command.Parameters.Add(new SqlParameter("@id", personAllowedToDrive.ID));
            command.Parameters.Add(new SqlParameter("@fullName", personAllowedToDrive.FullName));
            command.Parameters.Add(new SqlParameter("@drivingLicence",
personAllowedToDrive.DrivingLicence));

            con.Open();
            SqlDataReader reader = command1.ExecuteReader();
            if (reader.HasRows)

```

Изм.	Лист	№ докум.	Подпись	Дата

```

    {
        flag = true;
        throw new Exception("Данное водительское удостоверение уже используется");
    }
    reader.Close();

    command.ExecuteNonQuery();
    con.Close();
}
}
catch (Exception exp)
{
    if (flag)
    {
        throw exp;
    }
    else
    {
        throw new Exception("Ошибка в работе БД");
    }
}
}

public static void ChangePolicyWithConnections(Policy policy, List<PersonAllowedToDrive>
listDeletePersons, List<PersonAllowedToDrive> listAddPersons)
{
    try
    {
        using (SqlConnection con = new SqlConnection(ConnectionString))
        {
            string query = "BEGIN TRANSACTION " +
                "UPDATE Policies " +
                "SET InsurancePremium = @insurancePremium, " +
                "ExpirationDate = @expirationDate " +
                "WHERE ID = @id; ";
            if (listAddPersons.Count != 0)
            {
                query += "INSERT INTO Connections(PolicyID, PersonAllowedToDriveID) VALUES ";
                for (int i = 0; i < listAddPersons.Count - 1; i++)
                {
                    query += "(@id, @addPersonID" + i + "), ";
                }
                query += "(@id, @addPersonID" + (listAddPersons.Count - 1) + "); ";
            }
            if (listDeletePersons.Count != 0)
            {
                for (int i = 0; i < listDeletePersons.Count; i++)
                {
                    query += "DELETE FROM Connections WHERE PolicyID = @id AND
PersonAllowedToDriveID = @deletePersonID" + i + ";";
                }
            }
            query += "COMMIT TRANSACTION";

            SqlCommand command = new SqlCommand(query, con);
            command.Parameters.Add(new SqlParameter("@id", policy.ID));
            command.Parameters.Add(new SqlParameter("@insurancePremium", policy.InsurancePremium));
            command.Parameters.Add(new SqlParameter("@expirationDate", policy.ExpirationDate));
            for (int i = 0; i < listAddPersons.Count; i++)
            {
                command.Parameters.Add(new SqlParameter("@addPersonID" + i, listAddPersons[i].ID));
            }
            for (int i = 0; i < listDeletePersons.Count; i++)

```

Изм.	Лист	№ докум.	Подпись	Дата

```

    {
        command.Parameters.Add(new SqlParameter("@deletePersonID" + i, listDeletePersons[i].ID));
    }

    con.Open();
    command.ExecuteNonQuery();
    con.Close();
}
}
catch
{
    throw new Exception("Ошибка в работе БД");
}
}

public static void ChangePolicyholder(Policyholder policyholder)
{
    bool flag = false;
    try
    {
        using (SqlConnection con = new SqlConnection(ConnectionString))
        {
            string query1 = "SELECT ID FROM Policyholders WHERE Telephone = @telephone AND ID <> @id";
            SqlCommand command1 = new SqlCommand(query1, con);
            command1.Parameters.Add(new SqlParameter("@telephone", policyholder.Telephone));
            command1.Parameters.Add(new SqlParameter("@id", policyholder.ID));

            string query2 = "SELECT ID FROM Policyholders WHERE Passport = @passport AND ID <> @id";
            SqlCommand command2 = new SqlCommand(query2, con);
            command2.Parameters.Add(new SqlParameter("@passport", policyholder.Passport));
            command2.Parameters.Add(new SqlParameter("@id", policyholder.ID));

            string query = "UPDATE Policyholders " +
                "SET FullName = @fullName, " +
                "Birthday = @birthday, " +
                "Telephone = @telephone, " +
                "Passport = @passport " +
                "WHERE ID = @id";
            SqlCommand command = new SqlCommand(query, con);

            command.Parameters.Add(new SqlParameter("@id", policyholder.ID));
            command.Parameters.Add(new SqlParameter("@fullName", policyholder.FullName));
            command.Parameters.Add(new SqlParameter("@birthday", policyholder.Birthday));
            command.Parameters.Add(new SqlParameter("@telephone", policyholder.Telephone));
            command.Parameters.Add(new SqlParameter("@passport", policyholder.Passport));

            con.Open();
            SqlDataReader reader = command1.ExecuteReader();
            if (reader.HasRows)
            {
                flag = true;
                throw new Exception("Данный телефон уже используется");
            }
            reader.Close();

            reader = command2.ExecuteReader();
            if (reader.HasRows)
            {
                flag = true;
                throw new Exception("Данный паспорт уже используется");
            }
            reader.Close();
        }
    }
}

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        command.ExecuteNonQuery();
        con.Close();
    }
}
catch (Exception exp)
{
    if (flag)
    {
        throw exp;
    }
    else
    {
        throw new Exception("Ошибка в работе БД");
    }
}
}

```

```

//Функции удаления
public static void DeleteCar(int id)
{
    bool flag = false;
    try
    {
        using (SqlConnection con = new SqlConnection(ConnectionString))
        {
            string query1 = "SELECT ID FROM Policies WHERE CarID = @carID";
            SqlCommand command1 = new SqlCommand(query1, con);
            command1.Parameters.Add(new SqlParameter("@carID", id));

            string query = "DELETE FROM Cars WHERE ID = @id";
            SqlCommand command = new SqlCommand(query, con);
            command.Parameters.Add(new SqlParameter("@id", id));

            con.Open();
            SqlDataReader reader = command1.ExecuteReader();
            if (reader.HasRows)
            {
                flag = true;
                throw new Exception("Вы не можете удалить данный автомобиль, так как на него оформлен
полис");
            }
            reader.Close();

            command.ExecuteNonQuery();
            con.Close();
        }
    }
    catch (Exception exp)
    {
        if (flag)
        {
            throw exp;
        }
        else
        {
            throw new Exception("Ошибка в работе БД");
        }
    }
}

```

Изм.	Лист	№ докум.	Подпись	Дата

```

public static void DeleteEmployee(int id)
{
    bool flag = false;
    try
    {
        using (SqlConnection con = new SqlConnection(ConnectionString))
        {
            string query1 = "SELECT ID FROM Policies WHERE EmployeeID = @employeeID";
            SqlCommand command1 = new SqlCommand(query1, con);
            command1.Parameters.Add(new SqlParameter("@employeeID", id));

            string query = "DELETE FROM Employees WHERE ID = @id";
            SqlCommand command = new SqlCommand(query, con);
            command.Parameters.Add(new SqlParameter("@id", id));

            con.Open();
            SqlDataReader reader = command1.ExecuteReader();
            if (reader.HasRows)
            {
                flag = true;
                throw new Exception("Вы не можете удалить данного сотрудника, так как он оформил полис");
            }
            reader.Close();

            command.ExecuteNonQuery();
            con.Close();
        }
    }
    catch (Exception exp)
    {
        if (flag)
        {
            throw exp;
        }
        else
        {
            throw new Exception("Ошибка в работе БД");
        }
    }
}

public static void DeletePersonAllowedToDrive(int id)
{
    bool flag = false;
    try
    {
        using (SqlConnection con = new SqlConnection(ConnectionString))
        {
            string query1 = "SELECT * FROM Connections WHERE PersonAllowedToDriveID = @personAllowedToDriveID";
            SqlCommand command1 = new SqlCommand(query1, con);
            command1.Parameters.Add(new SqlParameter("@personAllowedToDriveID", id));

            string query = "DELETE FROM PersonsAllowedToDrive WHERE ID = @id";
            SqlCommand command = new SqlCommand(query, con);
            command.Parameters.Add(new SqlParameter("@id", id));

            con.Open();
            SqlDataReader reader = command1.ExecuteReader();
            if (reader.HasRows)
            {
                flag = true;
            }
        }
    }
}

```

Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

```

        throw new Exception("Вы не можете удалить данного водителя, так как на него оформлен
полис");
    }
    reader.Close();

    command.ExecuteNonQuery();
    con.Close();
}
}

catch (Exception exp)
{
    if (flag)
    {
        throw exp;
    }
    else
    {
        throw new Exception("Ошибка в работе БД");
    }
}
}

public static void DeletePolicyholder(int id)
{
    bool flag = false;
    try
    {
        using (SqlConnection con = new SqlConnection(ConnectionString))
        {
            string query1 = "SELECT ID FROM Policies WHERE PolicyholderID = @policyholderID";
            SqlCommand command1 = new SqlCommand(query1, con);
            command1.Parameters.Add(new SqlParameter("@policyholderID", id));

            string query = "DELETE FROM Policyholders WHERE ID = @id";
            SqlCommand command = new SqlCommand(query, con);

            command.Parameters.Add(new SqlParameter("@id", id));

            con.Open();
            SqlDataReader reader = command1.ExecuteReader();
            if (reader.HasRows)
            {
                flag = true;
                throw new Exception("Вы не можете удалить данного страхователя, так как он оформил
полис");
            }
            reader.Close();

            command.ExecuteNonQuery();
            con.Close();
        }
    }
    catch (Exception exp)
    {
        if (flag)
        {
            throw exp;
        }
        else
        {
            throw new Exception("Ошибка в работе БД");
        }
    }
}

```

Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

```

}

//Функции поиска
public static Car SearchCar(string vin)
{
    bool flag = false;
    try
    {
        using (SqlConnection con = new SqlConnection(ConnectionString))
        {
            string query = "SELECT * FROM Cars WHERE VIN = @vin";
            SqlCommand command = new SqlCommand(query, con);
            command.Parameters.Add(new SqlParameter("@vin", vin));

            con.Open();
            SqlDataReader reader = command.ExecuteReader();

            Car car;
            if (!reader.HasRows)
            {
                flag = true;
                throw new Exception("Данный автомобиль не существует");
            }
            while (reader.Read())
            {
                car = new Car(Convert.ToInt32(reader["ID"].ToString()),
                    reader["Model"].ToString(),
                    reader["VIN"].ToString(),
                    reader["RegistrationPlate"].ToString(),
                    reader["VehiclePassport"].ToString());
                reader.Close();
                con.Close();
                return car;
            }
            return null;
        }
    catch (Exception exp)
    {
        if (flag)
        {
            throw exp;
        }
        else
        {
            throw new Exception("Ошибка в работе БД");
        }
    }
}

public static Car SearchCarID(int id)
{
    bool flag = false;
    try
    {
        using (SqlConnection con = new SqlConnection(ConnectionString))
        {
            string query = "SELECT * FROM Cars WHERE ID = @id";
            SqlCommand command = new SqlCommand(query, con);
            command.Parameters.Add(new SqlParameter("@id", id));

```

Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

```

con.Open();
SqlDataReader reader = command.ExecuteReader();

Car car;
if (!reader.HasRows)
{
    flag = true;
    throw new Exception("Данный автомобиль не существует");
}
while (reader.Read())
{
    car = new Car(Convert.ToInt32(reader["ID"].ToString()),
                  reader["Model"].ToString(),
                  reader["VIN"].ToString(),
                  reader["RegistrationPlate"].ToString(),
                  reader["VehiclePassport"].ToString());
    reader.Close();
    con.Close();
    return car;
}
return null;
}
catch (Exception exp)
{
    if (flag)
    {
        throw exp;
    }
    else
    {
        throw new Exception("Ошибка в работе БД");
    }
}
}

```

```

public static List<Connection> SearchConnection(int policyID)
{
    try
    {
        using (SqlConnection con = new SqlConnection(ConnectionString))
        {
            var list = new List<Connection>();

            string query = "SELECT * FROM Connections WHERE PolicyID = @policyID";
            SqlCommand command = new SqlCommand(query, con);
            command.Parameters.Add(new SqlParameter("@policyID", policyID));

            con.Open();
            SqlDataReader reader = command.ExecuteReader();

            while (reader.Read())
            {
                var connection = new Connection(Convert.ToInt32(reader["PolicyID"].ToString()),
                                                Convert.ToInt32(reader["PersonAllowedToDriveID"].ToString()));
                list.Add(connection);
            }
            reader.Close();
            con.Close();
            return list;
        }
    }
    catch

```

Изм.	Лист	№ докум.	Подпись	Дата

```

    {
        throw new Exception("Ошибка в работе БД");
    }
}

public static Employee SearchEmployee(string telephoneOrPassport)
{
    bool flag = false;
    try
    {
        using (SqlConnection con = new SqlConnection(ConnectionString))
        {
            string query = "SELECT * FROM Employees WHERE Telephone = @telephoneOrPassport OR
Passport = @telephoneOrPassport";
            SqlCommand command = new SqlCommand(query, con);
            command.Parameters.Add(new SqlParameter("@telephoneOrPassport", telephoneOrPassport));

            con.Open();
            SqlDataReader reader = command.ExecuteReader();

            Employee employee;
            if (!reader.HasRows)
            {
                flag = true;
                throw new Exception("Данный сотрудник не существует");
            }
            while (reader.Read())
            {
                employee = new Employee(Convert.ToInt32(reader["ID"].ToString()),
                    reader["FullName"].ToString(),
                    Convert.ToDateTime(reader["Birthday"].ToString()),
                    reader["Telephone"].ToString(),
                    reader["Passport"].ToString(),
                    reader["Login"].ToString(),
                    reader["Password"].ToString(),
                    Convert.ToBoolean(reader["Admin"].ToString()),
                    Convert.ToBoolean(reader["Works"].ToString()));
                reader.Close();
                con.Close();
                return employee;
            }
            return null;
        }
    }
    catch (Exception exp)
    {
        if (flag)
        {
            throw exp;
        }
        else
        {
            throw new Exception("Ошибка в работе БД");
        }
    }
}

public static Employee SearchEmployeeID(int id)
{
    bool flag = false;
    try
    {
        using (SqlConnection con = new SqlConnection(ConnectionString))

```

Изм.	Лист	№ докум.	Подпись	Дата

```

{
    string query = "SELECT * FROM Employees WHERE ID = @id";
    SqlCommand command = new SqlCommand(query, con);
    command.Parameters.Add(new SqlParameter("@id", id));

    con.Open();
    SqlDataReader reader = command.ExecuteReader();

    Employee employee;
    if (!reader.HasRows)
    {
        flag = true;
        throw new Exception("Данный сотрудник не существует");
    }
    while (reader.Read())
    {
        employee = new Employee(Convert.ToInt32(reader["ID"].ToString()),
            reader["FullName"].ToString(),
            Convert.ToDateTime(reader["Birthday"].ToString()),
            reader["Telephone"].ToString(),
            reader["Passport"].ToString(),
            reader["Login"].ToString(),
            reader["Password"].ToString(),
            Convert.ToBoolean(reader["Admin"].ToString()),
            Convert.ToBoolean(reader["Works"].ToString()));

        reader.Close();
        con.Close();
        return employee;
    }
    return null;
}
catch (Exception exp)
{
    if (flag)
    {
        throw exp;
    }
    else
    {
        throw new Exception("Ошибка в работе БД");
    }
}
}

public static Employee SearchEmployeeLogin(string login)
{
    bool flag = false;
    try
    {
        using (SqlConnection con = new SqlConnection(ConnectionString))
        {
            string query = "SELECT * FROM Employees WHERE Login = @login";
            SqlCommand command = new SqlCommand(query, con);
            command.Parameters.Add(new SqlParameter("@login", login));

            con.Open();
            SqlDataReader reader = command.ExecuteReader();

            Employee employee;
            if (!reader.HasRows)
            {
                flag = true;

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        throw new Exception("Данный сотрудник не существует");
    }
    while (reader.Read())
    {
        employee = new Employee(Convert.ToInt32(reader["ID"].ToString()),
            reader["FullName"].ToString(),
            Convert.ToDateTime(reader["Birthday"].ToString()),
            reader["Telephone"].ToString(),
            reader["Passport"].ToString(),
            reader["Login"].ToString(),
            reader["Password"].ToString(),
            Convert.ToBoolean(reader["Admin"].ToString()),
            Convert.ToBoolean(reader["Works"].ToString()));

        reader.Close();
        con.Close();
        return employee;
    }
    return null;
}
catch (Exception exp)
{
    if (flag)
    {
        throw exp;
    }
    else
    {
        throw new Exception("Ошибка в работе БД");
    }
}
}

public static List<InsuranceEvent> SearchInsuranceEvent(int policyID)
{
    try
    {
        using (SqlConnection con = new SqlConnection(ConnectionString))
        {
            var list = new List<InsuranceEvent>();

            string query = "SELECT * FROM InsuranceEvents WHERE PolicyID = @policyID ORDER BY Date";
            SqlCommand command = new SqlCommand(query, con);
            command.Parameters.Add(new SqlParameter("@policyID", policyID));

            con.Open();
            SqlDataReader reader = command.ExecuteReader();

            while (reader.Read())
            {
                var insuranceEvent = new InsuranceEvent(Convert.ToInt32(reader["ID"].ToString()),
                    Convert.ToDateTime(reader["Date"].ToString()),
                    Convert.ToInt32(reader["InsurancePayment"].ToString()),
                    Convert.ToInt32(reader["PolicyID"].ToString()));

                list.Add(insuranceEvent);
            }
            reader.Close();
            con.Close();
            return list;
        }
    }
    catch

```

Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

```

        {
            throw new Exception("Ошибка в работе БД");
        }
    }

    public static DateTime SearchInsuranceEventMaxDate(int policyID)
    {
        try
        {
            using (SqlConnection con = new SqlConnection(ConnectionString))
            {
                string query = "SELECT MAX(Date) FROM InsuranceEvents WHERE PolicyID = @policyID";
                SqlCommand command = new SqlCommand(query, con);
                command.Parameters.Add(new SqlParameter("@policyID", policyID));

                con.Open();
                var temp = command.ExecuteScalar().ToString();
                if (temp == String.Empty)
                {
                    return default;
                }
                DateTime dateMax = Convert.ToDateTime(temp);
                con.Close();
                return dateMax;
            }
        }
        catch
        {
            throw new Exception("Ошибка в работе БД");
        }
    }

    public static PersonAllowedToDrive SearchPersonAllowedToDrive(string drivingLicence)
    {
        bool flag = false;
        try
        {
            using (SqlConnection con = new SqlConnection(ConnectionString))
            {
                string query = "SELECT * FROM PersonsAllowedToDrive WHERE DrivingLicence = @drivingLicence";
                SqlCommand command = new SqlCommand(query, con);
                command.Parameters.Add(new SqlParameter("@drivingLicence", drivingLicence));

                con.Open();
                SqlDataReader reader = command.ExecuteReader();

                PersonAllowedToDrive personAllowedToDrive;
                if (!reader.HasRows)
                {
                    flag = true;
                    throw new Exception("Данный водитель не существует");
                }
                while (reader.Read())
                {
                    personAllowedToDrive = new PersonAllowedToDrive(Convert.ToInt32(reader["ID"].ToString()),
                                                                    reader["FullName"].ToString(),
                                                                    reader["DrivingLicence"].ToString());
                    reader.Close();
                    con.Close();
                    return personAllowedToDrive;
                }
            }
            return null;
        }
    }
}

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        }
    }
    catch (Exception exp)
    {
        if (flag)
        {
            throw exp;
        }
        else
        {
            throw new Exception("Ошибка в работе БД");
        }
    }
}

public static PersonAllowedToDrive SearchPersonAllowedToDriveID(int id)
{
    bool flag = false;
    try
    {
        using (SqlConnection con = new SqlConnection(ConnectionString))
        {
            string query = "SELECT * FROM PersonsAllowedToDrive WHERE ID = @id";
            SqlCommand command = new SqlCommand(query, con);
            command.Parameters.Add(new SqlParameter("@id", id));

            con.Open();
            SqlDataReader reader = command.ExecuteReader();

            PersonAllowedToDrive personAllowedToDrive;
            if (!reader.HasRows)
            {
                flag = true;
                throw new Exception("Данный водитель не существует");
            }
            while (reader.Read())
            {
                personAllowedToDrive = new PersonAllowedToDrive(Convert.ToInt32(reader["ID"].ToString()),
                                                               reader["FullName"].ToString(),
                                                               reader["DrivingLicence"].ToString());
                reader.Close();
                con.Close();
                return personAllowedToDrive;
            }
            return null;
        }
    }
    catch (Exception exp)
    {
        if (flag)
        {
            throw exp;
        }
        else
        {
            throw new Exception("Ошибка в работе БД");
        }
    }
}

public static List<Photo> SearchPhoto(int carID)
{
    try

```

Изм.	Лист	№ докум.	Подпись	Дата

```

{
    using (SqlConnection con = new SqlConnection(ConnectionString))
    {
        var list = new List<Photo>();

        string query = "SELECT * FROM Photos WHERE CarID = @carID";
        SqlCommand command = new SqlCommand(query, con);
        command.Parameters.Add(new SqlParameter("@carID", carID));

        con.Open();
        SqlDataReader reader = command.ExecuteReader();

        while (reader.Read())
        {
            var policy = new Photo(Convert.ToInt32(reader["ID"].ToString()),
                reader["EncodedPhoto"].ToString(),
                Convert.ToInt32(reader["CarID"].ToString()));
            list.Add(policy);
        }
        reader.Close();
        con.Close();
        return list;
    }
}
catch
{
    throw new Exception("Ошибка в работе БД");
}
}

public static List<Policy> SearchPolicy(int policyholderID)
{
    try
    {
        using (SqlConnection con = new SqlConnection(ConnectionString))
        {
            var list = new List<Policy>();

            string query = "SELECT * FROM Policies WHERE PolicyholderID = @policyholderID ORDER BY DateOfConclusion DESC";
            SqlCommand command = new SqlCommand(query, con);
            command.Parameters.Add(new SqlParameter("@policyholderID", policyholderID));

            con.Open();
            SqlDataReader reader = command.ExecuteReader();

            while (reader.Read())
            {
                var policy = new Policy(Convert.ToInt32(reader["ID"].ToString()),
                    reader["InsuranceType"].ToString(),
                    Convert.ToInt32(reader["InsurancePremium"].ToString()),
                    Convert.ToInt32(reader["InsuranceAmount"].ToString()),
                    Convert.ToDateTime(reader["DateOfConclusion"].ToString()),
                    Convert.ToDateTime(reader["ExpirationDate"].ToString()),
                    Convert.ToInt32(reader["PolicyholderID"].ToString()),
                    Convert.ToInt32(reader["CarID"].ToString()),
                    Convert.ToInt32(reader["EmployeeID"].ToString()));

                list.Add(policy);
            }
            reader.Close();
            con.Close();
            return list;
        }
    }
}

```

Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

```

        }
    catch
    {
        throw new Exception("Ошибка в работе БД");
    }
}

public static Policy SearchPolicyID(int id)
{
    try
    {
        using (SqlConnection con = new SqlConnection(ConnectionString))
        {
            var list = new List<Policy>();

            string query = "SELECT * FROM Policies WHERE ID = @id";
            SqlCommand command = new SqlCommand(query, con);
            command.Parameters.Add(new SqlParameter("@id", id));

            con.Open();
            SqlDataReader reader = command.ExecuteReader();

            while (reader.Read())
            {
                var policy = new Policy(Convert.ToInt32(reader["ID"].ToString()),
                    reader["InsuranceType"].ToString(),
                    Convert.ToInt32(reader["InsurancePremium"].ToString()),
                    Convert.ToInt32(reader["InsuranceAmount"].ToString()),
                    Convert.ToDateTime(reader["DateOfConclusion"].ToString()),
                    Convert.ToDateTime(reader["ExpirationDate"].ToString()),
                    Convert.ToInt32(reader["PolicyholderID"].ToString()),
                    Convert.ToInt32(reader["CarID"].ToString()),
                    Convert.ToInt32(reader["EmployeeID"].ToString()));

                reader.Close();
                con.Close();
                return policy;
            }
            return null;
        }
    catch
    {
        throw new Exception("Ошибка в работе БД");
    }
}

public static Policyholder SearchPolicyholder(string telephoneOrPassport)
{
    bool flag = false;
    try
    {
        using (SqlConnection con = new SqlConnection(ConnectionString))
        {
            string query = "SELECT * FROM Policyholders WHERE Telephone = @telephoneOrPassport OR
Passport = @telephoneOrPassport";
            SqlCommand command = new SqlCommand(query, con);
            command.Parameters.Add(new SqlParameter("@telephoneOrPassport", telephoneOrPassport));

            con.Open();
            SqlDataReader reader = command.ExecuteReader();

            Policyholder policyholder;
            if (!reader.HasRows)

```

Изм.	Лист	№ докум.	Подпись	Дата

```

    {
        flag = true;
        throw new Exception("Данный страхователь не существует");
    }
    while (reader.Read())
    {
        policyholder = new Policyholder(Convert.ToInt32(reader["ID"].ToString()),
                                         reader["FullName"].ToString(),
                                         Convert.ToDateTime(reader["Birthday"].ToString()),
                                         reader["Telephone"].ToString(),
                                         reader["Passport"].ToString());
        reader.Close();
        con.Close();
        return policyholder;
    }
    return null;
}
catch (Exception exp)
{
    if (flag)
    {
        throw exp;
    }
    else
    {
        throw new Exception("Ошибка в работе БД");
    }
}
}

public static Policyholder SearchPolicyholderID(int id)
{
    bool flag = false;
    try
    {
        using (SqlConnection con = new SqlConnection(ConnectionString))
        {
            string query = "SELECT * FROM Policyholders WHERE ID = @id";
            SqlCommand command = new SqlCommand(query, con);
            command.Parameters.Add(new SqlParameter("@id", id));

            con.Open();
            SqlDataReader reader = command.ExecuteReader();

            Policyholder policyholder;
            if (!reader.HasRows)
            {
                flag = true;
                throw new Exception("Данный страхователь не существует");
            }
            while (reader.Read())
            {
                policyholder = new Policyholder(Convert.ToInt32(reader["ID"].ToString()),
                                                 reader["FullName"].ToString(),
                                                 Convert.ToDateTime(reader["Birthday"].ToString()),
                                                 reader["Telephone"].ToString(),
                                                 reader["Passport"].ToString());
                reader.Close();
                con.Close();
                return policyholder;
            }
        }
        return null;
    }
}

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        }
    }
    catch (Exception exp)
    {
        if (flag)
        {
            throw exp;
        }
        else
        {
            throw new Exception("Ошибка в работе БД");
        }
    }
}

//Функции получения всего списка
public static List<Car> AllCars()
{
    try
    {
        using (SqlConnection con = new SqlConnection(ConnectionString))
        {
            var list = new List<Car>();

            string query = "SELECT ID, Model, VIN, RegistrationPlate, VehiclePassport FROM Cars";
            SqlCommand command = new SqlCommand(query, con);

            con.Open();
            SqlDataReader reader = command.ExecuteReader();

            while (reader.Read())
            {
                var car = new Car(Convert.ToInt32(reader["ID"].ToString()),
                    reader["Model"].ToString(),
                    reader["VIN"].ToString(),
                    reader["RegistrationPlate"].ToString(),
                    reader["VehiclePassport"].ToString());
                list.Add(car);
            }
            reader.Close();
            con.Close();
            return list;
        }
    }
    catch
    {
        throw new Exception("Ошибка в работе БД");
    }
}

public static List<Employee> AllEmployees()
{
    try
    {
        using (SqlConnection con = new SqlConnection(ConnectionString))
        {
            var list = new List<Employee>();

            string query = "SELECT * FROM Employees ORDER BY Works DESC, FullName";
            SqlCommand command = new SqlCommand(query, con);

```

Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

```

        con.Open();
        SqlDataReader reader = command.ExecuteReader();

        while (reader.Read())
        {
            var employee = new Employee(Convert.ToInt32(reader["ID"].ToString()),
                reader["FullName"].ToString(),
                Convert.ToDateTime(reader["Birthday"].ToString()),
                reader["Telephone"].ToString(),
                reader["Passport"].ToString(),
                reader["Login"].ToString(),
                reader["Password"].ToString(),
                Convert.ToBoolean(reader["Admin"].ToString()),
                Convert.ToBoolean(reader["Works"].ToString()));

            list.Add(employee);
        }
        reader.Close();
        con.Close();
        return list;
    }
}
catch
{
    throw new Exception("Ошибка в работе БД");
}
}

public static List<PersonAllowedToDrive> AllPersonsAllowedToDrive()
{
    try
    {
        using (SqlConnection con = new SqlConnection(ConnectionString))
        {
            var list = new List<PersonAllowedToDrive>();

            string query = "SELECT * FROM PersonsAllowedToDrive ORDER BY FullName";
            SqlCommand command = new SqlCommand(query, con);

            con.Open();
            SqlDataReader reader = command.ExecuteReader();

            while (reader.Read())
            {
                var personAllowedToDrive = new
                    PersonAllowedToDrive(Convert.ToInt32(reader["ID"].ToString()),
                        reader["FullName"].ToString(),
                        reader["DrivingLicence"].ToString());

                list.Add(personAllowedToDrive);
            }
            reader.Close();
            con.Close();
            return list;
        }
    }
    catch
    {
        throw new Exception("Ошибка в работе БД");
    }
}

public static List<Policy> AllPolicies()
{
    try

```

Изм.	Лист	№ докум.	Подпись	Дата

```

{
    using (SqlConnection con = new SqlConnection(ConnectionString))
    {
        var list = new List<Policy>();

        string query = "SELECT * FROM Policies";
        SqlCommand command = new SqlCommand(query, con);

        con.Open();
        SqlDataReader reader = command.ExecuteReader();

        while (reader.Read())
        {
            var policy = new Policy(Convert.ToInt32(reader["ID"].ToString()),
                reader["InsuranceType"].ToString(),
                Convert.ToInt32(reader["InsurancePremium"].ToString()),
                Convert.ToInt32(reader["InsuranceAmount"].ToString()),
                Convert.ToDateTime(reader["DateOfConclusion"].ToString()),
                Convert.ToDateTime(reader["ExpirationDate"].ToString()),
                Convert.ToInt32(reader["PolicyholderID"].ToString()),
                Convert.ToInt32(reader["CarID"].ToString()),
                Convert.ToInt32(reader["EmployeeID"].ToString()));
            list.Add(policy);
        }
        reader.Close();
        con.Close();
        return list;
    }
}
catch
{
    throw new Exception("Ошибка в работе БД");
}
}

public static List<Policyholder> AllPolicyholders()
{
    try
    {
        using (SqlConnection con = new SqlConnection(ConnectionString))
        {
            var list = new List<Policyholder>();

            string query = "SELECT * FROM Policyholders ORDER BY FullName";
            SqlCommand command = new SqlCommand(query, con);

            con.Open();
            SqlDataReader reader = command.ExecuteReader();

            while (reader.Read())
            {
                var policyholder = new Policyholder(Convert.ToInt32(reader["ID"].ToString()),
                    reader["FullName"].ToString(),
                    Convert.ToDateTime(reader["Birthday"].ToString()),
                    reader["Telephone"].ToString(),
                    reader["Passport"].ToString());
                list.Add(policyholder);
            }
            reader.Close();
            con.Close();
            return list;
        }
    }
}

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        catch
    {
        throw new Exception("Ошибка в работе БД");
    }
}

//Отчёты
public static (int, int, int) Reports(string insuranceType, DateTime dateStart, DateTime dateEnd)
{
    try
    {
        using (SqlConnection con = new SqlConnection(ConnectionString))
        {
            string query;
            SqlCommand command;
            if (insuranceType == "ОСАГО и КАСКО")
            {
                query = "SELECT (SELECT SUM(InsurancePayment) " +
                    "FROM InsuranceEvents " +
                    "WHERE Date >= @dateStart AND Date <= @dateEnd), " +

                    "COUNT(ID), SUM(InsurancePremium) " +
                    "FROM Policies " +
                    "WHERE DateOfConclusion >= @dateStart AND DateOfConclusion <= @dateEnd";
                command = new SqlCommand(query, con);
                command.Parameters.Add(new SqlParameter("@dateStart", dateStart));
                command.Parameters.Add(new SqlParameter("@dateEnd", dateEnd));
            }
            else
            {
                query = "SELECT (SELECT SUM(InsurancePayment) " +
                    "FROM Policies as p LEFT JOIN InsuranceEvents as ie ON p.ID = ie.PolicyID " +
                    "WHERE p.InsuranceType = @insuranceType AND Date >= @dateStart AND Date <=
@dateEnd), " +

                    "COUNT(ID), SUM(InsurancePremium) " +
                    "FROM Policies " +
                    "WHERE InsuranceType = @insuranceType AND DateOfConclusion >= @dateStart AND
DateOfConclusion <= @dateEnd";
                command = new SqlCommand(query, con);
                command.Parameters.Add(new SqlParameter("@insuranceType", insuranceType));
                command.Parameters.Add(new SqlParameter("@dateStart", dateStart));
                command.Parameters.Add(new SqlParameter("@dateEnd", dateEnd));
            }
        }

        con.Open();
        SqlDataReader reader = command.ExecuteReader();

        int countContracts = 0, sumContracts = 0, sumInsuranceEvents = 0;
        while (reader.Read())
        {
            string tempSumInsuranceEvents = reader.GetValue(0).ToString();
            if (tempSumInsuranceEvents == String.Empty) tempSumInsuranceEvents = "0";
            string tempCountContracts = reader.GetValue(1).ToString();
            if (tempCountContracts == String.Empty) tempCountContracts = "0";
            string tempSumContracts = reader.GetValue(2).ToString();
            if (tempSumContracts == String.Empty) tempSumContracts = "0";

            countContracts = Convert.ToInt32(tempCountContracts);
            sumContracts = Convert.ToInt32(tempSumContracts);
            sumInsuranceEvents = Convert.ToInt32(tempSumInsuranceEvents);
        }
    }
}

```

Изм.	Лист	№ докум.	Подпись	Дата

```
        }
        reader.Close();
        var tuple = (countContracts, sumContracts, sumInsuranceEvents);
        con.Close();
        return tuple;
    }
}
catch
{
    throw new Exception("Ошибка в работе БД");
}
}
```

Листинг класса AddCar:

```

namespace InsuranceAgency.Pages
{
    public partial class AddCar : Page
    {
        List<BitmapImage> listPhotos = new List<BitmapImage>();
        List<string> listEncodedPhotos = new List<string>();
        int currentIndex = 0;

        public AddCar()
        {
            InitializeComponent();
        }

        private void tbVehiclePassportSeries_TextChanged(object sender, TextChangedEventArgs e)
        {
            if (tbVehiclePassportSeries.Text.Length == 0)
            {
                tbVehiclePassportSeriesHint.Visibility = Visibility.Visible;
            }
            else
            {
                tbVehiclePassportSeriesHint.Visibility = Visibility.Hidden;
            }
        }

        private void tbVehiclePassportNumber_TextChanged(object sender, TextChangedEventArgs e)
        {
            if (tbVehiclePassportNumber.Text.Length == 0)
            {
                tbVehiclePassportNumberHint.Visibility = Visibility.Visible;
            }
            else
            {
                tbVehiclePassportNumberHint.Visibility = Visibility.Hidden;
            }
        }

        private void btnAddImage_Click(object sender, RoutedEventArgs e)
        {
            OpenFileDialog openFileDialog = new OpenFileDialog();
            openFileDialog.Filter = "All Embroidery Files | *.bmp; *.gif; *.jpeg; *.jpg; " +
                " *.fif; *.fiff; *.png; *.wmf; *.emf" +
                "|Windows Bitmap (*.bmp)|*.bmp" +

```

<i>Изм.</i>	<i>Лист</i>	<i>№ докум.</i>	<i>Подпись</i>	<i>Дата</i>

```

    "|JPEG File Interchange Format (*.jpg)|*.jpg;*.jpeg" +
    "|Graphics Interchange Format (*.gif)|*.gif" +
    "|Portable Network Graphics (*.png)|*.png" +
    "|Tag Embroidery File Format (*.tif)|*.tif;*.tiff";
if (openFileDialog.ShowDialog() == true)
{
    var size = new FileInfo(openFileDialog.FileName).Length;
    if (size / 1024 > 512)
    {
        tbException.Visibility = Visibility.Visible;
        tbException.Text = "Размер фото не может быть больше 512Кб";
    }
    else
    {
        BitmapImage bi = new BitmapImage(new Uri(openFileDialog.FileName));
        imgCar.Source = bi;
        listPhotos.Add(bi);
        currentIndex = listPhotos.Count - 1;
        if (currentIndex == 1)
        {
            btnLeft.Visibility = Visibility.Visible;
            btnRight.Visibility = Visibility.Visible;
        }

        string image = DBImage.Encode(bi);
        listEncodedPhotos.Add(image);
    }
}
}

private void btnAddCar_Click(object sender, RoutedEventArgs e)
{
    try
    {
        string model = tbModel.Text.Trim();
        if (model == "")
        {
            throw new Exception("Заполните поле Модель");
        }

        string vin = tbVIN.Text.Trim();
        if (vin.Length != 17)
        {
            throw new Exception("VIN номер должен содержать 17 знаков");
        }
        foreach (var item in vin)
        {
            if (!char.IsDigit(item) && !(Convert.ToInt32(item) >= 65 && Convert.ToInt32(item) <= 90))
            {
                throw new Exception("VIN номер должен состоять из цифр и заглавных латинских букв");
            }
        }

        string registrationPlate = tbRegistrationPlate.Text.Trim();
        if (registrationPlate == "")
        {
            throw new Exception("Заполните поле Регистрационный знак");
        }

        string vehiclePassportSeries = tbVehiclePassportSeries.Text.Trim();
        if (vehiclePassportSeries.Length != 4)
        {
            throw new Exception("Серия паспорта должна содержать 4 знака");
        }
    }
}

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        }
        for(var i = 0; i < vehiclePassportSeries.Length; i++)
        {
            if ((i == 0 || i == 1) && !char.IsDigit(vehiclePassportSeries[i]))
            {
                throw new Exception("Первые два символа серии паспорта ТС должны быть цифрами");
            }
            if ((i == 3 || i == 4) && !char.IsLetter(vehiclePassportSeries[i]))
            {
                throw new Exception("Последние два символа серии паспорта ТС должны быть буквами");
            }
        }
        string vehiclePassportNumber = tbVehiclePassportNumber.Text.Trim();
        if (vehiclePassportNumber.Length != 6)
        {
            throw new Exception("Номер паспорта ТС должен содержать 6 цифр");
        }
        foreach (var item in vehiclePassportNumber)
        {
            if (!char.IsDigit(item))
            {
                throw new Exception("Номер паспорта ТС должен содержать только цифры");
            }
        }
        string vehiclePassport = vehiclePassportSeries + vehiclePassportNumber;

        if (listEncodedPhotos.Count == 0)
        {
            throw new Exception("Добавьте фотографию автомобиля");
        }

        Car car = new Car(model, vin, registrationPlate, vehiclePassport);

        Database.AddCarWithPhotos(car, listEncodedPhotos);

        MessageBox.Show("Автомобиль успешно добавлен", "", MessageBoxButtons.OK,
        MessageBoxIcon.Information);

        tbModel.Clear();
        tbVIN.Clear();
        tbRegistrationPlate.Clear();
        tbVehiclePassportSeries.Clear();
        tbVehiclePassportNumber.Clear();

        listPhotos.Clear();
        listEncodedPhotos.Clear();
        BitmapImage bi = new BitmapImage();
        bi.BeginInit();
        bi.UriSource = new Uri("/InsuranceAgency;component/Assets/Car.jpg", UriKind.RelativeOrAbsolute);
        bi.EndInit();
        imgCar.Source = bi;
        btnLeft.Visibility = Visibility.Hidden;
        btnRight.Visibility = Visibility.Hidden;

        tbException.Visibility = Visibility.Hidden;
    }
    catch (Exception exp)
    {
        tbException.Visibility = Visibility.Visible;
        tbException.Text = exp.Message;
    }
}

```

Изм.	Лист	№ докум.	Подпись	Дата

```

private void btnLeft_Click(object sender, RoutedEventArgs e)
{
    if(currentIndex == 0)
    {
        currentIndex = listPhotos.Count - 1;
    }
    else
    {
        currentIndex--;
    }

    imgCar.Source = listPhotos[currentIndex];
}

private void btnRight_Click(object sender, RoutedEventArgs e)
{
    if (currentIndex == listPhotos.Count - 1)
    {
        currentIndex = 0;
    }
    else
    {
        currentIndex++;
    }

    imgCar.Source = listPhotos[currentIndex];
}

private void btnDeleteImage_Click(object sender, RoutedEventArgs e)
{
    try
    {
        if (listEncodedPhotos.Count == 0)
        {
            throw new Exception("Добавьте фотографию автомобиля");
        }

        System.Windows.Forms.DialogResult result = System.Windows.Forms.MessageBox.Show("Вы уверены, что хотите удалить данную фотографию", "Удаление",
                                         System.Windows.Forms.MessageBoxButtons.YesNo,
                                         System.Windows.Forms.MessageBoxIcon.Information,
                                         System.Windows.Forms.MessageBoxDefaultButton.Button1,
                                         System.Windows.Forms.MessageBoxOptions.DefaultDesktopOnly);
        if (result == System.Windows.Forms.DialogResult.Yes)
        {
            listPhotos.RemoveAt(currentIndex);
            listEncodedPhotos.RemoveAt(currentIndex);

            currentIndex = 0;

            if (listPhotos.Count == 1)
            {
                currentIndex = 0;
                btnLeft.Visibility = Visibility.Hidden;
                btnRight.Visibility = Visibility.Hidden;
            }

            if (listPhotos.Count == 0)
            {
                currentIndex = 0;
                BitmapImage bi = new BitmapImage();
            }
        }
    }
}

```

Изм.	Лист	№ докум.	Подпись	Дата

```
        bi.BeginInit();
        bi.UriSource = new Uri("/InsuranceAgency;component/Assets/Car.jpg",
UriKind.RelativeOrAbsolute);
        bi.EndInit();
        imgCar.Source = bi;
    }
    else
    {
        imgCar.Source = listPhotos[currentIndex];
    }
}
catch (Exception exp)
{
    tbException.Visibility = Visibility.Visible;
    tbException.Text = exp.Message;
}
}
```

Листинг класса AddEmployee:

```
namespace InsuranceAgency.Pages
{
    public partial class AddEmployee : Page
    {
        public AddEmployee()
        {
            InitializeComponent();
        }

        private void tbPassportSeries_TextChanged(object sender, TextChangedEventArgs e)
        {
            if (tbPassportSeries.Text.Length == 0)
            {
                tbPassportSeriesHint.Visibility = Visibility.Visible;
            }
            else
            {
                tbPassportSeriesHint.Visibility = Visibility.Hidden;
            }
        }

        private void tbPassportNumber_TextChanged(object sender, TextChangedEventArgs e)
        {
            if (tbPassportNumber.Text.Length == 0)
            {
                tbPassportNumberHint.Visibility = Visibility.Visible;
            }
            else
            {
                tbPassportNumberHint.Visibility = Visibility.Hidden;
            }
        }

        private void btnAddEmployee_Click(object sender, RoutedEventArgs e)
        {
            try
            {

```

<i>Изм.</i>	<i>Лист</i>	<i>№ докум.</i>	<i>Подпись</i>	<i>Дата</i>

```

string fullName = tbFullName.Text.Trim();
if(fullName == "")
{
    throw new Exception("Заполните поле ФИО");
}

DateTime birthday = Convert.ToDateTime(dpBirthday.Text);

string telephone_temp = tbTelephone.Text.Trim();
string telephone = "";
if (telephone_temp == "")
{
    throw new Exception("Заполните поле Номер телефона");
}
foreach(var item in telephone_temp)
{
    if (char.IsDigit(item))
    {
        telephone += item;
    }
}
if (telephone.Length > 15)
{
    throw new Exception("Номер телефона не может быть больше 15 символов");
}

string passportSeries = tbPassportSeries.Text.Trim();
if (passportSeries.Length != 4)
{
    throw new Exception("Серия паспорта должна содержать 4 цифры");
}
string passportNumber = tbPassportNumber.Text.Trim();
if (passportNumber.Length != 6)
{
    throw new Exception("Номер паспорта должен содержать 6 цифр");
}
string passport = passportSeries + passportNumber;
foreach (var item in passport)
{
    if (!char.IsDigit(item))
    {
        throw new Exception("Серия и номер паспорта должны содержать только цифры");
    }
}

string login = tbLogin.Text.Trim();
if (login.Length < 4 || login.Length > 32)
{
    throw new Exception("Длина логина должна быть от 4 до 32 символов");
}
foreach (var item in telephone)
{
    if (char.IsNullOrWhiteSpace(item))
    {
        throw new Exception("Логин не может содержать пробелы");
    }
}

string password = tbPassword.Text.Trim();
if (password.Length < 4 || password.Length > 32)
{
    throw new Exception("Длина пароля должна быть от 4 до 32 символов");
}

```

Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

Листинг класса AddPersonAllowedToDrive

```
namespace InsuranceAgency.Pages
{
    public partial class AddPersonAllowedToDrive : Page
    {
        public AddPersonAllowedToDrive()
        {
            InitializeComponent();
        }

        private void tbDrivingLicenceSeries_TextChanged(object sender, TextChangedEventArgs e)
```

<i>Изм.</i>	<i>Лист</i>	<i>№ докум.</i>	<i>Подпись</i>	<i>Дата</i>

МИВУ 09.03.04-8.000 ПЗ

Лист

118

```

{
    if (tbDrivingLicenceSeries.Text.Length == 0)
    {
        tbDrivingLicenceSeriesHint.Visibility = Visibility.Visible;
    }
    else
    {
        tbDrivingLicenceSeriesHint.Visibility = Visibility.Hidden;
    }
}

private void tbDrivingLicenceNumber_TextChanged(object sender, TextChangedEventArgs e)
{
    if (tbDrivingLicenceNumber.Text.Length == 0)
    {
        tbDrivingLicenceNumberHint.Visibility = Visibility.Visible;
    }
    else
    {
        tbDrivingLicenceNumberHint.Visibility = Visibility.Hidden;
    }
}

private void btnAddPersonAllowedToDrive_Click(object sender, RoutedEventArgs e)
{
    try
    {
        string fullName = tbFullName.Text.Trim();
        if (fullName == "")
        {
            throw new Exception("Заполните поле ФИО");
        }

        string drivingLicenceSeries = tbDrivingLicenceSeries.Text.Trim();
        if (drivingLicenceSeries.Length != 4)
        {
            throw new Exception("Серия водительского удостоверения должна содержать 4 цифры");
        }
        string drivingLicenceNumber = tbDrivingLicenceNumber.Text.Trim();
        if (drivingLicenceNumber.Length != 6)
        {
            throw new Exception("Номер водительского удостоверения должен содержать 6 цифр");
        }
        string drivingLicence = drivingLicenceSeries + drivingLicenceNumber;
        foreach (var item in drivingLicence)
        {
            if (!char.IsDigit(item))
            {
                throw new Exception("Серия и номер водительского удостоверения должны содержать только цифры");
            }
        }
    }
}

```

PersonAllowedToDrive personAllowedToDrive = new PersonAllowedToDrive(fullName, drivingLicence);

Database.AddPersonAllowedToDrive(personAllowedToDrive);

MessageBox.Show("Водитель успешно добавлен", "", MessageBoxButton.OK, MessageBoxImage.Information);

tbFullName.Clear();

Изм.	Лист	№ докум.	Подпись	Дата

```
        tbDrivingLicenceSeries.Clear();
        tbDrivingLicenceNumber.Clear();
        tbException.Visibility = Visibility.Hidden;
    }
    catch (Exception exp)
    {
        tbException.Visibility = Visibility.Visible;
        tbException.Text = exp.Message;
    }
}
```

Листинг класса AddPolicy:

```

namespace InsuranceAgency.Pages
{
    public partial class AddPolicy : Page
    {
        int PolicyholderID;
        List<PersonAllowedToDrive> list = new List<PersonAllowedToDrive>();

        public AddPolicy(int policyholderID)
        {
            InitializeComponent();

            PolicyholderID = policyholderID;
            dpDateOfConclusion.SelectedDate = DateTime.Now;
        }

        public AddPolicy(Struct.Policy policy)
        {
            InitializeComponent();

            PolicyholderID = policy.PolicyholderID;
            AddInfoInTb(policy);
        }

        private void AddInfoInTb(Struct.Policy policy)
        {
            cbInsuranceType.Text = policy.InsuranceType;
            tbInsurancePremium.Text = policy.InsurancePremium.ToString();
            tbInsuranceAmount.Text = policy.InsuranceAmount.ToString();
            dpDateOfConclusion.SelectedDate = policy.ExpirationDate;
            tbVIN.Text = Database.SearchCarID(policy.CarID).VIN;

            List<Connection> connections = Database.SearchConnection(policy.ID);
            foreach (var item in connections)
            {
                PersonAllowedToDrive personAllowedToDrive =
                    Database.SearchPersonAllowedToDriveID(item.PersonAllowedToDriveID);
                list.Add(personAllowedToDrive);
                cbPersonsAllowedToDrive.Items.Add(personAllowedToDrive.FullName);
            }
            cbPersonsAllowedToDrive.Text = "";
            tbPersonsAllowedToDriveHint.Visibility = Visibility.Visible;
        }

        private void cbPersonsAllowedToDrive_GotFocus(object sender, RoutedEventArgs e)
        {
            tbPersonsAllowedToDriveHint.Visibility = Visibility.Hidden;
        }
    }
}

```

<i>Изм.</i>	<i>Лист</i>	<i>№ докум.</i>	<i>Подпись</i>	<i>Дата</i>

```

}

private void cbPersonsAllowedToDrive_LostFocus(object sender, RoutedEventArgs e)
{
    if (cbPersonsAllowedToDrive.Text.Length == 0)
    {
        tbPersonsAllowedToDriveHint.Visibility = Visibility.Visible;
    }
}

private void btnAddPolicy_Click(object sender, RoutedEventArgs e)
{
    try
    {
        string insuranceType = cbInsuranceType.Text;
        if (insuranceType == "")
        {
            throw new Exception("Заполните поле Вид страхования");
        }

        string insuranceAmount_temp = tbInsuranceAmount.Text.Trim();
        if (insuranceAmount_temp == "")
        {
            throw new Exception("Заполните поле Страховая сумма");
        }
        int insuranceAmount = 0;
        try { insuranceAmount = Convert.ToInt32(insuranceAmount_temp); }
        catch { throw new Exception("Страховая сумма должна быть целым числом"); }

        string insurancePremium_temp = tbInsurancePremium.Text.Trim();
        if (insurancePremium_temp == "")
        {
            throw new Exception("Заполните поле Страховая премия");
        }
        int insurancePremium = 0;
        try { insurancePremium = Convert.ToInt32(insurancePremium_temp); }
        catch { throw new Exception("Страховая премия должна быть целым числом"); }

        DateTime dateOfConclusion = Convert.ToDateTime(dpDateOfConclusion.Text);

        DateTime expirationDate = dateOfConclusion;
        if (cbExpirationDate.Text == "")
        {
            throw new Exception("Заполните поле Срок действия");
        }
        else if (cbExpirationDate.Text == "6 месяцев")
        {
            expirationDate = expirationDate.AddMonths(6);
        }
        else
        {
            expirationDate = expirationDate.AddYears(1);
        }

        string vin = tbVIN.Text.Trim();
        if (vin.Length != 17)
        {
            throw new Exception("VIN номер должен содержать 17 знаков");
        }
        foreach (var item in vin)
        {
            if (!char.IsDigit(item) && !(Convert.ToInt32(item) >= 65 && Convert.ToInt32(item) <= 90))
            {

```

Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

```

        throw new Exception("VIN номер должен состоять из цифр и заглавных латинских букв");
    }
}
Car car = Database.SearchCar(vin);
int carID = car.ID;

int employeeID = Database.SearchEmployeeLogin(Database.Login).ID;

Struct.Policy policy = new Struct.Policy(insuranceType, insurancePremium, insuranceAmount,
dateOfConclusion, expirationDate, PolicyholderID, carID, employeeID);

if(list.Count == 0)
{
    throw new Exception("Список лиц, допущенных к управлению пуст");
}

Database.AddPolicyWithConnections(policy, list);

MessageBox.Show("Полис успешно добавлен", "", MessageBoxButtons.OK,
MessageBoxImage.Information);

this.NavigationService.Navigate(new Pages.Policy(PolicyholderID));
}
catch (Exception exp)
{
    tbException.Visibility = Visibility.Visible;
    tbException.Text = exp.Message;
}
}

private void btnAddPersonAllowedToDrive_Click(object sender, RoutedEventArgs e)
{
try
{
    PersonAllowedToDrive personAllowedToDrive =
Database.SearchPersonAllowedToDrive(cbPersonsAllowedToDrive.Text.Trim());
    foreach(var item in list)
    {
        if(item.ID == personAllowedToDrive.ID)
        {
            throw new Exception("Данный водитель уже добавлен");
        }
    }
    list.Add(personAllowedToDrive);
    cbPersonsAllowedToDrive.Items.Add(personAllowedToDrive.FullName);

    MessageBox.Show("Водитель добавлен", "", MessageBoxButtons.OK, MessageBoxIcon.Information);
    cbPersonsAllowedToDrive.Text = "";
    tbPersonsAllowedToDriveHint.Visibility = Visibility.Visible;
}
catch (Exception exp)
{
    tbException.Visibility = Visibility.Visible;
    tbException.Text = exp.Message;
}
}

private void btnDeletePersonAllowedToDrive_Click(object sender, RoutedEventArgs e)
{
try
{
    System.Windows.Forms.DialogResult result = System.Windows.Forms.MessageBox.Show("Вы
уверены, что хотите удалить данного водителя", "Удаление",

```

Изм.	Лист	№ докум.	Подпись	Дата

```

System.Windows.Forms.MessageBoxButtons.YesNo,
System.Windows.Forms.MessageBoxIcon.Information,
System.Windows.Forms.MessageBoxDefaultButton.Button1,
System.Windows.Forms.MessageBoxOptions.DefaultDesktopOnly);
if (result == System.Windows.Forms.DialogResult.Yes)
{
    int index = cbPersonsAllowedToDrive.SelectedIndex;
    try
    {
        if (cbPersonsAllowedToDrive.Text == list[index].FullName)
        {
            list.RemoveAt(index);
        }
        else
        {
            throw new Exception("Данный водитель не существует в списке добавленных водителей");
        }
    }
    catch { throw new Exception("Данный водитель не существует в списке добавленных водителей"); }
}
cbPersonsAllowedToDrive.Items.Clear();
foreach (var item in list)
{
    cbPersonsAllowedToDrive.Items.Add(item.FullName);
}

MessageBox.Show("Водитель удалён", "", MessageBoxButtons.OK, MessageBoxIcon.Information);
cbPersonsAllowedToDrive.Text = "";
tbPersonsAllowedToDriveHint.Visibility = Visibility.Visible;
}
}
catch (Exception exp)
{
    tbException.Visibility = Visibility.Visible;
    tbException.Text = exp.Message;
}
}

private void btnBack_Click(object sender, RoutedEventArgs e)
{
    this.NavigationService.Navigate(new Pages.Policy(PolicyholderID));
}
}
}

```

Листинг класса AddPolicyholder:

```

namespace InsuranceAgency.Pages
{
    public partial class AddPolicyholder : Page
    {
        public AddPolicyholder()
        {
            InitializeComponent();
        }

        private void tbPassportSeries_TextChanged(object sender, TextChangedEventArgs e)
        {

```

Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

```

        if (tbPassportSeries.Text.Length == 0)
    {
        tbPassportSeriesHint.Visibility = Visibility.Visible;
    }
    else
    {
        tbPassportSeriesHint.Visibility = Visibility.Hidden;
    }
}

private void tbPassportNumber_TextChanged(object sender, TextChangedEventArgs e)
{
    if (tbPassportNumber.Text.Length == 0)
    {
        tbPassportNumberHint.Visibility = Visibility.Visible;
    }
    else
    {
        tbPassportNumberHint.Visibility = Visibility.Hidden;
    }
}

private void btnAddPolicyholder_Click(object sender, RoutedEventArgs e)
{
    try
    {
        string fullName = tbFullName.Text.Trim();
        if (fullName == "")
        {
            throw new Exception("Заполните поле ФИО");
        }

        DateTime birthday = Convert.ToDateTime(dpBirthday.Text);

        string telephone_temp = tbTelephone.Text.Trim();
        string telephone = "";
        if (telephone_temp == "")
        {
            throw new Exception("Заполните поле Номер телефона");
        }
        foreach (var item in telephone_temp)
        {
            if (char.IsDigit(item))
            {
                telephone += item;
            }
        }
        if (telephone.Length > 15)
        {
            throw new Exception("Номер телефона не может быть больше 15 символов");
        }

        string passportSeries = tbPassportSeries.Text.Trim();
        if (passportSeries.Length != 4)
        {
            throw new Exception("Серия паспорта должна содержать 4 цифры");
        }
        string passportNumber = tbPassportNumber.Text.Trim();
        if (passportNumber.Length != 6)
        {
            throw new Exception("Номер паспорта должен содержать 6 цифр");
        }
        string passport = passportSeries + passportNumber;
    }
}

```

Изм.	Лист	№ докум.	Подпись	Дата

```
foreach (var item in passport)
{
    if (!char.IsDigit(item))
    {
        throw new Exception("Серия и номер паспорта должны содержать только цифры");
    }
}

Policyholder policyholder = new Policyholder(fullName, birthday, telephone, passport);

Database.AddPolicyholder(policyholder);

MessageBox.Show("Страхователь успешно добавлен", "", MessageBoxButtons.OK,
MessageBoxImage.Information);

tbFullName.Clear();
dpBirthday.Text = "01.01.1990";
tbTelephone.Clear();
tbPassportSeries.Clear();
tbPassportNumber.Clear();
tbException.Visibility = Visibility.Hidden;
}
catch (Exception exp)
{
    tbException.Visibility = Visibility.Visible;
    tbException.Text = exp.Message;
}
}
```

Листинг класса AllCars:

```
namespace InsuranceAgency.Pages
{
    public partial class AllCars : Page
    {
        public AllCars()
        {
            InitializeComponent();
            AddDataInDG();
        }

        private void tbSearch_TextChanged(object sender, TextChangedEventArgs e)
        {
            if (tbSearch.Text.Length == 0)
            {
                tbSearchHint.Visibility = Visibility.Visible;
            }
            else
            {
                tbSearchHint.Visibility = Visibility.Hidden;
            }
        }

        private void btnSearch_Click(object sender, RoutedEventArgs e)
        {
            try
            {

```

<i>Изм.</i>	<i>Лист</i>	<i>№ докум.</i>	<i>Подпись</i>	<i>Дата</i>

МИВУ 09.03.04-8.000 ПЗ

Лист

125

```

        string search = tbSearch.Text.Trim();
        if (search == "")
        {
            throw new Exception("Строка поиска пуста");
        }

        Car searchCar = Database.SearchCar(search);

        DataTable dt = DTColumn();
        AddRow(dt, searchCar);

        DataView view = new DataView(dt);
        dgCars.ItemsSource = view;

        tbSearch.Text = "";
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message, "", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

private void AddDataInDG()
{
    List<Car> list = Database.AllCars();

    DataTable dt = DTColumn();
    foreach (var item in list)
    {
        AddRow(dt, item);
    }

    DataView view = new DataView(dt);
    dgCars.ItemsSource = view;
}

private DataTable DTColumn()
{
    DataTable dt = new DataTable();

    DataColumn column;

    column = new DataColumn();
    column.DataType = Type.GetType("System.String");
    column.ColumnName = "ID";
    dt.Columns.Add(column);

    column = new DataColumn();
    column.DataType = Type.GetType("System.String");
    column.ColumnName = "Model";
    dt.Columns.Add(column);

    column = new DataColumn();
    column.DataType = Type.GetType("System.String");
    column.ColumnName = "VIN";
    dt.Columns.Add(column);

    column = new DataColumn();
    column.DataType = Type.GetType("System.String");
    column.ColumnName = "RegistrationPlate";
    dt.Columns.Add(column);

    column = new DataColumn();
}

```

Изм.	Лист	№ докум.	Подпись	Дата

```

column.DataType = Type.GetType("System.String");
column.ColumnName = "VehiclePassport";
dt.Columns.Add(column);

return dt;
}

private void AddRow(DataTable dt, Car car)
{
    DataRow row;
    row = dt.NewRow();
    row["ID"] = car.ID;
    row["Model"] = car.Model;
    row["VIN"] = car.VIN;
    row["RegistrationPlate"] = car.RegistrationPlate;
    row["VehiclePassport"] = car.VehiclePassport.Insert(4, " ");
    dt.Rows.Add(row);
}

private void btnTable_Click(object sender, RoutedEventArgs e)
{
    AddDataInDG();
}

private void dgCars_CurrentCellChanged(object sender, EventArgs e)
{
    DataGrid dg = (sender as DataGrid);
    var row = dg.Items.IndexOf(dg.CurrentCell.Item);

    dg.SelectedIndex = row;
    var selectedRow = (DataRowView)dg.SelectedItem;

    int ID = Convert.ToInt32(selectedRow["ID"].ToString());
    this.NavigationService.Navigate(new Pages.ChangeCar(ID));
}
}
}
}

```

Листинг класса AllEmployees:

```

namespace InsuranceAgency.Pages
{
    public partial class AllEmployees : Page
    {
        List<Employee> list = Database.AllEmployees();

        public AllEmployees()
        {
            InitializeComponent();

            AddDataInDG();
        }

        private void tbSearch_TextChanged(object sender, TextChangedEventArgs e)
        {
            if (tbSearch.Text.Length == 0)
            {
                tbSearchHint.Visibility = Visibility.Visible;
            }
            else

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        {
            tbSearchHint.Visibility = Visibility.Hidden;
        }
    }

private void btnSearch_Click(object sender, RoutedEventArgs e)
{
    tbSortHint.Visibility = Visibility.Visible;
    cbSort.Text = "";

    try
    {
        string search_temp = tbSearch.Text.Trim();
        string search = "";
        if (search_temp == "")
        {
            throw new Exception("Строка поиска пуста");
        }
        foreach (var item in search_temp)
        {
            if (char.IsDigit(item))
            {
                search += item;
            }
        }
    }

    Employee searchEmployee = Database.SearchEmployee(search);

    DataTable dt = DTColumn();
    AddRow(dt, searchEmployee);

    DataView view = new DataView(dt);
    dgEmployees.ItemsSource = view;

    tbSearch.Text = "";
}
catch (Exception exp)
{
    MessageBox.Show(exp.Message, "", MessageBoxButton.OK, MessageBoxIcon.Error);
}
}

private void cbSort_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    tbSortHint.Visibility = Visibility.Hidden;

    DataTable dt = DTColumn();
    switch ((sender as ComboBox).SelectedIndex)
    {
        //Работает и администратор
        case 0:
            foreach (var item in list)
            {
                if (item.Admin && item.Works)
                {
                    AddRow(dt, item);
                }
            }
            break;
        //Работает и не администратор
        case 1:
            foreach (var item in list)
            {

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        if (!item.Admin && item.Works)
        {
            AddRow(dt, item);
        }
    }
    break;
//Работает
case 2:
    foreach (var item in list)
    {
        if (item.Works)
        {
            AddRow(dt, item);
        }
    }
    break;
//Не работает
case 3:
    foreach (var item in list)
    {
        if (!item.Works)
        {
            AddRow(dt, item);
        }
    }
    break;
//Администратор
case 4:
    foreach (var item in list)
    {
        if (item.Admin)
        {
            AddRow(dt, item);
        }
    }
    break;
//Не администратор
case 5:
    foreach (var item in list)
    {
        if (!item.Admin)
        {
            AddRow(dt, item);
        }
    }
    break;
}

DataView view = new DataView(dt);
dgEmployees.ItemsSource = view;
}

private void AddDataInDG()
{
    DataTable dt = DTColumn();
    foreach (var item in list)
    {
        AddRow(dt, item);
    }

    DataView view = new DataView(dt);
    dgEmployees.ItemsSource = view;
}

```

Изм.	Лист	№ докум.	Подпись	Дата

```

private DataTable DTColumn()
{
    DataTable dt = new DataTable();

    DataColumn column;
    column = new DataColumn();
    column.DataType = Type.GetType("System.String");
    column.ColumnName = "ID";
    dt.Columns.Add(column);

    column = new DataColumn();
    column.DataType = Type.GetType("System.String");
    column.ColumnName = "FullName";
    dt.Columns.Add(column);

    column = new DataColumn();
    column.DataType = Type.GetType("System.String");
    column.ColumnName = "Birthday";
    dt.Columns.Add(column);

    column = new DataColumn();
    column.DataType = Type.GetType("System.String");
    column.ColumnName = "Telephone";
    dt.Columns.Add(column);

    column = new DataColumn();
    column.DataType = Type.GetType("System.String");
    column.ColumnName = "Passport";
    dt.Columns.Add(column);

    column = new DataColumn();
    column.DataType = Type.GetType("System.String");
    column.ColumnName = "Login";
    dt.Columns.Add(column);

    column = new DataColumn();
    column.DataType = Type.GetType("System.String");
    column.ColumnName = "Password";
    dt.Columns.Add(column);

    column = new DataColumn();
    column.DataType = Type.GetType("System.String");
    column.ColumnName = "Admin";
    dt.Columns.Add(column);

    column = new DataColumn();
    column.DataType = Type.GetType("System.String");
    column.ColumnName = "Works";
    dt.Columns.Add(column);

    return dt;
}

private void AddRow(DataTable dt, Employee employee)
{
    DataRow row;
    row = dt.NewRow();
    row["ID"] = employee.ID;
    row["FullName"] = employee.FullName;
    row["Birthday"] = employee.Birthday.ToString("d");
    row["Telephone"] = employee.Telephone;
}

```

Изм.	Лист	№ докум.	Подпись	Дата

```

row["Passport"] = employee.Passport.Insert(4, " ");
row["Login"] = employee.Login;
row["Password"] = employee.Password;
row["Admin"] = employee.Admin;
row["Works"] = employee.Works;
dt.Rows.Add(row);
}

private void btnTable_Click(object sender, RoutedEventArgs e)
{
    tbSortHint.Visibility = Visibility.Visible;
    cbSort.Text = "";
    AddDataInDG();
}

private void dgEmployees_CurrentCellChanged(object sender, EventArgs e)
{
    DataGrid dg = (sender as DataGrid);
    int row = dg.Items.IndexOf(dg.CurrentCell.Item);

    dg.SelectedIndex = row;
    var selectedRow = (DataRowView)dg.SelectedItem;

    int ID = Convert.ToInt32(selectedRow["ID"].ToString());
    foreach (var item in list)
    {
        if (item.ID == ID)
        {
            this.NavigationService.Navigate(new Pages.ChangeEmployee(item));
        }
    }
}
}

```

Листинг класса AllPersonsAllowedToDrive:

```
namespace InsuranceAgency.Pages
{
    public partial class AllPersonsAllowedToDrive : Page
    {
        List<PersonAllowedToDrive> list = Database.AllPersonsAllowedToDrive();

        public AllPersonsAllowedToDrive()
        {
            InitializeComponent();
            AddDataInDG();
        }

        private void tbSearch_TextChanged(object sender, TextChangedEventArgs e)
        {
            if (tbSearch.Text.Length == 0)
            {
                tbSearchHint.Visibility = Visibility.Visible;
            }
            else
            {
                tbSearchHint.Visibility = Visibility.Hidden;
            }
        }
    }
}
```

<i>Изм.</i>	<i>Лист</i>	<i>№ докум.</i>	<i>Подпись</i>	<i>Дата</i>

```

}

private void btnSearch_Click(object sender, RoutedEventArgs e)
{
    try
    {
        string search_temp = tbSearch.Text.Trim();
        string search = "";
        if (search_temp == "")
        {
            throw new Exception("Строка поиска пуста");
        }
        foreach (var item in search_temp)
        {
            if (char.IsDigit(item))
            {
                search += item;
            }
        }
    }

    PersonAllowedToDrive searchPersonAllowedToDrive = Database.SearchPersonAllowedToDrive(search);

    DataTable dt = DTColumn();
    AddRow(dt, searchPersonAllowedToDrive);

    DataView view = new DataView(dt);
    dgPersonsAllowedToDrive.ItemsSource = view;

    tbSearch.Text = "";
}
catch (Exception exp)
{
    MessageBox.Show(exp.Message, "", MessageBoxButton.OK, MessageBoxImage.Error);
}
}

private void AddDataInDG()
{
    DataTable dt = DTColumn();
    foreach (var item in list)
    {
        AddRow(dt, item);
    }

    DataView view = new DataView(dt);
    dgPersonsAllowedToDrive.ItemsSource = view;
}

private DataTable DTColumn()
{
    DataTable dt = new DataTable();

    DataColumn column;

    column = new DataColumn();
    column.DataType = Type.GetType("System.String");
    column.ColumnName = "ID";
    dt.Columns.Add(column);

    column = new DataColumn();
    column.DataType = Type.GetType("System.String");
    column.ColumnName = "FullName";
    dt.Columns.Add(column);
}

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        column = new DataColumn();
        column.DataType = Type.GetType("System.String");
        column.ColumnName = "DrivingLicence";
        dt.Columns.Add(column);

        return dt;
    }

private void AddRow(DataTable dt, PersonAllowedToDrive personAllowedToDrive)
{
    DataRow row;
    row = dt.NewRow();
    row["ID"] = personAllowedToDrive.ID;
    row["FullName"] = personAllowedToDrive.FullName;
    row["DrivingLicence"] = personAllowedToDrive.DrivingLicence.Insert(4, " ");
    dt.Rows.Add(row);
}

private void btnTable_Click(object sender, RoutedEventArgs e)
{
    AddDataInDG();
}

private void PersonsAllowedToDrive_CurrentCellChanged(object sender, EventArgs e)
{
    DataGrid dg = (sender as DataGrid);
    int row = dg.Items.IndexOf(dg.CurrentCell.Item);

    dg.SelectedIndex = row;
    var selectedRow = (DataRowView)dg.SelectedItem;

    int ID = Convert.ToInt32(selectedRow["ID"].ToString());
    foreach (var item in list)
    {
        if (item.ID == ID)
        {
            this.NavigationService.Navigate(new Pages.ChangePersonAllowedToDrive(item));
        }
    }
}
}

```

Листинг класса AllPolicyholders:

```

namespace InsuranceAgency.Pages
{
    public partial class AllPolicyholders : Page
    {
        List<Policyholder> list = Database.AllPolicyholders();

        public AllPolicyholders()
        {
            InitializeComponent();
            AddDataInDG();
        }

        private void tbSearch_TextChanged(object sender, TextChangedEventArgs e)
        {

```

Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

```

        if (tbSearch.Text.Length == 0)
    {
        tbSearchHint.Visibility = Visibility.Visible;
    }
    else
    {
        tbSearchHint.Visibility = Visibility.Hidden;
    }
}

private void btnSearch_Click(object sender, RoutedEventArgs e)
{
    try
    {
        string search_temp = tbSearch.Text.Trim();
        string search = "";
        if (search_temp == "")
        {
            throw new Exception("Строка поиска пуста");
        }
        foreach (var item in search_temp)
        {
            if (char.IsDigit(item))
            {
                search += item;
            }
        }
    }

    Policyholder searchPolicyholder = Database.SearchPolicyholder(search);

    DataTable dt = DTColumn();
    AddRow(dt, searchPolicyholder);

    DataView view = new DataView(dt);
    dgPolicyholders.ItemsSource = view;

    tbSearch.Text = "";
}
catch (Exception exp)
{
    MessageBox.Show(exp.Message, "", MessageBoxButton.OK, MessageBoxImage.Error);
}
}

private void AddDataInDG()
{
    DataTable dt = DTColumn();
    foreach (var item in list)
    {
        AddRow(dt, item);
    }

    DataView view = new DataView(dt);
    dgPolicyholders.ItemsSource = view;
}

private DataTable DTColumn()
{
    DataTable dt = new DataTable();

    DataColumn column;

    column = new DataColumn();

```

Изм.	Лист	№ докум.	Подпись	Дата

```

column.DataType = Type.GetType("System.String");
column.ColumnName = "ID";
dt.Columns.Add(column);

column = new DataColumn();
column.DataType = Type.GetType("System.String");
column.ColumnName = "FullName";
dt.Columns.Add(column);

column = new DataColumn();
column.DataType = Type.GetType("System.String");
column.ColumnName = "Birthday";
dt.Columns.Add(column);

column = new DataColumn();
column.DataType = Type.GetType("System.String");
column.ColumnName = "Telephone";
dt.Columns.Add(column);

column = new DataColumn();
column.DataType = Type.GetType("System.String");
column.ColumnName = "Passport";
dt.Columns.Add(column);

return dt;
}

private void AddRow(DataTable dt, Policyholder policyholder)
{
    DataRow row;
    row = dt.NewRow();
    row["ID"] = policyholder.ID;
    row["FullName"] = policyholder.FullName;
    row["Birthday"] = policyholder.Birthday.ToString("d");
    row["Telephone"] = policyholder.Telephone;
    row["Passport"] = policyholder.Passport.Insert(4, " ");
    dt.Rows.Add(row);
}

private void btnTable_Click(object sender, RoutedEventArgs e)
{
    AddDataInDG();
}

private void dgPolicyholders_CurrentCellChanged(object sender, EventArgs e)
{
    DataGrid dg = (sender as DataGrid);
    int row = dg.Items.IndexOf(dg.CurrentCell.Item);

    dg.SelectedIndex = row;
    var selectedRow = (DataRowView)dg.SelectedItem;

    int ID = Convert.ToInt32(selectedRow["ID"].ToString());
    foreach (var item in list)
    {
        if (item.ID == ID)
        {
            this.NavigationService.Navigate(new Pages.ChangePolicyholder(item));
        }
    }
}

```

Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

Листинг класса ChangeCar:

```
namespace InsuranceAgency.Pages
{
    public partial class ChangeCar : Page
    {
        List<Photo> listPhotos = new List<Photo>();
        List<string> listEncodedPhotos = new List<string>();

        List<BitmapImage> listNewPhotos = new List<BitmapImage>();
        List<string> listNewEncodedPhotos = new List<string>();

        List<Photo> listDeletePhotos = new List<Photo>();
        List<string> listAddPhotos = new List<string>();

        int currentIndex = 0;

        public ChangeCar()
        {
            InitializeComponent();
        }

        public ChangeCar(int id)
        {
            InitializeComponent();

            Car car = Database.SearchCarID(id);
            searchCar = car;

            listPhotos = Database.SearchPhoto(car.ID);
            foreach(var item in listPhotos)
            {
                listNewEncodedPhotos.Add(item.EncodedPhoto);
                listEncodedPhotos.Add(item.EncodedPhoto);
                listNewPhotos.Add(DBImage.Decode(item.EncodedPhoto));
            }

            AddInfoInTb(car);
            flagSearchCar = true;
        }

        bool flagSearchCar = false;
        Car searchCar;

        private void tbSearch_TextChanged(object sender, TextChangedEventArgs e)
        {
            if (tbSearch.Text.Length == 0)
            {
                tbSearchHint.Visibility = Visibility.Visible;
            }
            else
            {
                tbSearchHint.Visibility = Visibility.Hidden;
            }
        }

        private void btnSearch_Click(object sender, RoutedEventArgs e)
        {
            tbException.Visibility = Visibility.Hidden;
            try
            {

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        string search = tbSearch.Text.Trim();
        if (search == "")
        {
            throw new Exception("Строка поиска пуста");
        }

        searchCar = Database.SearchCar(search);

        listPhotos = Database.SearchPhoto(searchCar.ID);
        foreach (var item in listPhotos)
        {
            listNewEncodedPhotos.Add(item.EncodedPhoto);
            listEncodedPhotos.Add(item.EncodedPhoto);
            listNewPhotos.Add(DBImage.Decode(item.EncodedPhoto));
        }

        AddInfoInTb(searchCar);

        tbSearch.Text = "";

        flagSearchCar = true;
    }
    catch(Exception exp)
    {
        MessageBox.Show(exp.Message, "", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

private void AddInfoInTb(Car car)
{
    tbModel.Text = car.Model;
    tbVIN.Text = car.VIN;
    tbRegistrationPlate.Text = car.RegistrationPlate;
    tbVehiclePassportSeries.Text = "";
    tbVehiclePassportNumber.Text = "";
    for (var i = 0; i < 4; i++) tbVehiclePassportSeries.Text += car.VehiclePassport[i];
    for (var i = 4; i < 10; i++) tbVehiclePassportNumber.Text += car.VehiclePassport[i];

    imgCar.Source = listNewPhotos[0];
    currentIndex = 0;
    if (listNewPhotos.Count - 1 > 0)
    {
        btnLeft.Visibility = Visibility.Visible;
        btnRight.Visibility = Visibility.Visible;
    }
}

private void tbVehiclePassportSeries_TextChanged(object sender, TextChangedEventArgs e)
{
    if (tbVehiclePassportSeries.Text.Length == 0)
    {
        tbVehiclePassportSeriesHint.Visibility = Visibility.Visible;
    }
    else
    {
        tbVehiclePassportSeriesHint.Visibility = Visibility.Hidden;
    }
}

private void tbVehiclePassportNumber_TextChanged(object sender, TextChangedEventArgs e)
{
    if (tbVehiclePassportNumber.Text.Length == 0)
    {

```

Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

```

        tbVehiclePassportNumberHint.Visibility = Visibility.Visible;
    }
    else
    {
        tbVehiclePassportNumberHint.Visibility = Visibility.Hidden;
    }
}

private void btnAddImage_Click(object sender, RoutedEventArgs e)
{
    OpenFileDialog openFileDialog = new OpenFileDialog();
    openFileDialog.Filter = "All Embroidery Files | *.bmp; *.gif; *.jpeg; *.jpg; " +
        "*.tif; *.tiff; *.png; *.wmf; *.emf" +
        "|Windows Bitmap (*.bmp)|*.bmp" +
        "|JPEG File Interchange Format (*.jpg)|*.jpg; *.jpeg" +
        "|Graphics Interchange Format (*.gif)|*.gif" +
        "|Portable Network Graphics (*.png)|*.png" +
        "|Tag Embroidery File Format (*.tif)|*.tif; *.tiff";
    if (openFileDialog.ShowDialog() == true)
    {
        var size = new FileInfo(openFileDialog.FileName).Length;
        if (size / 1024 > 512)
        {
            tbException.Visibility = Visibility.Visible;
            tbException.Text = "Размер фото не может быть больше 512Кб";
        }
        else
        {
            BitmapImage bi = new BitmapImage(new Uri(openFileDialog.FileName));
            imgCar.Source = bi;
            listNewPhotos.Add(bi);
            currentIndex = listNewPhotos.Count - 1;
            if (currentIndex == 1)
            {
                btnLeft.Visibility = Visibility.Visible;
                btnRight.Visibility = Visibility.Visible;
            }

            string image = DBImage.Encode(bi);
            listNewEncodedPhotos.Add(image);
            listAddPhotos.Add(image);
        }
    }
}

private void btnChangeCar_Click(object sender, RoutedEventArgs e)
{
    try
    {
        if (!flagSearchCar) throw new Exception("Выберите автомобиль");

        string registrationPlate = tbRegistrationPlate.Text.Trim();
        if (registrationPlate == "")
        {
            throw new Exception("Заполните поле Регистрационный знак");
        }

        string vehiclePassportSeries = tbVehiclePassportSeries.Text.Trim();
        if (vehiclePassportSeries.Length != 4)
        {
            throw new Exception("Серия паспорта должна содержать 4 знака");
        }
        for (var i = 0; i < vehiclePassportSeries.Length; i++)
    
```

Изм.	Лист	№ докум.	Подпись	Дата

```

{
    if ((i == 0 || i == 1) && !char.IsDigit(vehiclePassportSeries[i]))
    {
        throw new Exception("Первые два символа серии паспорта ТС должны быть цифрами");
    }
    if ((i == 3 || i == 4) && !char.IsLetter(vehiclePassportSeries[i]))
    {
        throw new Exception("Последние два символа серии паспорта ТС должны быть буквами");
    }
}
string vehiclePassportNumber = tbVehiclePassportNumber.Text.Trim();
if (vehiclePassportNumber.Length != 6)
{
    throw new Exception("Номер паспорта ТС должен содержать 6 цифр");
}
foreach (var item in vehiclePassportNumber)
{
    if (!char.IsDigit(item))
    {
        throw new Exception("Номер паспорта ТС должен содержать только цифры");
    }
}
string vehiclePassport = vehiclePassportSeries + vehiclePassportNumber;

if (listNewPhotos.Count == 0)
{
    throw new Exception("Добавьте фотографию автомобиля");
}

Car car = new Car(searchCar.ID, searchCar.Model, searchCar.VIN, registrationPlate, vehiclePassport);

Database.ChangeCarWithPhotos(car, listDeletePhotos, listAddPhotos);

MessageBox.Show("Автомобиль успешно изменён", "", MessageBoxButton.OK,
MessageBoxImage.Information);

tbModel.Clear();
tbVIN.Clear();
tbRegistrationPlate.Clear();
tbVehiclePassportSeries.Clear();
tbVehiclePassportNumber.Clear();

listPhotos.Clear();
listEncodedPhotos.Clear();
listNewPhotos.Clear();
listNewEncodedPhotos.Clear();
listDeletePhotos.Clear();
listAddPhotos.Clear();
BitmapImage bi = new BitmapImage();
bi.BeginInit();
bi.UriSource = new Uri("/InsuranceAgency;component/Assets/Car.jpg", UriKind.RelativeOrAbsolute);
bi.EndInit();
imgCar.Source = bi;
btnLeft.Visibility = Visibility.Hidden;
btnRight.Visibility = Visibility.Hidden;

tbException.Visibility = Visibility.Hidden;
flagSearchCar = false;
}
catch (Exception exp)
{
    tbException.Visibility = Visibility.Visible;
    tbException.Text = exp.Message;
}

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        }

    private void btnDeleteCar_Click(object sender, RoutedEventArgs e)
    {
        try
        {
            if (!flagSearchCar) throw new Exception("Выберите автомобиль");

            System.Windows.Forms.DialogResult result = System.Windows.Forms.MessageBox.Show("Вы уверены, что хотите удалить данный автомобиль", "Удаление",
                System.Windows.Forms.MessageBoxButtons.YesNo,
                System.Windows.Forms.MessageBoxIcon.Information,
                System.Windows.Forms.MessageBoxDefaultButton.Button1,
                System.Windows.Forms.MessageBoxOptions.DefaultDesktopOnly);
            if (result == System.Windows.Forms.DialogResult.Yes)
            {
                Database.DeleteCar(searchCar.ID);

                MessageBox.Show("Автомобиль успешно удалён", "", MessageBoxButton.OK,
                MessageBoxImage.Information);

                tbModel.Clear();
                tbVIN.Clear();
                tbRegistrationPlate.Clear();
                tbVehiclePassportSeries.Clear();
                tbVehiclePassportNumber.Clear();

                listPhotos.Clear();
                listEncodedPhotos.Clear();
                listNewPhotos.Clear();
                listNewEncodedPhotos.Clear();
                listDeletePhotos.Clear();
                listAddPhotos.Clear();
                BitmapImage bi = new BitmapImage();
                bi.BeginInit();
                bi.UriSource = new Uri("/InsuranceAgency;component/Assets/Car.jpg", UriKind.RelativeOrAbsolute);
                bi.EndInit();
                imgCar.Source = bi;
                btnLeft.Visibility = Visibility.Hidden;
                btnRight.Visibility = Visibility.Hidden;

                tbException.Visibility = Visibility.Hidden;
                flagSearchCar = false;
            }
        }
        catch (Exception exp)
        {
            tbException.Visibility = Visibility.Visible;
            tbException.Text = exp.Message;
        }
    }

    private void btnLeft_Click(object sender, RoutedEventArgs e)
    {
        if (currentIndex == 0)
        {
            currentIndex = listNewPhotos.Count - 1;
        }
        else
        {
    
```

Изм.	Лист	№ докум.	Подпись	Дата

```

        currentIndex--;
    }

    imgCar.Source = listNewPhotos[currentIndex];
}

private void btnRight_Click(object sender, RoutedEventArgs e)
{
    if (currentIndex == listNewPhotos.Count - 1)
    {
        currentIndex = 0;
    }
    else
    {
        currentIndex++;
    }

    imgCar.Source = listNewPhotos[currentIndex];
}

private void btnDeleteImage_Click(object sender, RoutedEventArgs e)
{
    try
    {
        if (listNewPhotos.Count == 0)
        {
            throw new Exception("Добавьте фотографию автомобиля");
        }

        System.Windows.Forms.DialogResult result = System.Windows.Forms.MessageBox.Show("Вы уверены, что хотите удалить данную фотографию", "Удаление",
                                         System.Windows.Forms.MessageBoxButtons.YesNo,
                                         System.Windows.Forms.MessageBoxIcon.Information,
                                         System.Windows.Forms.MessageBoxDefaultButton.Button1,
                                         System.Windows.Forms.MessageBoxOptions.DefaultDesktopOnly);
        if (result == System.Windows.Forms.DialogResult.Yes)
        {
            for (var i = 0; i < listEncodedPhotos.Count; i++)
            {
                if (listNewEncodedPhotos[currentIndex] == listEncodedPhotos[i])
                {
                    listDeletePhotos.Add(listPhotos[i]);
                }
            }
            for (var i = 0; i < listAddPhotos.Count; i++)
            {
                if (listNewEncodedPhotos[currentIndex] == listAddPhotos[i])
                {
                    listAddPhotos.RemoveAt(i);
                }
            }
            listNewPhotos.RemoveAt(currentIndex);
            listNewEncodedPhotos.RemoveAt(currentIndex);

            currentIndex = 0;

            if (listNewPhotos.Count == 1)
            {
                currentIndex = 0;
                btnLeft.Visibility = Visibility.Hidden;
                btnRight.Visibility = Visibility.Hidden;
            }
        }
    }
}

```

Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

```
        }

        if (listNewPhotos.Count == 0)
        {
            currentIndex = 0;
            BitmapImage bi = new BitmapImage();
            bi.BeginInit();
            bi.UriSource = new Uri("/InsuranceAgency;component/Assets/Car.jpg",
UriKind.RelativeOrAbsolute);
            bi.EndInit();
            imgCar.Source = bi;
        }
        else
        {
            imgCar.Source = listNewPhotos[currentIndex];
        }
    }

    catch (Exception exp)
    {
        tbException.Visibility = Visibility.Visible;
        tbException.Text = exp.Message;
    }
}
}
```

Листинг класса ChangeEmployee:

```
namespace InsuranceAgency.Pages
{
    public partial class ChangeEmployee : Page
    {
        public ChangeEmployee()
        {
            InitializeComponent();
        }

        public ChangeEmployee(Employee employee)
        {
            InitializeComponent();

            searchEmployee = employee;
            AddInfoInTb(employee);

            flagSearchEmployee = true;
        }

        private void tbPassportSeries_TextChanged(object sender, TextChangedEventArgs e)
        {
            if (tbPassportSeries.Text.Length == 0)
            {
                tbPassportSeriesHint.Visibility = Visibility.Visible;
            }
            else
            {
                tbPassportSeriesHint.Visibility = Visibility.Hidden;
            }
        }
    }
}
```

<i>Изм.</i>	<i>Лист</i>	<i>№ докум.</i>	<i>Подпись</i>	<i>Дата</i>

МИВУ 09.03.04-8.000 ПЗ

Лисс

142

```

private void tbPassportNumber_TextChanged(object sender, TextChangedEventArgs e)
{
    if (tbPassportNumber.Text.Length == 0)
    {
        tbPassportNumberHint.Visibility = Visibility.Visible;
    }
    else
    {
        tbPassportNumberHint.Visibility = Visibility.Hidden;
    }
}

private void tbPassword_TextChanged(object sender, TextChangedEventArgs e)
{
    if (tbPassword.Text.Length == 0)
    {
        tbPasswordHint.Visibility = Visibility.Visible;
    }
    else
    {
        tbPasswordHint.Visibility = Visibility.Hidden;
    }
}

private void tbSearch_TextChanged(object sender, TextChangedEventArgs e)
{
    if (tbSearch.Text.Length == 0)
    {
        tbSearchHint.Visibility = Visibility.Visible;
    }
    else
    {
        tbSearchHint.Visibility = Visibility.Hidden;
    }
}

bool flagSearchEmployee = false;
Employee searchEmployee;

private void btnSearch_Click(object sender, RoutedEventArgs e)
{
    try
    {
        string search_temp = tbSearch.Text.Trim();
        string search = "";
        if (search_temp == "")
        {
            throw new Exception("Строка поиска пуста");
        }
        foreach (var item in search_temp)
        {
            if (char.IsDigit(item))
            {
                search += item;
            }
        }
    }

    searchEmployee = Database.SearchEmployee(search);

    AddInfoInTb(searchEmployee);

    tbSearch.Text = "";
}

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        flagSearchEmployee = true;
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message, "", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

private void AddInfoInTb(Employee employee)
{
    tbFullName.Text = employee.FullName;
    tbBirthday.Text = employee.Birthday.ToString("d");
    tbTelephone.Text = employee.Telephone;
    tbPassportSeries.Text = "";
    tbPassportNumber.Text = "";
    for (var i = 0; i < 4; i++) tbPassportSeries.Text += employee.Passport[i];
    for (var i = 4; i < 10; i++) tbPassportNumber.Text += employee.Passport[i];
    tbLogin.Text = employee.Login;
    tbPassword.Text = "";
    if (employee.Admin == false)
    {
        cbAdmin.SelectedIndex = 0;
    }
    else
    {
        cbAdmin.SelectedIndex = 1;
    }
    if (employee.Works == true)
    {
        cbWorks.SelectedIndex = 0;
    }
    else
    {
        cbWorks.SelectedIndex = 1;
    }
}

private void btnChangeEmployee_Click(object sender, RoutedEventArgs e)
{
    try
    {
        if (!flagSearchEmployee) throw new Exception("Выберите сотрудника");

        string fullName = tbFullName.Text.Trim();
        if (fullName.Trim() == "")
        {
            throw new Exception("Заполните поле ФИО");
        }

        string telephone = tbTelephone.Text.Trim();
        if (telephone.Trim() == "")
        {
            throw new Exception("Заполните поле Номер телефона");
        }
        if (telephone.Length > 15)
        {
            throw new Exception("Номер телефона не может быть больше 15 символов");
        }
        foreach (var item in telephone)
        {
            if (!char.IsDigit(item))
            {
                throw new Exception("Номер телефона должен содержать только цифры");
            }
        }
    }
}

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        }

        string passportSeries = tbPassportSeries.Text.Trim();
        if (passportSeries.Length != 4)
        {
            throw new Exception("Серия паспорта должна содержать 4 цифры");
        }
        string passportNumber = tbPassportNumber.Text.Trim();
        if (passportNumber.Length != 6)
        {
            throw new Exception("Номер паспорта должен содержать 6 цифр");
        }
        string passport = passportSeries + passportNumber;
        foreach (var item in passport)
        {
            if (!char.IsDigit(item))
            {
                throw new Exception("Серия и номер паспорта должны содержать только цифры");
            }
        }

        string login = tbLogin.Text.Trim();
        if (login.Length < 4 || login.Length > 32)
        {
            throw new Exception("Длина логина должна быть от 4 до 32 символов");
        }
        foreach (var item in telephone)
        {
            if (char.IsNullOrWhiteSpace(item))
            {
                throw new Exception("Логин не может содержать пробелы");
            }
        }

        string password = tbPassword.Text.Trim();
        bool changePassword = false;
        if (tbPassword.Text.Trim() == "")
        {
            password = searchEmployee.Password;
        }
        else
        {
            changePassword = true;
            if (password.Length < 4 || password.Length > 32)
            {
                throw new Exception("Длина пароля должна быть от 4 до 32 символов");
            }
            foreach (var item in telephone)
            {
                if (char.IsNullOrWhiteSpace(item))
                {
                    throw new Exception("Пароль не может содержать пробелы");
                }
            }
        }

        bool admin;
        if (cbAdmin.Text == "")
        {
            throw new Exception("Заполните поле Администратор");
        }
        else if (cbAdmin.Text == "Не администратор")
    
```

Изм.	Лист	№ докум.	Подпись	Дата

```

{
    admin = false;
}
else
{
    admin = true;
}

bool works;
if (cbWorks.Text == "")
{
    throw new Exception("Заполните поле Работает ли");
}
else if (cbWorks.Text == "Не работает")
{
    works = false;
}
else
{
    works = true;
}

Employee employee = new Employee(searchEmployee.ID, fullName, searchEmployee.Birthday,
telephone, passport, login, password, admin, works);

Database.ChangeEmployee(employee, changePassword);

MessageBox.Show("Сотрудник успешно изменён", "", MessageBoxButtons.OK,
MessageBoxImage.Information);

tbFullName.Clear();
tbBirthday.Clear();
tbTelephone.Clear();
tbPassportSeries.Clear();
tbPassportNumber.Clear();
tbLogin.Clear();
tbPassword.Clear();
cbAdmin.Text = "";
cbWorks.Text = "";

tbException.Visibility = Visibility.Hidden;
flagSearchEmployee = false;

if(searchEmployee.Login == Database.Login)
{
    MessageBox.Show("Перезагрузите программу", "", MessageBoxButtons.OK,
MessageBoxImage.Information);
    System.Diagnostics.Process.GetCurrentProcess().Kill();
}
catch (Exception exp)
{
    tbException.Visibility = Visibility.Visible;
    tbException.Text = exp.Message;
}
}

private void btnDeleteEmployee_Click(object sender, RoutedEventArgs e)
{
try
{
    if (!flagSearchEmployee) throw new Exception("Выберите сотрудника");
}

```

Изм.	Лист	№ докум.	Подпись	Дата

```
System.Windows.Forms.DialogResult result = System.Windows.Forms.MessageBox.Show("Вы
уверены, что хотите удалить данного сотрудника", "Удаление",
System.Windows.Forms.MessageBoxButtons.YesNo,
System.Windows.Forms.MessageBoxIcon.Information,
System.Windows.Forms.MessageBoxDefaultButton.Button1,
System.Windows.Forms.MessageBoxOptions.DefaultDesktopOnly);
if (result == System.Windows.Forms.DialogResult.Yes)
{
    Database.DeleteEmployee(searchEmployee.ID);

    MessageBox.Show("Сотрудник успешно удалён", "", MessageBoxButtons.OK,
MessageBoxImage.Information);

    tbFullName.Clear();
    tbBirthday.Clear();
    tbTelephone.Clear();
    tbPassportSeries.Clear();
    tbPassportNumber.Clear();
    tbLogin.Clear();
    tbPassword.Clear();
    cbAdmin.Text = "";
    cbWorks.Text = "";

    tbException.Visibility = Visibility.Hidden;
    flagSearchEmployee = false;

    if (searchEmployee.Login == Database.Login)
    {
        MessageBox.Show("Перезагрузите программу", "", MessageBoxButtons.OK,
MessageBoxImage.Information);
        System.Diagnostics.Process.GetCurrentProcess().Kill();
    }
}
catch (Exception exp)
{
    tbException.Visibility = Visibility.Visible;
    tbException.Text = exp.Message;
}
}
```

Листинг класса ChangePersonAllowedToDoDrive:

```
namespace InsuranceAgency.Pages
{
    public partial class ChangePersonAllowedToDrive : Page
    {
        public ChangePersonAllowedToDrive()
        {
            InitializeComponent();
        }

        public ChangePersonAllowedToDrive(PersonAllowedToDrive personAllowedToDrive)
        {
            InitializeComponent();
        }
    }
}
```

<i>Изм.</i>	<i>Лист</i>	<i>№ докум.</i>	<i>Подпись</i>	<i>Дата</i>

```

searchPersonAllowedToDrive = personAllowedToDrive;
AddInfoInTb(personAllowedToDrive);

    flagSearchPersonAllowedToDrive = true;
}

private void tbSearch_TextChanged(object sender, TextChangedEventArgs e)
{
    if (tbSearch.Text.Length == 0)
    {
        tbSearchHint.Visibility = Visibility.Visible;
    }
    else
    {
        tbSearchHint.Visibility = Visibility.Hidden;
    }
}

bool flagSearchPersonAllowedToDrive = false;
PersonAllowedToDrive searchPersonAllowedToDrive;

private void btnSearch_Click(object sender, RoutedEventArgs e)
{
    try
    {
        string search_temp = tbSearch.Text.Trim();
        string search = "";
        if (search_temp == "")
        {
            throw new Exception("Строка поиска пуста");
        }
        foreach (var item in search_temp)
        {
            if (char.IsDigit(item))
            {
                search += item;
            }
        }
    }

    searchPersonAllowedToDrive = Database.SearchPersonAllowedToDrive(search);

    AddInfoInTb(searchPersonAllowedToDrive);

    tbSearch.Text = "";
    flagSearchPersonAllowedToDrive = true;
}
catch (Exception exp)
{
    MessageBox.Show(exp.Message, "", MessageBoxButton.OK, MessageBoxImage.Error);
}
}

private void AddInfoInTb(PersonAllowedToDrive personAllowedToDrive)
{
    tbFullName.Text = personAllowedToDrive.FullName;
    tbDrivingLicenceSeries.Text = "";
    tbDrivingLicenceNumber.Text = "";
    for (var i = 0; i < 4; i++) tbDrivingLicenceSeries.Text += personAllowedToDrive.DrivingLicence[i];
    for (var i = 4; i < 10; i++) tbDrivingLicenceNumber.Text += personAllowedToDrive.DrivingLicence[i];
}

private void tbDrivingLicenceSeries_TextChanged(object sender, TextChangedEventArgs e)
{

```

Изм.	Лист	№ докум.	Подпись	Дата

```

if (tbDrivingLicenceSeries.Text.Length == 0)
{
    tbDrivingLicenceSeriesHint.Visibility = Visibility.Visible;
}
else
{
    tbDrivingLicenceSeriesHint.Visibility = Visibility.Hidden;
}
}

private void tbDrivingLicenceNumber_TextChanged(object sender, TextChangedEventArgs e)
{
    if (tbDrivingLicenceNumber.Text.Length == 0)
    {
        tbDrivingLicenceNumberHint.Visibility = Visibility.Visible;
    }
    else
    {
        tbDrivingLicenceNumberHint.Visibility = Visibility.Hidden;
    }
}

private void btnChangePersonAllowedToDrive_Click(object sender, RoutedEventArgs e)
{
    try
    {
        if (!flagSearchPersonAllowedToDrive) throw new Exception("Выберите водителя");

        string fullName = tbFullName.Text.Trim();
        if (fullName == "")
        {
            throw new Exception("Заполните поле ФИО");
        }

        string drivingLicenceSeries = tbDrivingLicenceSeries.Text.Trim();
        if (drivingLicenceSeries.Length != 4)
        {
            throw new Exception("Серия водительского удостоверения должна содержать 4 цифры");
        }
        string drivingLicenceNumber = tbDrivingLicenceNumber.Text.Trim();
        if (drivingLicenceNumber.Length != 6)
        {
            throw new Exception("Номер водительского удостоверения должен содержать 6 цифр");
        }
        string drivingLicence = drivingLicenceSeries + drivingLicenceNumber;
        foreach (var item in drivingLicence)
        {
            if (!char.IsDigit(item))
            {
                throw new Exception("Серия и номер водительского удостоверения должны содержать только цифры");
            }
        }
    }

    PersonAllowedToDrive personAllowedToDrive = new
    PersonAllowedToDrive(searchPersonAllowedToDrive.ID, fullName, drivingLicence);

    Database.ChangePersonAllowedToDrive(personAllowedToDrive);

    MessageBox.Show("Водитель успешно изменён", "", MessageBoxButton.OK,
    MessageBoxImage.Information);
}

```

Изм.	Лист	№ докум.	Подпись	Дата

```

tbFullName.Clear();
tbDrivingLicenceSeries.Clear();
tbDrivingLicenceNumber.Clear();

tbException.Visibility = Visibility.Hidden;
flagSearchPersonAllowedToDrive = false;
}
catch (Exception exp)
{
    tbException.Visibility = Visibility.Visible;
    tbException.Text = exp.Message;
}
}

private void btnDeletePersonAllowedToDrive_Click(object sender, RoutedEventArgs e)
{
    try
    {
        if (!flagSearchPersonAllowedToDrive) throw new Exception("Выберите водителя");

        System.Windows.Forms.DialogResult result = System.Windows.Forms.MessageBox.Show("Вы
уверены, что хотите удалить данного водителя", "Удаление",
                                         System.Windows.Forms.MessageBoxButtons.YesNo,
                                         System.Windows.Forms.MessageBoxIcon.Information);

        System.Windows.Forms.MessageBoxDefaultButton.Button1,
        System.Windows.Forms.MessageBoxOptions.DefaultDesktopOnly);
        if (result == System.Windows.Forms.DialogResult.Yes)
        {
            Database.DeletePersonAllowedToDrive(searchPersonAllowedToDrive.ID);

            MessageBox.Show("Водитель успешно удалён", "", MessageBoxButton.OK,
                           MessageBoxIcon.Information);

            tbFullName.Clear();
            tbDrivingLicenceSeries.Clear();
            tbDrivingLicenceNumber.Clear();

            tbException.Visibility = Visibility.Hidden;
            flagSearchPersonAllowedToDrive = false;
        }
    }
    catch (Exception exp)
    {
        tbException.Visibility = Visibility.Visible;
        tbException.Text = exp.Message;
    }
}
}

```

Листинг класса ChangePolicy:

```
namespace InsuranceAgency.Pages  
{  
    public partial class ChangePolicy : Page  
    {
```

<i>Изм.</i>	<i>Лист</i>	<i>№ докум.</i>	<i>Подпись</i>	<i>Дата</i>

МИВУ 09.03.04-8.000 ПЗ

Лист

150

```

Struct.Policy Policy;

List<PersonAllowedToDrive> listPersons = new List<PersonAllowedToDrive>();
List<PersonAllowedToDrive> listNewPersons = new List<PersonAllowedToDrive>();
List<PersonAllowedToDrive> listDeletePersons = new List<PersonAllowedToDrive>();
List<PersonAllowedToDrive> listAddPersons = new List<PersonAllowedToDrive>();

public ChangePolicy(int policyID)
{
    InitializeComponent();

    Policy = Database.SearchPolicyID(policyID);
    AddInfoInTb(Policy);
}

private void AddInfoInTb(Struct.Policy policy)
{
    tbInsuranceType.Text = policy.InsuranceType;
    tbInsurancePremium.Text = policy.InsurancePremium.ToString();
    tbInsuranceAmount.Text = policy.InsuranceAmount.ToString();
    tbDateOfConclusion.Text = policy.DateOfConclusion.ToString("d");
    dpExpirationDate.SelectedDate = policy.ExpirationDate;
    tbVIN.Text = Database.SearchCarID(policy.CarID).VIN;
    tbEmployee.Text = Database.SearchEmployeeID(policy.EmployeeID).FullName;

    List<Connection> connections = Database.SearchConnection(policy.ID);
    foreach (var item in connections)
    {
        PersonAllowedToDrive personAllowedToDrive =
        Database.SearchPersonAllowedToDriveID(item.PersonAllowedToDriveID);
        listPersons.Add(personAllowedToDrive);
        listNewPersons.Add(personAllowedToDrive);
        cbPersonsAllowedToDrive.Items.Add(personAllowedToDrive.FullName);
    }
    cbPersonsAllowedToDrive.Text = "";
    tbPersonsAllowedToDriveHint.Visibility = Visibility.Visible;
}

private void cbPersonsAllowedToDrive_GotFocus(object sender, RoutedEventArgs e)
{
    tbPersonsAllowedToDriveHint.Visibility = Visibility.Hidden;
}

private void cbPersonsAllowedToDrive_LostFocus(object sender, RoutedEventArgs e)
{
    if (cbPersonsAllowedToDrive.Text.Length == 0)
    {
        tbPersonsAllowedToDriveHint.Visibility = Visibility.Visible;
    }
}

private void btnChangePolicy_Click(object sender, RoutedEventArgs e)
{
    try
    {
        string insurancePremium_temp = tbInsurancePremium.Text.Trim();
        if (insurancePremium_temp == "")
        {
            throw new Exception("Заполните поле Страховая премия");
        }
        int insurancePremium = 0;
        try { insurancePremium = Convert.ToInt32(insurancePremium_temp); }
        catch { throw new Exception("Страховая премия должна быть целым числом"); }
    }
}

```

Изм.	Лист	№ докум.	Подпись	Дата

```

DateTime expirationDate = Convert.ToDateTime(dpExpirationDate.Text);
if(expirationDate < Policy.DateOfConclusion)
{
    throw new Exception("Дата окончания действия не может быть меньше даты заключения");
}
DateTime maxDate = Database.SearchInsuranceEventMaxDate(Policy.ID);
if (maxDate != DateTime.MinValue)
{
    if(expirationDate < maxDate)
    {
        throw new Exception("Дата окончания действия не может быть меньше даты последнего
страхового случая");
    }
}

Struct.Policy policyChange = new Struct.Policy(Policy.ID, Policy.InsuranceType, insurancePremium,
Policy.InsuranceAmount, Policy.DateOfConclusion, expirationDate, Policy.PolicyholderID, Policy.CarID,
Policy.EmployeeID);

if (listNewPersons.Count == 0)
{
    throw new Exception("Список лиц, допущенных к управлению пуст");
}

Database.ChangePolicyWithConnections(policyChange, listDeletePersons, listAddPersons);
MessageBox.Show("Полис успешно изменён", "", MessageBoxButton.OK,
MessageBoxImage.Information);

this.NavigationService.Navigate(new Pages.Policy(Policy.PolicyholderID));
}
catch (Exception exp)
{
    tbException.Visibility = Visibility.Visible;
    tbException.Text = exp.Message;
}
}

private void btnAddPersonAllowedToDrive_Click(object sender, RoutedEventArgs e)
{
try
{
    PersonAllowedToDrive personAllowedToDrive =
Database.SearchPersonAllowedToDrive(cbPersonsAllowedToDrive.Text.Trim());
    foreach (var item in listNewPersons)
    {
        if (item.ID == personAllowedToDrive.ID)
        {
            throw new Exception("Данный водитель уже добавлен");
        }
    }
    listNewPersons.Add(personAllowedToDrive);
    listAddPersons.Add(personAllowedToDrive);
    cbPersonsAllowedToDrive.Items.Add(personAllowedToDrive.FullName);

    MessageBox.Show("Водитель добавлен", "", MessageBoxButton.OK, MessageBoxImage.Information);
    cbPersonsAllowedToDrive.Text = "";
    tbPersonsAllowedToDriveHint.Visibility = Visibility.Visible;
}
catch (Exception exp)
{
    tbException.Visibility = Visibility.Visible;
    tbException.Text = exp.Message;
}
}

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        }

    private void btnDeletePersonAllowedToDrive_Click(object sender, RoutedEventArgs e)
    {
        try
        {
            System.Windows.Forms.DialogResult result = System.Windows.Forms.MessageBox.Show("Вы уверены, что хотите удалить данного водителя", "Удаление",
                System.Windows.Forms.MessageBoxButtons.YesNo,
                System.Windows.Forms.MessageBoxIcon.Information,
                System.Windows.Forms.MessageBoxDefaultButton.Button1,
                System.Windows.Forms.MessageBoxOptions.DefaultDesktopOnly);
            if (result == System.Windows.Forms.DialogResult.Yes)
            {
                int index = cbPersonsAllowedToDrive.SelectedIndex;

                try
                {
                    if (cbPersonsAllowedToDrive.Text == listNewPersons[index].FullName)
                    {
                        for (var i = 0; i < listPersons.Count; i++)
                        {
                            if (listNewPersons[index] == listPersons[i])
                            {
                                listDeletePersons.Add(listPersons[i]);
                            }
                        }
                        for (var i = 0; i < listAddPersons.Count; i++)
                        {
                            if (listNewPersons[index] == listAddPersons[i])
                            {
                                listAddPersons.RemoveAt(i);
                            }
                        }
                    }
                    listNewPersons.RemoveAt(index);
                }
                else
                {
                    throw new Exception("Данный водитель не существует в списке добавленных водителей");
                }
            }
            catch { throw new Exception("Данный водитель не существует в списке добавленных водителей"); }

            cbPersonsAllowedToDrive.Items.Clear();
            foreach (var item in listNewPersons)
            {
                cbPersonsAllowedToDrive.Items.Add(item.FullName);
            }

            MessageBox.Show("Водитель удалён", "", MessageBoxButton.OK, MessageBoxIcon.Information);
            cbPersonsAllowedToDrive.Text = "";
            tbPersonsAllowedToDriveHint.Visibility = Visibility.Visible;
        }
    }
    catch (Exception exp)
    {
        tbException.Visibility = Visibility.Visible;
        tbException.Text = exp.Message;
    }
}

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        }

    private void btnBack_Click(object sender, RoutedEventArgs e)
    {
        this.NavigationService.Navigate(new Pages.Policy(Policy.PolicyholderID));
    }

    private void btnInsuranceEvents_Click(object sender, RoutedEventArgs e)
    {
        this.NavigationService.Navigate(new Pages.InsuranceEvents(Policy));
    }

    private void btnExtendPolicy_Click(object sender, RoutedEventArgs e)
    {
        this.NavigationService.Navigate(new Pages.AddPolicy(Policy));
    }
}

```

Листинг класса ChangePolicyholder:

```

namespace InsuranceAgency.Pages
{
    public partial class ChangePolicyholder : Page
    {
        public ChangePolicyholder()
        {
            InitializeComponent();
        }

        public ChangePolicyholder(Policyholder policyholder)
        {
            InitializeComponent();

            searchPolicyholder = policyholder;
            AddInfoInTb(policyholder);

            flagSearchPolicyholder = true;
        }

        private void tbPassportSeries_TextChanged(object sender, TextChangedEventArgs e)
        {
            if (tbPassportSeries.Text.Length == 0)
            {
                tbPassportSeriesHint.Visibility = Visibility.Visible;
            }
            else
            {
                tbPassportSeriesHint.Visibility = Visibility.Hidden;
            }
        }

        private void tbPassportNumber_TextChanged(object sender, TextChangedEventArgs e)
        {
            if (tbPassportNumber.Text.Length == 0)
            {
                tbPassportNumberHint.Visibility = Visibility.Visible;
            }
            else

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        {
            tbPassportNumberHint.Visibility = Visibility.Hidden;
        }
    }

private void tbSearch_TextChanged(object sender, TextChangedEventArgs e)
{
    if (tbSearch.Text.Length == 0)
    {
        tbSearchHint.Visibility = Visibility.Visible;
    }
    else
    {
        tbSearchHint.Visibility = Visibility.Hidden;
    }
}

bool flagSearchPolicyholder = false;
Policyholder searchPolicyholder;

private void btnSearch_Click(object sender, RoutedEventArgs e)
{
    try
    {
        string search_temp = tbSearch.Text.Trim();
        string search = "";
        if (search_temp == "")
        {
            throw new Exception("Строка поиска пуста");
        }
        foreach (var item in search_temp)
        {
            if (char.IsDigit(item))
            {
                search += item;
            }
        }
    }

    searchPolicyholder = Database.SearchPolicyholder(search);

    AddInfoInTb(searchPolicyholder);

    tbSearch.Text = "";
    flagSearchPolicyholder = true;
}
catch (Exception exp)
{
    MessageBox.Show(exp.Message, "", MessageBoxButton.OK, MessageBoxImage.Error);
}
}

private void AddInfoInTb(Policyholder policyholder)
{
    tbFullName.Text = policyholder.FullName;
    tbBirthday.Text = policyholder.Birthday.ToString("d");
    tbTelephone.Text = policyholder.Telephone;
    tbPassportSeries.Text = "";
    tbPassportNumber.Text = "";
    for (var i = 0; i < 4; i++) tbPassportSeries.Text += policyholder.Passport[i];
    for (var i = 4; i < 10; i++) tbPassportNumber.Text += policyholder.Passport[i];
}

private void btnChangePolicyholder_Click(object sender, RoutedEventArgs e)

```

Изм.	Лист	№ докум.	Подпись	Дата

```

{
try
{
    if (!flagSearchPolicyholder) throw new Exception("Выберите страхователя");

    string fullName = tbFullName.Text.Trim();
    if (fullName == "")
    {
        throw new Exception("Заполните поле ФИО");
    }

    string telephone = tbTelephone.Text.Trim();
    if (telephone == "")
    {
        throw new Exception("Заполните поле Номер телефона");
    }
    if (telephone.Length > 15)
    {
        throw new Exception("Номер телефона не может быть больше 15 символов");
    }
    foreach (var item in telephone)
    {
        if (!char.IsDigit(item))
        {
            throw new Exception("Номер телефона должен содержать только цифры");
        }
    }

    string passportSeries = tbPassportSeries.Text.Trim();
    if (passportSeries.Length != 4)
    {
        throw new Exception("Серия паспорта должна содержать 4 цифры");
    }
    string passportNumber = tbPassportNumber.Text.Trim();
    if (passportNumber.Length != 6)
    {
        throw new Exception("Номер паспорта должен содержать 6 цифр");
    }
    string passport = passportSeries + passportNumber;
    foreach (var item in passport)
    {
        if (!char.IsDigit(item))
        {
            throw new Exception("Серия и номер паспорта должны содержать только цифры");
        }
    }
}

```

Policyholder policyholder = new Policyholder(searchPolicyholder.ID, fullName,
searchPolicyholder.Birthday, telephone, passport);

Database.ChangePolicyholder(policyholder);

MessageBox.Show("Страхователь успешно изменён", "", MessageBoxButtons.OK,
MessageBoxImage.Information);

tbFullName.Clear();
tbBirthday.Clear();
tbTelephone.Clear();
tbPassportSeries.Clear();
tbPassportNumber.Clear();

tbException.Visibility = Visibility.Hidden;

Изм.	Лист	№ докум.	Подпись	Дата

```

        flagSearchPolicyholder = false;
    }
    catch (Exception exp)
    {
        tbException.Visibility = Visibility.Visible;
        tbException.Text = exp.Message;
    }
}

private void btnDeletePolicyholder_Click(object sender, RoutedEventArgs e)
{
    try
    {
        if (!flagSearchPolicyholder) throw new Exception("Выберите страхователя");

        System.Windows.Forms.DialogResult result = System.Windows.Forms.MessageBox.Show("Вы уверены, что хотите удалить данного страхователя", "Удаление",
            System.Windows.Forms.MessageBoxButtons.YesNo,
            System.Windows.Forms.MessageBoxIcon.Information);

        System.Windows.Forms.MessageBoxDefaultButton.Button1,
        System.Windows.Forms.MessageBoxOptions.DefaultDesktopOnly);
        if (result == System.Windows.Forms.DialogResult.Yes)
        {
            Database.DeletePolicyholder(searchPolicyholder.ID);

            MessageBox.Show("Страхователь успешно удалён", "", MessageBoxButton.OK,
            MessageBoxImage.Information);

            tbFullName.Clear();
            tbBirthday.Clear();
            tbTelephone.Clear();
            tbPassportSeries.Clear();
            tbPassportNumber.Clear();

            tbException.Visibility = Visibility.Hidden;
            flagSearchPolicyholder = false;
        }
    }
    catch (Exception exp)
    {
        tbException.Visibility = Visibility.Visible;
        tbException.Text = exp.Message;
    }
}
}

```

Листинг класса Policy:

```
namespace InsuranceAgency.Pages  
{  
    public partial class Policy : Page  
    {  
        public Policy()  
        {  
            InitializeComponent();  
        }  
    }  
}
```

<i>Изм.</i>	<i>Лист</i>	<i>№ докум.</i>	<i>Подпись</i>	<i>Дата</i>

МИВУ 09.03.04-8.000 ПЗ

Лист

157

```

public Policy(int policyholderID)
{
    InitializeComponent();

    searchPolicyholder = Database.SearchPolicyholderID(policyholderID);
    AddDataInDG();
    btnAddPolicy.Visibility = Visibility.Visible;
}

private void tbSearch_TextChanged(object sender, TextChangedEventArgs e)
{
    if (tbSearch.Text.Length == 0)
    {
        tbSearchHint.Visibility = Visibility.Visible;
    }
    else
    {
        tbSearchHint.Visibility = Visibility.Hidden;
    }
}

Policyholder searchPolicyholder;

private void btnSearch_Click(object sender, RoutedEventArgs e)
{
    try
    {
        string search_temp = tbSearch.Text.Trim();
        string search = "";
        if (search_temp == "")
        {
            throw new Exception("Строка поиска пуста");
        }
        foreach (var item in search_temp)
        {
            if (char.IsDigit(item))
            {
                search += item;
            }
        }
    }

    searchPolicyholder = Database.SearchPolicyholder(search);

    AddDataInDG();

    tbSearch.Text = "";
    btnAddPolicy.Visibility = Visibility.Visible;
}
catch (Exception exp)
{
    MessageBox.Show(exp.Message, "", MessageBoxButton.OK, MessageBoxImage.Error);
}
}

private void AddDataInDG()
{
    DataTable dt1 = DTColumnPolicyholder();
    AddRow(dt1, searchPolicyholder);
    DataView view1 = new DataView(dt1);
    dgPolicyholders.ItemsSource = view1;

    List<Struct.Policy> listPolicy = Database.SearchPolicy(searchPolicyholder.ID);
}

```

Изм.	Лист	№ докум.	Подпись	Дата

```

DataTable dt = DTColumn();
foreach (var item in listPolicy)
{
    AddRow(dt, item);
}

DataView view = new DataView(dt);
dgPolicies.ItemsSource = view;
}

private DataTable DTColumn()
{
    DataTable dt = new DataTable();

    DataColumn column;

    column = new DataColumn();
    column.DataType = Type.GetType("System.String");
    column.ColumnName = "ID";
    dt.Columns.Add(column);

    column = new DataColumn();
    column.DataType = Type.GetType("System.String");
    column.ColumnName = "InsuranceType";
    dt.Columns.Add(column);

    column = new DataColumn();
    column.DataType = Type.GetType("System.String");
    column.ColumnName = "InsurancePremium";
    dt.Columns.Add(column);

    column = new DataColumn();
    column.DataType = Type.GetType("System.String");
    column.ColumnName = "InsuranceAmount";
    dt.Columns.Add(column);

    column = new DataColumn();
    column.DataType = Type.GetType("System.String");
    column.ColumnName = "DateOfConclusion";
    dt.Columns.Add(column);

    column = new DataColumn();
    column.DataType = Type.GetType("System.String");
    column.ColumnName = "ExpirationDate";
    dt.Columns.Add(column);

    column = new DataColumn();
    column.DataType = Type.GetType("System.String");
    column.ColumnName = "VIN";
    dt.Columns.Add(column);

    return dt;
}

private void AddRow(DataTable dt, Struct.Policy policy)
{
    DataRow row;
    row = dt.NewRow();
    row["ID"] = policy.ID;
    row["InsuranceType"] = policy.InsuranceType;
    row["InsurancePremium"] = policy.InsurancePremium.ToString();
    row["InsuranceAmount"] = policy.InsuranceAmount.ToString();
}

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        row["DateOfConclusion"] = policy.DateOfConclusion.ToString("d");
        row["ExpirationDate"] = policy.ExpirationDate.ToString("d");
        Car car = Database.SearchCarID(policy.CarID);
        row["VIN"] = car.VIN;
        dt.Rows.Add(row);
    }

private DataTable DTColumnPolicyholder()
{
    DataTable dt = new DataTable();

    DataColumn column;

    column = new DataColumn();
    column.DataType = Type.GetType("System.String");
    column.ColumnName = "ID";
    dt.Columns.Add(column);

    column = new DataColumn();
    column.DataType = Type.GetType("System.String");
    column.ColumnName = "FullName";
    dt.Columns.Add(column);

    column = new DataColumn();
    column.DataType = Type.GetType("System.String");
    column.ColumnName = "Birthday";
    dt.Columns.Add(column);

    column = new DataColumn();
    column.DataType = Type.GetType("System.String");
    column.ColumnName = "Telephone";
    dt.Columns.Add(column);

    column = new DataColumn();
    column.DataType = Type.GetType("System.String");
    column.ColumnName = "Passport";
    dt.Columns.Add(column);

    return dt;
}

private void AddRow(DataTable dt, Policyholder policyholder)
{
    DataRow row;
    row = dt.NewRow();
    row["ID"] = policyholder.ID;
    row["FullName"] = policyholder.FullName;
    row["Birthday"] = policyholder.Birthday.ToString("d");
    row["Telephone"] = policyholder.Telephone;
    row["Passport"] = policyholder.Passport.Insert(4, " ");
    dt.Rows.Add(row);
}

private void dgPolicies_CurrentCellChanged(object sender, EventArgs e)
{
    DataGrid dg = (sender as DataGrid);
    int row = dg.Items.IndexOf(dg.CurrentCell.Item);

    dg.SelectedIndex = row;
    var selectedRow = (DataRowView)dg.SelectedItem;

    int ID = Convert.ToInt32(selectedRow["ID"].ToString());
    this.NavigationService.Navigate(new Pages.ChangePolicy(ID));
}

```

Изм.	Лист	№ докум.	Подпись	Дата

```
}

private void btnAddPolicy_Click(object sender, RoutedEventArgs e)
{
    this.NavigationService.Navigate(new Pages.AddPolicy(searchPolicyholder.ID));
}

private void dgPolicyholders_CurrentCellChanged(object sender, EventArgs e)
{
    DataGrid dg = (sender as DataGrid);
    int row = dg.Items.IndexOf(dg.CurrentCell.Item);

    dg.SelectedIndex = row;
    var selectedRow = (DataRowView)dg.SelectedItem;

    this.NavigationService.Navigate(new Pages.ChangePolicyholder(searchPolicyholder));
}
}
```

Листинг программы DBImage:

```
namespace InsuranceAgency
{
    public static class DBImage
    {
        public static string Encode(BitmapImage image)
        {
            string base64 = Convert.ToBase64String(CopyImageToByteArray(image));
            return base64;
        }

        private static byte[] CopyImageToByteArray(BitmapImage image)
        {
            using (MemoryStream memoryStream = new MemoryStream())
            {
                ConvertToImage(image).Save(memoryStream, ImageFormat.Png);
                return memoryStream.ToArray();
            }
        }

        private static Image ConvertToImage(BitmapImage bitmapImage)
        {
            MemoryStream ms = new MemoryStream();
            PngBitmapEncoder encoder = new PngBitmapEncoder();
            encoder.Frames.Add(BitmapFrame.Create(bitmapImage));
            encoder.Save(ms);

            Bitmap bmp = new Bitmap(ms);
            Image image = bmp;
            return image;
        }
    }

    public static BitmapImage Decode(string base64)
    {
        Image image = GetImageFromByteArray(base64);
        return ConvertToBitmapImage(image);
    }

    private static Image GetImageFromByteArray(string base64)
    {
```

<i>Изм.</i>	<i>Лист</i>	<i>№ докум.</i>	<i>Подпись</i>	<i>Дата</i>

МИВУ 09.03.04-8.000 ПЗ

Лисс

161

```

        var image = Image.FromStream(new MemoryStream(Convert.FromBase64String(base64)));
        return image;
    }
    private static BitmapImage ConvertToBitmapImage(Image img)
    {
        BitmapImage bmImg = new BitmapImage();

        using (MemoryStream memStream2 = new MemoryStream())
        {
            img.Save(memStream2, System.Drawing.Imaging.ImageFormat.Png);
            memStream2.Position = 0;

            bmImg.BeginInit();
            bmImg.CacheOption = BitmapCacheOption.OnLoad;
            bmImg.UriSource = null;
            bmImg.StreamSource = memStream2;
            bmImg.EndInit();
        }

        return bmImg;
    }
}
}

```

Листинг класса Reports:

```

namespace InsuranceAgency.Pages
{
    public partial class Reports : Page
    {
        public Reports()
        {
            InitializeComponent();

            dpDateStart.SelectedDate = DateTime.Now.AddYears(-1);
            dpDateEnd.SelectedDate = DateTime.Now;
        }

        private void btnReports_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                string insuranceType = cbInsuranceType.Text;
                if (insuranceType == "")
                {
                    throw new Exception("Заполните поле Вид страхования");
                }

                DateTime dateStart = Convert.ToDateTime(dpDateStart.Text);
                DateTime dateEnd = Convert.ToDateTime(dpDateEnd.Text);
                if (dateStart >= dateEnd)
                {
                    throw new Exception("Дата начала не может быть больше даты окончания");
                }

                (int CountContracts, int SumContracts, int SumInsuranceEvents) tuple = Database.Reports(insuranceType,
                dateStart, dateEnd);

                tbCountContracts.Text = tuple.CountContracts.ToString();
                tbSumContracts.Text = tuple.SumContracts.ToString();
            }
        }
    }
}

```

Изм.	Лист	№ докум.	Подпись	Дата

```
        tbSumInsuranceEvents.Text = tuple.SumInsuranceEvents.ToString();

        tbException.Visibility = Visibility.Hidden;
    }
    catch (Exception exp)
    {
        tbException.Visibility = Visibility.Visible;
        tbException.Text = exp.Message;
    }
}
}
```

Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------