МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ

ДОНЕЦКОЙ НАРОДНОЙ РЕСПУБЛИКИ

ГОУ ВПО «ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Интеллектуальных систем и программирования

Кафедра "Программная инженерия" им. Л.П. Фельдмана.

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

по курсу “Базы данных”

Тема работы:

“Страховые компании”

Руководители: Щедрин С.В.

Рычка О.В.

Незамова Л.В.

(подпись) (дата)

Разработал: Сергушин О.С.

ст.гр. ПИ-20а (подпись) (дата)

Донецк - 2022

РЕФЕРАТ

Пояснительная записка содержит 107 с., 93 рис., 1 табл., 3 источника, 5 прил.

Объектом исследования данной работы является учет договоров страховых компаний и способы его оптимизации.

Цель работы – проектирование и разработка приложения для оптимизации учета договоров страховых компаний, получение теоретических знаний и практического опыта работы с СУБД, составление запросов на языке SQL.

Результатом работы является созданная реляционная база данных, содержащая информацию для работы страховых компаний, спроектированная в соответствии с требованиями к курсовой работе. Разработанная база данных написана в СУБД PostgreSQL, запросы реализованы средствами языка SQL.

Для разработки системы использована СУБД PostgreSQL 4 и оболочка для нее PGAdmin 4.0; пользовательский интерфейс реализован средствами C# с поддержкой WinForms для .NET Framework 4.7.

БАЗЫ ДАННЫХ, SQL, POSTGRESQL, ТАБЛИЦЫ, ЗАПРОСЫ, C#, СТРАХОВЫЕ КОМПАНИИ

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 6](#_Toc104239834)

[1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 7](#_Toc104239835)

[1.1 Описание предметной области 7](#_Toc104239836)

[1.2 Постановка задачи 8](#_Toc104239837)

[1.2.1 Описание системы 8](#_Toc104239838)

[1.2.2 Описание входных документов 9](#_Toc104239839)

[1.2.3 Описание выходных данных 10](#_Toc104239840)

[1.2.4 Список ограничений 11](#_Toc104239841)

[1.2.5 Основные функции разрабатываемой системы 11](#_Toc104239842)

[2 ОБОСНОВАНИЕ ВЫБОРА СУБД 12](#_Toc104239843)

[2.1 Аргументация в пользу выбранной СУБД 12](#_Toc104239844)

[2.2 Описание возможностей СУБД 12](#_Toc104239845)

[3 ПРОЕКТИРОВАНИЕ СТРУКТУРЫ ПО 14](#_Toc104239846)

[3.1 Обоснование выбора инструментальные средств для написания клиентской части 14](#_Toc104239847)

[3.2 Невизуальные компоненты для работы с данными 15](#_Toc104239848)

[3.3 Визуальные компоненты отображения данных 18](#_Toc104239849)

[4 ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ В ВЫБРАННОЙ СУБД 22](#_Toc104239850)

[4.1 Проектирование концептуальной модели БД 22](#_Toc104239851)

[4.2 Создание таблиц 25](#_Toc104239852)

[4.3 Создание доменов 27](#_Toc104239853)

[4.4 Создание индексов 27](#_Toc104239854)

[4.5 Разработка триггеров 28](#_Toc104239855)

[4.6 Создание представлений 33](#_Toc104239856)

[4.7 Проектирование запросов к базе данных 34](#_Toc104239857)

[4.7.1 Симметричное внутреннее соединение с условием (два запроса с условием отбора по внешнему ключу, два – по дате) 34](#_Toc104239858)

[4.7.2 Симметричное внутреннее соединение без условия (три запроса) 38](#_Toc104239859)

[4.7.3 Левое внешнее соединение 39](#_Toc104239860)

[4.7.4 Правое внешнее соединение 40](#_Toc104239861)

[4.7.5 Запрос на запросе по принципу левого соединения 41](#_Toc104239862)

[4.7.6 Итоговый запрос без условия 42](#_Toc104239863)

[4.7.7 Итоговый запрос без условия c итоговыми данными вида: «всего», «в том числе» 43](#_Toc104239864)

[4.7.8 Итоговые запросы с условием на данные (по значению, по маске, с использованием индекса, без использования индекса) 44](#_Toc104239865)

[4.7.9 Итоговый запрос с условием на группы 48](#_Toc104239866)

[4.7.10 Итоговый запрос с условием на данные и на группы 49](#_Toc104239867)

[4.7.11 Запрос на запросе по принципу итогового запроса 50](#_Toc104239868)

[4.7.12 Запрос с использованием объединения 51](#_Toc104239869)

[4.7.13 Запросы с подзапросами (с использованием in, not in, case, операциями над итоговыми данными) 52](#_Toc104239870)

[4.7.13 Специальные запросы курсового проекта 55](#_Toc104239871)

[5 РАЗРАБОТКА КЛИЕНТСКОГО ПРИЛОЖЕНИЯ 59](#_Toc104239872)

[5.1 Форма авторизации 59](#_Toc104239873)

[5.2 Формы и компоненты для работы с таблицами 59](#_Toc104239874)

[5.3 Формы и компоненты для отображения результатов запросов 64](#_Toc104239875)

[6 ТЕСТИРОВАНИЕ РАЗРАБОТАННОЙ ИНФОРМАЦИОННОЙ СИСТЕМЫ 69](#_Toc104239876)

[ЗАКЛЮЧЕНИЕ 70](#_Toc104239877)

[СПИСОК ЛИТЕРАТУРЫ 71](#_Toc104239878)

[ПРИЛОЖЕНИЕ А. ТЕХНИЧЕСКОЕ ЗАДАНИЕ 72](#_Toc104239879)

[ПРИЛОЖЕНИЕ Б. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 77](#_Toc104239880)

[ПРИЛОЖЕНИЕ В. ЛИСТИНГ СЕРВЕРНОГО ПРИЛОЖЕНИЯ 80](#_Toc104239881)

[ПРИЛОЖЕНИЕ Д. ЛИСТИНГ КЛИЕНТСКОГО ПРИЛОЖЕНИЯ 86](#_Toc104239882)

# ВВЕДЕНИЕ

Целью курсовой работы является проектирование и программная реализация реляционной базы данных PostgreSQL для учета договоров страховых компаний и закрепление практических навыков разработки и проектирования моделей баз данных.

Разработанная в ходе выполнения курсовой работы система может быть использована в жизни для работы с настоящими страховыми компаниями и оптимизации работы их сотрудников.

Одним из способов взаимодействия с базой данных PostgreSQL является графический пользовательский. Среда разработки Microsoft Visual Studio 2019 и язык программирования C# предоставляют достаточные возможности для создания удобных графических интерфейсов.

В ходе курсовой работы необходимо спроектировать модель взаимодействия пользователя с интерфейсом, связать базу данных с разработанным интерфейсом, организовать удобные способы взаимодействия (добавление, удаление, редактирование записей); дополнительно необходимо обеспечить возможность сохранения результатов определенных запросов в форме отчета и подготовить программу генерации тестовых данных для пробного заполнения базы данных псевдослучайными записями.

# 1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

## 1.1 Описание предметной области

Во время работы специалисты страховых компаний формируют множество страховых договоров и взаимодействуют с большим объемом данных. Для оптимизации их работы создаются различные программы с шаблонами договоров. Такое ПО значительно упрощает и ускоряет составление договоров. Однако существует другая проблема – отчетность.

Для начальника управления должны формироваться отчеты, содержащие разные данные и статистику по договорам, сотрудникам и клиентам. В большинстве своем такие отчеты приходится делать вручную. Такой подход является неоптимальным, трудоемким и дает пространство для допущения множества неточностей.

Для решения всех проблем разрабатываются автоматизированные информационные системы – специальное ПО, работающее с централизованным хранилищем данных. Преимущество такого подход: все данные хранятся в единой удаленной базе данных, а сотрудники страховых компаний обращаются к данным через терминалы с графическим интерфейсом. Данный подход позволит разделить предшествующие таблицы на целостные логические структуры (например, разделение таблиц «договоры» и «клиенты»). Это значительно ускорит взаимодействие сотрудников с данными, позволит нескольким пользователям работать в базе одновременно, также такой подход облегчит отчетную деятельность, т.к. каждый сотрудник сможет составлять необходимые отчеты автоматически и используя заданные фильтры.

Вышеизложенные аргументы дают возможность сделать вывод о том, что разработка автоматизированной системы учета договоров страховых компаний является наилучшим способом реализации программной среды для облегчения и оптимизации работы сотрудников страховых компаний.

## 1.2 Постановка задачи

### 1.2.1 Описание системы

Разработанная система ведет учет договоров страховых компаний. Также система содержит информацию о страховых компаниях, их филиалах, сотрудниках, договорах и клиентах.

Для добавления договора страховая компания должна иметь не менее одного филиала, один главным офис, не менее одного сотрудника в филиале и не менее одного клиента. Договор содержит следующую информацию: вид страхования, ФИО сотрудника, ФИО клиента, текст договора, сумму страховки и дату заключения договора.

Основным пользователем системы являются старшие сотрудники. Они вносят данные о заключенных договорах в базу данных. В случае необходимости старший сотрудник должен иметь возможность редактировать данные клиентов (например, если клиент изменил социальное положение), договорах. Также старший сотрудник должен иметь доступ к данным других таблиц для формирования отчетов.

Таким образом, для обеспечения работы страховых компаний старшим сотрудникам должна быть известна полная информация об страховой компании, клиентах, договорах, других сотрудниках и множестве дополнительных таблиц (социальные положения, виды страхования и т.д.). В связи с этим можно охарактеризовать входные данные разрабатываемой системы.

### 1.2.2 Описание входных документов

Основные элементы разрабатываемой базы данных: компании, филиалы, договоры, сотрудники, клиенты.

Информация о компании содержит:

- название компании;

- тип собственности компании;

- номер лицензии;

- фото лицензии;

- дату окончания лицензии;

- город;

- адрес

- телефон

- год открытия

Информация о филиале содержит:

- название филиала;

- компанию, которой принадлежит филиал;

- город;

- статус филиала (главный офис или нет);

- адрес;

- телефон;

- год открытия;

- количество сотрудников.

Информация о договоре содержит:

- вид страхования;

- ФИО сотрудника, заключившего договор;

- ФИО клиента, заключившего договор;

- текст договора;

- сумму страховки;

- дату заключения договора.

Информация о сотруднике содержит:

- ФИО сотрудника;

- филиал, в котором работает сотрудник;

Информация о клиенте содержит:

- ФИО клиента;

- дату рождения клиента;

- социальное положение;

- город;

- адрес;

- телефон.

Страховая компания, через филиалы, может заключать неограниченное количество договоров. Клиент может заключать неограниченное количество договоров.

### 1.2.3 Описание выходных данных

Выходными данными являются диаграммы, построенные на основе статистических показателей текущих данных в базе данных. Эти диаграммы представляют: компании и общее число сотрудников в их филиалах; общее количество клиентов, в том числе по возрастам; количество договоров у компаний по месяцам.

Сверх диаграмм, выходными данными являются различные отчеты, сформированные по конкретным требованиями пользователя. Они сохраняются в формат .xlsx (Excel) и содержат полную информацию по интересующему запросу в удобном для чтения виде. Место сохранения документа Excel выбирается пользователем. Также пользователю предоставляется возможность сразу открыть сохраненный документ.

### 1.2.4 Список ограничений

В базе должны быть предусмотрены следующие ограничения:

- год открытия компании и филиала не больше текущего и не меньше 1850

- клиент старше 18 лет и младше 120 лет

- сумма договора больше 0

- дата заключения контракта не раньше 1 года назад, не позже текущей даты

- у одной компании только один филиал-главный офис

### 1.2.5 Основные функции разрабатываемой системы

- просмотр, добавление, удаление, редактирование записей;

- поиск по конкретному значению;

- группировка по различным критериям;

- генерация диаграмм для текущего состояния базы данных;

- генерация отчетов по избранным запросам.

# 2 ОБОСНОВАНИЕ ВЫБОРА СУБД

## 2.1 Аргументация в пользу выбранной СУБД

Для разработки системы была использована СУБД PostgreSQL. PostgreSQL обеспечивает очень хорошую поддержку стандарта языка SQL и также предоставляет интересные и практически полезные дополнительные возможности. Одним из главных достоинств PostgreSQL является расширяемость. Это означает, например, что пользователь может разработать свои собственные типы данных. Эти типы данных будут обладать всеми свойствами встроенных типов данных и могут быть введены в работу без останова сервера. Кроме того, PostgreSQL является свободно распространяемым продуктом с открытым исходным кодом, который доступен на большом числе платформ [1].

Также в комплект PostgreSQL входит и оболочка PGAdmin. Она представляет графический клиент для работы с сервером, через который в удобном виде можер создавать, удалять, изменять базы данных и управлять ими.

## 2.2 Описание возможностей СУБД

PostgreSQL – это объектно-реляционная СУБД. Благодаря этому осуществляется поддержка пользовательских объектов и их поведения, включая типы данных, функции, операции, домены и индексы. PostgreSQL умеет создавать, хранить и извлекать сложные структуры данных.

PostgreSQL поддерживает обширный список типов данных. Кроме стандартных числовых, с плавающей точкой, текстовых, булевых и других ожидаемых типов данных, PostgreSQL имеет такие типы данных, как: uuid, денежный, перечисляемый, геометрический, бинарный, сетевые адреса, битовые строки, текстовые поиска, xml, json, массивы, композитные типы и диапазоны, а также некоторые внутренние типы для идентификации объектов и местоположения логов.

PostgreSQL исполняет функции на сервере и допускает в них дополнительную логику, например, условные переходы и циклы. Также СУБД допускает использование функций, возвращающих набор записей.

В PostgreSQL можно создавать триггерные процедуры, которые будут вызываться при изменениях данных или событиях в базе данных. Реализация триггеров происходит через функции, у которых типом возвращаемого значения является trigger.

В данной СУБД присутствует поддержка доменов. Домен представляет собой тип данных с дополнительными условиями. Многоразовые повторяющиеся ограничения CHECK заменяются доменами, что упрощает сопровождение базы данных.

PostgreSQL поддерживает несколько типов индексов: B-дерево, хеш, GiST, SP-GiST, GIN и BRIN. Для разных типов индексов применяются разные алгоритмы, ориентированные на определённые типы запросов [2]. Индексом по умолчанию является B-дерево. B-деревья могут работать в условиях на равенство и в проверках диапазонов с данными, которые можно отсортировать в некотором порядке.

PostgreSQL соответствовует стандарту ANSI-SQL:2008, отвечает требованиям ACID имеет ссылочную и транзакционную целостностью. Первичные ключи, ограничивающие и каскадные внешние ключи, уникальные ограничения, ограничения NOT NULL, проверочные ограничения и другие функции обеспечения целостности данных дают уверенность, что только корректные данные будут сохранены.

# 3 ПРОЕКТИРОВАНИЕ СТРУКТУРЫ ПО

## 3.1 Обоснование выбора инструментальные средств для написания клиентской части

Для разработки приложения был выбрана среда разработки Microsoft Visual Studio 2019. Среда Visual Studio 2019 – это универсальный инструмент для разработки программного обеспечения с помощью платформы .NET и языка C# [3].

Язык C# был выбран по причине современности, многофункциональности и гибкости. Также данный язык программирования имеет графический пакет WinForms для .NET Framework 4.7, что обеспечит относительную простоту процесса разработки и дальнейшей поддержки и сопровождения программного продукта.

Сам C# удобен, так как это объектно-ориентированный язык программирования, из чего вытекаю все необходимые следствия, например наследования, множественные реализации интерфейсов, полиморфизм и т.д.

C# поддерживает сборщик мусора. Что позволяет не беспокоится об утечках памяти, если не использовать неуправляемый код.

Данный язык программирования имеет лямбда выражения, который позволяют сократить код и повысить его читаемость.

Также В C# для хранения набора однотипных данных можно использовать

коллекции. Они позволяют динамически изменять свой размер. Также они удобны тем, что некоторые из них представляют из себя готовые реализации стандартных структур данных, таких как список, хеш таблица, стек, очередь. Для работы с коллекциями предусмотренные LINQ выражения. Коллекции и LINQ выражения позволяют манипулировать данными, полученными из базы данных.

## 3.2 Невизуальные компоненты для работы с данными

Основным пакетом для взаимодействия с СУБД PostgreSQL используя язык программирования C# является пакет Npgsql. Данный пакет включает в себя класс NpgsqlConnection и позволяет установить соединение с базой данных PostgreSQL, выполнить команду базы данных, считать результаты, имеет методы анализа значений записи считываемых данных, например GetInt32. Пакет Npgsql предусматривает использование пула команд, благодаря чем можно одновременно обращаться к базе данным разными командами.

Основным элементом для считывания данных с базы данных был выбран класс DataTable. DataTable представляет одну таблицу данных в памяти. В объект данного класса можно поместить сразу всю таблицу-результат команды отправленной в базу данных. На рисунке 3.1 представлена функция выполнения команды и чтения данных с использованием класса NpgsqlConnection из пакета Npgsql и класса DataTable.



Рисунок 3.1 – Функция выполнения запроса и считывая результата

Кроме того, что класс DataTable дает доступ к строкам, столбцам и ячейкам, он еще может вернуть последовательность объектов DataRow. DataRow представляет строку данных в DataTable. С помощью LINQ можно всячески манипулировать полученными данными. Например, на рисунке 3.2 показано как, после выполнения запроса на выборку ID компаний и их названий из таблицы «Компании», происходит распределение полученных данных по столбцам в массивы.

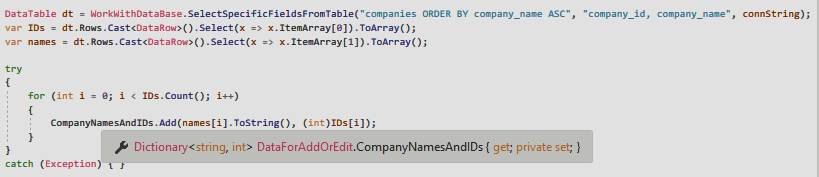


Рисунок 3.2 – Распределение данных из DataTable по массивам

В данном примере элементы массива «IDs» и массива «names» добавляются в объект класса Dictionary. Ключом выступает имя компании, а значением – ее ID.

Решение использовать класс Dictionary было принято во избежание повторного запроса к базе данным за поиском ID выбранной компании. То есть пользователю предоставляется список имен компаний (ключи), после выбора имени компании поиск ID компании происходит внутри объекта класса Dictionary.

У страховых компаний есть необязательной поле «фото копии лицензии». Для работы с изображениями был выбран класс Bitmap. Bitmap удобен для дальнейших преобразований и вывода изображений. Однако в разработанной базе данных изображения хранятся как массив байтов. Поэтому при считывании изображений дополнительно был использован класс MemoryStream.

На рисунке 3.3 приведение функция считывания изображения из базы данных.



Рисунок 3.3 – Функция считывания изображения из базы данных

Для манипуляции данными даты и времени была выбрана структура DateTime. Структура DateTime представляет текущее время, обычно выраженное как дата и время суток. Данная структура предоставляет множество данных о времени и гибкое управление. Из данной структуры удобно извлекать отдельных элементы даты, например, год, месяц или день.

Для сохранения результатов запросов (таблиц) в Excel документ был использован класс ExcelPackage, так как он находит в свободном доступе. На рисунке 3.4 приведена функция сохранения данных в Excel документ.

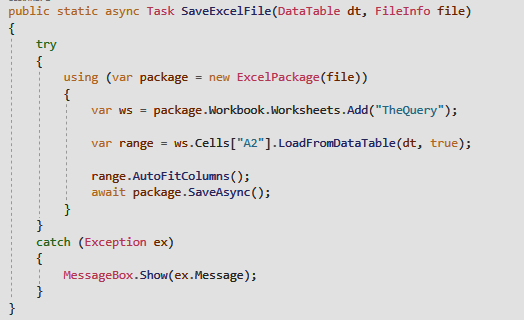


Рисунок 3.4 – Функция сохранения данных в Excel документ

## 3.3 Визуальные компоненты отображения данных

В качестве основного компонента отображения данных таблиц базы данных был выбрал элемент управления DataGridView. Элемент управления DataGridView предоставляет мощный и гибкий способ отображения данных в табличном формате. Элемент управления DataGridView можно использовать для отображения представлений небольшого объема данных только для чтения, либо можно масштабировать его для отображения редактируемого представления очень больших наборов данных.

Значительным аргументом в пользу выбора DataGridView является простая привязка к объекту класс DataTable. Никаких дополнительных преобразований данных не требуется. На рисунке 3.5 изображен пример помещения результата запроса из класса DataTable в элемент управления DataGridView.

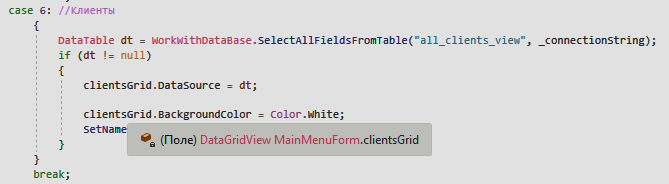


Рисунок 3.5 – Вывод таблицы «Клиенты» в DataGridView

Следующим по важности выбранным визуальным компонентом является TabControl. Элемент TabControl позволяет создать элемент управления с несколькими вкладками. И каждая вкладка будет хранить некоторый набор других элементов управления, как кнопки, текстовые поля и т.д. Каждая вкладка представлена классом TabPage.

TabControl позволяет отобразить каждую таблицу базы данных на одной форме, что существенно упрощает работу.

В качестве основной панели главного меню был выбран элемент ToolStrip. Элемент ToolStrip представляет панель инструментов. В качестве ключевых объектов панели ToolStrip были выбраны:

- ToolStripButton: аналогичен элементу Button. Также имеет событие Click, с помощью которого можно обработать нажатие пользователя на кнопку;

- ToolStripDropDownButton: представляет кнопку, по нажатию на которую открывается выпадающее меню.

Использование панели ToolStrip способствовало разделению основных элементов главного меню, таких как: «Таблицы», «Справочники», «Запросы» и «Другое». На рисунке 3.6 представлена панель главного меню, реализованная через ToolStrip.

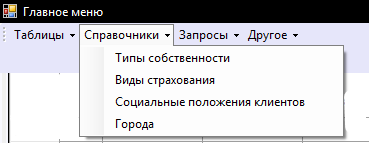


Рисунок 3.6 – Панель главного меню

Для отображения изображений, например, фото лицензии, был выбран элемент PictureBox. PictureBox предназначен для показа изображений и обладает множеством полезных свойств, например, свойство SizeMode, которое может подстроить изображение под размеры PictureBox. На рисунке 3.7 приведен пример добавления фото лицензии в PictureBox.

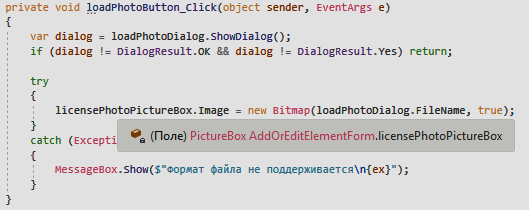


Рисунок 3.7 – Функция добавления фото лицензии компании

Для визуализации диаграмм был выбран класс Chart. Данный класс предоставляет все свойства, методы и события элемента управления Chart Windows. Свойства Chart являются свойствами коллекции, что существенно облегчает добавление данных на диаграмму. На рисунке 3.8 показан пример добавления данных на диаграмму Chart.

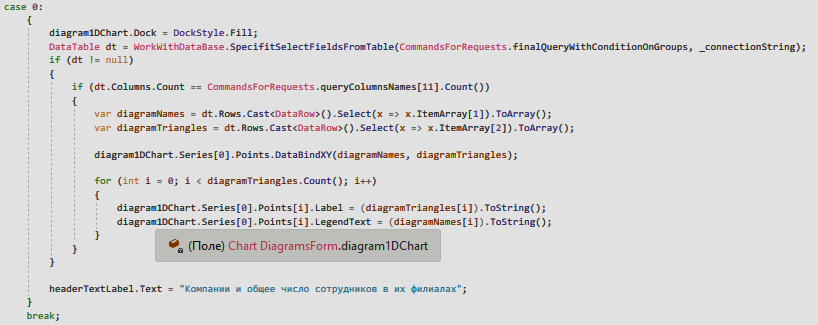


Рисунок 3.8 – Добавление данных на диаграмму

Помимо упомянутых компонентов, также были использованы: текстовые поля (TextBox), кнопки (Button), выпадающие списки (ComboBox), даты (DateTimePicker), чекбоксы (CheckBox) и другие.

# 4 ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ В ВЫБРАННОЙ СУБД

## 4.1 Проектирование концептуальной модели БД

Для создания концептуальной модели необходимо выделить сущности, свойства и связи между ними.

Первая сущность разрабатываемой системы – Компании. Среди свойств этой сущность должны быть: название, тип собственности, номер лицензии, фото лицензии, дата окончания лицензии, город, адрес, телефон, год открытия.

Вторая сущность – Филиалы. Филиалы обладают следующими свойствами: название, компания, которой принадлежит филиал, город, является ли филиал главным офисом, адрес, телефон, год открытия, количество сотрудников.

Третья сущность – Сотрудники. У каждого сотрудника должны быть такие свойства: ФИО и филиал, в котором он работает.

Четвертая сущность – Клиенты. Клиент имеет свойства: ФИО, дата рождения, социальное положение, город, адрес, телефон.

Пятая важная сущность – Договоры. Договор определяется следующими свойствами: вид страхования, ФИО сотрудника, ФИО клиента, текст договора, сумма страховки, дата заключения договора.

Между сущностями Компании и Филиалы установлена связь «один ко многим» соответственно. Между сущностями Филиалы и Сотрудники связь «один ко многим» соответственно. Между сущностями Сотрудники и Договоры, и Клиенты и Договоры связи «один ко многим» соответственно. Сущность Договоры содержит связь «многие ко многим, то есть множество сотрудников ко множеству клиентов.

По итогу получаются основные таблицы: «Компании», «Филиалы», «Сотрудники», «Клиенты», «Договоры».

Данные таблицы дополняются справочниками: «Города», «Типы собственности», «Виды страхования», «Социальные положения», содержащими только код записи и само значения.

Результатом проектирования базы данных является концептуальная модель, представленная на рисунке 4.1.

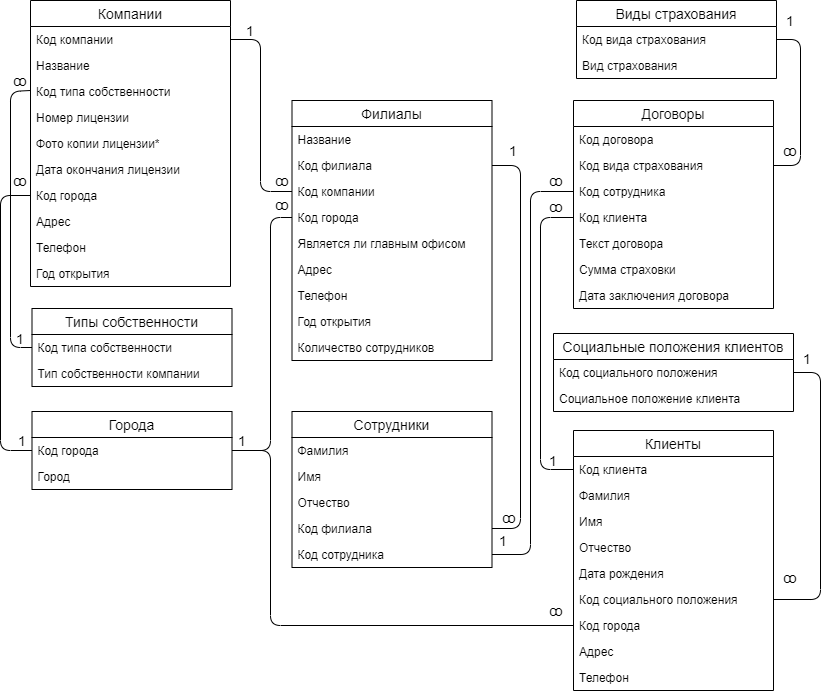


Рисунок 4.1 – Концептуальная модель базы данных

Определения типов данных, которые будут заложены в полях таблиц представлены в таблице 4.1.

В представленной таблице все внешние ключи заканчиваются на «\_id».

Таблица 4.1 – Таблицы и поля базы данных

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица | Поле | Назначение поля | Тип данных |
| companies | company\_id | Код компании | serial |
| company\_name | Название компании | text |
| type\_id | Код типа собственности | integer |
| license\_number | Номер лицензии | integer |
| license\_expiration\_date | Дата окончания лицензии | date |
| city\_id | Код города | integer |
| address | Адрес | text |
| phone\_number | Телефон | varchar(10) |
| opening\_year | Год открытия | op\_year |
| license\_photo | Фото лицензии | bytea |
| branches | branch\_id | Код филиала | serial |
| branch\_name | Название филиала | text |
| company\_id | Код компании | integer |
| city\_id | Код города | integer |
| is\_main | Является ли главным офисом | boolean |
| address | Адрес | text |
| phone\_number | Телефон | varchar(10) |
| opening\_year | Год открытия | op\_year |
| number\_of\_employees | Количество сотрудников | integer |
| employees | employee\_id | Код сотрудника | serial |
| branch\_id | Код филиала | integer |
| surname | Фамилия | text |
| firstname | Имя | text |
| lastname | Отчество | text |
| clients | client\_id | Код клиента | serial |
| surname | Фамилия | text |
| firstname | Имя | text |
| lastname | Отчество | text |
| date\_of\_birth | Дата рождения | date |
| social\_status\_id | Код социального положения | integer |
| city\_id | Код города | integer |
| address | Адрес | text |
| phone\_number | Номер телефона | varchar(10) |

Продолжение таблицы 4.1

|  |  |  |  |
| --- | --- | --- | --- |
| contracts | contract\_id | Код договора | serial |
| type\_of\_insurance\_id | Код вида страхования | integer |
| employee\_id | Код сотрудника | integer |
| client\_id | Код филиала | integer |
| text\_of\_contract | Текст договора | text |
| sum\_of\_contract | Сумма страховки | integer |
| date\_of\_onclusion  \_contract | Дата заключения договора | date |
| property  \_types | type\_id | Код типа собственности | serial |
| property\_type | Тип собственности | text |
| cities | city\_id | Код города | serial |
| city | Город | text |
| types\_of  \_insurance | type\_of\_insurance\_id | Код вида страхования | serial |
| type\_of\_insurance | Вид страхования | text |
| social\_status  \_of\_clients | social\_status\_id | Код социального положения | serial |
| social\_status | Социальное положение | text |

Разработанная модель будет применена на этапе создания таблиц в СУБД PostgreSQL.

## 4.2 Создание таблиц

С помощью ручного ввода SQL-скрипта были созданы таблицы. На рисунках 4.2-4.4 представлены скриншоты создания таблиц. Остальные таблицы созданы аналогичным образом.

Для поддержания ссылочной целостности на все внешние ключи в базе было наложено правило CASCADE. То есть при удалении родительской записи будут удалены и все ее дочерние элементы.

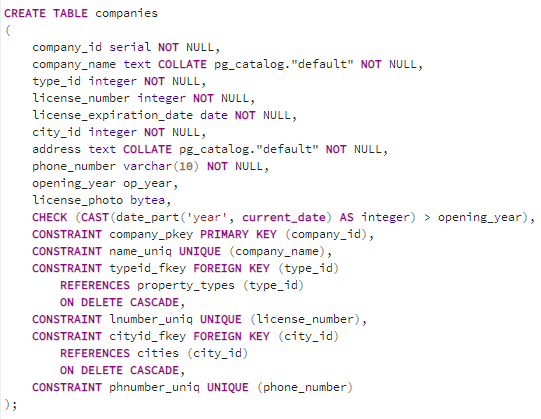


Рисунок 4.2 – Скрипт создания таблицы «Компании»

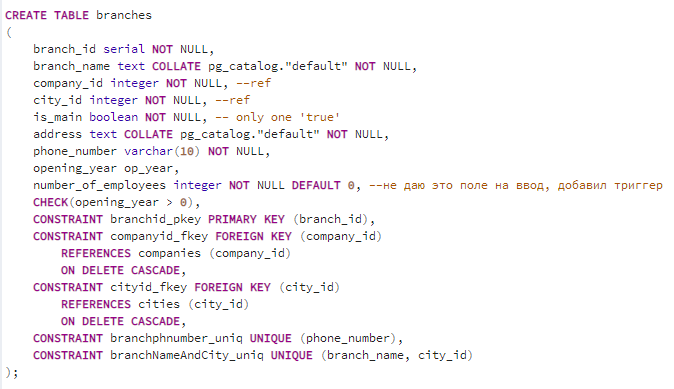


Рисунок 4.3 – Скрипт создания таблицы «Филиалы»

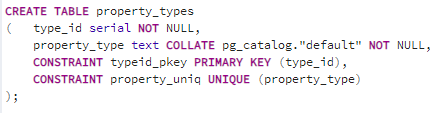


Рисунок 4.4 – Скрипт создания таблицы «Типы собственности»

## 4.3 Создание доменов

Домен представляет собой тип данных с дополнительными условиями, ограничивающими допустимый набор значений.

При создании ограничений для полей таблиц было выделено повторяющееся ограничение, на его основе был создан домен. На рисунке 4.5 приведен домен, который ограничивает данные типа integer на значение не меньше 1850 и не больше значения текущего года. Применяется для полей «год открытия» таблицы «Компании» и таблицы «Филиалы».



Рисунок 4.5 – Скрипт создания таблицы домена «Год открытия»

## 4.4 Создание индексов

Индексы – специальные объекты базы данных, предназначенные в основном для ускорения доступа к данным.

Для получения быстрого доступа было созданы дополнительные индексы «B-дерево» типа для следующих таблиц: «Компании», «Филиалы», «Клиенты» и «Договоры». На рисунке 4.6 приведен скриншот создания указанных индексов.

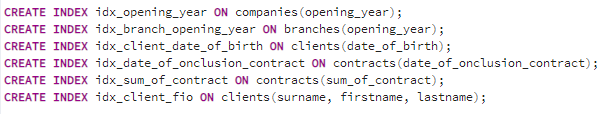


Рисунок 4.6 – Скрипты создания индексов

## 4.5 Разработка триггеров

Триггер определяет операцию, которая должна выполняться при наступлении некоторого события в базе данных.

Для созданной базы данных были разработаны 7 триггеров. На таблицу «Филиалы» 3 триггера для событий: INSERT, DELETE и UPDATE. На таблицу «Сотрудники» 3 триггеры для событий: INSERT, DELETE и UPDATE. На представление «all\_property\_types\_view» (рис. 4.17) 1 триггеры для события INSERT.

На рисунке 4.7 представлен скриншот создания триггера на таблицу «Сотрудники» для события INSERT. Данный триггер увеличивает число сотрудников на 1 в филиале, к которому был добавлен новый сотрудник.

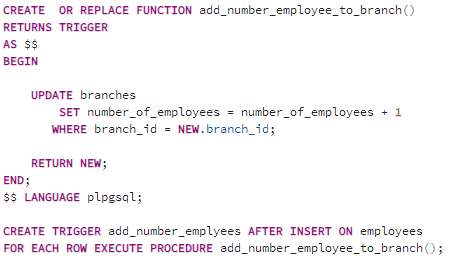


Рисунок 4.7 – Триггер типа INSERT на таблицу «Сотрудники»

На рисунке 4.8 изображено создание триггера на таблицу «Сотрудники» для события DELETE. Данный триггер уменьшает число сотрудников на 1 в филиале, из которого был удален сотрудник.

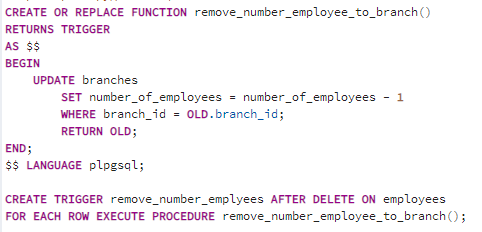


Рисунок 4.8 – Триггер типа DELETE на таблицу «Сотрудники»

На рисунке 4.9 изображено создание триггера на таблицу «Сотрудники» для события UPDATE. Данный триггер увеличивает число сотрудников на 1 в филиале, к которому был присвоен новый сотрудник сотрудников на 1 в филиале, из которого был убран сотрудник, и уменьшает число сотрудников на 1 в филиале, из которого был удален сотрудник.

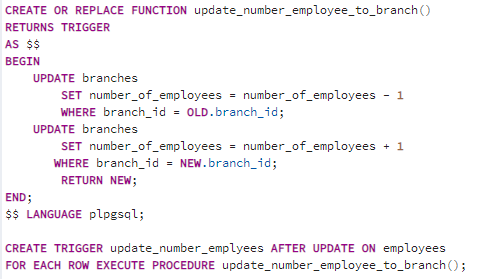


Рисунок 4.9 – Триггер типа UPDATE на таблицу «Сотрудники»

На рисунке 4.10 изображено создание триггера на таблицу «Филиалы» для события INSERT. Данный триггер после добавление нового филиала ставит полю «является ли офис главным» значение «ложь», если количество филиалов у определенной компании больше или равно 1, или значение «истина» в ином случае.

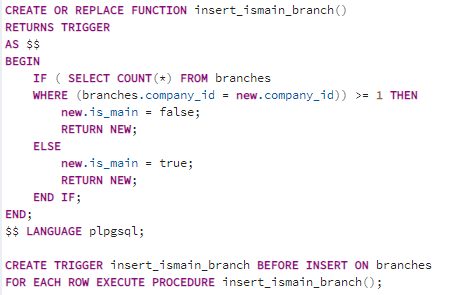


Рисунок 4.10 – Триггер типа INSERT на таблицу «Филиалы»

На рисунке 4.11 изображено создание триггера на таблицу «Филиалы» для события DELETE. Данный триггер после удаления филиала, который является главным офисом, ставит полю филиала определенной компании, который открылся раньше других, «является ли офис главным» значение «истина». В ином случае ничего не меняет.

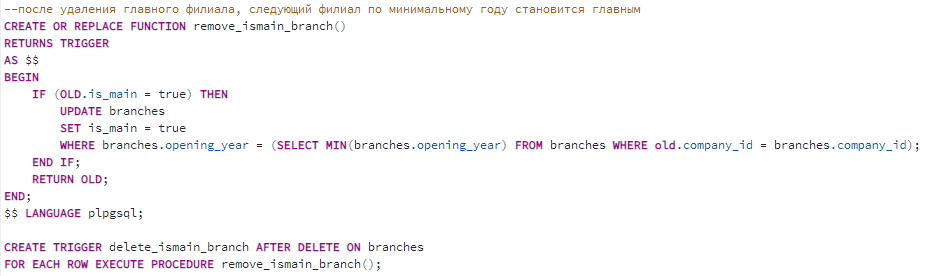


Рисунок 4.11 – Триггер типа DELETE на таблицу «Филиалы»

На рисунке 4.12 приведен скриншот создания триггера на таблицу «Филиалы» для события UPDATE с условием: «если значение кода компании изменилось». Данный триггер после перемещения филиала, который является главным офисом, ставит полю филиала бывшей компании, который открылся раньше других, «является ли офис главным» значение «истина». В ином случае ничего не меняет. Ставит полю «является ли офис главным» значение «ложь», если количество филиалов у новой компании больше или равно 1, или значение «истина» в ином случае.



Рисунок 4.12 – Триггер типа UPDATE на таблицу «Филиалы»

На рисунке 4.13 приведен скриншот создания триггера на представление «all\_property\_types\_view» для события INSERT. Данный триггер, вместо добавления данных в представление, добавляет данный в таблицу «Типы собственности» входные значения, преобразуя их в верхний регистр.

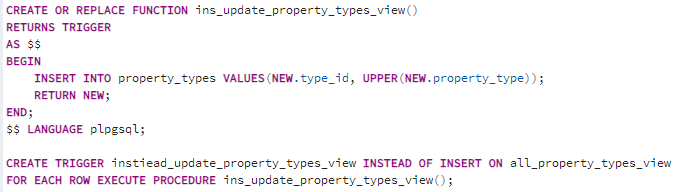


Рисунок 4.13 – Триггер типа INSERT на представление «all\_property\_types\_view»

## 4.6 Создание представлений

Представления – это таблицы чье содержание выбирается или получается из других таблиц.

Было создано 9 представлений, по одному на каждую таблицу. В проекте эти представления используются для вывода основной информации и таблиц в главном меню. Также данные представления используются в некоторых запросах во избежание дублирования кода. На рисунках 4.14-4.17 представлены скриншоты создания некоторых представлений. Остальные представления созданы аналогично.

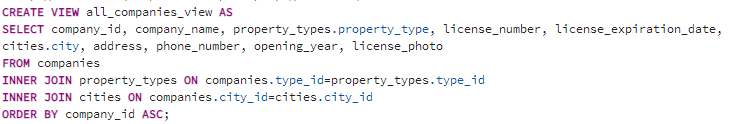


Рисунок 4.14 – Представление для таблицы «Компании»

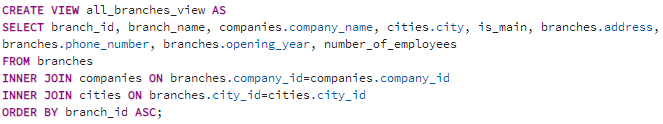


Рисунок 4.15 – Представление для таблицы «Филиалы»

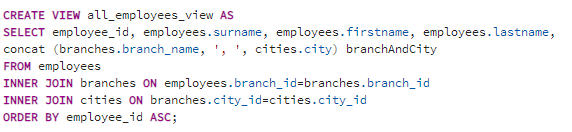


Рисунок 4.16 – Представление для таблицы «Сотрудники»

В данном представлении (рис. 4.16) соединены названия филиалов, в которых работают сотрудники, с городами. Сделано это по причине того, что филиалы одной компании носят одинаковое название.



Рисунок 4.17 – Представление для таблицы «Типы собственности»

## 4.7 Проектирование запросов к базе данных

В требованиях к курсовому проекту необходимо было реализовать несколько сложных запросов на выборку.

### 4.7.1 Симметричное внутреннее соединение с условием (два запроса с условием отбора по внешнему ключу, два – по дате)

1. отбор по внешнему ключу – вывести филиалы в определенном городе, рисунки 4.18-4.19.

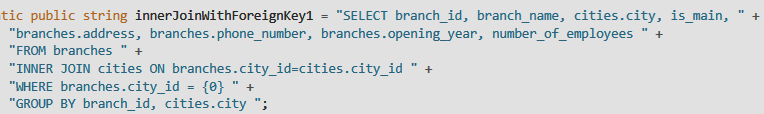


Рисунок 4.18 – Симметричное внутреннее соединение (запрос 1)

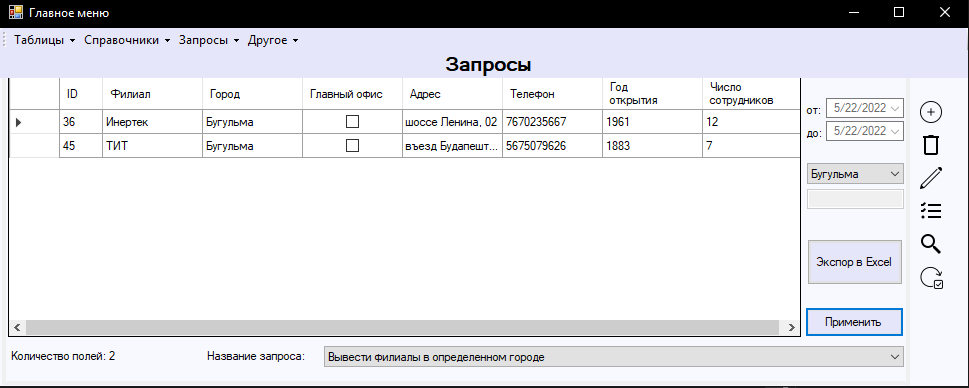


Рисунок 4.19 – Пример выполнения запроса на симметричное внутреннее соединение (запрос 1)

1. отбор по внешнему ключу – вывести сотрудников определенного филиала, рисунки 4.20-4.21.

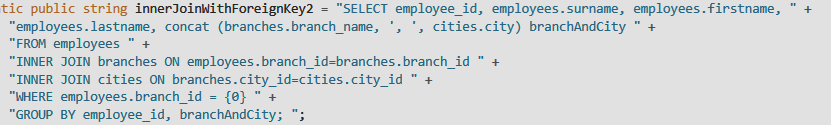


Рисунок 4.20 – Симметричное внутреннее соединение (запрос 2)

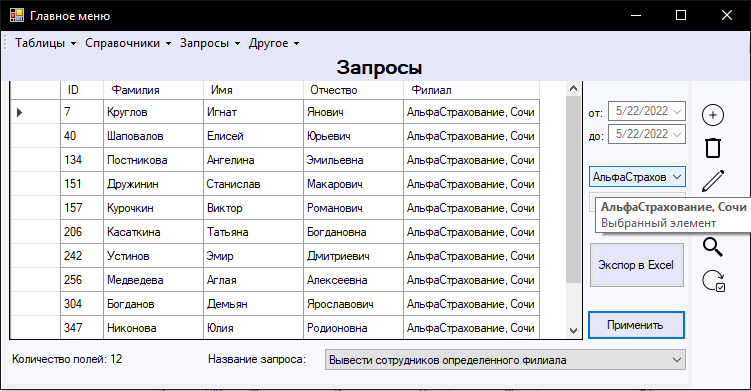


Рисунок 4.21 – Пример выполнения запроса на симметричное внутреннее соединение (запрос 2)

1. отбор по дате – вывести договоры клиента заключенные за определенный период времени, рисунки 4.22-4.23.

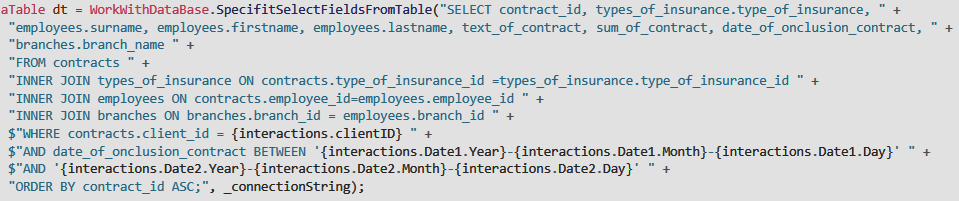


Рисунок 4.22 – Симметричное внутреннее соединение (запрос 3)

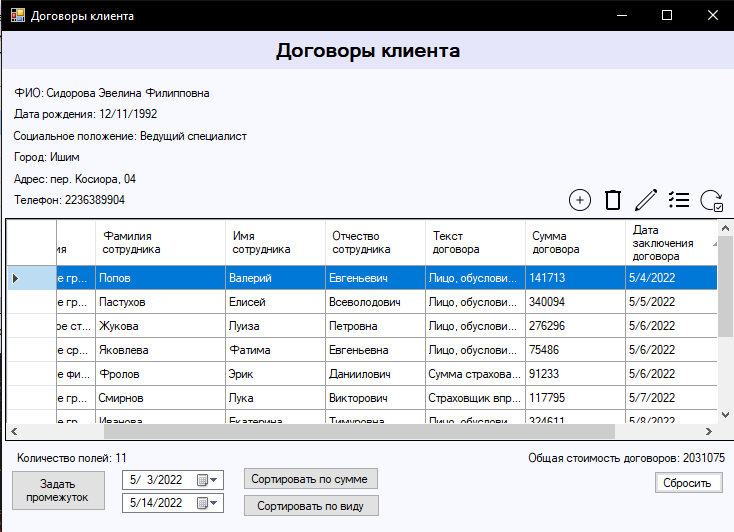


Рисунок 4.23 – Пример выполнения запроса на симметричное внутреннее соединение (запрос 3)

1. отбор по дате – вывести клиентов, младше и старше определенного возраста, рисунки 4.24-4.26.



Рисунок 4.24 – Симметричное внутреннее соединение (запрос 4)

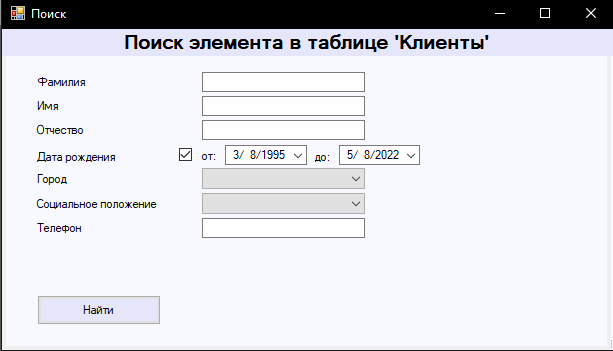


Рисунок 4.25 – Задание нужного периода времени в форме поиска

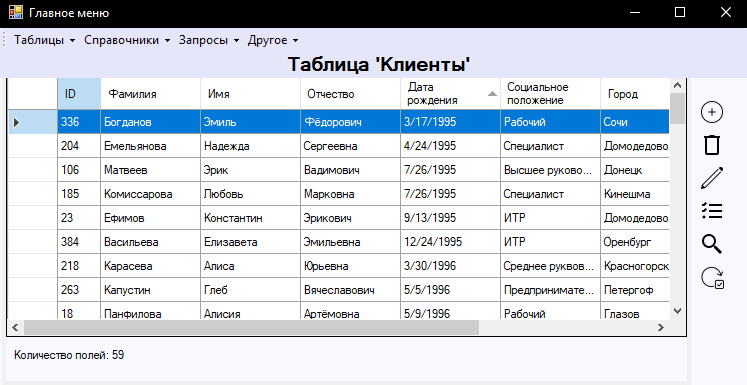


Рисунок 4.26 – Пример выполнения запроса на симметричное внутреннее соединение (запрос 4)

### 4.7.2 Симметричное внутреннее соединение без условия (три запроса)

Примерами запросов на симметричное внутреннее соединение без условия являются созданные ранее представления (рис. 4.14-4.16). На рисунках 4.27-4.29 приведены примеры выполнения первых трех представлений, которые включаю в себя симметричное внутреннее соединение без условия. Происходит вывод полной информации из определенной таблицы.

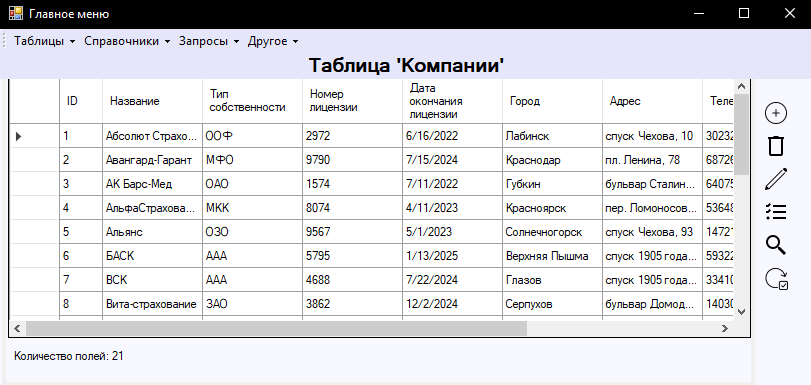


Рисунок 4.27 – Пример выполнения запроса на симметричное внутреннее соединение (запрос 1)



Рисунок 4.28 – Пример выполнения запроса на симметричное внутреннее соединение (запрос 2)

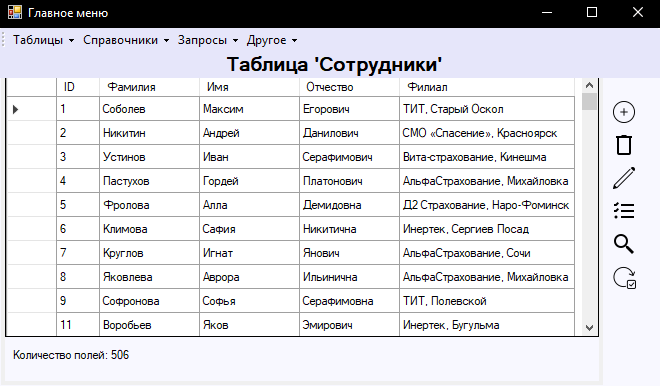


Рисунок 4.29 – Пример выполнения запроса на симметричное внутреннее соединение (запрос 3)

### 4.7.3 Левое внешнее соединение

Вывести информацию о филиалах и названия компаний у тех филиалов, компании чьих имеют фото лицензии, рисунки 4.30-4.31.

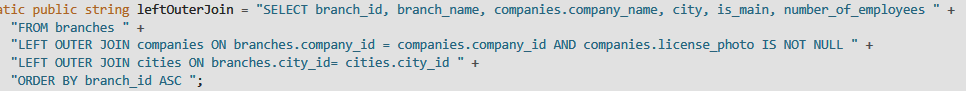


Рисунок 4.30 – Левое внешнее соединение

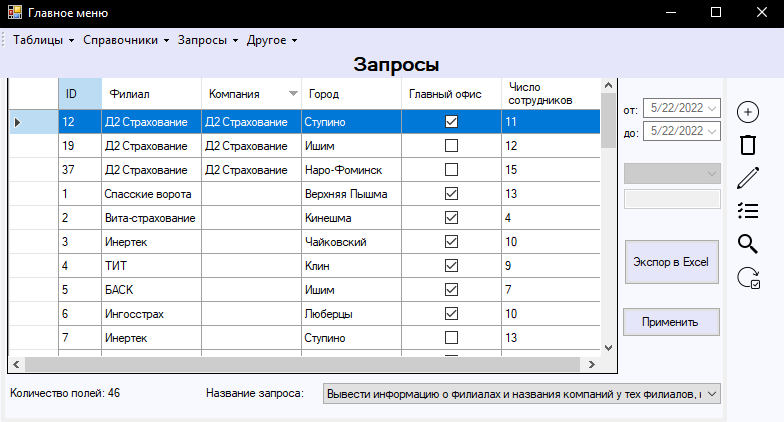


Рисунок 4.31 – Пример выполнения запроса на левое внешнее соединение

### 4.7.4 Правое внешнее соединение

Вывести типы собственности и, у компаний которых имеются фото лицензии, рисунки 4.32-4.33.



Рисунок 4.32 – Правое внешнее соединение

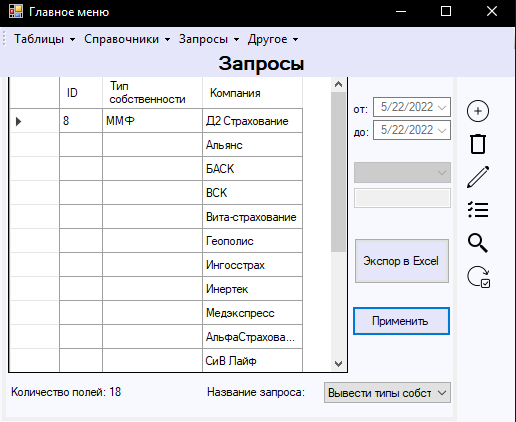


Рисунок 4.33 – Пример выполнения запроса на правое внешнее соединение

### 4.7.5 Запрос на запросе по принципу левого соединения

Вывести компании, у которых есть главный офис, рисунки 4.34-4.35.

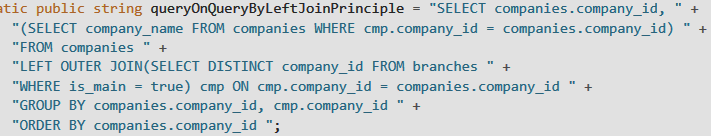


Рисунок 4.34 – Запрос на запросе по принципу левого соединения

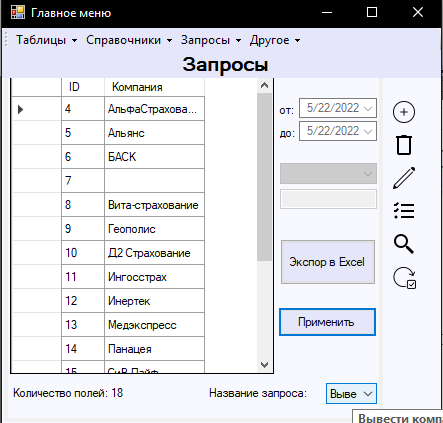


Рисунок 4.35 – Пример выполнения запроса на запросе по принципу левого соединения

### 4.7.6 Итоговый запрос без условия

Вывести количество договоров у клиентов, рисунки 4.36-4.37.

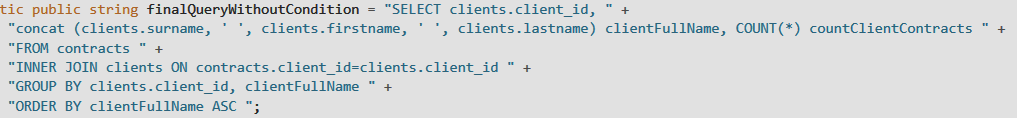


Рисунок 4.36 – Итоговый запрос без условия

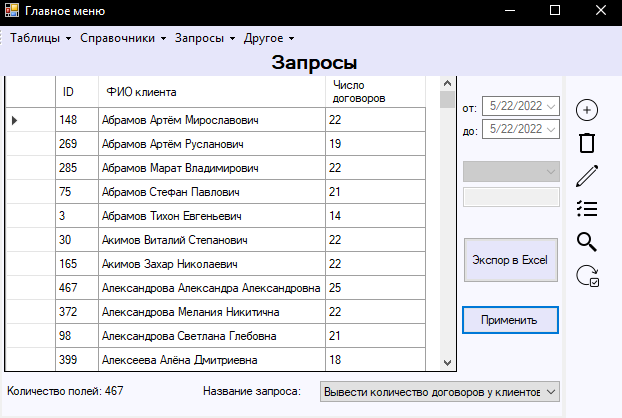


Рисунок 4.37 – Пример выполнения итогового запроса без условия

### 4.7.7 Итоговый запрос без условия c итоговыми данными вида: «всего», «в том числе»

Вывести общее количество клиентов, в том числе по возрастам, рисунки 4.38-4.39.

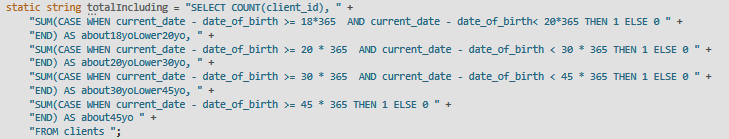
****

Рисунок 4.38 – Итоговый запрос без условия c итоговыми данными вида: «всего», «в том числе»

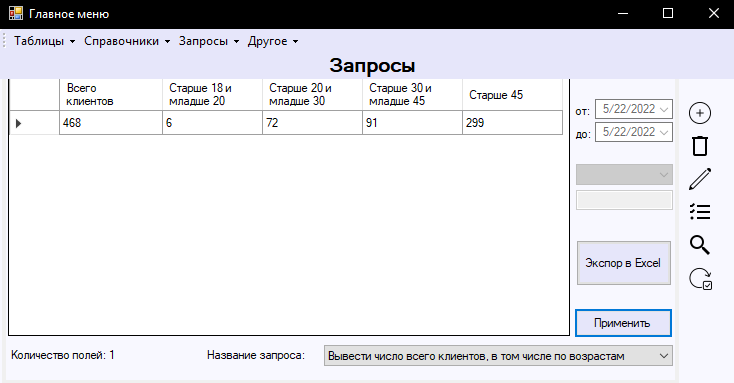


Рисунок 4.39 – Пример выполнения итогового запроса без условия c итоговыми данными вида: «всего», «в том числе»

### 4.7.8 Итоговые запросы с условием на данные (по значению, по маске, с использованием индекса, без использования индекса)

1. по значению – вывести клиентов, у которых общая сумма, на которую заключены договоры больше, чем определенное число, рисунки 4.40-4.41.

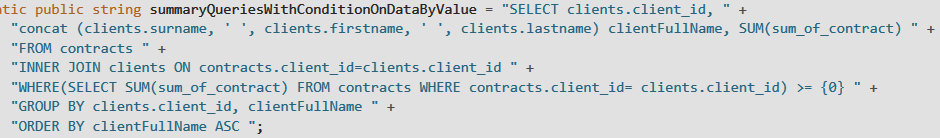


Рисунок 4.40 – Итоговый запрос с условием на данные по значению

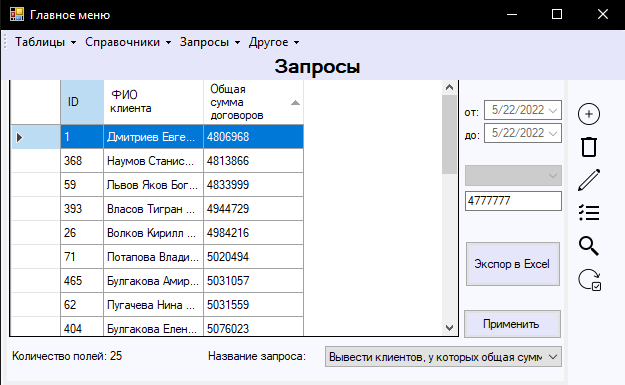


Рисунок 4.41 – Пример выполнения итогового запроса с условием на данные по значению

1. по маске – вывести филиалы, у который номер телефона начинается с определенного значения, рисунки 4.42-4.43.

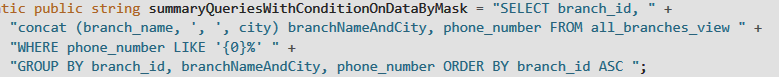


Рисунок 4.42 – Итоговый запрос с условием на данные по маске

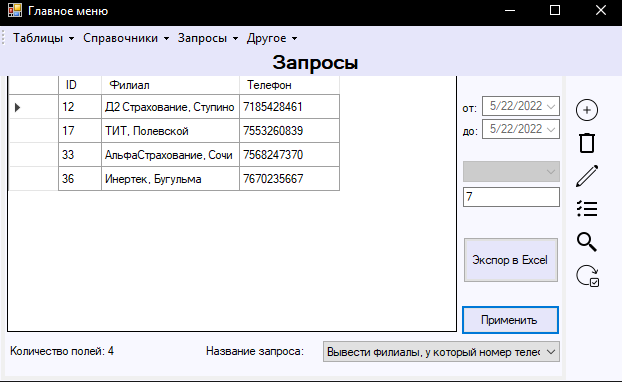


Рисунок 4.43 – Пример выполнения итогового запроса с условием на данные по маске

1. с использованием индекса – вывести клиентов, у которых сумма договора меньше чем 10000, рисунки 4.44-4.46. Ограничение в 10000 гарантирует использование индекса.

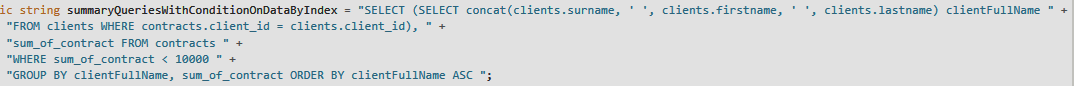


Рисунок 4.44 – Итоговый запрос с использованием индекса

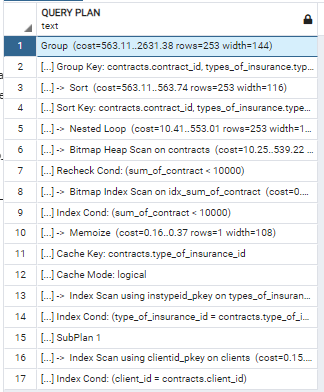


Рисунок 4.45 – План итогового запроса с использованием индекса

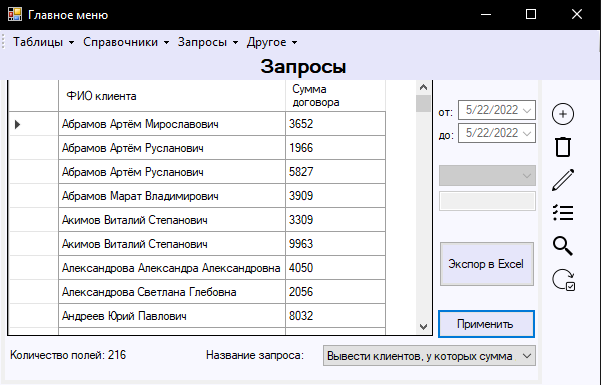


Рисунок 4.46 – Пример выполнения итогового запроса с использованием индекса

1. без использования индекса – вывести договоры определенного сотрудника, рисунки 4.47-4.49.

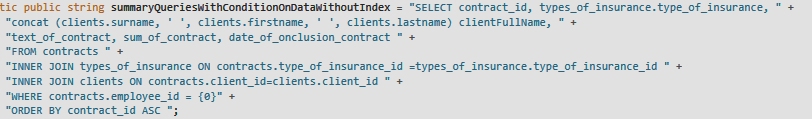


Рисунок 4.47 – Итоговый запрос без использования индекса

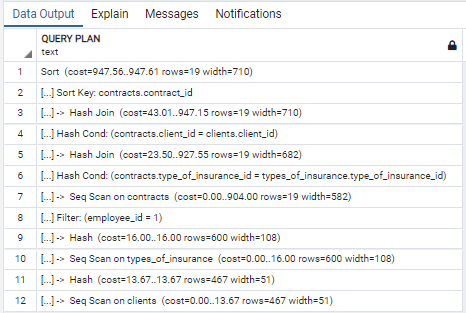


Рисунок 4.48 – План итогового запроса без использования индекса

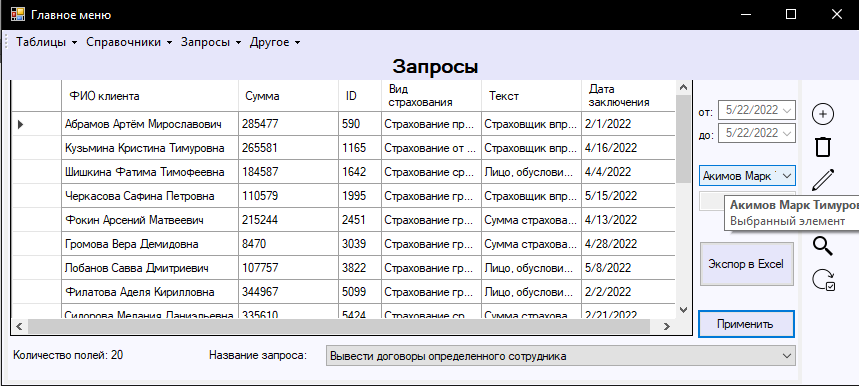


Рисунок 4.49 – Пример выполнения итогового запроса без использования индекса

### 4.7.9 Итоговый запрос с условием на группы

Вывести компании и общее число сотрудников в их филиалах, рисунки 4.50-4.51.

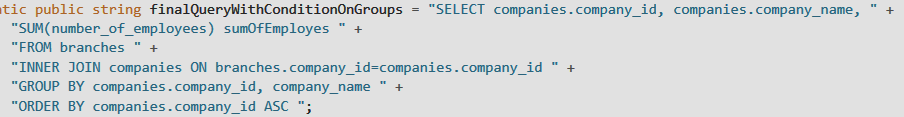


Рисунок 4.50 – Итоговый запрос с условием на группы

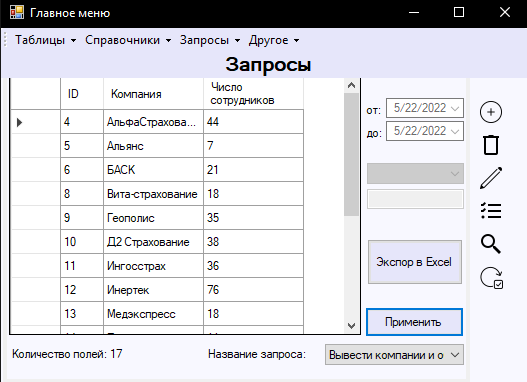


Рисунок 4.51 – Пример выполнения итогового запроса с условием на группы

### 4.7.10 Итоговый запрос с условием на данные и на группы

Вывести число филиалов у определенной компании, который были открыты в определенный период времени (годы), рисунки 4.52-4.53.



Рисунок 4.52 – Итоговый запрос с условием на данные и на группы

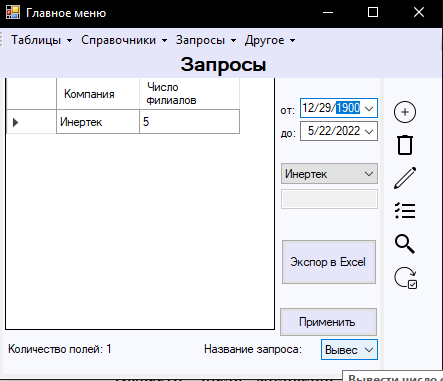


Рисунок 4.53 – Пример выполнения итогового запроса с условием на данные и на группы

### 4.7.11 Запрос на запросе по принципу итогового запроса

Вывести информацию о договорах, сумма которых превышает среднюю сумму, рисунки 4.54-4.55.

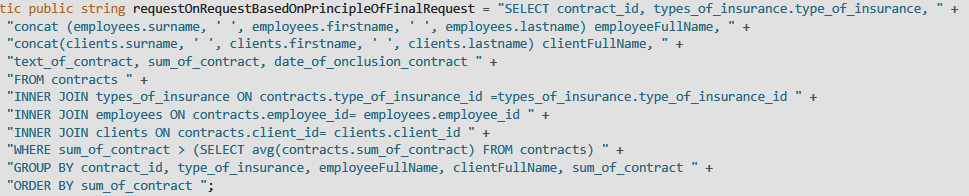


Рисунок 4.54 – Запрос на запросе по принципу итогового запроса

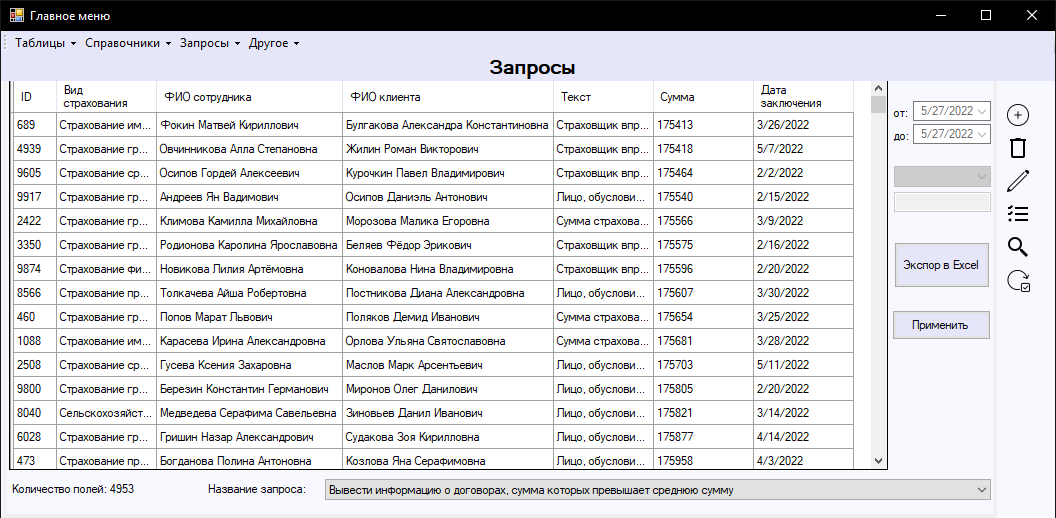


Рисунок 4.55 – Пример выполнения запроса на запросе по принципу итогового запроса

### 4.7.12 Запрос с использованием объединения

Объединить сумму договоров каждого сотрудника филиала и общую сумму договоров по филиалу, рисунки 4.56-4.57.

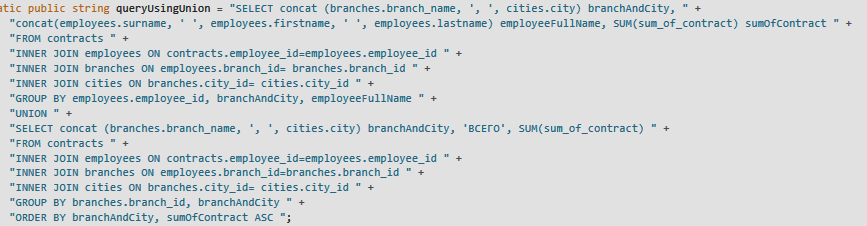
****

Рисунок 4.56 – Запрос с использованием объединения

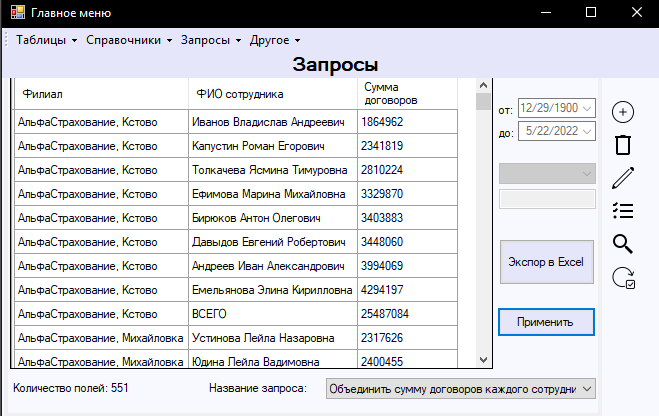


Рисунок 4.57 – Пример выполнения запроса с использованием объединения

### 4.7.13 Запросы с подзапросами (с использованием in, not in, case, операциями над итоговыми данными)

1. с использованием in – вывести договоры всех филиалов определенной компании, рисунки 4.58-4.59.

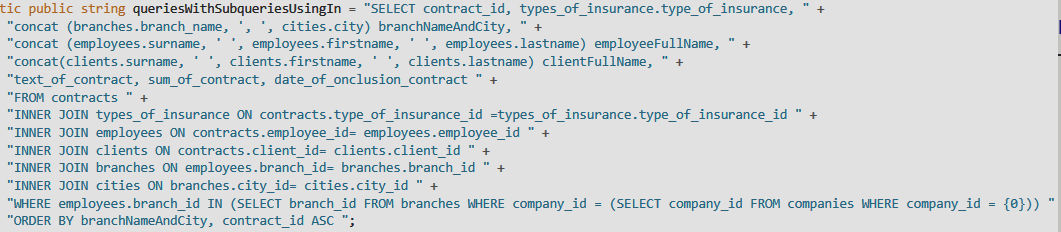


Рисунок 4.58 – Запрос с подзапросами (запрос 1)

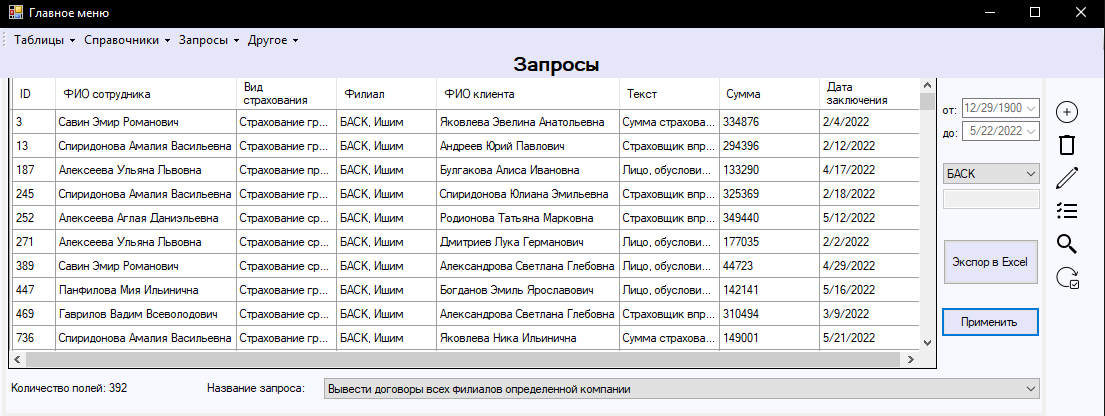


Рисунок 4.59 – Пример выполнения запроса с подзапросами (запрос 1)

1. с использованием not in – вывести договоры филиалов из всех городов, кроме определенного город, рисунки 4.60-4.61.

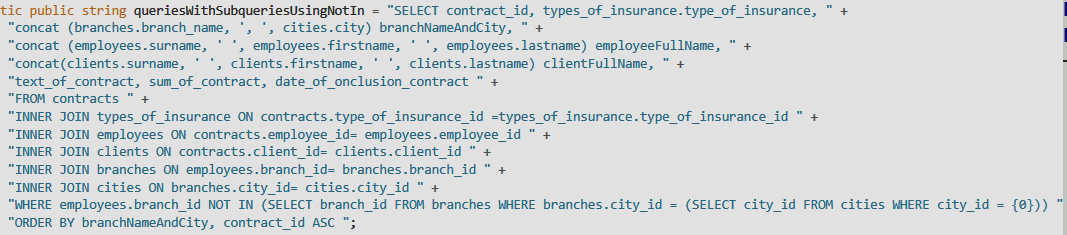


Рисунок 4.60 – Запрос с подзапросами (запрос 2)

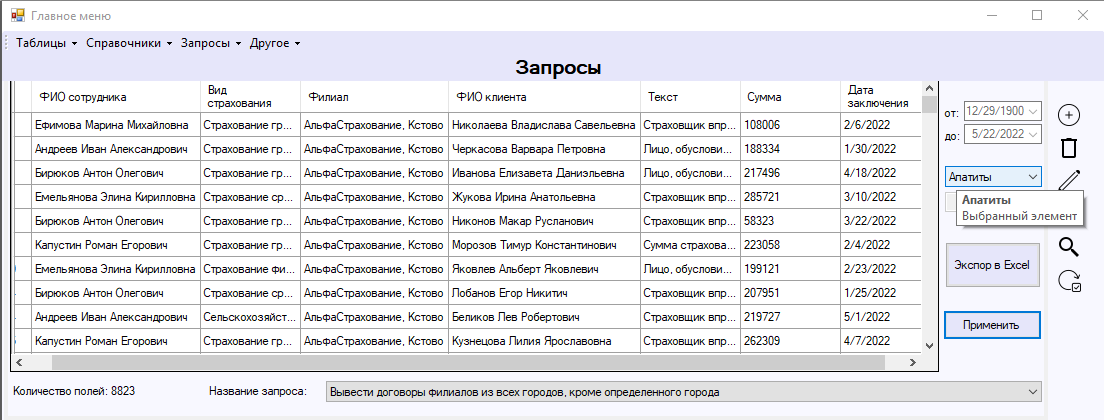


Рисунок 4.61 – Пример выполнения запроса с подзапросами (запрос 2)

1. с использованием case – вывести компании, которые заключили сделки на самую максимальную и минимальную сумму, рисунки 4.62-4.63.

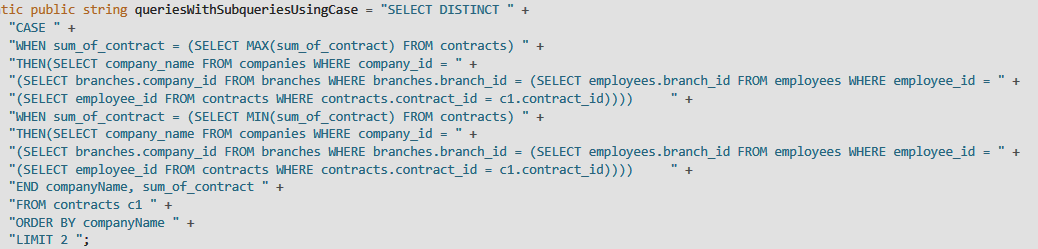


Рисунок 4.62 – Запрос с подзапросами (запрос 3)

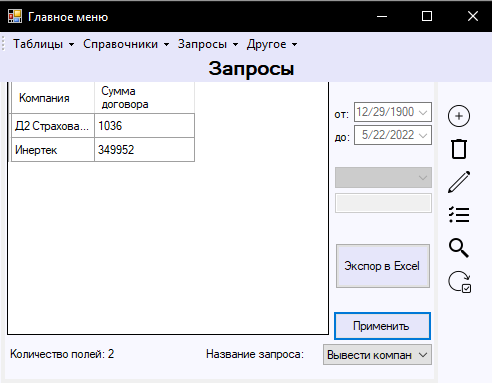


Рисунок 4.63 – Пример выполнения запроса с подзапросами (запрос 3)

1. с операциями над итоговыми данными (with) – вывести топовых сотрудников по количеству заключенных договоров, рисунки 4.64-4.65.

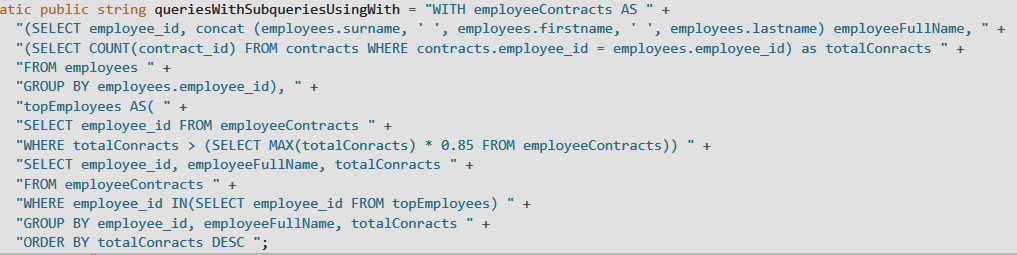


Рисунок 4.64 – Запрос с подзапросами (запрос 4)

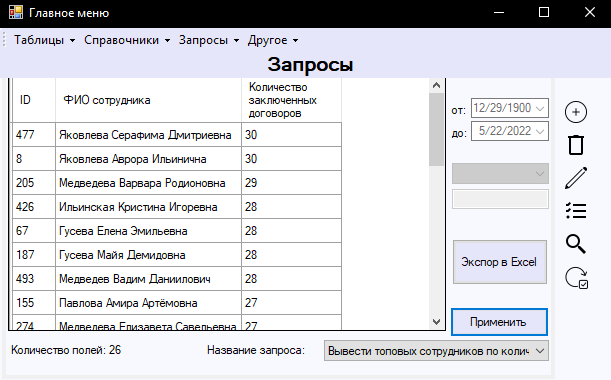


Рисунок 4.65 – Пример выполнения запроса с подзапросами (запрос 4)

### 4.7.13 Специальные запросы курсового проекта

1. Определить три филиала в каждом городе, пользующиеся наибольшим спросом и всем городам в целом, рисунки 4.66-4.68.

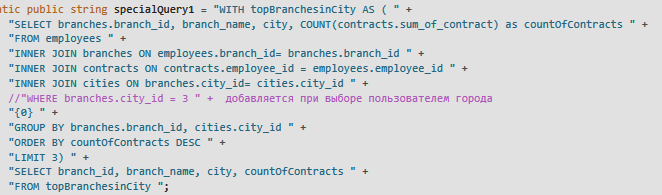


Рисунок 4.66 – Специальный запрос 1

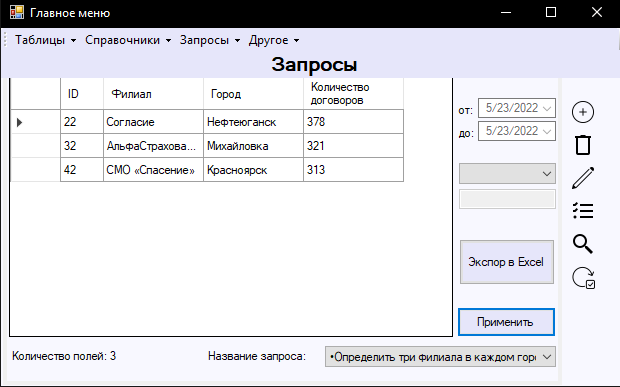


Рисунок 4.67 – Пример выполнения специального запроса 1 по всем городам

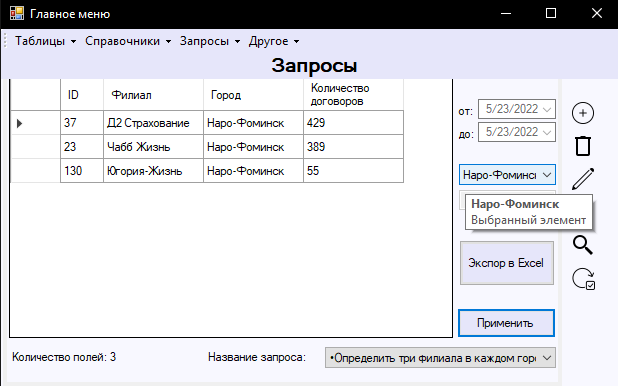


Рисунок 4.68 – Пример выполнения специального запроса 1 по определенному городу

1. Определить среднее количество клиентов по каждому филиалу и по каждой компании, рисунки 4.69-4.70.



Рисунок 4.69 – Специальный запрос 2

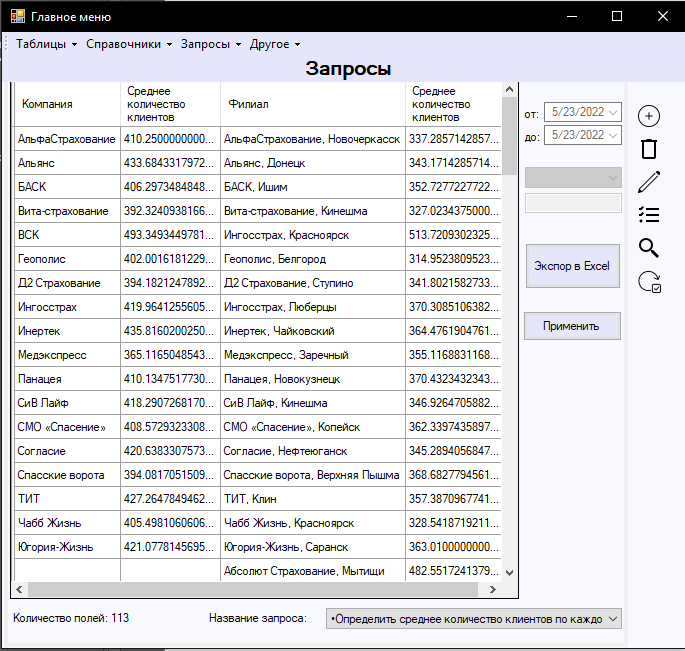


Рисунок 4.70 – Пример выполнения специального запроса 2

1. Определить количество клиентов привлеченных страховыми компаниями за указанный период (несколько лет) и доход по этим сделкам, рисунки 4.71-4.72.

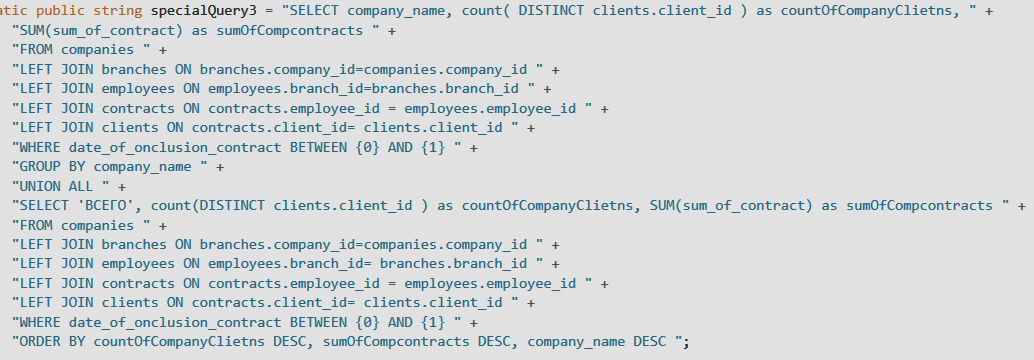


Рисунок 4.71 – Специальный запрос 3

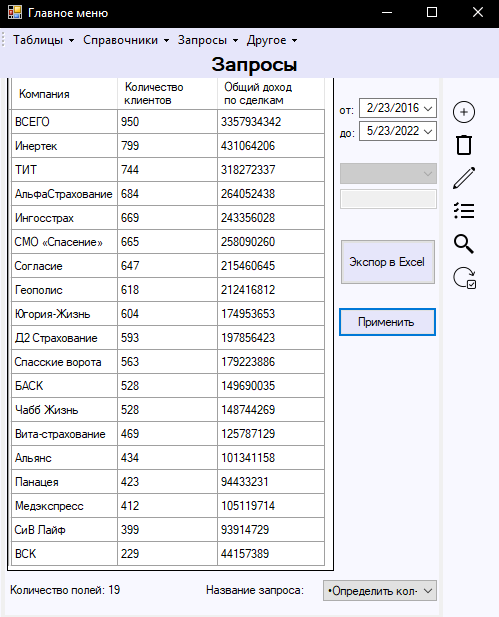


Рисунок 4.72 – Пример выполнения специального запроса 3

# 5 РАЗРАБОТКА КЛИЕНТСКОГО ПРИЛОЖЕНИЯ

## 5.1 Форма авторизации

В разрабатываемом клиентском приложении предусмотрена форма авторизации. Данная форма содержит в себе: информацию об авторе приложения, теме курсового проекта, поля для ввода логина и пароля и кнопку «войти». Доступ к приложению осуществляется только после ввода правильного логина и пароля.

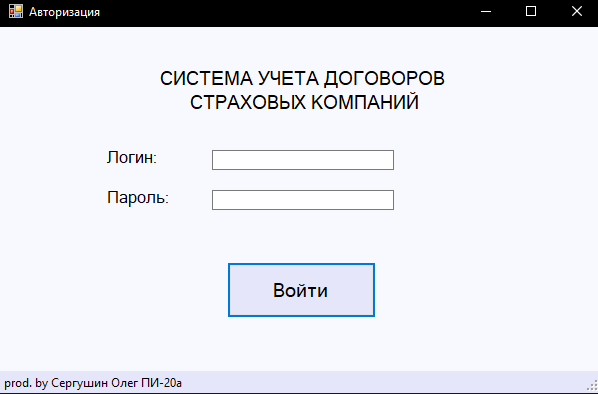


Рисунок 5.1 – Форма авторизации

## 5.2 Формы и компоненты для работы с таблицами

Взаимодействие с таблицами доступно из главного меню. Все таблицы размещены на одной форме, при это они разбиты по вкладкам. Для переключения между таблицами, справочниками, запросами и другими элементами существует верхняя панель главного меню. На рисунке 5.2 изображена форма главного меню.



Рисунок 5.2 – Форма главного меню

Справа от каждой таблицы находятся кнопки: добавления записи, удаления записи, редактирования записи, выделения всех элементов, поиска, обновления. Данные кнопки визуализированы в виде иконок.

По нажатию на кнопку добавления записи (иконка плюсика), откроется форма добавления записи для соответствующей таблицы. На рисунке 5.3 изображена форма добавления записи в таблицу «Компании».

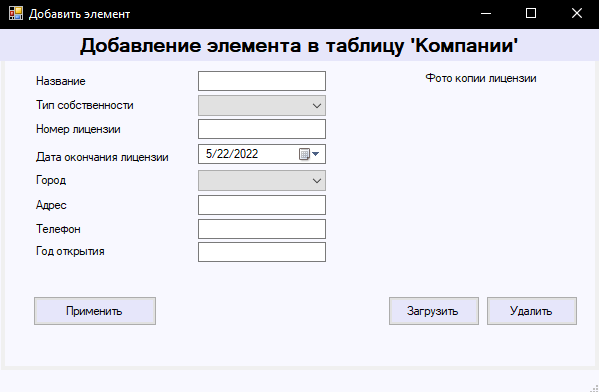


Рисунок 5.3 – Форма добавления записи

В форме добавления осуществляется проверка данных обязательных полей. Поля с некорректными данными подсвечиваются красным. На рисунке 5.4 приведен пример проверки входных данных. Остальные формы имеют подобную проверку

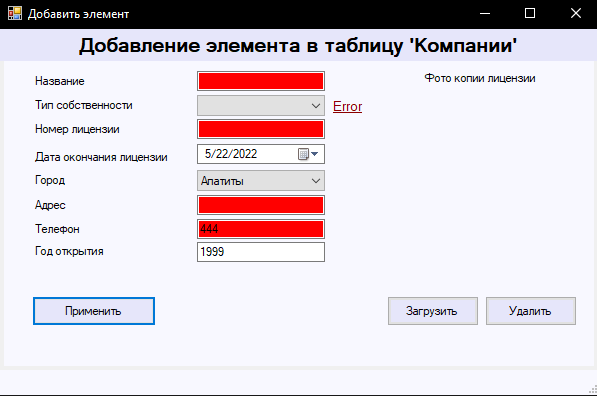


Рисунок 5.4 – Проверка данных на форме добавления записи

По нажатию на кнопку редактирования записи (иконка карандаша), откроется форма редактирования записи для соответствующей таблицы. На рисунке 5.5 изображена форма редактирования записи в таблицу «Компании». В форму редактирования записи помещаются данные выбранного для редактирования элемента.

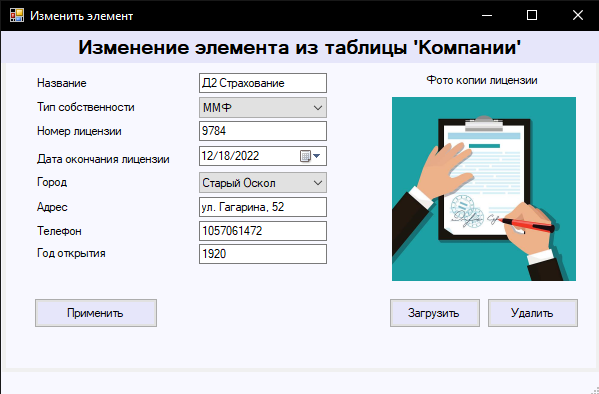


Рисунок 5.5 – Форма редактирования записи

По нажатию на кнопку поиска (иконка лупы), откроется форма поиска записей для соответствующей таблицы. На рисунке 5.6 изображена форма поиск записей по таблице «Компании». Результат поиск выводит на таблицу по который был осуществлен поиск.

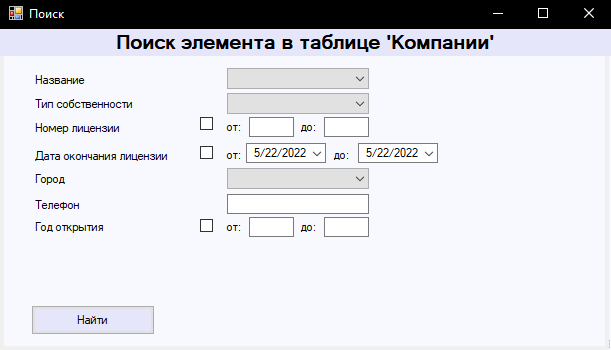


Рисунок 5.6 – Форма поиска записей

Для элементов из таблицы «Клиенты» создана составная форма «Договоры клиента». Данная форма содержит в себе информацию о выбранном клиенте и список его договоров. Чтобы открыть составную форму достаточно сделать двойное нажатие мышью по нужному элементу из таблицы «Клиенты». На рисунке 5.7 изображена составная форма.

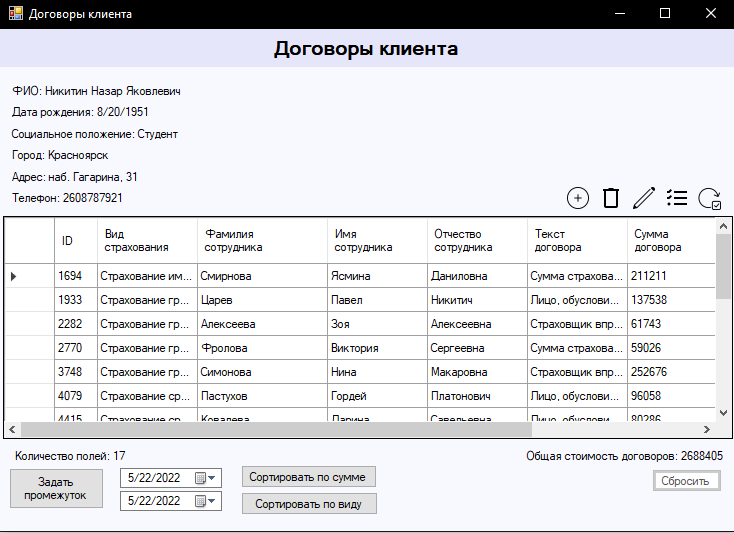


Рисунок 5.7 – Составная форма

Составная форма обладает тремя кнопками сортировки: по дате, по сумме и по виду. На составной форме присутствуют аналогичные главному меню кнопки добавления, удаления, редактирования выделения всех элементов и обновления. По нажатию на кнопку добавления записи откроется форма добавления контракта, аналогичная форме добавления любой другой записи, за исключением того, что не льзя изменить данные клиента. На рисунке 5.8 представлена форма добавления записи из составной формы.

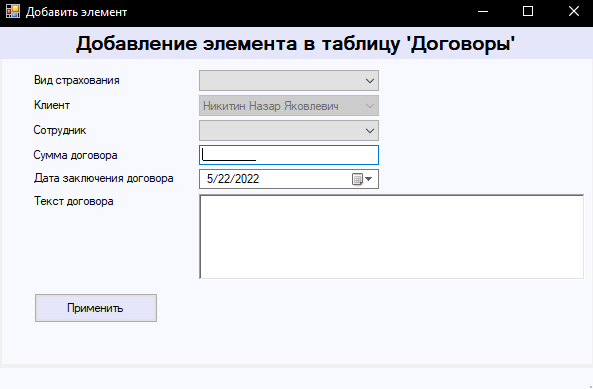


Рисунок 5.8 – Форма добавления записи из составной формы

По нажатию на кнопку удаления удаляются только выделенные строки соответствующей таблицы. Удалением можно пользоваться после применения поиска.

Формы для остальных таблиц и справочников имеют аналогичный вид.

## 5.3 Формы и компоненты для отображения результатов запросов

Форма, она же является одной из вкладок главного меню, отображения запросов доступна из главного меню. Во время работы со формой «Запросы» кнопки добавления, удаления, редактирования и поиска записей недоступны. На рисунке 5.9 представлена форма «Запросы».

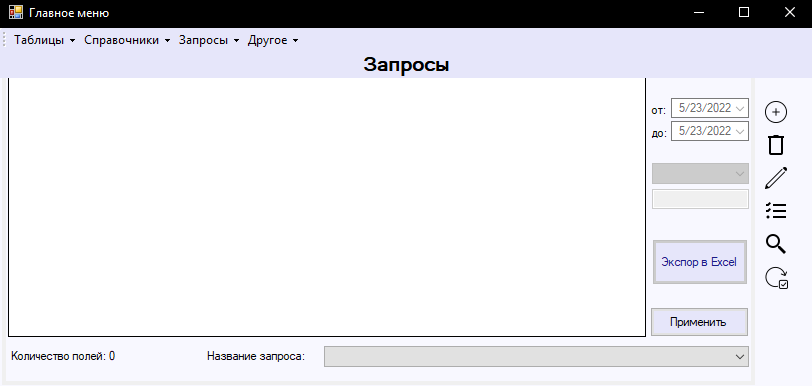


Рисунок 5.9 – Форма «Запросы»

На данной форме есть: выпадающий список с реализованными запросами, различные поля для ввода данных, копка «Применить» и кнопка «Экспорт в Excel».

Поля для ввода данных становятся доступными и наоборот в зависимости от выбранного запроса.

Кнопка «Экспорт в Excel» становится доступна после успешного выполнения запроса. Экспортировать в документ Excel можно результат каждого из представленных запросов. При экспорте пользователь сам выбирает место и названия документа Excel. После сохранения документа пользователю предлагается сразу открыть его.

Примеры запросов и результатов приведены в пункте 4.7 раздела 4.

На рисунке 5.10 приведен скриншот результата экспорта итогового запроса с условием на данные по значению (рис. 4.41).

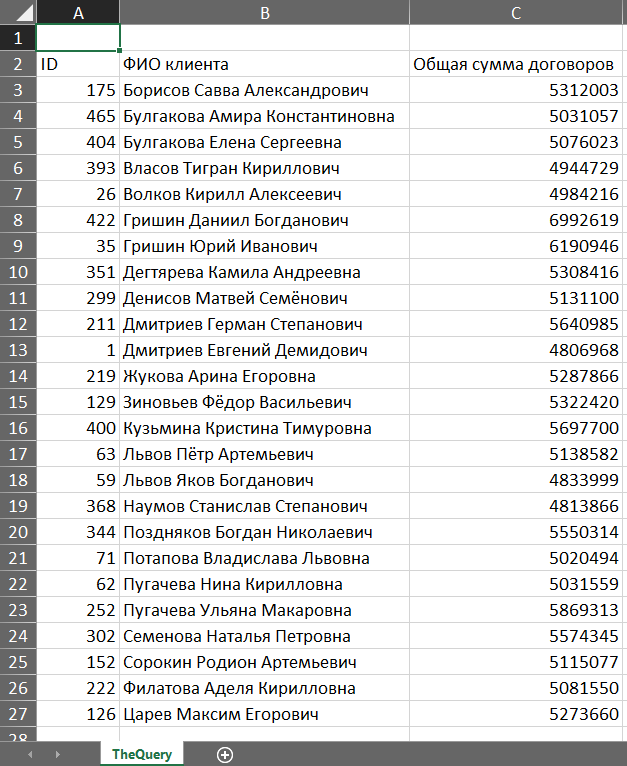


Рисунок 5.10 – Результат экспорта в Excel

Клиентское приложение предусматривает наличие трех диаграмм, которые доступны в панели главного меню в списке «Запросы». Созданы диаграммы таких видов: одномерная (в виде пирамиды), столбчатая, трехмерная столбчатая.

На рисунках 5.11-5.13 изображены формы с перечисленными диаграммами.

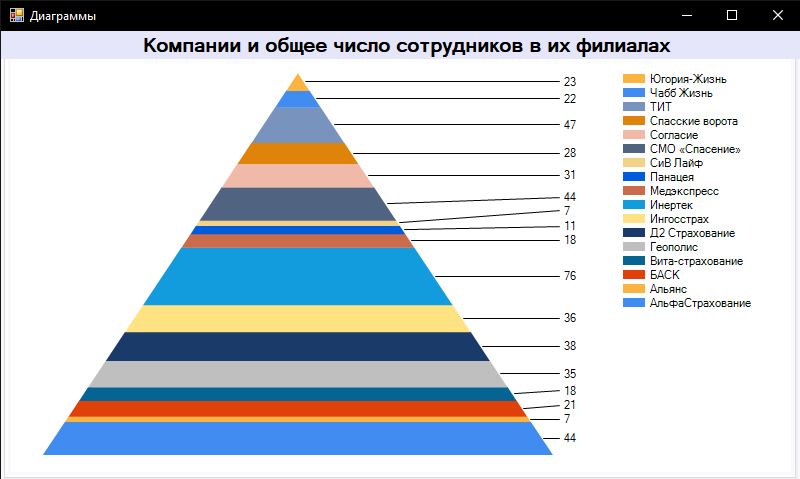


Рисунок 5.11 – Одномерная диаграмма



Рисунок 5.12 – Столбчатая диаграмма

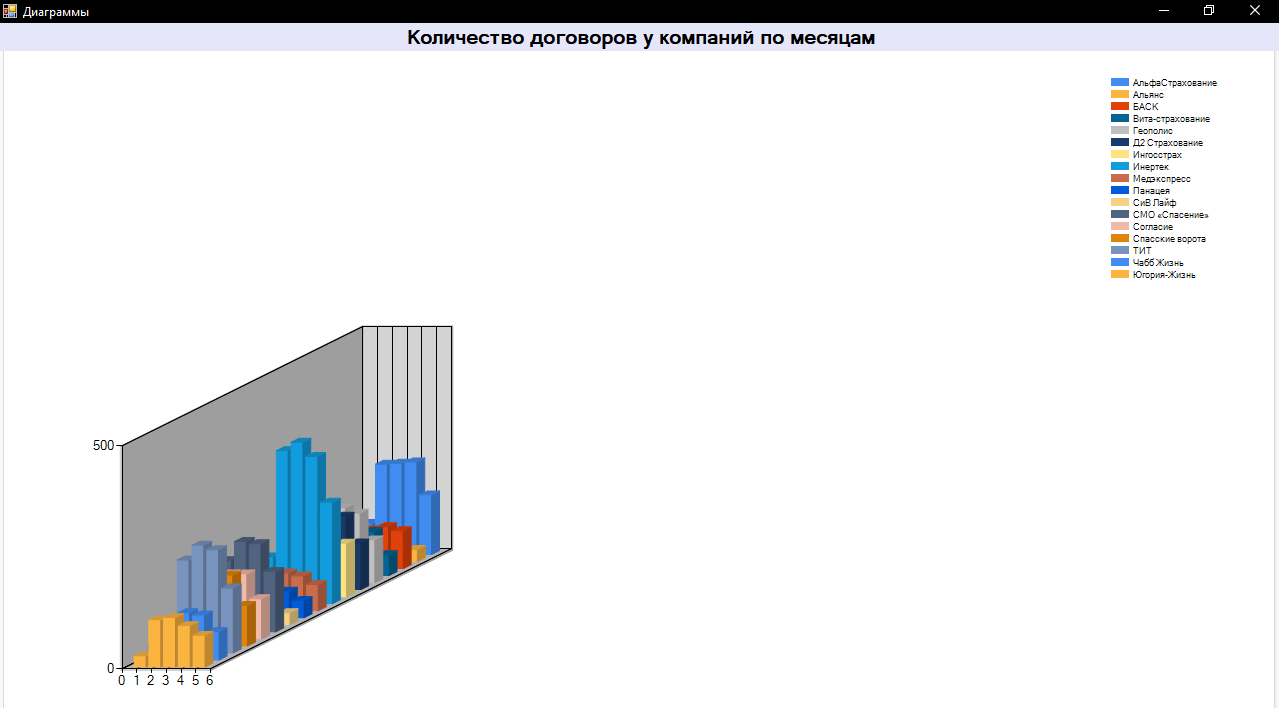


Рисунок 5.13 – Трехмерная диаграмма

# 6 ТЕСТИРОВАНИЕ РАЗРАБОТАННОЙ ИНФОРМАЦИОННОЙ СИСТЕМЫ

Для корректного тестирования системы был разработан простейший алгоритм генерации псевдослучайных записей для последующего добавления их в базу данных. Программа генерирует случайные наборы и добавляет их в базу данных.

Данные для генерации содержатся в текстовых файлах каталога проекта.

Генерация заполняет все таблицы и справочники. При этом данные справочников заполняются данными для справочников из файлов, а не генерируются. Поэтому нет необходимости заполнять справочники до генерации.

Генерировать данные можно множество раз. При этом, если файлы справочником не будут обновлены, то содержания справочников останется неизменны. В таблицах, данные которых генерируются, останутся прежние значения и добавятся новые.

Предусмотрена быстра очистка всех таблиц базы данных.

Приложение позволяет выполнять просмотр данных из разных таблиц во время генерации, одна не позволяет удалять данные.

По завершении разработки и отладки приложения было написано руководство пользователя, в котором описаны основные положения и рекомендации по использованию программы. Руководство пользователя приведено в приложении Б.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы были изучены основы языка структурированных запросов SQL, получены практические навыки проектирования, моделирования и создания баз данных средствами СУБД PostgreSQL и графического пакета pgAdmin4.

В результате выполнения курсовой работы было проведено проектирование модели базы данных, был создан проект базы данных, добавлены таблицы, домены, индексы, триггеры и представления. Были заполнены соответствующим образом все необходимые таблицы. Также были разработаны предоставленные требованиями к курсовой работе сложные запросы на выборку. Результатами некоторых запросов стали диаграммы статистических показателей. В программе реализована возможность сохранения результатов всех реализованных запросов в документ Excel.

Разработанная система может быть использована для обслуживания реальных страховых компаний. Для пользователей разработанной системы было написано соответствующее руководство пользователя.

В качестве руководства для программиста может быть использован данный отчет по курсовой работе.

# СПИСОК ЛИТЕРАТУРЫ

1. PostgreSQL. Основы языка SQL: учеб. пособие / Е. П. Моргунов; под ред. Е. В. Рогова, П. В. Лузанова. — СПб.: БХВ-Петербург, 2018. — 336 с.: ил.

2. https://postgrespro.ru/docs/postgresql/9.6

3. Язык программирования C# 7 и платформы .NET и .NET Core, 8-е изд. : Пер. с англ. — СПб. : ООО “Диалектика”, 2018 — 1328 с. : ил. — Парал. тит. англ.

# ПРИЛОЖЕНИЕ А

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

Министерство образования и науки

Донецкой народной республики

ГОУ ВПО «ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Интеллектуальных систем и программирования

Кафедра "Программная инженерия" им. Л.П. Фельдмана

Утверждаю

Зори С.А.

10.02.2022 г.

## **ТЕХНИЧЕСКОЕ ЗАДАНИЕ**

**на курсовой проект по дисциплине**

**«Базы данных»**

выдано студенту группы ПИ-20а Сергушину Олегу Сергеевичу

**Тема:** **«Создание клиент-серверной информационной системы средствами СУБД»**

**Описание предметной области:**

разработать БД учета договоров страховых компаний. Страховая компания имеет лицензию на осуществление своей деятельности (центральный офис). Страховая компания имеет центральное отделение в некотором городе и свои филиалы в ряде городов страны. Поле *тип собственности* может содержать информацию: государственная, частная, ЗАО, ОАО, ООО, … Клиент может заключать договора с разными компаниями. Договора заключают только с совершеннолетними клиентами. Все поля таблицы являются обязательными для ввода.

Для учета договоров страховых компаний БД должна содержать следующую информацию: название страховой компании, тип собственности компании, № лицензии, фото копии лицензии\*, дата окончания лицензии, город (главный офис компании), адрес (главный офис компании, телефон (главный офис компании), год открытия (офис компании), название (филиал), город (филиал), адрес (филиал), телефон (филиал), год открытия (филиал), кол-во сотрудников(филиал), ФИО клиента(заключившего договор), дата рождения (клиент), социальное положение (клиент), город (клиент), адрес (клиент), телефон (клиент), вид страхования, сумма страховки, дата заключения договора, ФИО сотрудника, составившего, договор, текст договора.

Дата защиты26.05.2022Подпись **\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Донецк – 2022 г.

Задание на курсовой проект

1. Спроектировать концептуальную модель базы данных (БД) для заданной предметной области и представить её в виде взаимосвязанных таблиц, находящихся в третьей нормальной форме (в случае денормализации БД – обосновать необходимость). Выделить базовые таблицы и таблицы - справочники, указать для них первичные и внешние ключи.
2. Создать базу данных в среде СУБД средствами языка SQL. Добавить таблицы, домены, индексы.
3. Разработать не менее шести триггеров (по одному для каждого типа события), как минимум для двух различных таблиц БД. Триггеры типа BEFORE INSERT должны быть созданы для всех таблиц и с использованием генераторов задавать значение первичного ключа для вновь добавляемой записи.
4. Заполнить таблицы БД с использованием соответствующих запросов на языке SQL (не менее десяти записей в каждом справочнике, не менее 10 000 - 50 000 псевдослучайных записей в таблицах).
5. Сформулировать и реализовать следующие виды запросов:

* симметричное внутреннее соединение с условием (два запроса с условием отбора по внешнему ключу, два – по датам);
* симметричное внутреннее соединение без условия (три запроса);
* левое внешнее соединение;
* правое внешнее соединение;
* запрос на запросе по принципу левого соединения;
* итоговый запрос без условия;
* итоговый запрос без условия c итоговыми данными вида: «всего», «в том числе»;
* итоговые запросы с условием на данные (по значению, по маске, с использованием индекса, без использования индекса);
* итоговый запрос с условием на группы;
* итоговый запрос с условием на данные и на группы;
* запрос на запросе по принципу итогового запроса;
* запрос с использованием объединения;
* запросы с подзапросами (с использованием in, not in, case, операциями над итоговыми данными);
* определить три филиала в каждом городе, пользующиеся наибольшим спросом и всем городам в целом;
* определить среднее количество клиентов по каждому филиалу и по каждой компании;
* определить количество клиентов привлеченных страховыми компаниями за указанный период (несколько лет) и доход по этим сделкам.

1. Запросы без параметров реализовать в виде представлений, остальные запросы – в виде хранимых процедур и/или функций. Создать, по меньшей мере, одно модифицируемое представление, используя механизм триггеров. Вся логика проектируемого ПО – на сервере.
2. Разработать клиентское приложение, которое предоставляет следующие возможности для работы с созданной базой данных:

* добавление, редактирование и удаление записей таблиц и модифицируемых представлений;
* работа с наборами данных, находящимися в отношении «один - ко -многим» (создать составную форму для просмотра и редактирования данных родительской и дочерней таблиц);
* поиск и фильтрация данных отображаемых таблиц;
* просмотр результатов выполнения запросов;
* визуализация результатов одного из итоговых запросов (диаграммы, экспорт в Excel).

1. Обеспечить защиту данных, информации от несанкционированного доступа.

График выполнения курсового проекта

|  |  |
| --- | --- |
| Неделя | Этап выполнения работы |
| 1-2 | Выдача и изучение задания |
| 3 | Анализ требований к системе и способов их реализации |
| 4-5 | Проектирование и реализация БД (таблицы, домены, индексы) |
| 6-7 | Создание триггеров и заполнение таблиц БД |
| 8-9 | Создание представлений и хранимых процедур |
| 10-13 | Разработка клиентского приложения |
| 14 | Тестирование и отладка системы |
| 15 | Оформление пояснительной записки |
| 16-17 | Защита курсового проекта |

Дата выдачи задания 07.02.2022

Задание принял Сергушин Олег Сергеевич

Руководители проекта Щедрин Сергей Валерьевич

Незамова Лариса Викторовна

Рычка Ольга Валентиновна

# ПРИЛОЖЕНИЕ Б

РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Для корректной работы программы необходима ЭВМ с системой Windows 7 или выше.

Для работы с программой необходимо запустить исполняемый файл DB\_Kursach.exe.

Для продолжения работы с программой необходимо выполнить авторизацию. В случае ввода правильных данных окно авторизации закроется и запустится главное меню. В ином случае программа выдаст ошибку авторизации.

Для переключения между таблицами, справочниками, запросами и другими элементами существует верхняя панель главного меню. Все элементы панели главного меню имеют говорящие названия и являются интуитивно понятными.

Справа от каждой таблицы находятся кнопки: добавления записи, удаления записи, редактирования записи, выделения всех элементов, поиска, обновления. Данные кнопки визуализированы в виде иконок.

Нажатие на кнопки добавления записи, редактирования записи, поиска, откроет новую форму соответствующего действия и таблицы, на которой они были нажаты. В этой форме необходимо заполнить предложенные поля и нажать итоговую кнопку.

Форма поиска предусматривает незаполненные поля, форма добавления и редактирования сообщает об ошибке при пустых полях или некорректных данных. Результат поиска выводит на соответствующую таблицу. Вернуться к исходной таблице можно выбрав ее в панели главного меню.

По нажатию на кнопку удаления удаляются только выделенные строки соответствующей таблицы. Удалением можно пользоваться после применения поиска.

Кнопка выделениях всех элементов выделит все элементы таблицы, на которой была нажата данная кнопка. Если все элементы уже были выделены, выделение снимется.

Выделить один или множество элементов можно прямо на таблице использую ЛКМ.

Кнопка обновления обновляет соответствующую таблицу, однако не отсылает какой-либо запрос.

Для элементов из таблицы "Клиенты" существует составная форма. Для открытия составной формы достаточно сделать двойное нажатие мышью по нужному элементу из таблицы «Клиенты». Составная форма обладает тремя кнопками сортировки: по дате, по сумме и по виду. На составной форме присутствуют аналогичные главному меню кнопки добавления, удаления, редактирования выделения всех элементов и обновления. По нажатию на кнопку добавления записи откроется форма добавления контракта, аналогичная форме добавления любой другой записи, за исключением того, что не льзя изменить данные клиента.

Через панель главного меню можно попасть на форму "Запросы". Во время работы со формой «Запросы» кнопки добавления, удаления, редактирования и поиска записей недоступны. На данной форме есть: выпадающий список с реализованными запросами, различные поля для ввода данных, копка «Применить» и кнопка «Экспорт в Excel». Поля для ввода данных становятся доступными и наоборот в зависимости от выбранного запроса. Кнопка «Экспорт в Excel» становится доступна после успешного выполнения запроса. Экспортировать в документ Excel можно результат каждого из представленных запросов. При экспорте пользователь сам выбирает место и названия документа Excel. После сохранения документа пользователю предлагается сразу открыть его.

Панель главного меню в разделе "Запросы" содержит таких видов: одномерная (в виде пирамиды), столбчатая, трехмерная столбчатая. Диаграммы открываются на в отдельной окне и имеют цель визуализировать некоторые запросы.

В списке другое находятся кнопки генерации элементов, очистки элементов и помощь.

Генерация заполняет все таблицы и справочники. При этом данные справочников заполняются данными для справочников из файлов, а не генерируются. Поэтому нет необходимости заполнять справочники до генерации. Генерировать данные можно множество раз. При этом, если файлы справочником не будут обновлены, то содержания справочников останется неизменны. В таблицах, данные которых генерируются, останутся прежние значения и добавятся новые.

Очистка всех элементов удаляет данных всех таблиц и обновляет значения всех ID.

Кнопка помощи открывает файл, содержащий руководство пользователя.

# ПРИЛОЖЕНИЕ В

ЛИСТИНГ СЕРВЕРНОГО ПРИЛОЖЕНИЯ

CREATE DOMAIN op\_year AS integer

CHECK(VALUE BETWEEN 1850 AND CAST(date\_part('year', current\_date) AS integer)) NOT NULL;

--создание таблиц

CREATE TABLE property\_types

( type\_id serial NOT NULL,

property\_type text COLLATE pg\_catalog."default" NOT NULL,

CONSTRAINT typeid\_pkey PRIMARY KEY (type\_id),

CONSTRAINT property\_uniq UNIQUE (property\_type)

);

CREATE TABLE cities

(

city\_id serial NOT NULL,

city text COLLATE pg\_catalog."default" NOT NULL,

CONSTRAINT cityid\_pkey PRIMARY KEY (city\_id),

CONSTRAINT city\_uniq UNIQUE (city)

);

CREATE TABLE companies

(

company\_id serial NOT NULL,

company\_name text COLLATE pg\_catalog."default" NOT NULL,

type\_id integer NOT NULL,

license\_number integer NOT NULL,

license\_expiration\_date date NOT NULL,

city\_id integer NOT NULL,

address text COLLATE pg\_catalog."default" NOT NULL,

phone\_number varchar(10) NOT NULL,

opening\_year op\_year,

license\_photo bytea,

CHECK (CAST(date\_part('year', current\_date) AS integer) > opening\_year),

CONSTRAINT company\_pkey PRIMARY KEY (company\_id),

CONSTRAINT name\_uniq UNIQUE (company\_name),

CONSTRAINT typeid\_fkey FOREIGN KEY (type\_id)

REFERENCES property\_types (type\_id)

ON DELETE CASCADE,

CONSTRAINT lnumber\_uniq UNIQUE (license\_number),

CONSTRAINT cityid\_fkey FOREIGN KEY (city\_id)

REFERENCES cities (city\_id)

ON DELETE CASCADE,

CONSTRAINT phnumber\_uniq UNIQUE (phone\_number)

);

CREATE TABLE types\_of\_insurance

(

type\_of\_insurance\_id serial NOT NULL,

type\_of\_insurance text COLLATE pg\_catalog."default" NOT NULL,

CONSTRAINT instypeid\_pkey PRIMARY KEY (type\_of\_insurance\_id),

CONSTRAINT instype\_uniq UNIQUE (type\_of\_insurance)

);

CREATE TABLE social\_status\_of\_clients

(

social\_status\_id serial NOT NULL,

social\_status text COLLATE pg\_catalog."default" NOT NULL,

CONSTRAINT socstatus\_pkey PRIMARY KEY (social\_status\_id),

CONSTRAINT socstatus\_uniq UNIQUE (social\_status)

);

CREATE TABLE branches

(

branch\_id serial NOT NULL,

branch\_name text COLLATE pg\_catalog."default" NOT NULL,

company\_id integer NOT NULL, --ref

city\_id integer NOT NULL, --ref

is\_main boolean NOT NULL, -- only one 'true'

address text COLLATE pg\_catalog."default" NOT NULL,

phone\_number varchar(10) NOT NULL,

opening\_year op\_year,

number\_of\_employees integer NOT NULL DEFAULT 0, --не даю это поле на ввод, добавил триггер

CHECK(opening\_year > 0),

CONSTRAINT branchid\_pkey PRIMARY KEY (branch\_id),

--CONSTRAINT branchname\_uniq UNIQUE (branch\_name),

CONSTRAINT companyid\_fkey FOREIGN KEY (company\_id)

REFERENCES companies (company\_id)

ON DELETE CASCADE,

CONSTRAINT cityid\_fkey FOREIGN KEY (city\_id)

REFERENCES cities (city\_id)

ON DELETE CASCADE,

CONSTRAINT branchphnumber\_uniq UNIQUE (phone\_number),

CONSTRAINT branchNameAndCity\_uniq UNIQUE (branch\_name, city\_id)

);

CREATE TABLE employees

(

employee\_id serial NOT NULL,

branch\_id integer NOT NULL,

surname text COLLATE pg\_catalog."default" NOT NULL,

firstname text COLLATE pg\_catalog."default" NOT NULL,

lastname text COLLATE pg\_catalog."default" NOT NULL,

CONSTRAINT employee\_pkey PRIMARY KEY (employee\_id),

CONSTRAINT branchid\_fkey FOREIGN KEY (branch\_id)

REFERENCES branches (branch\_id)

ON DELETE CASCADE,

CONSTRAINT fullname\_uniq UNIQUE (surname, firstname, lastname)

);

CREATE TABLE clients

(

client\_id serial NOT NULL,

surname text COLLATE pg\_catalog."default" NOT NULL,

firstname text COLLATE pg\_catalog."default" NOT NULL,

lastname text COLLATE pg\_catalog."default" NOT NULL,

date\_of\_birth date NOT NULL,

social\_status\_id integer NOT NULL, --ref

city\_id integer NOT NULL, --ref

address text NOT NULL,

phone\_number varchar(10) NOT NULL,

CHECK (date\_of\_birth BETWEEN (current\_date - (365\*120))::date AND (current\_date - (365\*18))), -- 18-120 лет

CONSTRAINT clientid\_pkey PRIMARY KEY (client\_id),

CONSTRAINT social\_statusid\_fkey FOREIGN KEY (social\_status\_id)

REFERENCES social\_status\_of\_clients (social\_status\_id)

ON DELETE CASCADE,

CONSTRAINT clientcityid\_fkey FOREIGN KEY (city\_id)

REFERENCES cities (city\_id)

ON DELETE CASCADE,

CONSTRAINT clientchphnumber\_uniq UNIQUE (phone\_number),

CONSTRAINT clientfullname\_uniq UNIQUE (surname, firstname, lastname)

);

CREATE TABLE contracts

(

contract\_id serial NOT NULL,

type\_of\_insurance\_id integer NOT NULL, --ref

employee\_id integer NOT NULL, --ref

client\_id integer NOT NULL, --ref

text\_of\_contract text NOT NULL,

sum\_of\_contract integer NOT NULL,

date\_of\_onclusion\_contract date DEFAULT(current\_date) NOT NULL,

CHECK (sum\_of\_contract > 0),

CHECK (date\_of\_onclusion\_contract BETWEEN (current\_date - 365) AND current\_date), --365 дней

CONSTRAINT contract\_id PRIMARY KEY (contract\_id),

CONSTRAINT instype\_fkey FOREIGN KEY (type\_of\_insurance\_id)

REFERENCES types\_of\_insurance (type\_of\_insurance\_id)

ON DELETE CASCADE,

CONSTRAINT employeeid\_fkey FOREIGN KEY (employee\_id)

REFERENCES employees (employee\_id)

ON DELETE CASCADE,

CONSTRAINT contract\_clientid\_fkey FOREIGN KEY (client\_id)

REFERENCES clients (client\_id)

ON DELETE CASCADE

);

DROP TABLE IF EXISTS property\_types CASCADE;

DROP TABLE IF EXISTS cities CASCADE;

DROP TABLE IF EXISTS companies CASCADE;

DROP TABLE IF EXISTS types\_of\_insurance CASCADE;

DROP TABLE IF EXISTS social\_status\_of\_clients CASCADE;

DROP TABLE IF EXISTS branches CASCADE;

DROP TABLE IF EXISTS employees CASCADE;

DROP TABLE IF EXISTS clients CASCADE;

DROP TABLE IF EXISTS contracts CASCADE;

CREATE INDEX idx\_opening\_year ON companies(opening\_year);

CREATE INDEX idx\_branch\_opening\_year ON branches(opening\_year);

CREATE INDEX idx\_client\_date\_of\_birth ON clients(date\_of\_birth);

CREATE INDEX idx\_date\_of\_onclusion\_contract ON contracts(date\_of\_onclusion\_contract);

CREATE INDEX idx\_sum\_of\_contract ON contracts(sum\_of\_contract);

CREATE INDEX idx\_client\_fio ON clients(surname, firstname, lastname);

--триггер сотрудники - филиалы 1

CREATE OR REPLACE FUNCTION add\_number\_employee\_to\_branch()

RETURNS TRIGGER

AS $$

BEGIN

UPDATE branches

SET number\_of\_employees = number\_of\_employees + 1

WHERE branch\_id = NEW.branch\_id;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

DROP TRIGGER add\_number\_emplyees ON employees;

CREATE TRIGGER add\_number\_emplyees AFTER INSERT ON employees

FOR EACH ROW EXECUTE PROCEDURE add\_number\_employee\_to\_branch();

--триггер сотрудники - филиалы 2

CREATE OR REPLACE FUNCTION remove\_number\_employee\_to\_branch()

RETURNS TRIGGER

AS $$

BEGIN

UPDATE branches

SET number\_of\_employees = number\_of\_employees - 1

WHERE branch\_id = OLD.branch\_id;

RETURN OLD;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER remove\_number\_emplyees AFTER DELETE ON employees

FOR EACH ROW EXECUTE PROCEDURE remove\_number\_employee\_to\_branch();

--триггер сотрудники - филиалы 3

CREATE OR REPLACE FUNCTION update\_number\_employee\_to\_branch()

RETURNS TRIGGER

AS $$

BEGIN

UPDATE branches

SET number\_of\_employees = number\_of\_employees - 1

WHERE branch\_id = OLD.branch\_id;

UPDATE branches

SET number\_of\_employees = number\_of\_employees + 1

WHERE branch\_id = NEW.branch\_id;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER update\_number\_emplyees AFTER UPDATE ON employees

FOR EACH ROW EXECUTE PROCEDURE update\_number\_employee\_to\_branch();

--триггер для филиала 1

--если уже был 1 филиал у компании, значит он был главным, значит остальные не главные

CREATE OR REPLACE FUNCTION insert\_ismain\_branch()

RETURNS TRIGGER

AS $$

BEGIN

IF ( SELECT COUNT(\*) FROM branches

WHERE (branches.company\_id = new.company\_id)) >= 1 THEN

new.is\_main = false;

RETURN NEW;

ELSE

new.is\_main = true;

RETURN NEW;

END IF;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER insert\_ismain\_branch BEFORE INSERT ON branches

FOR EACH ROW EXECUTE PROCEDURE insert\_ismain\_branch();

--триггер для филиала 2

--после удаления главного филиала, следующий филиал по минимальному году становится главным

CREATE OR REPLACE FUNCTION remove\_ismain\_branch()

RETURNS TRIGGER

AS $$

BEGIN

IF (OLD.is\_main = true) THEN

UPDATE branches

SET is\_main = true

WHERE branches.opening\_year = (SELECT MIN(branches.opening\_year) FROM branches WHERE old.company\_id = branches.company\_id);

END IF;

RETURN OLD;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER delete\_ismain\_branch AFTER DELETE ON branches

FOR EACH ROW EXECUTE PROCEDURE remove\_ismain\_branch();

--триггер для филиала 3

--после обвновления главного филиала, следующий филиал по минимальному году становится главным п этот становится обычным

CREATE OR REPLACE FUNCTION update\_ismain\_branch()

RETURNS TRIGGER

AS $$

BEGIN --если изменяемый филиал был главным -> главным станет другой

IF (OLD.is\_main = true) THEN

UPDATE branches

SET is\_main = true

WHERE branches.opening\_year = (SELECT MIN(branches.opening\_year) FROM branches WHERE old.company\_id = branches.company\_id);

END IF;

IF ( SELECT COUNT(\*) FROM branches -- если у другой компании уже больше 1 филиала (первый всегда главный)

WHERE (branches.company\_id = new.company\_id)) > 1 THEN -- то обновленный будет обычным

UPDATE branches

SET is\_main = false

WHERE branch\_id = new.branch\_id;

ELSE -- иначе обновленный станет главным

UPDATE branches

SET is\_main = true

WHERE branch\_id = new.branch\_id;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER update\_ismain\_branch AFTER UPDATE ON branches

FOR EACH ROW

WHEN (OLD.company\_id is DISTINCT FROM NEW.company\_id)

EXECUTE PROCEDURE update\_ismain\_branch();

-- представления (запрос без параметров) для таблиц

CREATE VIEW all\_companies\_view AS

SELECT company\_id, company\_name, property\_types.property\_type, license\_number, license\_expiration\_date, cities.city, address, phone\_number, opening\_year, license\_photo FROM companies

INNER JOIN property\_types ON companies.type\_id=property\_types.type\_id

INNER JOIN cities ON companies.city\_id=cities.city\_id

ORDER BY company\_id ASC;

CREATE VIEW all\_property\_types\_view AS

SELECT type\_id, property\_type FROM property\_types;

CREATE VIEW all\_branches\_view AS

SELECT branch\_id, branch\_name, companies.company\_name, cities.city, is\_main, branches.address, branches.phone\_number, branches.opening\_year, number\_of\_employees FROM branches

INNER JOIN companies ON branches.company\_id=companies.company\_id

INNER JOIN cities ON branches.city\_id=cities.city\_id

ORDER BY branch\_id ASC;

CREATE VIEW all\_employees\_view AS

SELECT employee\_id, employees.surname, employees.firstname, employees.lastname, concat (branches.branch\_name, ', ', cities.city) branchAndCity

FROM employees

INNER JOIN branches ON employees.branch\_id=branches.branch\_id

INNER JOIN cities ON branches.city\_id=cities.city\_id

ORDER BY employee\_id ASC;

CREATE VIEW all\_contracts\_view AS

SELECT contract\_id, types\_of\_insurance.type\_of\_insurance, concat (employees.surname, ' ', employees.firstname, ' ', employees.lastname) employeeFullName, concat (clients.surname, ' ', clients.firstname, ' ', clients.lastname) clientFullName, text\_of\_contract, sum\_of\_contract, date\_of\_onclusion\_contract

FROM contracts

INNER JOIN types\_of\_insurance ON contracts.type\_of\_insurance\_id =types\_of\_insurance.type\_of\_insurance\_id

INNER JOIN employees ON contracts.employee\_id=employees.employee\_id

INNER JOIN clients ON contracts.client\_id=clients.client\_id

ORDER BY contract\_id ASC;

CREATE VIEW all\_types\_of\_insurance\_view AS

SELECT type\_of\_insurance\_id, type\_of\_insurance FROM types\_of\_insurance;

CREATE VIEW all\_clients\_view AS

SELECT client\_id, clients.surname, clients.firstname, clients.lastname, date\_of\_birth, social\_status\_of\_clients.social\_status, cities.city, address, phone\_number FROM clients

INNER JOIN social\_status\_of\_clients ON clients.social\_status\_id=social\_status\_of\_clients.social\_status\_id

INNER JOIN cities ON clients.city\_id=cities.city\_id

ORDER BY client\_id ASC;

CREATE VIEW all\_social\_status\_of\_clients\_view AS

SELECT social\_status\_id, social\_status FROM social\_status\_of\_clients;

CREATE VIEW all\_cities\_view AS

SELECT city\_id, city FROM cities;

-- модифицируемое представление, используя механизм триггеров

-- ставит все символы поля тип\_собсвтенности в верхний герист. Однако при добавлении в представление нужн оуказывать ID!!!!!!

CREATE OR REPLACE FUNCTION ins\_update\_property\_types\_view()

RETURNS TRIGGER

AS $$

BEGIN

INSERT INTO property\_types VALUES(NEW.type\_id, UPPER(NEW.property\_type));

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER instiead\_update\_property\_types\_view INSTEAD OF INSERT ON all\_property\_types\_view

FOR EACH ROW EXECUTE PROCEDURE ins\_update\_property\_types\_view();

--пример

INSERT INTO all\_property\_types\_view VALUES(12, 'SSS');

# ПРИЛОЖЕНИЕ Д

ЛИСТИНГ КЛИЕНТСКОГО ПРИЛОЖЕНИЯ

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Windows.Forms;

namespace DataGenerator{

public static class Generation{

public static string[] \_maleSurnames { get; private set; }public static string[] \_maleFirstnames { get; private set; }

public static string[] \_maleLastnames { get; private set; }public static string[] \_femaleSurnames { get; private set; }

public static string[] \_femaleFirstnames { get; private set; }public static string[] \_femaleLastnames { get; private set; }

public static string[] \_cities { get; private set; }public static string[] \_typesOfInsurance { get; private set; }

public static string[] \_addresses { get; private set; }public static string[] \_propertyTypes { get; private set; }

public static string[] \_companyNames { get; private set; }public static string[] \_contractTexts { get; private set; }

public static string[] \_socialStatusOfClients { get; private set; }static Random random = new Random();

static bool isFilesLoaded = false; // загружены ли все файлы для генерации

static Generation() {

try{\_maleSurnames = File.ReadAllLines(@"FilesForGeneration\М\_фамилия.txt", Encoding.GetEncoding(1251));

\_maleFirstnames = File.ReadAllLines(@"FilesForGeneration\М\_имя.txt", Encoding.GetEncoding(1251));

\_maleLastnames = File.ReadAllLines(@"FilesForGeneration\М\_Отчество.txt", Encoding.GetEncoding(1251));\_femaleSurnames = File.ReadAllLines(@"FilesForGeneration\Ж\_фамилия.txt", Encoding.GetEncoding(1251));\_femaleFirstnames = File.ReadAllLines(@"FilesForGeneration\Ж\_имя.txt", Encoding.GetEncoding(1251));\_femaleLastnames = File.ReadAllLines(@"FilesForGeneration\Ж\_Отчество.txt", Encoding.GetEncoding(1251));\_cities = File.ReadAllLines(@"FilesForGeneration\Города.txt", Encoding.GetEncoding(1251));\_typesOfInsurance = File.ReadAllLines(@"FilesForGeneration\Виды страхования.txt", Encoding.GetEncoding(1251));\_addresses = File.ReadAllLines(@"FilesForGeneration\Адреса.txt", Encoding.GetEncoding(1251));\_propertyTypes = File.ReadAllLines(@"FilesForGeneration\Типы собственности.txt", Encoding.GetEncoding(1251));\_companyNames = File.ReadAllLines(@"FilesForGeneration\Названия\_компаний.txt", Encoding.GetEncoding(1251));\_contractTexts = File.ReadAllLines(@"FilesForGeneration\Тексты\_договоров.txt", Encoding.GetEncoding(1251));\_socialStatusOfClients = File.ReadAllLines(@"FilesForGeneration\Социальные положения клиентов.txt", Encoding.GetEncoding(1251));

isFilesLoaded = true;}catch(Exception ex) {MessageBox.Show(ex.ToString()); }}

public static string[] GenerateFullName(bool sex){if (!isFilesLoaded) return null;string[] fullName = new string[3];if (!sex) // Женщина {

fullName[0] = \_femaleSurnames[random.Next(0, \_femaleSurnames.Count())];fullName[1] = \_femaleFirstnames[random.Next(0, \_femaleFirstnames.Count())];fullName[2] = \_femaleLastnames[random.Next(0, \_femaleLastnames.Count())];return fullName;}

//мужчинаfullName[0] = \_maleSurnames[random.Next(0, \_maleSurnames.Count())];fullName[1] = \_maleFirstnames[random.Next(0, \_maleFirstnames.Count())];

fullName[2] = \_maleLastnames[random.Next(0, \_maleLastnames.Count())];return fullName;}

public static string GenerateLine(string lineName) {

if (!isFilesLoaded) return null;

string generatedLine;

switch (lineName){

case "city":generatedLine = \_cities[random.Next(0, \_cities.Count())];break;case "typeOfInsurance":generatedLine = \_typesOfInsurance[random.Next(0, \_typesOfInsurance.Count())];break;

case "address":generatedLine = \_addresses[random.Next(0, \_addresses.Count())];break;case "propertyType":generatedLine = \_propertyTypes[random.Next(0, \_propertyTypes.Count())];break;

case "companyName":generatedLine = \_companyNames[random.Next(0, \_companyNames.Count())];break;case "contractText":generatedLine = \_contractTexts[random.Next(0, \_contractTexts.Count())];break;

case "socialStatusOfClient":generatedLine = \_socialStatusOfClients[random.Next(0, \_socialStatusOfClients.Count())];break;

default:generatedLine = null;break;}return generatedLine;}

public static string[] GenerateFullDirectory(string directoryName){

if (!isFilesLoaded) return null;

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

switch (directoryName){

case "cities":foreach (var line in \_cities) {directory.Add(line);}break;

case "typesOfInsurance":foreach (var line in \_typesOfInsurance){directory.Add(line); }break;

case "propertyTypes":foreach (var line in \_propertyTypes){directory.Add(line); }break;

case "socialStatusOfClients":foreach (var line in \_socialStatusOfClients) {directory.Add(line);}break;

default:directory = null;break;}return directory.ToArray();}

public static string GeneratePhoneNumber(){

if (!isFilesLoaded) return null;

string number = random.Next(10000, 100000).ToString();

number += random.Next(10000, 100000).ToString();return number;}}}

namespace DB\_Kursach{

static class WorkWithDataBase{

public static Bitmap SelectImageFromTable(string cmd, string connString){

try{using (var conn = new NpgsqlConnection(connString)) {

conn.Open();Bitmap bmp = new Bitmap(1,1);

using (var command = new NpgsqlCommand(cmd, conn)){var reader = command.ExecuteReader();

reader.Read();using (var ms = new MemoryStream((byte[])reader[0])){

bmp = new Bitmap(ms); }

reader.Close();}return bmp;} }

catch (Exception) {return null;}}

public static DataTable SpecifitSelectFieldsFromTable(string cmd, string connString){

try {using (var conn = new NpgsqlConnection(connString)) {conn.Open();

DataTable dt = new DataTable();using (var command = new NpgsqlCommand(cmd, conn)) {

var reader = command.ExecuteReader();if (reader.HasRows) {

dt.Load(reader);}

reader.Close();}return dt; }}

catch (Exception ex){MessageBox.Show(ex.Message);return null;}}

public static bool InsertValuesIntoTable(string tableNameAndFields, string values, string connString){

try{using (var conn = new NpgsqlConnection(connString)) {

conn.Open();using (var command = new NpgsqlCommand($"INSERT INTO {tableNameAndFields} VALUES ({values}) ON CONFLICT DO NOTHING", conn)){

command.ExecuteNonQuery(); }return true;} }

catch (Exception ex){MessageBox.Show(ex.ToString());return false;}}

public static bool SpecificInsertValuesWithImageIntoTable(string tableNameAndFields, string values, string connString, byte[] img){

try{using (var conn = new NpgsqlConnection(connString)){ conn.Open();

if (img != null){NpgsqlParameter param = new NpgsqlParameter(":param", NpgsqlDbType.Bytea);param.Value = img;

using (var command = new NpgsqlCommand($"INSERT INTO {tableNameAndFields} VALUES ({values}, :param) ON CONFLICT DO NOTHING", conn)){command.Parameters.Add(param);command.ExecuteReader(); } }

else{using (var command = new NpgsqlCommand($"INSERT INTO {tableNameAndFields} VALUES ({values}, '{img}') ON CONFLICT DO NOTHING", conn)){

command.ExecuteNonQuery();}}return true;}}

catch (Exception ex){MessageBox.Show(ex.ToString());return false;}}

public static bool ClearTableCascade(string tableName, string connString){

try{using (var conn = new NpgsqlConnection(connString)){

conn.Open();

using (var command = new NpgsqlCommand($"TRUNCATE TABLE {tableName} RESTART IDENTITY CASCADE", conn)){

command.ExecuteNonQuery();}return true;}}

catch (Exception ex){MessageBox.Show(ex.ToString());return false;}}

public static bool DeleteTableCascade(string tableName, string connString){

try{using (var conn = new NpgsqlConnection(connString)){conn.Open();

using (var command = new NpgsqlCommand($"DROP TABLE IF EXISTS {tableName} CASCADE", conn)){

command.ExecuteNonQuery();}return true;}}

catch (Exception ex){MessageBox.Show(ex.ToString());return false;}}

public static int SelectRowsCountFromTable(string tableName, string connString){

int count = 0;

try{using (var conn = new NpgsqlConnection(connString)){

conn.Open();using (var command = new NpgsqlCommand($"SELECT COUNT (\*) FROM {tableName}", conn)){count = Convert.ToInt32(command.ExecuteScalar());}return count;}}catch (Exception ex){MessageBox.Show(ex.ToString());return count;}}

public static string SelectLineFromTable(string lineName, string tableName, string connString, string func = "", string where = ""){

try{using (var conn = new NpgsqlConnection(connString)){

conn.Open();string line = null;

using (var command = new NpgsqlCommand($"SELECT {func}({lineName}) FROM {tableName} {where}", conn)){

line = command.ExecuteScalar().ToString();return line;}}}catch (Exception ex){MessageBox.Show(ex.ToString());return null;}}

public static bool RemoveLineFromTable(string tableName, string where, string connString){

try{using (var conn = new NpgsqlConnection(connString)){conn.Open();

using (var command = new NpgsqlCommand($"DELETE FROM {tableName} WHERE {where}", conn)){

command.ExecuteReader();}return true;}}

catch (Exception ex){MessageBox.Show(ex.ToString() + "\n" + tableName);return false;}}

public static bool SpecificUpdateValuesWithImageFromTable(string tableName, string valuesNamesDatas, string where, string connString, byte[] img){

try{using (var conn = new NpgsqlConnection(connString)){

conn.Open();

if (img != null){NpgsqlParameter param = new NpgsqlParameter(":param", NpgsqlDbType.Bytea);

param.Value = img;

using (var command = new NpgsqlCommand($"UPDATE {tableName} SET {valuesNamesDatas} :param WHERE {where} ", conn)){

command.Parameters.Add(param);command.ExecuteReader();}}

else{using (var command = new NpgsqlCommand($"UPDATE {tableName} SET {valuesNamesDatas} '{img}' WHERE {where} ", conn)){

command.ExecuteNonQuery();}}return true;}}

catch (Exception ex){

MessageBox.Show(ex.ToString() + "\n////");return false;}}

public static bool UpdateValuesFromTable(string tableName, string valuesNamesDatas, string where, string connString){

try{using (var conn = new NpgsqlConnection(connString)){conn.Open();

using (var command = new NpgsqlCommand($"UPDATE {tableName} SET {valuesNamesDatas} WHERE {where} ", conn)){command.ExecuteNonQuery();}return true;}}

catch (Exception ex){

MessageBox.Show(ex.ToString() + "\n////");return false;}}}}

namespace DB\_Kursach{

static public string innerJoinWithForeignKey1 = "SELECT branch\_id, branch\_name, cities.city, is\_main, branches.address, branches.phone\_number, branches.opening\_year, number\_of\_employees " +"FROM branches " +

"INNER JOIN cities ON branches.city\_id=cities.city\_id " +

"WHERE branches.city\_id = {0} " +"GROUP BY branch\_id, cities.city ";

static public string innerJoinWithForeignKey2 = "SELECT employee\_id, employees.surname, employees.firstname, employees.lastname, concat (branches.branch\_name, ', ', cities.city) branchAndCity " +

"FROM employees " +"INNER JOIN branches ON employees.branch\_id=branches.branch\_id " +

"INNER JOIN cities ON branches.city\_id=cities.city\_id " +"WHERE employees.branch\_id = {0} " +"GROUP BY employee\_id, branchAndCity; ";

static public string leftOuterJoin = "SELECT branch\_id, branch\_name, companies.company\_name, city, is\_main, number\_of\_employees FROM branches " + "LEFT OUTER JOIN companies ON branches.company\_id = companies.company\_id AND companies.license\_photo IS NOT NULL " +

"LEFT OUTER JOIN cities ON branches.city\_id= cities.city\_id " +"ORDER BY branch\_id ASC ";

static public string rightOuterJoin = "SELECT property\_types.type\_id , property\_type , companies.company\_name FROM property\_types " +

"RIGHT OUTER JOIN companies ON companies.type\_id = property\_types.type\_id AND companies.license\_photo IS NOT NULL " +"ORDER BY property\_types.type\_id ASC ";static public string queryOnQueryByLeftJoinPrinciple = "SELECT companies.company\_id, (SELECT company\_name FROM companies WHERE cmp.company\_id = companies.company\_id) FROM companies " +

"LEFT OUTER JOIN(SELECT DISTINCT company\_id FROM branches " +

"WHERE is\_main = true) cmp ON cmp.company\_id = companies.company\_id " +

"GROUP BY companies.company\_id, cmp.company\_id " +"ORDER BY companies.company\_id ";

static public string finalQueryWithoutCondition = "SELECT clients.client\_id, concat (clients.surname, ' ', clients.firstname, ' ', clients.lastname) clientFullName, COUNT(\*) countClientContracts " +"FROM contracts " +

"INNER JOIN clients ON contracts.client\_id=clients.client\_id " +

"GROUP BY clients.client\_id, clientFullName " +"ORDER BY clientFullName ASC ";

//всего в том числе

static public string totalIncluding = "SELECT COUNT(client\_id), " +

"SUM(CASE WHEN current\_date - date\_of\_birth >= 18\*365 AND current\_date - date\_of\_birth< 20\*365 THEN 1 ELSE 0 " +

"END) AS about18yoLower20yo, " +"SUM(CASE WHEN current\_date - date\_of\_birth >= 20 \* 365 AND current\_date - date\_of\_birth < 30 \* 365 THEN 1 ELSE 0 " +

"END) AS about20yoLower30yo, " +"SUM(CASE WHEN current\_date - date\_of\_birth >= 30 \* 365 AND current\_date - date\_of\_birth < 45 \* 365 THEN 1 ELSE 0 " +"END) AS about30yoLower45yo, " +"SUM(CASE WHEN current\_date - date\_of\_birth >= 45 \* 365 THEN 1 ELSE 0 " +

"END) AS about45yo " +"FROM clients ";static public string summaryQueriesWithConditionOnDataByValue = "SELECT clients.client\_id, concat (clients.surname, ' ', clients.firstname, ' ', clients.lastname) clientFullName, SUM(sum\_of\_contract) " +"FROM contracts " +"INNER JOIN clients ON contracts.client\_id=clients.client\_id " +

"WHERE(SELECT SUM(sum\_of\_contract) FROM contracts WHERE contracts.client\_id= clients.client\_id) >= {0} " +

"GROUP BY clients.client\_id, clientFullName " +"ORDER BY clientFullName ASC ";

static public string summaryQueriesWithConditionOnDataByMask = "SELECT branch\_id, concat (branch\_name, ', ', city) branchNameAndCity, phone\_number FROM all\_branches\_view " +

"WHERE phone\_number LIKE '{0}%' " +"GROUP BY branch\_id, branchNameAndCity, phone\_number ORDER BY branch\_id ASC ";

static public string summaryQueriesWithConditionOnDataByIndex = "SELECT (SELECT concat(clients.surname, ' ', clients.firstname, ' ', clients.lastname) clientFullName " +"FROM clients WHERE contracts.client\_id = clients.client\_id), " +

"sum\_of\_contract FROM contracts " +"WHERE sum\_of\_contract < 10000 " +

"GROUP BY clientFullName, sum\_of\_contract ORDER BY clientFullName ASC ";static public string summaryQueriesWithConditionOnDataWithoutIndex = "SELECT contract\_id, types\_of\_insurance.type\_of\_insurance, " +

"concat (clients.surname, ' ', clients.firstname, ' ', clients.lastname) clientFullName, " +

"text\_of\_contract, sum\_of\_contract, date\_of\_onclusion\_contract " +"FROM contracts " +

"INNER JOIN types\_of\_insurance ON contracts.type\_of\_insurance\_id =types\_of\_insurance.type\_of\_insurance\_id " +

"INNER JOIN clients ON contracts.client\_id=clients.client\_id " +

"WHERE contracts.employee\_id = {0}" +"ORDER BY contract\_id ASC ";

static public string finalQueryWithConditionOnGroups = "SELECT companies.company\_id, companies.company\_name, SUM(number\_of\_employees) sumOfEmployes FROM branches " +"INNER JOIN companies ON branches.company\_id=companies.company\_id " +

"GROUP BY companies.company\_id, company\_name " +"ORDER BY companies.company\_id ASC ";

static public string finalQueryWithConditionOnDataAndGroups = "SELECT company\_name, COUNT(branches.branch\_id) " +

"FROM companies " +"INNER JOIN branches ON branches.company\_id=companies.company\_id " +

"WHERE companies.company\_id = {0} AND branches.opening\_year BETWEEN {1} AND {2} " +"GROUP BY company\_name " +"ORDER BY company\_name ";static public string requestOnRequestBasedOnPrincipleOfFinalRequest = "SELECT contract\_id, types\_of\_insurance.type\_of\_insurance, concat (employees.surname, ' ', employees.firstname, ' ', employees.lastname) employeeFullName, " +

"concat(clients.surname, ' ', clients.firstname, ' ', clients.lastname) clientFullName, text\_of\_contract, sum\_of\_contract, date\_of\_onclusion\_contract " +"FROM contracts " +

"INNER JOIN types\_of\_insurance ON contracts.type\_of\_insurance\_id =types\_of\_insurance.type\_of\_insurance\_id " +

"INNER JOIN employees ON contracts.employee\_id= employees.employee\_id " +

"INNER JOIN clients ON contracts.client\_id= clients.client\_id " +

"WHERE sum\_of\_contract > (SELECT avg(contracts.sum\_of\_contract) FROM contracts) " +

"GROUP BY contract\_id, type\_of\_insurance, employeeFullName, clientFullName, sum\_of\_contract " +"ORDER BY sum\_of\_contract ";

static public string queryUsingUnion = "SELECT concat (branches.branch\_name, ', ', cities.city) branchAndCity, " +

"concat(employees.surname, ' ', employees.firstname, ' ', employees.lastname) employeeFullName, SUM(sum\_of\_contract) sumOfContract " +"FROM contracts " +"INNER JOIN employees ON contracts.employee\_id=employees.employee\_id " +

"INNER JOIN branches ON employees.branch\_id= branches.branch\_id " +

"INNER JOIN cities ON branches.city\_id= cities.city\_id " +

"GROUP BY employees.employee\_id, branchAndCity, employeeFullName " +"UNION " +

"SELECT concat (branches.branch\_name, ', ', cities.city) branchAndCity, 'ВСЕГО', SUM(sum\_of\_contract) " +"FROM contracts " +

"INNER JOIN employees ON contracts.employee\_id=employees.employee\_id " +

"INNER JOIN branches ON employees.branch\_id=branches.branch\_id " +"INNER JOIN cities ON branches.city\_id= cities.city\_id " +"GROUP BY branches.branch\_id, branchAndCity " +"ORDER BY branchAndCity, sumOfContract ASC ";

static public string queriesWithSubqueriesUsingIn = "SELECT contract\_id, types\_of\_insurance.type\_of\_insurance, concat (branches.branch\_name, ', ', cities.city) branchNameAndCity, " +

"concat (employees.surname, ' ', employees.firstname, ' ', employees.lastname) employeeFullName, " +"concat(clients.surname, ' ', clients.firstname, ' ', clients.lastname) clientFullName, text\_of\_contract, sum\_of\_contract, date\_of\_onclusion\_contract " +"FROM contracts " +

"INNER JOIN types\_of\_insurance ON contracts.type\_of\_insurance\_id =types\_of\_insurance.type\_of\_insurance\_id " +

"INNER JOIN employees ON contracts.employee\_id= employees.employee\_id " +"INNER JOIN clients ON contracts.client\_id= clients.client\_id " +

"INNER JOIN branches ON employees.branch\_id= branches.branch\_id " +"INNER JOIN cities ON branches.city\_id= cities.city\_id " +

"WHERE employees.branch\_id IN (SELECT branch\_id FROM branches WHERE company\_id = (SELECT company\_id FROM companies WHERE company\_id = {0})) " +

"ORDER BY branchNameAndCity, contract\_id ASC ";

static public string queriesWithSubqueriesUsingNotIn = "SELECT contract\_id, types\_of\_insurance.type\_of\_insurance, concat (branches.branch\_name, ', ', cities.city) branchNameAndCity, " +"concat (employees.surname, ' ', employees.firstname, ' ', employees.lastname) employeeFullName, " +"concat(clients.surname, ' ', clients.firstname, ' ', clients.lastname) clientFullName, text\_of\_contract, sum\_of\_contract, date\_of\_onclusion\_contract " +"FROM contracts " +

"INNER JOIN types\_of\_insurance ON contracts.type\_of\_insurance\_id =types\_of\_insurance.type\_of\_insurance\_id " +

"INNER JOIN employees ON contracts.employee\_id= employees.employee\_id " +"INNER JOIN clients ON contracts.client\_id= clients.client\_id " +"INNER JOIN branches ON employees.branch\_id= branches.branch\_id " +"INNER JOIN cities ON branches.city\_id= cities.city\_id " +"WHERE employees.branch\_id NOT IN (SELECT branch\_id FROM branches WHERE branches.city\_id = (SELECT city\_id FROM cities WHERE city\_id = {0})) " +"ORDER BY branchNameAndCity, contract\_id ASC ";

static public string queriesWithSubqueriesUsingCase = "SELECT DISTINCT " +"CASE " +"WHEN sum\_of\_contract = (SELECT MAX(sum\_of\_contract) FROM contracts) " +

"THEN(SELECT company\_name FROM companies WHERE company\_id = (SELECT branches.company\_id FROM branches WHERE branches.branch\_id = (SELECT employees.branch\_id FROM employees WHERE employee\_id = " +"(SELECT employee\_id FROM contracts WHERE contracts.contract\_id = c1.contract\_id)))) " +

"WHEN sum\_of\_contract = (SELECT MIN(sum\_of\_contract) FROM contracts) " +

"THEN(SELECT company\_name FROM companies WHERE company\_id = (SELECT branches.company\_id FROM branches WHERE branches.branch\_id = (SELECT employees.branch\_id FROM employees WHERE employee\_id = " +

"(SELECT employee\_id FROM contracts WHERE contracts.contract\_id = c1.contract\_id)))) " +"END companyName, sum\_of\_contract " +

"FROM contracts c1 " +"ORDER BY companyName " +"LIMIT 2 ";//Вывести топовых сотрудников по количеству заключенных договоров

static public string queriesWithSubqueriesUsingWith = "WITH employeeContracts AS (SELECT employee\_id, concat (employees.surname, ' ', employees.firstname, ' ', employees.lastname) employeeFullName, " +

"(SELECT COUNT(contract\_id) FROM contracts WHERE contracts.employee\_id = employees.employee\_id) as totalConracts " +"FROM employees " +

"GROUP BY employees.employee\_id), " +"topEmployees AS( " +

"SELECT employee\_id FROM employeeContracts " +

"WHERE totalConracts > (SELECT MAX(totalConracts) \* 0.85 FROM employeeContracts)) " +"SELECT employee\_id, employeeFullName, totalConracts " +"FROM employeeContracts " +

"WHERE employee\_id IN(SELECT employee\_id FROM topEmployees) " +

"GROUP BY employee\_id, employeeFullName, totalConracts " +"ORDER BY totalConracts DESC ";

// СПЕЦИАЛЬНЫЙ ЗАПРОСЫ

static public string specialQuery1 = "WITH topBranchesinCity AS ( " +

"SELECT branches.branch\_id, branch\_name, city, COUNT(contracts.sum\_of\_contract) as countOfContracts " +"FROM employees " +

"INNER JOIN branches ON employees.branch\_id= branches.branch\_id " +

"INNER JOIN contracts ON contracts.employee\_id = employees.employee\_id " +

"INNER JOIN cities ON branches.city\_id= cities.city\_id " +

//"WHERE branches.city\_id = 3 " + добавляется при выборе пользователем города"{0} " +"GROUP BY branches.branch\_id, cities.city\_id " +

"ORDER BY countOfContracts DESC " +"LIMIT 3) " +"SELECT branch\_id, branch\_name, city, countOfContracts " +"FROM topBranchesinCity ";

static public string specialQuery2 = "WITH companiesAndClients AS ( " +

"SELECT companies.company\_id, company\_name, AVG(DISTINCT clients.client\_id ) as countOfCimpanyClietns FROM companies " +

"LEFT JOIN branches ON branches.company\_id=companies.company\_id " +"LEFT JOIN employees ON employees.branch\_id= branches.branch\_id " +

"LEFT JOIN contracts ON contracts.employee\_id = employees.employee\_id " +

"LEFT JOIN clients ON contracts.client\_id= clients.client\_id " +"GROUP BY companies.company\_id, company\_name), " +

"branchesAndClients AS( " +

"SELECT branches.company\_id, branches.branch\_id, concat (branch\_name, ', ', cities.city) branchAndCity, " +

"AVG(DISTINCT clients.client\_id ) as countOfBranchClietns, is\_main " +

"FROM branches " +"INNER JOIN cities ON branches.city\_id=cities.city\_id " +"LEFT JOIN employees ON employees.branch\_id= branches.branch\_id " +

"LEFT JOIN contracts ON contracts.employee\_id = employees.employee\_id " +

"LEFT JOIN clients ON contracts.client\_id= clients.client\_id " +"GROUP BY branches.branch\_id, branchAndCity) " +"SELECT DISTINCT company\_name, countOfCimpanyClietns, branchesAndClients.branchAndCity, branchesAndClients.countOfBranchClietns " +

"FROM companiesAndClients " +

"RIGHT OUTER JOIN branchesAndClients ON companiesAndClients.company\_id = branchesAndClients.company\_id AND branchesAndClients.is\_main = true " +

"ORDER BY company\_name, branchAndCity ";

static public string specialQuery3 = "SELECT company\_name, count( DISTINCT clients.client\_id ) as countOfCompanyClietns, SUM(sum\_of\_contract) as sumOfCompcontracts " +"FROM companies " +

"LEFT JOIN branches ON branches.company\_id=companies.company\_id " +

"LEFT JOIN employees ON employees.branch\_id=branches.branch\_id " +"LEFT JOIN contracts ON contracts.employee\_id = employees.employee\_id " +

"LEFT JOIN clients ON contracts.client\_id= clients.client\_id " +

"WHERE date\_of\_onclusion\_contract BETWEEN {0} AND {1} " +"GROUP BY company\_name " +"UNION ALL " +"SELECT 'ВСЕГО', count(DISTINCT clients.client\_id ) as countOfCompanyClietns, SUM(sum\_of\_contract) as sumOfCompcontracts FROM companies " +

"LEFT JOIN branches ON branches.company\_id=companies.company\_id " +"LEFT JOIN employees ON employees.branch\_id= branches.branch\_id " +

"LEFT JOIN contracts ON contracts.employee\_id = employees.employee\_id " +"LEFT JOIN clients ON contracts.client\_id= clients.client\_id " +

"WHERE date\_of\_onclusion\_contract BETWEEN {0} AND {1} " +

"ORDER BY countOfCompanyClietns DESC, sumOfCompcontracts DESC, company\_name DESC ";}}

namespace DB\_Kursach{

public partial class LogInForm : Form{

public LogInForm(){InitializeComponent();}

private void logInButton\_Click(object sender, EventArgs e){

//using (var con = new NpgsqlConnection($"Server=127.0.0.1;Port=5432;User Id={logInTextBox.Text.Trim()};Password={passwordTextBox.Text.Trim()};Database=InsuranceCompanies;"))

string str = "Server=127.0.0.1;Port=5432;User Id=postgres;Password=1111;Database=InsuranceCompanies;";if (String.IsNullOrEmpty(logInTextBox.Text.Trim()) || String.IsNullOrEmpty(passwordTextBox.Text.Trim())) return;

NpgsqlConnection con = new NpgsqlConnection();

try{string connectionString = $"Server=127.0.0.1;Port=5432;User Id={logInTextBox.Text.Trim()};Password={passwordTextBox.Text.Trim()};Database=InsuranceCompanies;";

con = new NpgsqlConnection(str); //con = new NpgsqlConnection(connectionString);con.Open();con.Close();MainMenuForm mm = new MainMenuForm(str);mm.Show();this.Hide();}catch (Exception){con.Close();MessageBox.Show("Неверный логин или пароль!");}}}}

namespace DB\_Kursach{

public partial class MainMenuForm : Form{

private readonly string \_connectionString = null;

private InteractionsWithForms interaction;

private DataForAddOrEdit DFAOE;

public MainMenuForm(string connect){

InitializeComponent();

\_connectionString = connect;

interaction = new InteractionsWithForms();

DFAOE = new DataForAddOrEdit();}

private void MainMenu\_Load(object sender, EventArgs e){

LoadTable(DBTablesTab.SelectedIndex);

exportToExelButton.Enabled = false;}

private void MainMenu\_Closed(object sender, FormClosedEventArgs e){Application.Exit();}

private void toolStrip1\_ItemClicked(object sender, ToolStripItemClickedEventArgs e){}

private void DBTablesTab\_Changed(object sender, EventArgs e){

exportToExelButton.Enabled = false;

LoadTable(DBTablesTab.SelectedIndex);}

private void LoadTable(int index, bool defaultQueryOrSpecial = true) // defaultQueryOrSpecial - вклкдка с запросами будет для обычных запросов (true), или для специальных(false) {

switch(index){case 0: //Компании{DataTable dt = WorkWithDataBase.SelectAllFieldsFromTable("all\_companies\_view", \_connectionString);

if (dt != null){companiesGrid.DataSource = dt;companiesGrid.BackgroundColor = Color.White;

if (companiesGrid.Columns.Count == 10){companiesGrid.Columns.RemoveAt(9);}

string cmd = "SELECT company\_id, license\_photo FROM companies ORDER BY company\_id ASC "; SetImagesFromTable(cmd);SetNamesOfAllColumnsOfAllTables(0);}}break;

case 1: //Типы собственности{ DataTable dt = WorkWithDataBase.SelectAllFieldsFromTable("all\_property\_types\_view", \_connectionString);

if (dt != null){propertyTypesGrid.DataSource = dt;propertyTypesGrid.BackgroundColor = Color.White;SetNamesOfAllColumnsOfAllTables(1);}}break;

case 2: //Филиалы{DataTable dt = WorkWithDataBase.SelectAllFieldsFromTable("all\_branches\_view", \_connectionString);

if (dt != null){branchesGrid.DataSource = dt;branchesGrid.BackgroundColor = Color.White;SetNamesOfAllColumnsOfAllTables(2);}}break;

case 3: //Сотрудники{DataTable dt = WorkWithDataBase.SelectAllFieldsFromTable("all\_employees\_view", \_connectionString);

if (dt != null){employeesGrid.DataSource = dt;employeesGrid.BackgroundColor = Color.White;SetNamesOfAllColumnsOfAllTables(3);}}break;

case 4: //Договоры{DataTable dt = WorkWithDataBase.SelectAllFieldsFromTable("all\_contracts\_view", \_connectionString);

if (dt != null){contractsGrid.DataSource = dt;contractsGrid.BackgroundColor = Color.White;SetNamesOfAllColumnsOfAllTables(4);}}break;

case 9: //Запросы{

headerTextLabel.Text = $"Запросы";queryNamesComboBox.Items.Clear();queryNamesComboBox.Items.AddRange(CommandsForRequests.queryNames);queryNamesComboBox.Items.AddRange(CommandsForRequests.specialQueryNames);

label10.Text = "Количество полей: " + requestsGrid.Rows.Count;}break;

default:break;}}

private void SetImagesFromTable(string cmd){

if (companiesGrid.Columns.Count != 9) return;

DataTable dt = WorkWithDataBase.SpecifitSelectFieldsFromTable(cmd, \_connectionString);

List<Bitmap> bmps = new List<Bitmap>();

var dataArray = dt.Rows.Cast<DataRow>().Select(x => x.ItemArray[1]);

foreach (var i in dataArray){

try{using (var ms = new MemoryStream((byte[])i)){

bmps.Add(new Bitmap(ms));}}

catch (Exception){bmps.Add(new Bitmap(10, 10));}}

//добавил столбец для изображений

DataGridViewImageColumn imageColumn = new DataGridViewImageColumn();

imageColumn.ImageLayout = DataGridViewImageCellLayout.Zoom;

companiesGrid.Columns.Add(imageColumn);

for (int i = 0; i < dataArray.Count(); i++){

companiesGrid.Rows[i].Cells[9].Value = bmps[i];}}

//соблюдается последовательно добавлений строк в таблицы, для того чтобы избеждать дополнительных изменений при добавлениях

private async void generationToolStripMenuItem\_Click(object sender, EventArgs e){

clearAllToolStripMenuItem.Enabled = false;

deleteButton.Enabled = false;editButton.Enabled = false;

string msg = await interaction.GenerateAllTablesAsync(\_connectionString);

LoadTable(DBTablesTab.SelectedIndex);

MessageBox.Show(msg);}

case 0: //Компании{if (companiesGrid.SelectedRows.Count == 0) return;

if (MessageBox.Show("Удаление данных из текущей таблицы может повлечь за собой удаление данных из других таблиц: Филиалы, Сотрудники, Договоры\n\nВсе равно далить?",

"Warning!", MessageBoxButtons.YesNo) == DialogResult.No) return;

if (companiesGrid.SelectedRows.Count == WorkWithDataBase.SelectRowsCountFromTable("companies", \_connectionString)){

WorkWithDataBase.ClearTableCascade("companies", \_connectionString);}

else // если выбраны не все элементы - проходимся (удаляем) выбранные по одному{

for (int i = 0; i < companiesGrid.SelectedRows.Count; i++){

WorkWithDataBase.RemoveLineFromTable("companies", $"company\_id = {companiesGrid.SelectedRows[i].Cells[0].Value}", \_connectionString);}}}break;}

LoadTable(DBTablesTab.SelectedIndex);}

private void editButton\_Click(object sender, EventArgs e){

if (DBTablesTab.SelectedIndex == 9) return;

int ID = 0;

switch (DBTablesTab.SelectedIndex){

case 0: //Компании{if (companiesGrid.SelectedRows.Count != 1){MessageBox.Show("Выберите один элемент для редактирования!", "Error!");return;}

ID = Convert.ToInt32(companiesGrid.SelectedRows[0].Cells[0].Value);}break;}

ID = Convert.ToInt32(citiesGrid.SelectedRows[0].Cells[0].Value);}break;

default:return;}

AddOrEditElementForm editElem = new AddOrEditElementForm(HowAddOrEdit.DefaultEdit, DBTablesTab.SelectedIndex, \_connectionString, ID);

editElem.ShowDialog();

LoadTable(DBTablesTab.SelectedIndex);}

private void clientsGrid\_CellDoubleClick(object sender, DataGridViewCellEventArgs e){

try {DataGridViewTextBoxCell cell = (DataGridViewTextBoxCell)clientsGrid.Rows[e.RowIndex].Cells[0]; //узнаю айди клиента на которого кликнули

ClientContractsForm CC = new ClientContractsForm((int)cell.Value, \_connectionString);

CC.Show();}

catch (Exception){}}

private void searchButton\_Click(object sender, EventArgs e){

if (DBTablesTab.SelectedIndex == 9) return;

int index = DBTablesTab.SelectedIndex;

SearchForm theSearch = new SearchForm(index, \_connectionString);

theSearch.ShowDialog();

if (theSearch.\_stringOfSelect == null) return;

LoadSearchTalbe(index, theSearch.\_stringOfSelect);}

private void applyQueryButton\_Click(object sender, EventArgs e){

exportToExelButton.Enabled = false;

switch (queryNamesComboBox.SelectedIndex){

case 0:{

if (queryComboBox.SelectedItem == null) return;

DFAOE.CitiesNamesAndIDs.TryGetValue(queryComboBox.Text, out int cityID);

string cmd = String.Format(CommandsForRequests.innerJoinWithForeignKey1, cityID);

DataTable dt = WorkWithDataBase.SpecifitSelectFieldsFromTable(cmd, \_connectionString);

if (dt != null){

requestsGrid.DataSource = dt;

if (requestsGrid.Columns.Count == CommandsForRequests.queryColumnsNames[0].Count()){

requestsGrid.BackgroundColor = Color.White;

requestsGrid.ClearSelection();

for (int i = 0; i < CommandsForRequests.queryColumnsNames[0].Count(); i++){

requestsGrid.Columns[i].HeaderText = CommandsForRequests.queryColumnsNames[0][i];}

exportToExelButton.Enabled = true;

interaction.indexOfQueriesComboBox = 0;

interaction.isSpecialQuery = false; }}

label10.Text = "Количество полей: " + requestsGrid.Rows.Count;}break;

case 1:{

if (queryComboBox.SelectedItem == null) return;

DFAOE.BranchesNamesAndIDs.TryGetValue(queryComboBox.Text, out int branchID);

string cmd = String.Format(CommandsForRequests.innerJoinWithForeignKey2, branchID);

DataTable dt = WorkWithDataBase.SpecifitSelectFieldsFromTable(cmd, \_connectionString);

if (dt != null){

requestsGrid.DataSource = dt;

if (requestsGrid.Columns.Count == CommandsForRequests.queryColumnsNames[1].Count()){

requestsGrid.BackgroundColor = Color.White;

requestsGrid.ClearSelection();

for (int i = 0; i < CommandsForRequests.queryColumnsNames[1].Count(); i++){

requestsGrid.Columns[i].HeaderText = CommandsForRequests.queryColumnsNames[1][i];}

requestsGrid.Columns[4].AutoSizeMode = DataGridViewAutoSizeColumnMode.DisplayedCells;

exportToExelButton.Enabled = true;interaction.indexOfQueriesComboBox = 1;

interaction.isSpecialQuery = false;}}

label10.Text = "Количество полей: " + requestsGrid.Rows.Count;}break;

case 2:{

DataTable dt = WorkWithDataBase.SpecifitSelectFieldsFromTable(CommandsForRequests.leftOuterJoin, \_connectionString);

if (dt != null){

requestsGrid.DataSource = dt;

if (requestsGrid.Columns.Count == CommandsForRequests.queryColumnsNames[2].Count()){

requestsGrid.BackgroundColor = Color.White;

requestsGrid.ClearSelection();

for (int i = 0; i < CommandsForRequests.queryColumnsNames[2].Count(); i++){

requestsGrid.Columns[i].HeaderText = CommandsForRequests.queryColumnsNames[2][i];}

exportToExelButton.Enabled = true;interaction.indexOfQueriesComboBox = 2;

interaction.isSpecialQuery = false;}}label10.Text = "Количество полей: " + requestsGrid.Rows.Count;}

break;

case 3:{

DataTable dt = WorkWithDataBase.SpecifitSelectFieldsFromTable(CommandsForRequests.rightOuterJoin, \_connectionString);

if (dt != null){

requestsGrid.DataSource = dt;

if (requestsGrid.Columns.Count == CommandsForRequests.queryColumnsNames[3].Count()) {

requestsGrid.BackgroundColor = Color.White;

requestsGrid.ClearSelection();

for (int i = 0; i < CommandsForRequests.queryColumnsNames[3].Count(); i++){

requestsGrid.Columns[i].HeaderText = CommandsForRequests.queryColumnsNames[3][i];}

exportToExelButton.Enabled = true;interaction.indexOfQueriesComboBox = 3;

interaction.isSpecialQuery = false;}}label10.Text = "Количество полей: " + requestsGrid.Rows.Count;}

break;

case 4:{

DataTable dt = WorkWithDataBase.SpecifitSelectFieldsFromTable(CommandsForRequests.queryOnQueryByLeftJoinPrinciple, \_connectionString);

if (dt != null){

requestsGrid.DataSource = dt;

if (requestsGrid.Columns.Count == CommandsForRequests.queryColumnsNames[4].Count()){

requestsGrid.BackgroundColor = Color.White;

requestsGrid.ClearSelection();

for (int i = 0; i < CommandsForRequests.queryColumnsNames[4].Count(); i++){

requestsGrid.Columns[i].HeaderText = CommandsForRequests.queryColumnsNames[4][i];}

exportToExelButton.Enabled = true;interaction.indexOfQueriesComboBox = 4;

interaction.isSpecialQuery = false;}}

label10.Text = "Количество полей: " + requestsGrid.Rows.Count; }break;

case 5:{DataTable dt = WorkWithDataBase.SpecifitSelectFieldsFromTable(CommandsForRequests.finalQueryWithoutCondition, \_connectionString);

if (dt != null){

requestsGrid.DataSource = dt;

if (requestsGrid.Columns.Count == CommandsForRequests.queryColumnsNames[5].Count()){

requestsGrid.BackgroundColor = Color.White;

requestsGrid.ClearSelection();

for (int i = 0; i < CommandsForRequests.queryColumnsNames[5].Count(); i++){

requestsGrid.Columns[i].HeaderText = CommandsForRequests.queryColumnsNames[5][i];}

exportToExelButton.Enabled = true;interaction.indexOfQueriesComboBox = 5;

interaction.isSpecialQuery = false; }}

label10.Text = "Количество полей: " + requestsGrid.Rows.Count;}break;

case 6:{

DataTable dt = WorkWithDataBase.SpecifitSelectFieldsFromTable(CommandsForRequests.totalIncluding, \_connectionString);

if (dt != null){

requestsGrid.DataSource = dt;

if (requestsGrid.Columns.Count == CommandsForRequests.queryColumnsNames[6].Count()){

requestsGrid.BackgroundColor = Color.White;

requestsGrid.ClearSelection();

for (int i = 0; i < CommandsForRequests.queryColumnsNames[6].Count(); i++){

requestsGrid.Columns[i].HeaderText = CommandsForRequests.queryColumnsNames[6][i];}

exportToExelButton.Enabled = true;interaction.indexOfQueriesComboBox = 6;

interaction.isSpecialQuery = false;}}

label10.Text = "Количество полей: " + requestsGrid.Rows.Count;}break;

case 7:{if (!Int32.TryParse(queryMaskedBox.Text.Trim('\_'), out int sum)) return;

string cmd = String.Format(CommandsForRequests.summaryQueriesWithConditionOnDataByValue, sum);

DataTable dt = WorkWithDataBase.SpecifitSelectFieldsFromTable(cmd, \_connectionString);

if (dt != null){

requestsGrid.DataSource = dt;

if (requestsGrid.Columns.Count == CommandsForRequests.queryColumnsNames[7].Count()){

requestsGrid.BackgroundColor = Color.White;

requestsGrid.ClearSelection();

for (int i = 0; i < CommandsForRequests.queryColumnsNames[7].Count(); i++){

requestsGrid.Columns[i].HeaderText = CommandsForRequests.queryColumnsNames[7][i];}

exportToExelButton.Enabled = true;interaction.indexOfQueriesComboBox = 7;

interaction.isSpecialQuery = false;}}

label10.Text = "Количество полей: " + requestsGrid.Rows.Count;}break;

case 8:{

if (!Int32.TryParse(queryMaskedBox.Text.Trim('\_'), out int phoneNumber)) return;

string cmd = String.Format(CommandsForRequests.summaryQueriesWithConditionOnDataByMask, phoneNumber);

DataTable dt = WorkWithDataBase.SpecifitSelectFieldsFromTable(cmd, \_connectionString);

if (dt != null){

requestsGrid.DataSource = dt;

if (requestsGrid.Columns.Count == CommandsForRequests.queryColumnsNames[8].Count()){

requestsGrid.BackgroundColor = Color.White;

requestsGrid.ClearSelection();

for (int i = 0; i < CommandsForRequests.queryColumnsNames[8].Count(); i++){

requestsGrid.Columns[i].HeaderText = CommandsForRequests.queryColumnsNames[8][i];}

exportToExelButton.Enabled = true;interaction.indexOfQueriesComboBox = 8;

interaction.isSpecialQuery = false;}}

label10.Text = "Количество полей: " + requestsGrid.Rows.Count;}break;

case 9:{DataTable dt = WorkWithDataBase.SpecifitSelectFieldsFromTable(CommandsForRequests.summaryQueriesWithConditionOnDataByIndex, \_connectionString);

if (dt != null){

requestsGrid.DataSource = dt;

if (requestsGrid.Columns.Count == CommandsForRequests.queryColumnsNames[9].Count()){

requestsGrid.BackgroundColor = Color.White;

requestsGrid.ClearSelection();

for (int i = 0; i < CommandsForRequests.queryColumnsNames[9].Count(); i++){

requestsGrid.Columns[i].HeaderText = CommandsForRequests.queryColumnsNames[9][i];}

exportToExelButton.Enabled = true;interaction.indexOfQueriesComboBox = 9;

interaction.isSpecialQuery = false;}}

label10.Text = "Количество полей: " + requestsGrid.Rows.Count;}break;

case 10:{

if (queryComboBox.SelectedItem == null) return;

DFAOE.EmployeesNamesAndIDs.TryGetValue(queryComboBox.Text, out int branchID);

string cmd = String.Format(CommandsForRequests.summaryQueriesWithConditionOnDataWithoutIndex, branchID);

DataTable dt = WorkWithDataBase.SpecifitSelectFieldsFromTable(cmd, \_connectionString);

if (dt != null){

requestsGrid.DataSource = dt;

if (requestsGrid.Columns.Count == CommandsForRequests.queryColumnsNames[10].Count()){

requestsGrid.BackgroundColor = Color.White;

requestsGrid.ClearSelection();

for (int i = 0; i < CommandsForRequests.queryColumnsNames[10].Count(); i++){

requestsGrid.Columns[i].HeaderText = CommandsForRequests.queryColumnsNames[10][i];}

exportToExelButton.Enabled = true;interaction.indexOfQueriesComboBox = 10;

interaction.isSpecialQuery = false;}}

label10.Text = "Количество полей: " + requestsGrid.Rows.Count;}break;

// СПЕЦИАЛЬНЫЕ ЗАПРОСЫ

case 19:{

string cmd = "";

if (queryComboBox.SelectedItem == null || (queryComboBox.SelectedItem != null && queryComboBox.Text == "По всем городам")){

cmd = String.Format(CommandsForRequests.specialQuery1, "");}

else{

DFAOE.CitiesNamesAndIDs.TryGetValue(queryComboBox.Text, out int cityID);

cmd = String.Format(CommandsForRequests.specialQuery1, $"WHERE branches.city\_id = {cityID} ");}

DataTable dt = WorkWithDataBase.SpecifitSelectFieldsFromTable(cmd, \_connectionString);

if (dt != null){

requestsGrid.DataSource = dt;

if (requestsGrid.Columns.Count == CommandsForRequests.specialQueryColumnsNames[0].Count()){

requestsGrid.BackgroundColor = Color.White;

requestsGrid.ClearSelection();

for (int i = 0; i < CommandsForRequests.specialQueryColumnsNames[0].Count(); i++){

requestsGrid.Columns[i].HeaderText = CommandsForRequests.specialQueryColumnsNames[0][i];}

exportToExelButton.Enabled = true;interaction.indexOfQueriesComboBox = 0;

interaction.isSpecialQuery = true;}}

label10.Text = "Количество полей: " + requestsGrid.Rows.Count;}break;

case 20:{

DataTable dt = WorkWithDataBase.SpecifitSelectFieldsFromTable(CommandsForRequests.specialQuery2, \_connectionString);

if (dt != null){

requestsGrid.DataSource = dt;

if (requestsGrid.Columns.Count == CommandsForRequests.specialQueryColumnsNames[1].Count()){

requestsGrid.BackgroundColor = Color.White;

requestsGrid.ClearSelection();

for (int i = 0; i < CommandsForRequests.specialQueryColumnsNames[1].Count(); i++){

requestsGrid.Columns[i].HeaderText = CommandsForRequests.specialQueryColumnsNames[1][i];}

//exportToExelButton.Visible = true;

exportToExelButton.Enabled = true;

interaction.indexOfQueriesComboBox = 1;

interaction.isSpecialQuery = true;}}

label10.Text = "Количество полей: " + requestsGrid.Rows.Count;}break;

case 21:{

string[] dates ={

$"'{queryDatePicker1.Value.Year}-{queryDatePicker1.Value.Month}-{queryDatePicker1.Value.Day}'",

$"'{queryDatePicker2.Value.Year}-{queryDatePicker2.Value.Month}-{queryDatePicker2.Value.Day}'"};

string cmd = String.Format(CommandsForRequests.specialQuery3, dates[0], dates[1]);

DataTable dt = WorkWithDataBase.SpecifitSelectFieldsFromTable(cmd, \_connectionString);

if (dt != null){

requestsGrid.DataSource = dt;

if (requestsGrid.Columns.Count == CommandsForRequests.specialQueryColumnsNames[2].Count()){

requestsGrid.BackgroundColor = Color.White;

for (int i = 0; i < CommandsForRequests.specialQueryColumnsNames[2].Count(); i++){

requestsGrid.Columns[i].HeaderText = CommandsForRequests.specialQueryColumnsNames[2][i];}

exportToExelButton.Enabled = true;interaction.indexOfQueriesComboBox = 2;

interaction.isSpecialQuery = true;}}

label10.Text = "Количество полей: " + requestsGrid.Rows.Count;}break;}}

private async void queryNamesComboBox\_SelectedIndexChanged(object sender, EventArgs e){

queryComboBox.Items.Clear();

queryMaskedBox.Clear();

exportToExelButton.Enabled = false;

switch (queryNamesComboBox.SelectedIndex){

case 0:{queryComboBox.Items.AddRange(await DFAOE.SelectCitiesNamesAndIDsAsync(\_connectionString));}break;

case 1:{queryComboBox.Items.AddRange(await DFAOE.SelectBranchesNamesAndIDsAsync(\_connectionString));}break;

case 10: {queryComboBox.Items.AddRange(await DFAOE.SelectEmployeesNamesAndIDsAsync(\_connectionString));}break;

case 12:{queryComboBox.Items.AddRange(await DFAOE.SelectCompnaiesNamesAndIDsAsync(\_connectionString));}break;

case 15:{queryComboBox.Items.AddRange(await DFAOE.SelectCompnaiesNamesAndIDsAsync(\_connectionString));}break;

case 16:{queryComboBox.Items.AddRange(await DFAOE.SelectCitiesNamesAndIDsAsync(\_connectionString));}break;

case 19:{queryComboBox.Items.Add("По всем городам");queryComboBox.Items.AddRange(await DFAOE.SelectCitiesNamesAndIDsAsync(\_connectionString));}break;

case 21:{}break;}}

private void specialQueriesToolStriptem\_Click(object sender, EventArgs e){

LoadTable(9, false); // false = для специальных запросов}

private void exportToExelButton\_Click(object sender, EventArgs e){

var dialog = saveExceleDialog.ShowDialog();

if (dialog != DialogResult.OK && dialog != DialogResult.Yes) return;

if (!SaveTableToExcel(saveExceleDialog.FileName)){

MessageBox.Show("Ошибка при экспорте таблицы в Excel");return;}

var res = MessageBox.Show("Экспорт завершен.\nYes - открыть сгенерированный файл." +

"\nNo - не открывать сгенерированный файл.", "Экспорт в Excel", MessageBoxButtons.YesNo);

if (res == DialogResult.Yes){

Process.Start(saveExceleDialog.FileName);}}

private bool SaveTableToExcel(string fileName){

try{DataTable dt = (DataTable)requestsGrid.DataSource;

if (!interaction.isSpecialQuery) {

for (int i = 0; i < CommandsForRequests.queryColumnsNames[interaction.indexOfQueriesComboBox].Count(); i++){

dt.Columns[i].ColumnName = CommandsForRequests.queryColumnsNames[interaction.indexOfQueriesComboBox][i];} }else{

for (int i = 0; i < CommandsForRequests.specialQueryColumnsNames[interaction.indexOfQueriesComboBox].Count(); i++){

dt.Columns[i].ColumnName = CommandsForRequests.specialQueryColumnsNames[interaction.indexOfQueriesComboBox][i];}}

var file = new FileInfo(fileName);

Task t = MyDataTableExtensions.SaveExcelFile(dt, file);return true;}

catch (Exception ex){return false;}}

private void diagram1DToolStripMenuItem\_Click(object sender, EventArgs e){DiagramsForm DF = new DiagramsForm(\_connectionString, 0); DF.Show();}

private void diagramСolumnarToolStripMenuItem\_Click(object sender, EventArgs e){

DiagramsForm DF = new DiagramsForm(\_connectionString, 1);DF.Show();}

private void diagram3DToolStripMenuItem\_Click(object sender, EventArgs e){

DiagramsForm DF = new DiagramsForm(\_connectionString, 2);DF.Show();}private void refreshGridButton\_Click(object sender, EventArgs e){

RefreshDataGrid(DBTablesTab.SelectedIndex);}}}

namespace DB\_Kursach{

public partial class ClientContractsForm : Form{

private readonly string \_connectionString;

InteractionsWithForms interactions = new InteractionsWithForms();

public ClientContractsForm(int clientID, string connString){

InitializeComponent();

interactions = new InteractionsWithForms(clientID);

\_connectionString = connString;}

private void ClientContracts\_Load(object sender, EventArgs e){

resetButton.Enabled = false;

DataTable dt = WorkWithDataBase.SpecifitSelectFieldsFromTable("SELECT surname, firstname, lastname, date\_of\_birth, social\_status\_of\_clients.social\_status, cities.city, address, phone\_number " +

"FROM clients " +

"INNER JOIN social\_status\_of\_clients ON social\_status\_of\_clients.social\_status\_id = clients.social\_status\_id " +

"INNER JOIN cities ON cities.city\_id = clients.city\_id " +

$"WHERE clients.client\_id = {interactions.clientID}", \_connectionString);

var stringArr = dt.Rows[0].ItemArray.Select(x => x.ToString()).ToArray();

stringArr[3] = stringArr[3].Substring(0, 10); //достаетм дату без времени 1/1/1111 вместо 1/1/1111 AM: 12:12:12

cNameLabel.Text = "ФИО: " + stringArr[0] + " " + stringArr[1] + " " + stringArr[2];

dateOfBirthLabel.Text = "Дата рождения: " + stringArr[3];

socialStatusLabel.Text = "Социальное положение: " + stringArr[4];

cityLabel.Text = "Город: " + stringArr[5];

addressLabel.Text = "Адрес: " + stringArr[6];

phoneLabel.Text = "Телефон: " + stringArr[7];

// Итоговый запрос с условием

dt = WorkWithDataBase.SpecifitSelectFieldsFromTable("SELECT contract\_id, types\_of\_insurance.type\_of\_insurance, employees.surname, employees.firstname, employees.lastname, text\_of\_contract, sum\_of\_contract, date\_of\_onclusion\_contract, branches.branch\_name " +

"FROM contracts " +

"INNER JOIN types\_of\_insurance ON contracts.type\_of\_insurance\_id =types\_of\_insurance.type\_of\_insurance\_id " +

"INNER JOIN employees ON contracts.employee\_id=employees.employee\_id " +

"INNER JOIN branches ON branches.branch\_id = employees.branch\_id " +

$"WHERE contracts.client\_id = {interactions.clientID} " +

"ORDER BY contract\_id ASC;", \_connectionString);

if (dt != null){

clientContractsGrid.DataSource = dt;

clientContractsGrid.BackgroundColor = Color.White;

SetColumnsNames();}

countOfClientContractsLabel.Text = "Количество полей: " + clientContractsGrid.Rows.Count;

sumLabel.Text = "Общая стоимость договоров: " + WorkWithDataBase.SelectLineFromTable("sum\_of\_contract", "contracts", \_connectionString, "SUM", $"WHERE client\_id={interactions.clientID}");}

private void sortBySumutton\_Click(object sender, EventArgs e){

string howSorting = "ASC;";

// изменения вида сортировки для следующего щелчка

if (interactions.sortSumASCorDESC){

interactions.sortSumASCorDESC = false;howSorting = "DESC;";}

else{interactions.sortSumASCorDESC = true;howSorting = "ASC;";}

if (interactions.Date2.Year == 1){

interactions.SetDate2(DateTime.Now);}

DataTable dt = WorkWithDataBase.SpecifitSelectFieldsFromTable("SELECT contract\_id, types\_of\_insurance.type\_of\_insurance, employees.surname, employees.firstname, employees.lastname, text\_of\_contract, sum\_of\_contract, date\_of\_onclusion\_contract, branches.branch\_name " +

"FROM contracts " +"INNER JOIN types\_of\_insurance ON contracts.type\_of\_insurance\_id =types\_of\_insurance.type\_of\_insurance\_id " +

"INNER JOIN employees ON contracts.employee\_id=employees.employee\_id " +

"INNER JOIN branches ON branches.branch\_id = employees.branch\_id " +

$"WHERE contracts.client\_id = {interactions.clientID} " +

$"AND date\_of\_onclusion\_contract BETWEEN '{interactions.Date1.Year}-{interactions.Date1.Month}-{interactions.Date1.Day}' AND '{interactions.Date2.Year}-{interactions.Date2.Month}-{interactions.Date2.Day}' " +

"ORDER BY sum\_of\_contract " + howSorting, \_connectionString);

if (dt != null){clientContractsGrid.DataSource = dt;SetColumnsNames();}

resetButton.Enabled = true;}

private void sortByTypeButton\_Click(object sender, EventArgs e){

if (interactions.Date2.Year == 1){

interactions.SetDate2(DateTime.Now);}

DataTable dt = WorkWithDataBase.SpecifitSelectFieldsFromTable("SELECT contract\_id, types\_of\_insurance.type\_of\_insurance, employees.surname, employees.firstname, employees.lastname, text\_of\_contract, sum\_of\_contract, date\_of\_onclusion\_contract, branches.branch\_name " +

"FROM contracts " +

"INNER JOIN types\_of\_insurance ON contracts.type\_of\_insurance\_id = types\_of\_insurance.type\_of\_insurance\_id " +

"INNER JOIN employees ON contracts.employee\_id=employees.employee\_id " +

"INNER JOIN branches ON branches.branch\_id = employees.branch\_id " +

$"WHERE contracts.client\_id = {interactions.clientID} " +

$"AND date\_of\_onclusion\_contract BETWEEN '{interactions.Date1.Year}-{interactions.Date1.Month}-{interactions.Date1.Day}' AND '{interactions.Date2.Year}-{interactions.Date2.Month}-{interactions.Date2.Day}' " +

"ORDER BY types\_of\_insurance.type\_of\_insurance ASC;", \_connectionString);

if (dt != null){

clientContractsGrid.DataSource = dt;

SetColumnsNames();}

resetButton.Enabled = true;}

private void resetButton\_Click(object sender, EventArgs e){

interactions.SetDate1(DateTime.MinValue);

interactions.SetDate2(DateTime.Now);dateTimePicker1.ResetText();dateTimePicker1.ResetText();

interactions.sortSumASCorDESC = false;ClientContracts\_Load(sender, e);}

private void setTimeSpanButton\_Click(object sender, EventArgs e){interactions.SetDate1(dateTimePicker1.Value);interactions.SetDate2(dateTimePicker2.Value);

DataTable dt = WorkWithDataBase.SpecifitSelectFieldsFromTable("SELECT contract\_id, types\_of\_insurance.type\_of\_insurance, employees.surname, employees.firstname, employees.lastname, text\_of\_contract, sum\_of\_contract, date\_of\_onclusion\_contract, branches.branch\_name " +

"FROM contracts " +

"INNER JOIN types\_of\_insurance ON contracts.type\_of\_insurance\_id =types\_of\_insurance.type\_of\_insurance\_id " +

"INNER JOIN employees ON contracts.employee\_id=employees.employee\_id " +

"INNER JOIN branches ON branches.branch\_id = employees.branch\_id " +$"WHERE contracts.client\_id = {interactions.clientID} " +

$"AND date\_of\_onclusion\_contract BETWEEN '{interactions.Date1.Year}-{interactions.Date1.Month}-{interactions.Date1.Day}' AND '{interactions.Date2.Year}-{interactions.Date2.Month}-{interactions.Date2.Day}' " +

"ORDER BY contract\_id ASC;", \_connectionString);

if (dt != null){

clientContractsGrid.DataSource = dt;

SetColumnsNames();}

countOfClientContractsLabel.Text = "Количество полей: " + clientContractsGrid.Rows.Count;

sumLabel.Text = "Общая стоимость договоров: " + WorkWithDataBase.SelectLineFromTable("sum\_of\_contract", "contracts", \_connectionString, "SUM",

$"WHERE client\_id={interactions.clientID} AND date\_of\_onclusion\_contract BETWEEN '{interactions.Date1.Year}-{interactions.Date1.Month}-{interactions.Date1.Day}' AND '{interactions.Date2.Year}-{interactions.Date2.Month}-{interactions.Date2.Day}'");resetButton.Enabled = true;}

private void addContractButton\_Click(object sender, EventArgs e){

AddOrEditElementForm addElem = new AddOrEditElementForm(HowAddOrEdit.CompoundFormAdd, 4, \_connectionString, interactions.clientID);

addElem.ShowDialog();

ClientContracts\_Load(sender, e);}

private void deleteContractButton\_Click(object sender, EventArgs e){

if (clientContractsGrid.SelectedRows.Count == 0) return;

for (int i = 0; i < clientContractsGrid.SelectedRows.Count; i++){

WorkWithDataBase.RemoveLineFromTable("contracts", $"contract\_id = {clientContractsGrid.SelectedRows[i].Cells[0].Value}", \_connectionString);}

ClientContracts\_Load(sender, e);

clientContractsGrid.Refresh();}

private void editContractButton\_Click(object sender, EventArgs e){

if (clientContractsGrid.SelectedRows.Count != 1){

MessageBox.Show("Выберите один элемент для редактирования!", "Error!");return;}

int ID = Convert.ToInt32(clientContractsGrid.SelectedRows[0].Cells[0].Value);

AddOrEditElementForm editElem = new AddOrEditElementForm(HowAddOrEdit.CompoundFormEdit, 4, \_connectionString, ID);

editElem.ShowDialog();

ClientContracts\_Load(sender, e);}

private void selectAllContractsButton\_Click(object sender, EventArgs e){

if (clientContractsGrid.SelectedRows.Count == clientContractsGrid.Rows.Count){

clientContractsGrid.ClearSelection();}

else{clientContractsGrid.SelectAll();}}

private void ClientContractsForm\_Resize(object sender, EventArgs e){

panel1.Size = new Size(this.Width, panel1.Size.Height);} }}

namespace DB\_Kursach{

public partial class AddOrEditElementForm : Form{

private readonly string \_connectionString;

private DataForAddOrEdit DFAOE;

private HowAddOrEdit HAOE;

public AddOrEditElementForm(HowAddOrEdit howAddOrEdit, int tabIndex, string connstring, int id = -1){

InitializeComponent();

HAOE = howAddOrEdit;

\_connectionString = connstring;

DFAOE = new DataForAddOrEdit(tabIndex, id);}

private void AddElement\_Load(object sender, EventArgs e){

addOrEditElementTab.SelectedIndex = DFAOE.tabIndex;

string tableOrDirectory = (HAOE == HowAddOrEdit.DefaultEdit) ? "таблицы" : "таблицу"; // добавление/удаление в таблицу или справочник

if (addOrEditElementTab.SelectedIndex == 1 || addOrEditElementTab.SelectedIndex == 5 || addOrEditElementTab.SelectedIndex == 7 || addOrEditElementTab.SelectedIndex == 8){

panel1.Size = new Size(this.Width, panel1.Size.Height);headerTextLabel.Size = new Size(this.Size.Width, headerTextLabel.Size.Height);

tableOrDirectory = (HAOE == HowAddOrEdit.DefaultEdit) ? "справочника" : "справочник";}

if (HAOE == HowAddOrEdit.DefaultAdd){

Task t = LoadTableForAdd(addOrEditElementTab.SelectedIndex);this.Text = "Добавить элемент";

headerTextLabel.Text = $"Добавление элемента в {tableOrDirectory} '{addOrEditElementTab.SelectedTab.Text}'";}

else if (HAOE == HowAddOrEdit.DefaultEdit) // если запрос на редактирование - загружаем форму с текущими данными{

LoadTableForEdit(addOrEditElementTab.SelectedIndex);

headerTextLabel.Text = $"Изменение элемента из {tableOrDirectory} '{addOrEditElementTab.SelectedTab.Text}'";}

else if (HAOE == HowAddOrEdit.CompoundFormAdd) // если запрос на добавление из составной формы{

Task t = LoadTableForAdd(4);this.Text = "Добавить элемент";

headerTextLabel.Text = $"Добавление элемента в таблицу '{addOrEditElementTab.SelectedTab.Text}'";}

else if (HAOE == HowAddOrEdit.CompoundFormEdit) // если запрос на добавление из составной формы{

headerTextLabel.Text = $"Изменение элемента из таблицы '{addOrEditElementTab.SelectedTab.Text}'";}}

private void addCompleteButton\_Click(object sender, EventArgs e){

bool isGoodData = true; // ввел ли пользователь подходящие данные

if (String.IsNullOrEmpty(companyNameBox.Text.Trim())){

companyNameBox.BackColor = Color.Red;isGoodData = false;}

if (typeBox.SelectedItem == null){errorCompanyPropertyTypeLabel.Visible = true;isGoodData = false;}

if (!Int32.TryParse(licenseNumberBox.Text.Trim('\_'), out int licenseNumber)){licenseNumberBox.BackColor = Color.Red;isGoodData = false;}

if (citiesBox.SelectedItem == null){errorCompanyCityLabel.Visible = true;isGoodData = false;}

if (String.IsNullOrEmpty(addressBox.Text.Trim())){addressBox.BackColor = Color.Red;isGoodData = false;}

if (!phoneNumberBox.MaskFull){phoneNumberBox.BackColor = Color.Red;isGoodData = false;}

if (!openingYearBox.MaskFull || Convert.ToInt32(openingYearBox.Text) < 1850 || Convert.ToInt32(openingYearBox.Text) > DateTime.Now.Year){openingYearBox.BackColor = Color.Red;isGoodData = false;}

if (!isGoodData) return;

DFAOE.CitiesNamesAndIDs.TryGetValue(citiesBox.Text, out int cityID);

DFAOE.PTypesNamesAndIDs.TryGetValue(typeBox.Text, out int pTypeID);

byte[] img = null;

if (licensePhotoPictureBox.Image != null){

Bitmap tmp = new Bitmap(licensePhotoPictureBox.Image);

using (MemoryStream ms = new MemoryStream()){

tmp.Save(ms, System.Drawing.Imaging.ImageFormat.Jpeg);

img = ms.ToArray();}}

if (HAOE == HowAddOrEdit.DefaultAdd || HAOE == HowAddOrEdit.CompoundFormAdd){

if (WorkWithDataBase.SpecificInsertValuesWithImageIntoTable("public.companies(company\_name, type\_id, license\_number, license\_expiration\_date, city\_id, address, phone\_number, opening\_year, license\_photo)",

$"'{companyNameBox.Text.Trim()}', {pTypeID}, {licenseNumber}, '{endOfLicensePicker.Value.Year}-{endOfLicensePicker.Value.Month}-{endOfLicensePicker.Value.Day}', {cityID}, '{addressBox.Text.Trim()}', '{phoneNumberBox.Text}', {Convert.ToInt32(openingYearBox.Text)}", \_connectionString, img)){

MessageBox.Show("Запись успешно добавленна"); }

else{MessageBox.Show("При добавлении записи возникла ошибка!"); }}

else// редактирование элемента{

if (WorkWithDataBase.SpecificUpdateValuesWithImageFromTable("public.companies",

$"company\_name='{companyNameBox.Text.Trim()}', type\_id={pTypeID}, license\_number={licenseNumber}, license\_expiration\_date='{endOfLicensePicker.Value.Year}-{endOfLicensePicker.Value.Month}-{endOfLicensePicker.Value.Day}', city\_id={cityID}, address='{addressBox.Text.Trim()}', phone\_number='{phoneNumberBox.Text}', opening\_year={Convert.ToInt32(openingYearBox.Text)}, license\_photo=", $"company\_id={DFAOE.ID}", \_connectionString, img)){

MessageBox.Show("Запись успешно изменена");}

else{MessageBox.Show("При изменении записи возникла ошибка!");}}}

private void addElementTab\_Changed(object sender, EventArgs e){

if (addOrEditElementTab.SelectedIndex == 1 || addOrEditElementTab.SelectedIndex == 5 || addOrEditElementTab.SelectedIndex == 7 || addOrEditElementTab.SelectedIndex == 8){

this.Size = new Size(500, 300);}

else{this.Size = new Size(616, 405);}

if (HAOE == HowAddOrEdit.DefaultAdd){

Task task = LoadTableForAdd(addOrEditElementTab.SelectedIndex);}}

private async Task LoadTableForAdd(int index){

switch (index){

case 0: //Компании{

typeBox.Items.AddRange(await DFAOE.SelectPropertyTypesNamesAndIDsAsync(\_connectionString));

citiesBox.Items.AddRange(await DFAOE.SelectCitiesNamesAndIDsAsync(\_connectionString));

endOfLicensePicker.MinDate = DateTime.Now.Date;}break;

case 2: //Филиалы{//

ownedByCompanyBox.Items.AddRange(await DFAOE.SelectCompnaiesNamesAndIDsAsync(\_connectionString));

branchCityBox.Items.AddRange(await DFAOE.SelectCitiesNamesAndIDsAsync(\_connectionString));}break;

case 3: //Сотрудники{employeeBranchBox.Items.AddRange(await DFAOE.SelectBranchesNamesAndIDsAsync(\_connectionString));}break;

case 4: //Договоры{

contractInsuranceTypeComboBox.Items.AddRange(await DFAOE.SelectTypesOfInsuranceNamesAndIDsAsync(\_connectionString));

contractClientComboBox.Items.AddRange(await DFAOE.SelectClientsNamesAndIDsAsync(\_connectionString));

contractEmployeeComboBox.Items.AddRange(await DFAOE.SelectEmployeesNamesAndIDsAsync(\_connectionString));

contractDateTime.MaxDate = DateTime.Now;contractDateTime.MinDate = DateTime.Now;

if (HAOE == HowAddOrEdit.CompoundFormAdd){

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

if (i != index){addOrEditElementTab.TabPages[i].Enabled = false;}}

string fullnameOfClient = WorkWithDataBase.SelectLineFromTable("concat (surname, ' ', firstname, ' ', lastname)", "all\_clients\_view", \_connectionString, where: $"WHERE client\_id = {DFAOE.ID}");

if (fullnameOfClient != null){

contractClientComboBox.SelectedIndex = contractClientComboBox.FindStringExact(fullnameOfClient); // dropbox

contractClientComboBox.Enabled = false;}}

else if (HAOE == HowAddOrEdit.CompoundFormEdit){

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

if (i != index){

addOrEditElementTab.TabPages[i].Enabled = false; }}

string fullnameOfClient = WorkWithDataBase.SelectLineFromTable("concat (surname, ' ', firstname, ' ', lastname)",

"clients", \_connectionString, where: $"WHERE (SELECT client\_id FROM contracts WHERE contract\_id = {DFAOE.ID}) = clients.client\_id");

if (fullnameOfClient != null){

contractClientComboBox.SelectedIndex = contractClientComboBox.FindStringExact(fullnameOfClient); // dropbox

contractClientComboBox.Enabled = false;}}}break;

case 6: //Клиенты{

clientDateOfBirthTimePicker.MaxDate = new DateTime(DateTime.Now.Year - 18, DateTime.Now.Month, DateTime.Now.Day);

clientDateOfBirthTimePicker.MinDate = new DateTime(DateTime.Now.Year - 120, DateTime.Now.Month, DateTime.Now.Day);

clientCityBox.Items.AddRange(await DFAOE.SelectCitiesNamesAndIDsAsync(\_connectionString));

clientSocailStatusBox.Items.AddRange(await DFAOE.SelectSocialStatusesNamesAndIDsAsync(\_connectionString));}break;

default:break; }}

private async void LoadTableForEdit(int index){

switch (index){

case 0: //Компании{

if (!addOrEditElementTab.TabPages[index].Enabled) return; // если изначально выбран другой элемент для редактирвоания

for (int i = 1; i < addOrEditElementTab.TabCount; i++){

addOrEditElementTab.TabPages[i].Enabled = false;}

await LoadTableForAdd(index);

DataTable dt = WorkWithDataBase.SelectAllFieldsFromTable($"all\_companies\_view WHERE company\_id = {DFAOE.ID}", \_connectionString);

if (dt != null){

var dataRows = dt.Rows[0].ItemArray.Select(x => x.ToString()).ToArray();

companyNameBox.Text = dataRows[1];

typeBox.SelectedIndex = typeBox.FindStringExact(dataRows[2]);

licenseNumberBox.Text = dataRows[3];endOfLicensePicker.MinDate = DateTime.Parse(dataRows[4]);endOfLicensePicker.Value = DateTime.Parse(dataRows[4]);

citiesBox.SelectedIndex = citiesBox.FindStringExact(dataRows[5]);addressBox.Text = dataRows[6];phoneNumberBox.Text = dataRows[7];openingYearBox.Text = dataRows[8];

//добаление изображения

string cmd = $"SELECT license\_photo FROM companies WHERE company\_id = {DFAOE.ID}";

licensePhotoPictureBox.Image = WorkWithDataBase.SelectImageFromTable(cmd, \_connectionString);}}break;

default:break;}}

private void applyBranchButton\_Click(object sender, EventArgs e){

bool isGoodData = true; // ввел ли пользователь подходящие данные

if (ownedByCompanyBox.SelectedItem == null){

errorOwnedByCompanyLabel.Visible = true;isGoodData = false;}

if (branchCityBox.SelectedItem == null){

errorBranchCityLabel.Visible = true;isGoodData = false;}

if (String.IsNullOrEmpty(branchAddressBox.Text.Trim())){

branchAddressBox.BackColor = Color.Red;isGoodData = false;}

if (!branchPhoneMaskedBox.MaskFull) //---{

branchPhoneMaskedBox.BackColor = Color.Red;isGoodData = false;}

if (!branchOpeningYearMaskedBox.MaskFull || Convert.ToInt32(branchOpeningYearMaskedBox.Text) < 1850 || Convert.ToInt32(branchOpeningYearMaskedBox.Text) > DateTime.Now.Year){

branchOpeningYearMaskedBox.BackColor = Color.Red;isGoodData = false;}

if (!isGoodData) return;

string openYearForMinDate = WorkWithDataBase.SelectLineFromTable("companies.opening\_year", "companies", \_connectionString, where: "INNER JOIN branches ON companies.company\_id=branches.company\_id " +

$"WHERE branch\_id = {DFAOE.ID} " +"LIMIT 1 ");

if (!Int32.TryParse(openYearForMinDate, out int YearForMinDate)) return;

if (Convert.ToInt32(branchOpeningYearMaskedBox.Text) < YearForMinDate){

MessageBox.Show($"Компания открылась в {YearForMinDate} году. Филиал не может быть открыть раньше!", "Error!");return;}

DFAOE.CitiesNamesAndIDs.TryGetValue(branchCityBox.Text, out int cityID);

DFAOE.CompanyNamesAndIDs.TryGetValue(ownedByCompanyBox.Text, out int companyID);

if (HAOE == HowAddOrEdit.DefaultAdd || HAOE == HowAddOrEdit.CompoundFormAdd){

if (WorkWithDataBase.InsertValuesIntoTable("public.branches(branch\_name, company\_id, city\_id, is\_main, address, phone\_number, opening\_year, number\_of\_employees)",

$"'{ownedByCompanyBox.Text}', {companyID}, {cityID}, 'false', '{branchAddressBox.Text.Trim()}', '{branchPhoneMaskedBox.Text}', {branchOpeningYearMaskedBox.Text}, DEFAULT", \_connectionString)){

MessageBox.Show("Запись успешно добавленна");}

else{

MessageBox.Show("При добавлении записи возникла ошибка!");}}

else // редактирование элемента{

if (WorkWithDataBase.UpdateValuesFromTable("branches",

$"branch\_name='{ownedByCompanyBox.Text}', company\_id={companyID}, city\_id={cityID}, is\_main='false', address='{branchAddressBox.Text.Trim()}', phone\_number='{branchPhoneMaskedBox.Text}', opening\_year={branchOpeningYearMaskedBox.Text}, number\_of\_employees=DEFAULT",

$"branch\_id = {DFAOE.ID}", \_connectionString)){

MessageBox.Show("Запись успешно изменена");}

else{MessageBox.Show("При изменении записи возникла ошибка!");}}}

private void contracInsuranceTypeComboBox\_SelectedIndexChanged(object sender, EventArgs e){

errorContractInsuranceTypeLabel.Visible = false;}

private void deleteImageButton\_Click(object sender, EventArgs e){

if (licensePhotoPictureBox.Image != null){

licensePhotoPictureBox.Image = null;}}

private void AddOrEditElementForm\_Resize(object sender, EventArgs e){

panel1.Size = new Size(this.Width, panel1.Size.Height);}}}

namespace DB\_Kursach{

public partial class SearchForm : Form{

private readonly string \_connectionString;

public string \_stringOfSelect = null; // строк запроса поиска

private DataForAddOrEdit DFAOE;

public SearchForm(int index, string connString){

InitializeComponent();

\_connectionString = connString;

DFAOE = new DataForAddOrEdit(index);}

private void SearchForm\_Load(object sender, EventArgs e){

searchTab.SelectedIndex = DFAOE.tabIndex;

Task t = LoadTableForSearchAsync(DFAOE.tabIndex);

if (searchTab.SelectedIndex == 1 || searchTab.SelectedIndex == 5 || searchTab.SelectedIndex == 7 || searchTab.SelectedIndex == 8){

headerTextLabel.Text = $"Поиск элемента в таблице '{searchTab.SelectedTab.Text}'";}

private void searchCompanyButton\_Click(object sender, EventArgs e){

string searchCommand = "SELECT \* FROM all\_companies\_view ";

if (companyCNamesComboBox.SelectedItem != null){

searchCommand += $"WHERE company\_name = '{companyCNamesComboBox.Text}' ";}

if (companyTypeBox.SelectedItem != null){

searchCommand += (searchCommand.Contains("WHERE")) ? "AND " : "WHERE ";

searchCommand += $"property\_type = '{companyTypeBox.Text}' ";}

if (licenseNumberCheckBox.Checked){

searchCommand += (searchCommand.Contains("WHERE")) ? "AND " : "WHERE ";

if (Int32.TryParse(licenseNumberBox1.Text.Trim('\_'), out int licenseNumber1) && Int32.TryParse(licenseNumberBox2.Text.Trim('\_'), out int licenseNumber2)){

searchCommand += $"license\_number >= {licenseNumber1} AND license\_number <= {licenseNumber2} ";}

else if (Int32.TryParse(licenseNumberBox1.Text.Trim('\_'), out licenseNumber1)){

searchCommand += $"license\_number >= {licenseNumber1} ";}

else if (Int32.TryParse(licenseNumberBox2.Text.Trim('\_'), out licenseNumber2)){

searchCommand += $"license\_number <= {licenseNumber2} ";}}

if (licenseDateCheckBox.Checked){

searchCommand += (searchCommand.Contains("WHERE")) ? "AND " : "WHERE ";

searchCommand += $"license\_expiration\_date >= '{endOfLicensePicker1.Value.Year}-{endOfLicensePicker1.Value.Month}-{endOfLicensePicker1.Value.Day}' " +

$"AND license\_expiration\_date <= '{endOfLicensePicker2.Value.Year}-{endOfLicensePicker2.Value.Month}-{endOfLicensePicker2.Value.Day}' ";}

if (companyCitiesBox.SelectedItem != null){

searchCommand += (searchCommand.Contains("WHERE")) ? "AND " : "WHERE ";

searchCommand += $"city = '{companyCitiesBox.Text}' ";}

if (companyPhoneNumberBox.MaskFull){

searchCommand += (searchCommand.Contains("WHERE")) ? "AND " : "WHERE ";

searchCommand += $"phone\_number = {companyCitiesBox.Text} ";}

if (openingYearCheckBox.Checked){

searchCommand += (searchCommand.Contains("WHERE")) ? "AND " : "WHERE ";

if (openingYearBox1.MaskFull && openingYearBox2.MaskFull){

searchCommand += $"opening\_year >= {openingYearBox1.Text} AND opening\_year <= {openingYearBox2.Text} ";}

else if (openingYearBox1.MaskFull){

searchCommand += $"opening\_year >= {openingYearBox1.Text} ";}

else if (openingYearBox2.MaskFull){

searchCommand += $"opening\_year <= {openingYearBox2.Text} ";}}

\_stringOfSelect = searchCommand;this.Close();}