ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту по курсу «Программирование систем с серверами баз данных»

Тема работы:

«Разработка информационной системы “Страховые компании”»

|  |  |
| --- | --- |
| Руководители: | Щедрин С.В. |
|  | Ногтев Е.А. |
|  | Филипишин Д.А. |
| (подпись) (дата) | |

|  |  |
| --- | --- |
| Разработала: | Суворина А. В. |
| ст. гр. ПИ-19а (подпись) (дата) | |

Донецк – 2022

РЕФЕРАТ

Пояснительная записка к курсовому проекту содержит: 88 страниц, 90 рисунков, 9 таблиц, 5 источников, 7 приложений.

Объект исследования – информационная система «Страховые компании».

Цель курсового проекта – рассмотреть основы проектирования баз данных, изучить следующие механизмы СУБД: роли, защита на уровне строк, триггеры, домены, индексы, функции, представления и партицирование.

Результат выполнения проекта – разработанная информационная система «Страховые компании», использующая вышеперечисленные механизмы СУБД.

БАЗА ДАННЫХ, RLS, СУБД, POSTGRESQL, ЗАЩИТА НА УРОВНЕ СТРОК, РОЛЬ, ТРИГГЕР, ЗАПРОС, ДОМЕН, ПАРТИЦИРОВАНИЕ, PYTHON

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc104561829)

[1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 6](#_Toc104561830)

[2 ОБОСНОВАНИЕ ВЫБОРА СУБД, ОПИСАНИЕ ВОЗМОЖНОСТЕЙ СУБД 8](#_Toc104561831)

[3 ОБОСНОВАНИЕ ВЫБОРА ИНСТРУМЕНТАЛЬНЫХ СРЕДСТВ ДЛЯ НАПИСАНИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ 9](#_Toc104561832)

[3.1 Невизуальные компоненты для работы с данными 9](#_Toc104561833)

[3.2 Визуальные компоненты для работы с данными 9](#_Toc104561834)

[3.3 Разработка шаблонов приложений для работы с шаблонами базы данных 11](#_Toc104561835)

[4 ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ 14](#_Toc104561836)

[4.1 Проектирование концептуальной модели БД 14](#_Toc104561837)

[4.2 Создание таблиц, доменов, индексов, последовательностей 18](#_Toc104561838)

[4.3 Разработка триггеров 21](#_Toc104561839)

[4.4 Организация многоролевого доступа к данным 24](#_Toc104561840)

[4.5 Разграничение доступа к данным на уровне строк 25](#_Toc104561841)

[4.6 Партицирование одной из основных таблиц БД 26](#_Toc104561842)

[4.7 Проектирование запросов к базе данных 26](#_Toc104561843)

[4.8 Разработка модифицируемого представления 42](#_Toc104561844)

[5 РАЗРАБОТКА ПРИЛОЖЕНИЯ 45](#_Toc104561845)

[5.1 Формы и компоненты для работы роли «Сотрудник филиала» 45](#_Toc104561846)

[5.2 Формы и компоненты для работы роли «Клиент» 47](#_Toc104561847)

[5.3 Формы и компоненты для работы роли «Администратор» 48](#_Toc104561848)

[5.4 Формы и компоненты для работы роли «Запросы» 50](#_Toc104561849)

[5.5 Экспортирование результата запроса в Excel 51](#_Toc104561850)

[6 ТЕСТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ 52](#_Toc104561851)

[ВЫВОДЫ 54](#_Toc104561852)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 55](#_Toc104561853)

[ПРИЛОЖЕНИЕ А ТЕХНИЧЕСКОЕ ЗАДАНИЕ 56](#_Toc104561854)

[ПРИЛОЖЕНИЕ Б ЛИСТИНГ ШАБЛОНОВ 62](#_Toc104561855)

[ПРИЛОЖЕНИЕ В ЛИСТИНГ СЕРВЕРНОГО ПРИЛОЖЕНИЯ 65](#_Toc104561856)

[ПРИЛОЖЕНИЕ Г ЛИСТИНГ КЛИЕНТСКОГО ПРИЛОЖЕНИЯ 73](#_Toc104561857)

[ПРИЛОЖЕНИЕ Д РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 86](#_Toc104561858)

[ПРИЛОЖЕНИЕ Е РУКОВОДСТВО АДМИНИСТРАТОРА 87](#_Toc104561859)

[ПРИЛОЖЕНИЕ Ж ОТЧЕТ О ПРОВЕРКЕ НА ЗАИМСТВОВАНИЯ 88](#_Toc104561860)

# ВВЕДЕНИЕ

База данных – это упорядоченный набор структурированной информации или данных, которые обычно хранятся в электронном виде в компьютерной системе. База данных обычно управляется системой управления базами данных (СУБД).

Данные в современных базах данных обычно хранятся в виде строк и столбцов, формирующих таблицу. Этими данными можно легко управлять, изменять, обновлять, контролировать и упорядочивать. В большинстве баз данных для записи и запросов данных используется язык структурированных запросов (SQL). [1]

В ходе выполнения курсового проекта в соответствии с техническим заданием необходимо спроектировать систему взаимодействия пользователя с БД и разработать экранные формы для нескольких ролей пользователей. На стороне сервера необходимо предусмотреть защиту на уровне строк, используя роли и политики защиты. Создать индексы, домены, разработать триггеры, выполнить партицирование одной из основных таблиц. Реализовать SQL запросы к базе данных в виде представлений и функций, создать модифицируемое представление, используя механизм триггеров и визуализировать результат запроса в Excel.

Также важной частью выполнения курсового проекта является выбор СУБД. Данный комплекс программных средств позволяет быстро и удобно создавать структуру базы, добавлять содержимое в БД и редактировать информацию.

Для разработки системы были использованы следующие инструменты: СУБД PostgreSQL, язык программирования Python 3.10, интегрированная среда разработки PyCharm, редактор SQL и менеджер баз данных Navicat.

Разработанная в ходе выполнения курсового проекта информационная система может быть использована в реальных условиях для учета деятельности страховых компаний.

# 1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

В задании курсовой работы требуется разработать базу данных о деятельности страховых компаний. Рассмотрим особенности данной системы.

Система обработки информации о деятельности страховых компаний Украины использует сведения о компаниях (название компании, тип (частная, государственная, акционерная,…), год получения лицензии, город, в котором находится центральное отделение, телефон, филиалы (название филиала, город, адрес, телефон, количество сотрудников)) и заключенных с филиалами договорах (клиент (ФИО, дата рождения, социальное положение (служащий, предприниматель, учащийся, рабочий, пенсионер,…), телефон), вид страхования (недвижимость, имущество, от несчастного случая, к бракосочетанию, до совершеннолетия,…), сумма страхования, дата заключения договора).

В ходе декомпозиции предметной области были выделены следующие таблицы: компании, филиалы, клиенты, договоры, города, типы компаний, социальные положения, виды страхования.

Исходя из условия многопользовательской системы, были выделены три группы пользователей: администратор базы данных и исходя из предметной области – сотрудник филиала и клиент, а также дополнительная роль для просмотра запросов.

Основными задачами администратора являются:

* Создание новых пользователей БД;
* Добавление/удаление прав пользователей;
* Управление справочниками БД.

К основным задачам сотрудника относятся:

* Просмотр списка договоров и клиентов своего филиала;
* Просмотр данных о своем филиале;
* Просмотр, добавление, удаление, изменение договоров.

Основные задачи клиента:

* Просмотр своих данных;
* Просмотр, добавление своих контрактов;
* Просмотр списка филиалов и компаний.

В результате анализа, необходимо реализовать приложение для многопользовательского использования базы данных, с вариативным доступом к интерфейсу клиентской части системы, а также ограничить доступ к данным в СУБД для варианта доступа к данным вне пользовательского интерфейса.

# 2 ОБОСНОВАНИЕ ВЫБОРА СУБД, ОПИСАНИЕ ВОЗМОЖНОСТЕЙ СУБД

В качестве системы управления базой данных была выбрана PostgreSQL.

PostgreSQL – это мощная СУБД, использующаяся для выполнения самых разных типов задач. Она обладает следующими преимуществами использования:

– поддержка БД неограниченного размера;

– мощные и надёжные механизмы транзакций и репликации;

– расширяемая система встроенных языков программирования и поддержка загрузки C-совместимых модулей;

– наследование;

– легкая расширяемость [2].

Созданный с использованием объектно-реляционной модели, PostgreSQL поддерживает сложные структуры и широкий спектр встроенных и определяемых пользователем типов данных. Он обеспечивает расширенную ёмкость данных и крайне бережно относится к целостности данных.

Помимо этих возможностей PostgreSQL обладает всеми необходимыми для выполнения курсового проекта механизмами: триггеры, представления, индексы, роли, домены, правила, защита на уровне строк и партицирование.

# 3 ОБОСНОВАНИЕ ВЫБОРА ИНСТРУМЕНТАЛЬНЫХ СРЕДСТВ ДЛЯ НАПИСАНИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ

В качестве языка программирования для написания информационной системы был выбран Python. Данный язык был выбран по ряду преимуществ, среди которых следует выделить совместимость с огромным количеством БД, высокую производительность, наличие динамической типизации данных, высокую читаемость итоговой кодификации и наличие множества библиотек.

Для разработки пользовательского графического интерфейса был выбран набор расширений графического фреймворка Qt для Python – PyQt5.

Для взаимодействия с СУБД используется самый популярный адаптер для PostgreSQL – Navicat, он обеспечивает доступ ко многим функциям СУБД.

Чтобы генерировать Excel файлы с диаграммами был выбран Python модуль xlsxwriter.

Данный набор инструментов позволяет удобно и быстро разрабатывать информационную систему, которая будет корректно работать на всех популярных операционных системах: Windows, Mac OS и Unix-подобные ОС.

## 3.1 Невизуальные компоненты для работы с данными

Для создания проекта был скачан модуль «psycopg» и импортирован в проект, что позволило в дальнейшем работать с базами данных.

## 

## 3.2 Визуальные компоненты для работы с данными

Для отображения данных из таблиц был выбран элемент управления QTableWidget, который является достаточно гибким и позволяет автоматизировать вывод информации пользователю. Пример использования данного элемента управления для отображения таблицы из базы данных представлен на рисунках 3.1 и 3.2.



Рисунок 3.1 – Пример использования QTableWidget для отображения данных таблицы

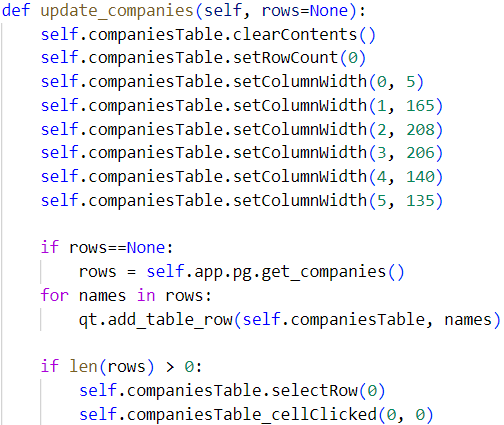


Рисунок 3.2 – Пример использования QTableWidget для отображения данных таблицы (код программы)

Для отображения справочников при добавлении/редактировании записей в базе данных был использован элемент управления QComboBox, который обеспечивает удобное отображение и выбор данных из справочника. Примеры использования данного элемента управления приведен на рисунках 3.3 и 3.4.

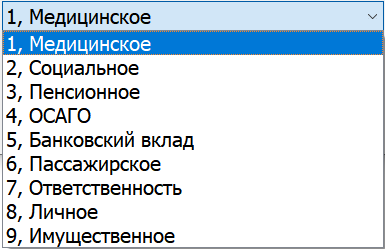


Рисунок 3.3 – Пример использования QComboBox для отображения справочников

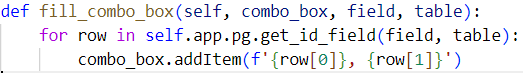


Рисунок 3.4 – Пример использования QComboBox для отображения справочников (код программы)

## 3.3 Разработка шаблонов приложений для работы с шаблонами базы данных

Данный программный продукт оснащён простым, интуитивно понятным пользовательским интерфейсом. В программе были созданы окна авторизации пользователя, работы с таблицами и работы с запросами.

Начало работы происходит в окне авторизации. Если авторизация происходит успешно, появляется окно для работы (добавление, удаление, изменение записей) с таблицами. Работа с таблицами происходит на основе разрешений конкретного пользователя.

Из окна для работы с таблицами пользователь может перейти на окно с перечнем запросов. В нём пользователь может выполнить запрос, получить результат, экспортировать результаты в excel.

Рассмотрим особенности каждой формы на рисунках 3.5 - 3.7.

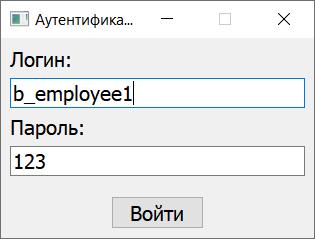


Рисунок 3.5 – Форма авторизации

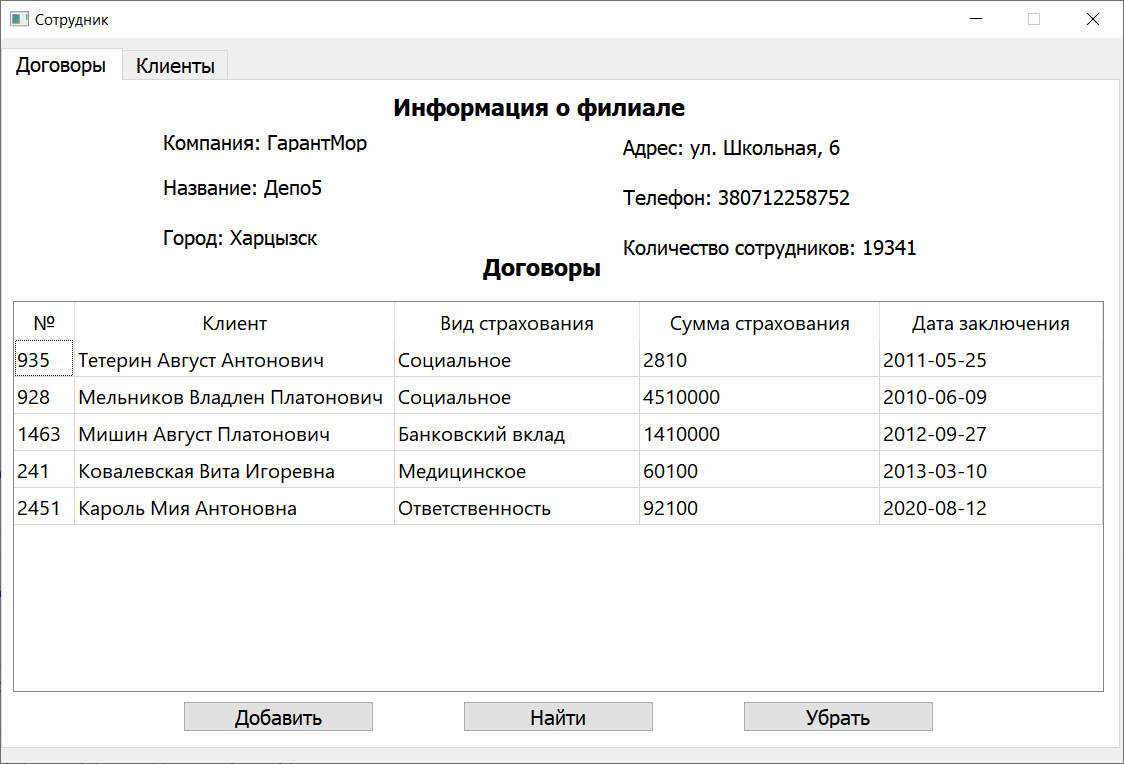


Рисунок 3.6 – Форма работы с таблицами

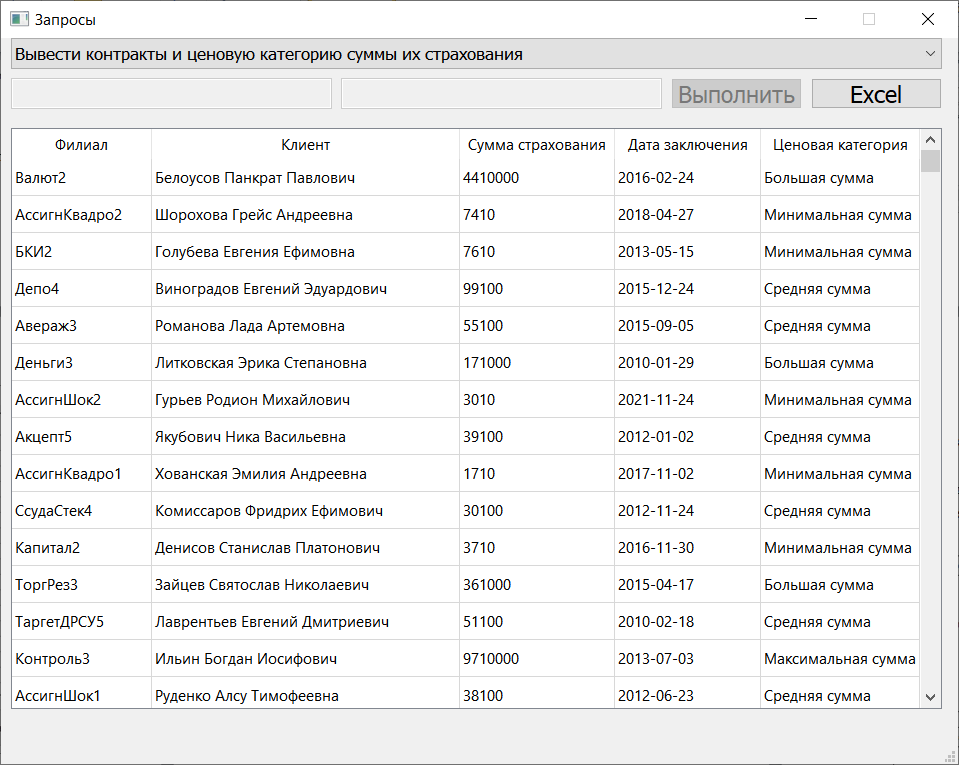


Рисунок 3.7 – Форма запросов

# 4 ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ

## 4.1 Проектирование концептуальной модели БД

В ходе анализа предметной области, данные были распределены по различным таблицам, между которыми установлены связи. Также была рассмотрена нормализация таблиц и добавление новых таблиц, для реализации многопользовательской защиты. Помимо ранее выделенных таблиц, была добавлена таблица с пользователями (users), с помощью которой реализована защита на уровне строк. В итоговом варианте база данных включает следующие таблицы:

* companies (компании);
* branches (филиалы);
* clients (клиенты);
* contracts (договоры);
* cities (города);
* company\_types (типы компаний);
* social\_statuses (социальные положения);
* insurance\_types (виды страхования);
* users (пользователи).

Структура таблиц приведена в таблицах 4.1 – 4.10.

Таблица 4.1 – Структура таблицы «companies»

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| id | integer | Первичный ключ |
| name | text | Название компании |
| company\_type\_id | integer | Внешний ключ, определяющий тип компании |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| year | integer | Год получения лицензии |
| city\_id | integer | Внешний ключ, определяющий город, в котором находится компания |
| phone | varchar(13) | Телефон компании |

Таблица 4.2 – Структура таблицы «branches»

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| id | integer | Первичный ключ |
| company\_id | integer | Внешний ключ, определяющий компанию, к которой относится филиал |
| name | text | Название филиала |
| city\_id | integer | Внешний ключ, определяющий город, в котором находится филиал |
| address | text | Адрес филиала |
| phone | varchar(13) | Телефон филиала |
| workers\_count | integer | Количество работников филиала |

Таблица 4.3 – Структура таблицы «clients»

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| id | integer | Первичный ключ |
| name | text | ФИО клиента |
| date | date | Дата рождения клиента |
| social\_status\_id | integer | Внешний ключ, определяющий социальное положение клиента |
| phone | varchar(13) | Телефон клиента |

Таблица 4.4 – Структура таблицы «contracts»

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| id | integer | Первичный ключ |
| branch\_id | integer | Внешний ключ, определяющий филиал, в котором был заключен договор |
| client\_id | integer | Внешний ключ, определяющий клиента, заключившего договор |
| insurance\_type\_id | integer | Внешний ключ, определяющий тип страхования |
| sum | integer | Сумма страхования |
| date | date | Дата заключения договора |

Таблица 4.5 – Структура таблицы «cities»

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| id | integer | Первичный ключ |
| city | text | Город |

Таблица 4.6 – Структура таблицы «company\_types»

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| id | integer | Первичный ключ |
| type | text | Тип компании |

Таблица 4.7 – Структура таблицы «social\_statuses»

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| id | integer | Первичный ключ |
| status | text | Социальное положение |

Таблица 4.8 – Структура таблицы «insurance\_types»

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| id | integer | Первичный ключ |
| type | text | Тип страхования |

Таблица 4.9 – Структура таблицы «users»

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| login | text | Логин пользователя |
| id | integer | Ключ, определяющий место работы пользователя |
| role | text | Роль пользователя |

Приведенная структура таблиц, позволяет удобно работать с таблицами посредством запросов и получать всю необходимую информацию для пользователя без необходимости построение очень сложных запросов. Концептуальная модель БД изображена на рисунке 4.1.

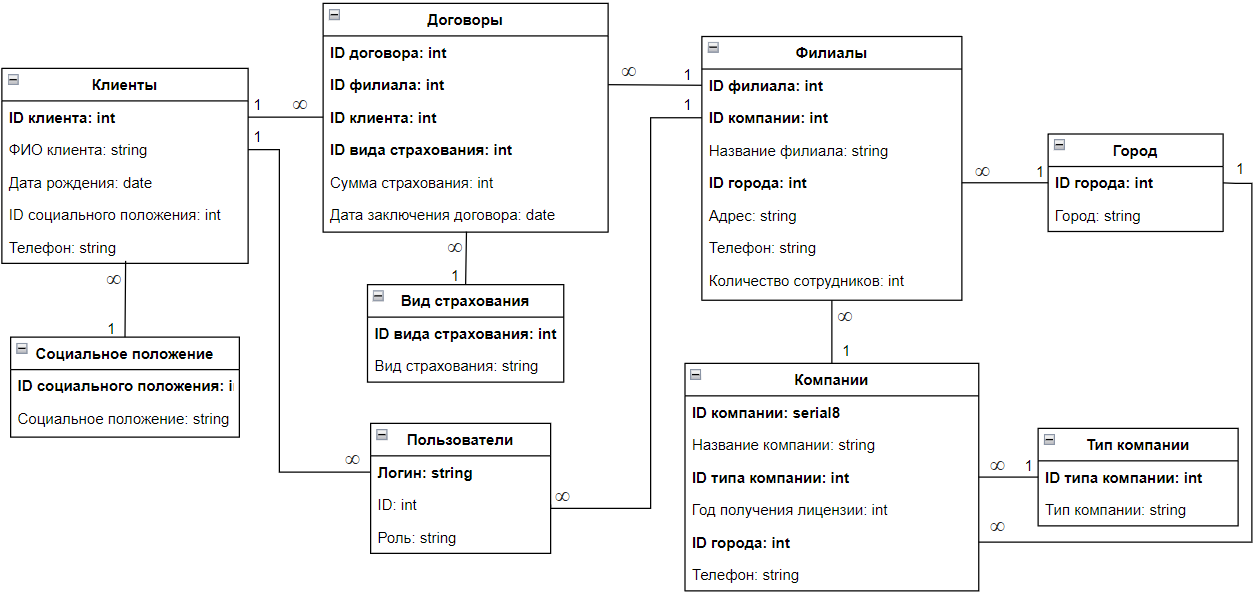


Рисунок 4.1 – Концептуальная модель БД

## 

## 4.2 Создание таблиц, доменов, индексов, последовательностей

SQL-запрос для создания таблицы «Компании» изображён на рисунке 4.2.

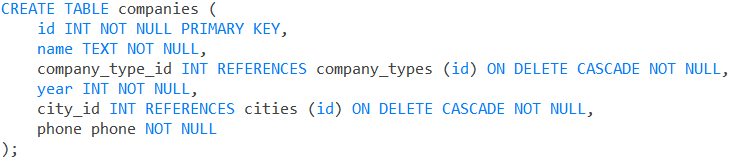


Рисунок 4.2 – SQL-запрос создания таблицы «Компании»

SQL-запрос для создания таблицы «Филиалы» изображён на рисунке 4.3.



Рисунок 4.3 – SQL-запрос создания таблицы «Филиалы»

SQL-запрос для создания таблицы «Клиенты» изображён на рисунке 4.4.

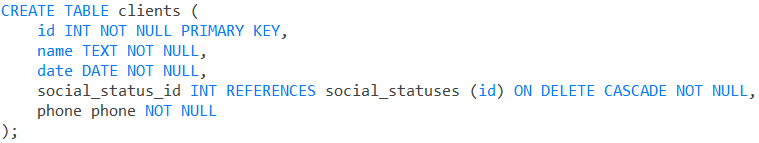


Рисунок 4.4 – SQL-запрос создания таблицы «Клиенты»

SQL-запрос для создания таблицы «Договоры» изображён на рисунке 4.5.

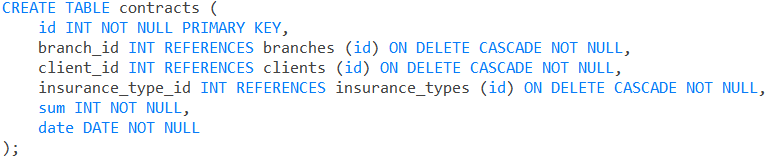


Рисунок 4.5 – SQL-запрос создания таблицы «Договоры»

SQL-запрос для создания таблицы «Города» изображён на рисунке 4.6.

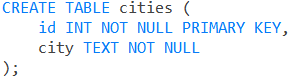


Рисунок 4.6 – SQL-запрос создания таблицы «Города»

SQL-запрос для создания таблицы «Типы компаний» изображён на рисунке 4.7.

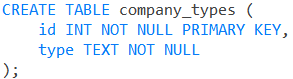


Рисунок 4.7 – SQL-запрос создания таблицы «Типы компаний»

SQL-запрос для создания таблицы «Социальные положения» изображён на рисунке 4.8.

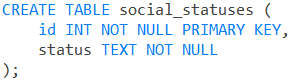


Рисунок 4.8 – SQL-запрос создания таблицы «Социальные положения»

SQL-запрос для создания таблицы «Типы страхования» изображён на рисунке 4.9.

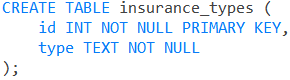


Рисунок 4.9 – SQL-запрос создания таблицы «Типы страхования»

SQL-запрос для создания таблицы «Пользоватлеи» изображён на рисунке 4.10.

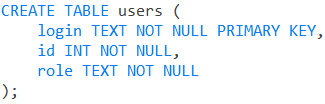


Рисунок 4.10 – SQL-запрос создания таблицы «Пользователи»

В созданных таблицах использован домен phone, изображённый на рисунке 4.11. Домен в реляционной модели данных – тип данных, то есть множество допустимых значений [2].

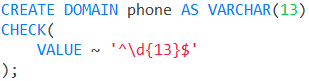


Рисунок 4.11 – Домен “Номер телефона” (13 цифр)

Также для таблиц были созданы индексы (рис. 4.12). Индекс - это структура данных значения определенного столбца в хранимой таблице. Индекс создается по столбцу таблицы. Поэтому следует помнить, что индекс содержит значения столбцов в таблице, и эти значения хранятся в структуре данных [3].

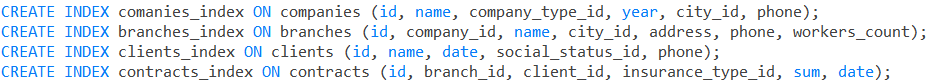


Рисунок 4.12 – SQL-запросы для создания индексов

Согласно техническому заданию разработаны последовательности (рис. 4.13) для таблиц, которые с помощью триггеров задают значения первичного ключа для вновь добавляемой записи.

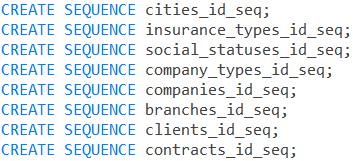


Рисунок 4.13 – SQL-запросы для создания последовательностей

## 4.3 Разработка триггеров

Для сохранения целостности данных и проверки внесенных пользователем изменений использовались триггеры – хранимые процедуры особого типа, исполнение которых запускаются сервером автоматически при модификации данных в таблице. Для всех таблицы были добавлены триггеры типа BEFORE INSERT, которые используют созданные последовательности и генерируют идентификаторы новых записей. SQL-запрос создания одного из этих триггеров представлен на рисунке 4.14.

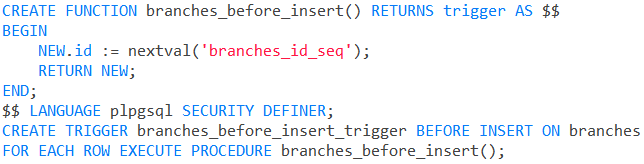


Рисунок 4.14 – SQL-запрос создания триггера BEFORE INSERT для таблицы «Филиалы»

Для таблицы пользователей был создан триггер типа AFTER UPDATE, который удаляет пользователя из старой группы и добавляет в новую (в случае если администратор поменял его роль). SQL-запрос этого триггера изображён на рисунке 4.15.

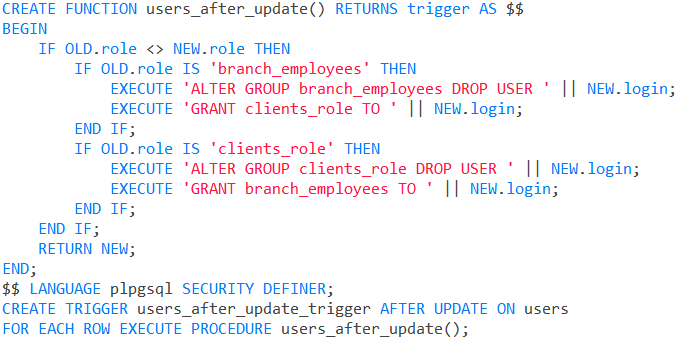


Рисунок 4.15 – Триггер типа AFTER UPDATE для таблицы “Пользователи”

Триггер типа AFTER INSERT для таблицы пользователей (рис. 4.16) создаёт роль для нового пользователя.

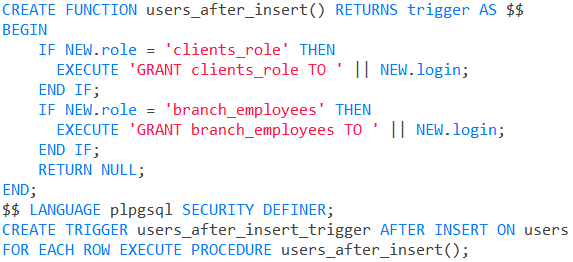


Рисунок 4.16 – Триггер типа AFTER INSERT для таблицы «Пользователи»

Триггер типа AFTER DELETE для таблицы пользователей (рис. 4.17) удаляет роль удаленного пользователя.

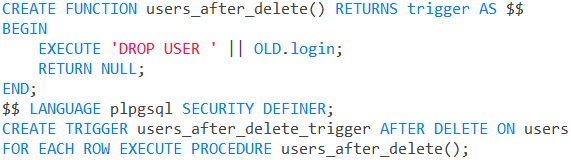


Рисунок 4.17 – Триггер типа AFTER DELETE для таблицы «Пользователи»

Триггер типа AFTER DELETE для таблицы «Филиалы» (рис. 4.18) удаляет всех пользователей филиала и их роли.

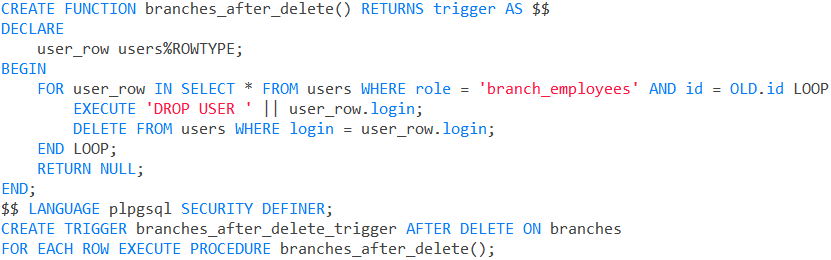


Рисунок 4.18 – Триггер типа AFTER DELETE для таблицы «Филиалы»

Триггер типа AFTER DELETE для таблицы «Клиенты» (рис. 4.19) удаляет всех клиентов филиала и их роли.

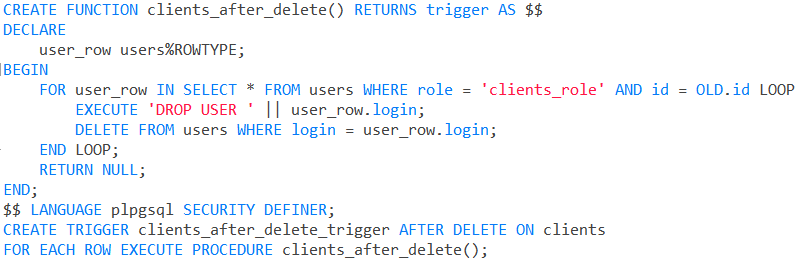


Рисунок 4.19 – Триггер типа AFTER DELETE для таблицы «Клиенты»

Триггер типа BEFORE INSERT для таблицы «Договоры» (рис. 4.20) выполняет проверку того, существует ли уже договор с заданными параметрами.

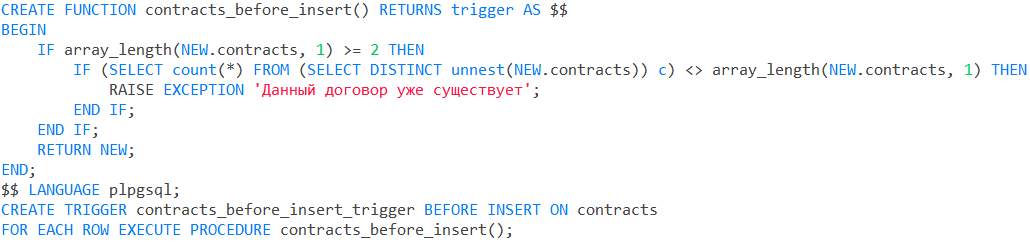


Рисунок 4.20 – Триггер типа BEFORE INSERT для таблицы «Договоры»

Триггер типа BEFORE UPDATE для таблицы «Договоры» (рис. 4.21) выполняет проверку того, существует ли уже договор с заданными параметрами.

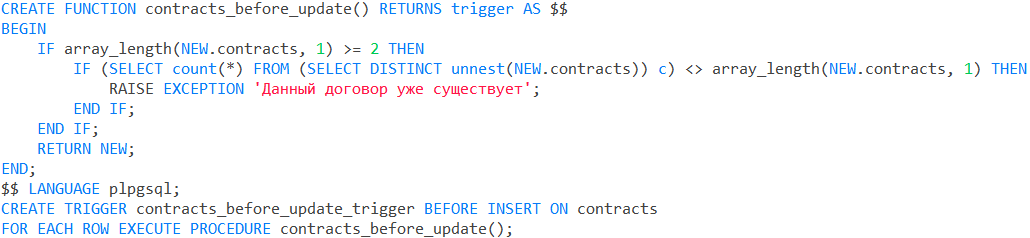


Рисунок 4.21 – Триггер типа BEFORE UPDATE для таблицы «Договоры»

## 

## 4.4 Организация многоролевого доступа к данным

Для информационной системы были созданы следующие роли: сотрудники филиалов, клиенты, администраторы и запросы. SQL-запросы для создания этих ролей изображены на рисунке 4.22.

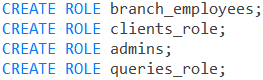


Рисунок 4.22 – SQL-запросы создания ролей пользователей

На рисунке 4.23 на примере роли «Сотрудники филиалов» продемонстрированы SQL-запросы создания прав доступа к таблицам.

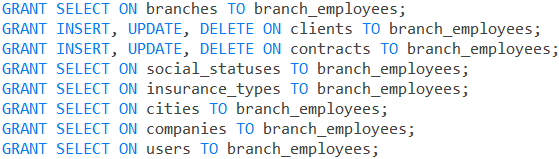


Рисунок 4.23 – SQL-запросы создания прав доступа к таблицам для сотрудников компаний

## 4.5 Разграничение доступа к данным на уровне строк

Чтобы разграничить данные на уровне строк для начала необходимо включить защиту на уровне строк для нужных таблиц. SQL-запросы изображены на рисунке 4.24.

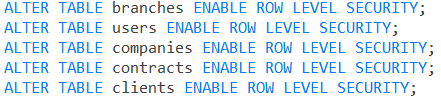


Рисунок 4.24 – SQL-запросы включения защиты на уровне строк для таблиц

Затем для каждой роли пользователей создаются политики защиты, где описано, к каким строкам таблицы роль имеет доступ. В выражении USING прописывается условие доступа к строке. На рисунке 4.25 на примере роли «Сотрудник филиала» продемонстрировано создание таких политик.

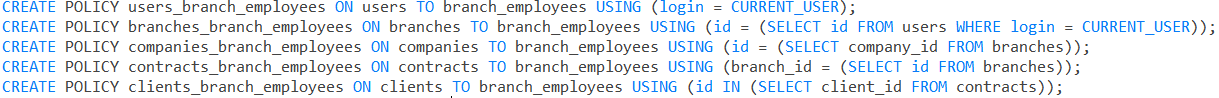


Рисунок 4.25 – SQL-запросы создания политик защиты для сотрудников филиала

## 4.6 Партицирование одной из основных таблиц БД

В соответствии с критериями технического задания было выполнено партицирование по диапазону таблицы «Договоры» (рис. 4.26). Партиционирование – это разбиение больших таблиц на логические части по выбранным критериям. Разбиение таблицы на разделы очень полезно, если таблица содержит большое количество данных. Разбиение ускорит выборку и запись в таблицу [4].

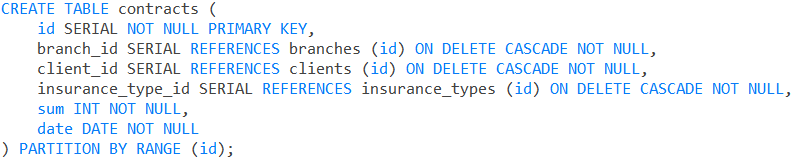


Рисунок 4.26 – SQL-запрос партицирования таблицы «Договоры» по диапазону

Для этой таблицы также были созданы секции (рис. 4.27), в том числе и секция default.

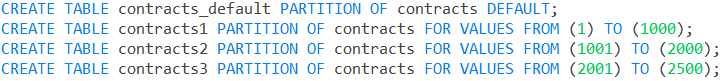


Рисунок 4.27 – SQL-запросы создания секций

В зависимости от идентификатора запись попадает в нужную секцию.

## 4.7 Проектирование запросов к базе данных

В ходе выполнения курсового проекта были разработаны различные запросы (в количестве 21 штуки), реализованные в виде представлений (запросы без параметров) и функций (запросы с параметрами):

– симметричное внутренне соединение с условием отбора по внешнему ключу: вывести филиалы в заданном городе (рис. 4.28-4.29);

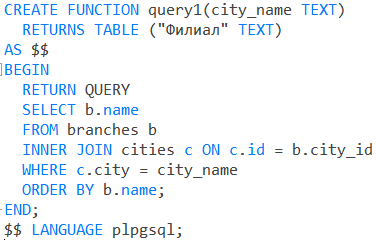


Рисунок 4.28 – Код запроса

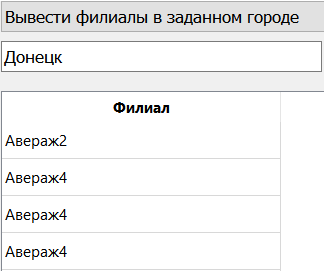


Рисунок 4.29 – Результат запроса

– симметричное внутренне соединение с условием отбора по внешнему ключу: вывести клиентов с заданным социальным положением (рис. 4.30-4.31);

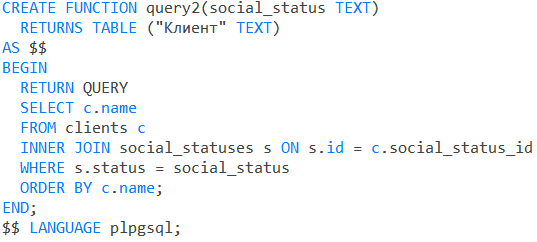


Рисунок 4.30 – Код запроса

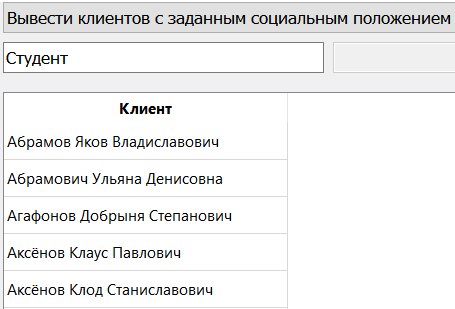


Рисунок 4.31 – Результат запроса

– симметричное внутренне соединение с условием отбора по дате: вывести клиентов, которые заключили договоры в заданную дату (рис. 4.32-4.33);

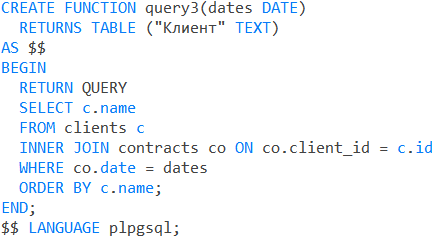


Рисунок 4.32 – Код запроса

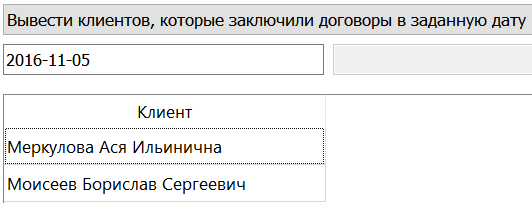


Рисунок 4.33 – Результат запроса

– симметричное внутренне соединение с условием отбора по дате: вывести филиалы, которые заключили договоры в заданную дату (рис. 4.34‑4.35);

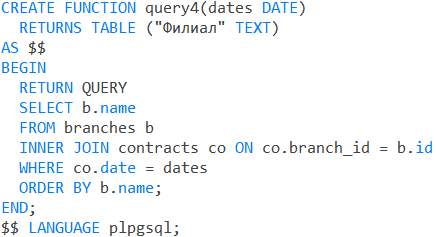


Рисунок 4.34 – Код запроса

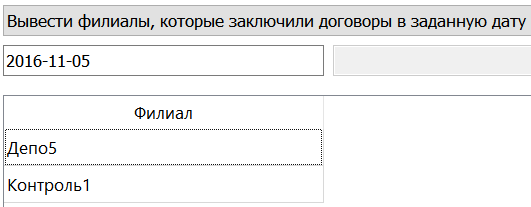


Рисунок 4.35 – Результат запроса

– симметричное внутренне соединение без условия: вывести компании и их филиалы (рис. 4.36‑4.37);

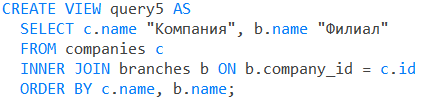


Рисунок 4.36 – Код запроса

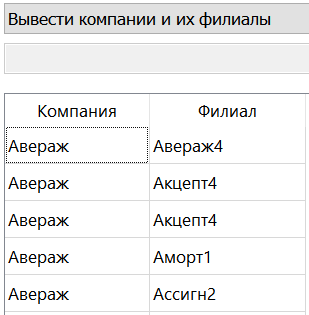


Рисунок 4.37 – Результат запроса

– симметричное внутренне соединение без условия: вывести компании и их типы (рис. 4.38‑4.39);

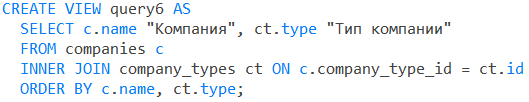


Рисунок 4.38 – Код запроса

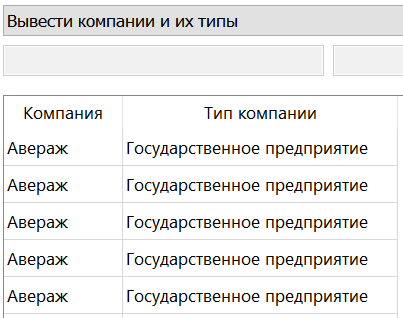


Рисунок 4.39 – Результат запроса

– симметричное внутренне соединение без условия: вывести клиентов и их договоры после увеличения суммы страхования в полтора раза (рис. 4.40‑4.41);

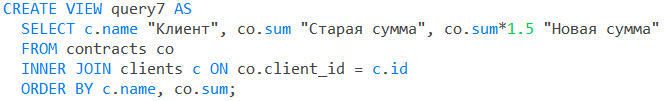


Рисунок 4.40 – Код запроса

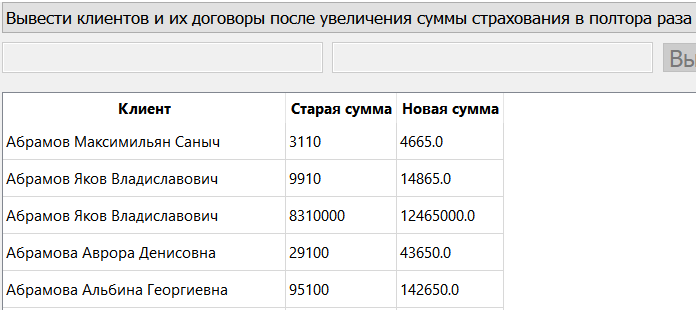


Рисунок 4.41 – Результат запроса

– левое внешнее соединение: вывести компании, у которых нет филиалов (рис. 4.42‑4.43);

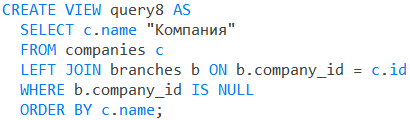


Рисунок 4.42 – Код запроса

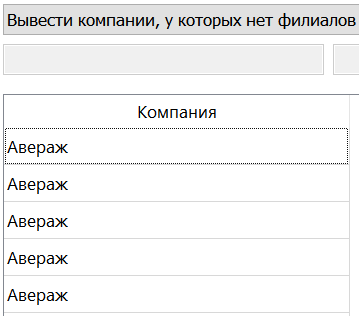


Рисунок 4.43 – Результат запроса

– правое внешнее соединение: вывести филиалы и их сотрудников (рис. 4.44‑4.45);

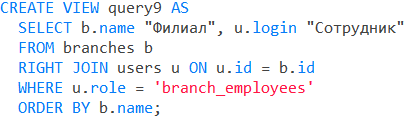


Рисунок 4.44 – Код запроса

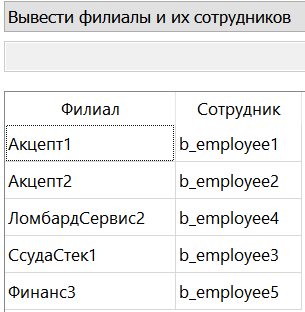


Рисунок 4.45 – Результат запроса

– запрос на запросе по принципу левого соединения: вывести компании, у которых нет филиалов, получившие лицензию в указанный период (рис. 4.46‑4.47);

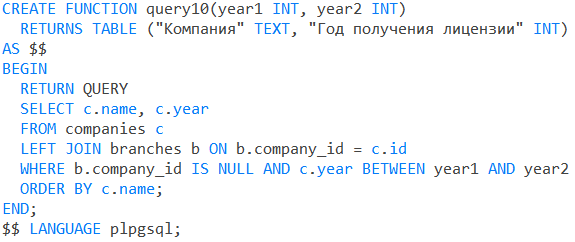


Рисунок 4.46 – Код запроса

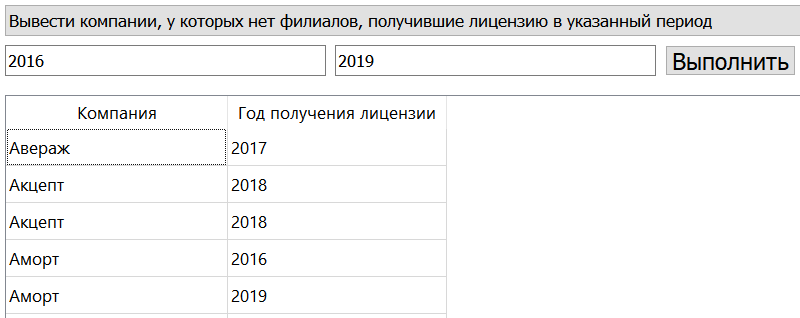


Рисунок 4.47 – Результат запроса

– итоговый запрос без условия: вывести количество компаний в каждом городе (рис. 4.48‑4.49);

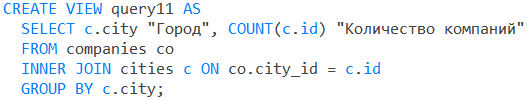


Рисунок 4.48 – Код запроса



Рисунок 4.49 – Результат запроса

– итоговый запрос без условия c итоговыми данными вида: «всего», «в том числе»: вывести количество компаний в заданной стране (рис. 4.50‑4.51);

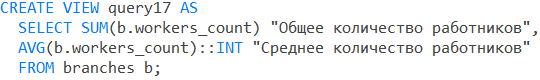


Рисунок 4.50 – Код запроса

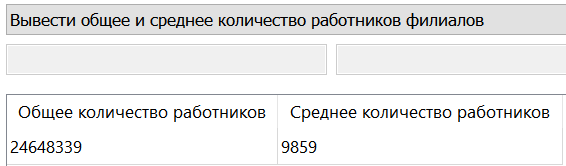


Рисунок 4.51 – Результат запроса

– итоговый запрос с условием на данные по значению: вывести количество компаний, получивших лицензию в заданном году, в каждом городе (рис. 4.52‑4.53);

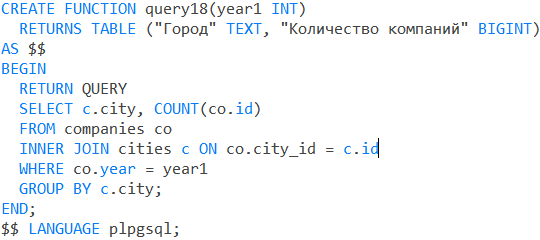


Рисунок 4.52 – Код запроса

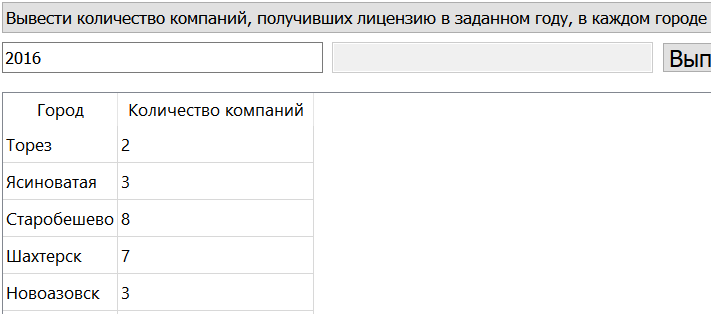


Рисунок 4.53 – Результат запроса

– итоговый запрос с условием на данные по маске: вывести количество клиентов, у которых в имени есть заданные буквы (рис. 4.54‑4.55);

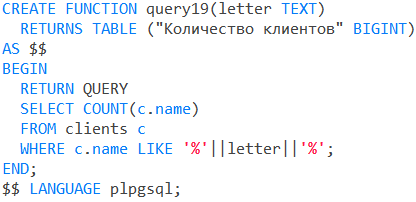


Рисунок 4.54 – Код запроса

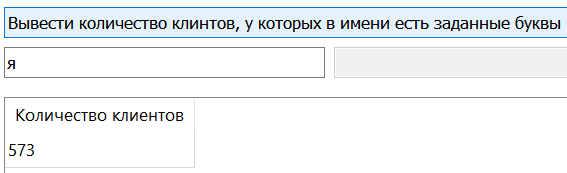


Рисунок 4.55 – Результат запроса

– итоговый запрос с условием на данные с использованием индекса: вывести количество компаний указанного типа (по индексу) (рис. 4.56‑4.57);

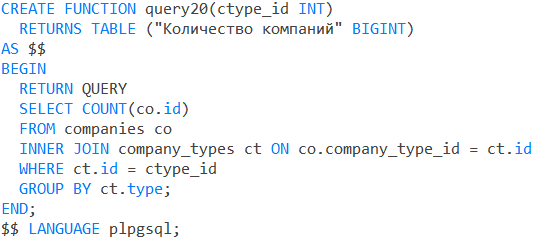


Рисунок 4.56 – Код запроса

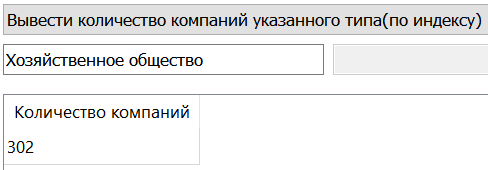


Рисунок 4.57 – Результат запроса

– итоговый запрос с условием на данные без использования индекса: вывести количество компаний указанного типа (рис. 4.58‑4.59);

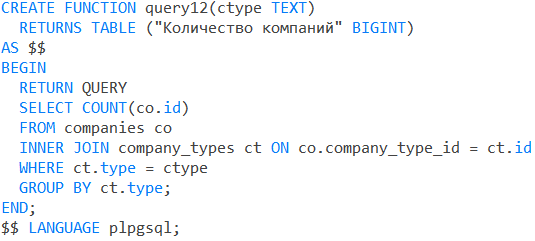


Рисунок 4.58 – Код запроса

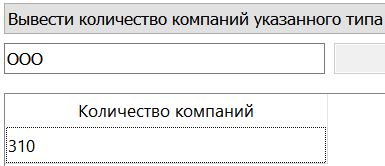


Рисунок 4.59 – Результат запроса

– итоговый запрос с условием на группы: вывести клиентов, сумма страхования всех контрактов которых больше заданной (рис. 4.60‑4.61);

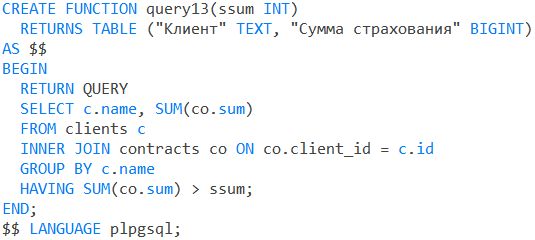


Рисунок 4.60 – Код запроса

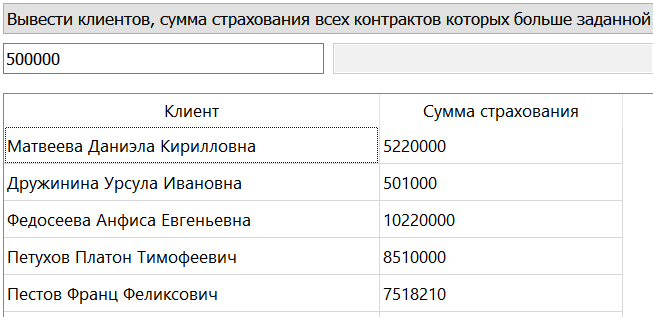


Рисунок 4.61 – Результат запроса

– итоговый запрос с условием на данные и на группы: вывести клиентов, сумма страхования всех контрактов которых больше заданной, рожденных в заданную дату (рис. 4.62‑4.63);

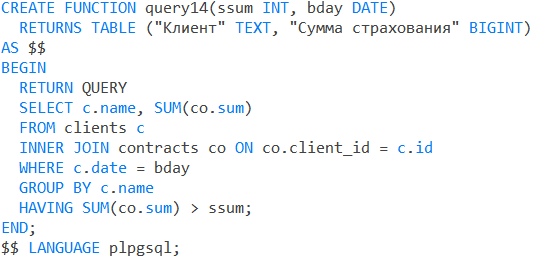


Рисунок 4.62 – Код запроса

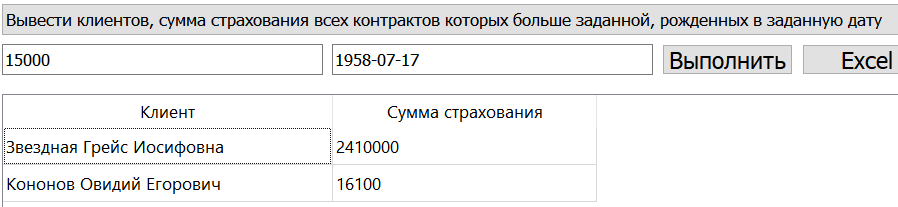


Рисунок 4.63 – Результат запроса

– запрос на запросе по принципу итогового запроса: вывести филиалы и сумму страхования всех контрактов, заключенных в период за указанные даты (рис. 4.64‑4.65);

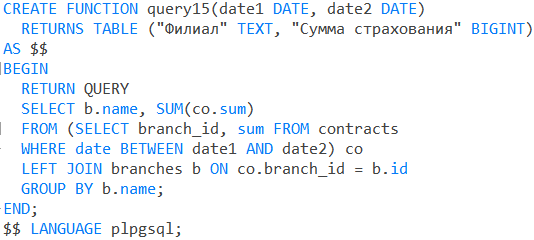


Рисунок 4.64 – Код запроса

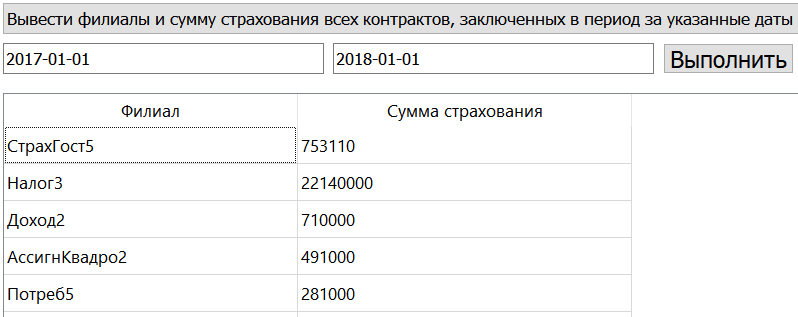


Рисунок 4.65 – Результат запроса

– запрос с использованием объединения: вывести названия компаний и филиалов (рис. 4.66‑4.67);

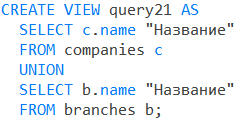


Рисунок 4.66 – Код запроса

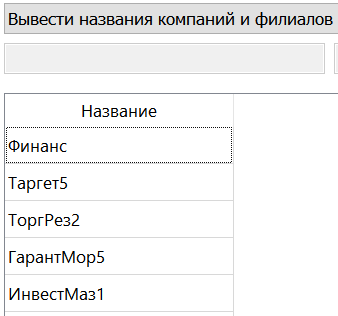


Рисунок 4.67 – Результат запроса

– запрос с подзапросом с использованием in: вывести клиентов с указанными социальными положениями (рис. 4.68‑4.69);

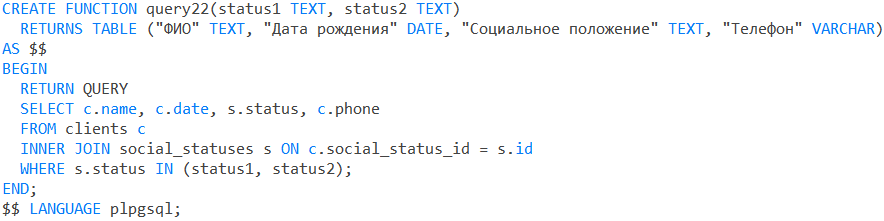


Рисунок 4.68 – Код запроса

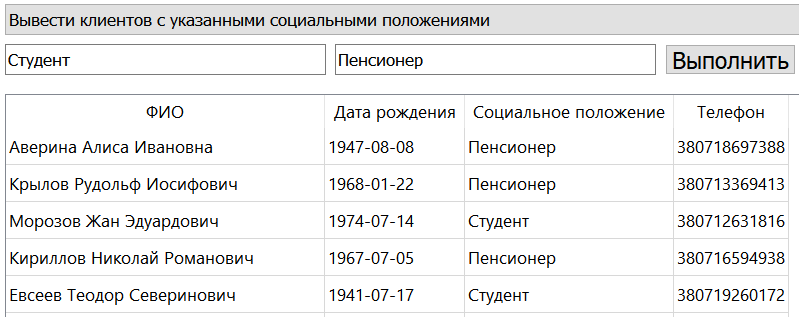


Рисунок 4.69 – Результат запроса

– запрос с подзапросом с использованием not in: вывести клиентов с социальными положениями кроме указанных (рис. 4.70‑4.71);

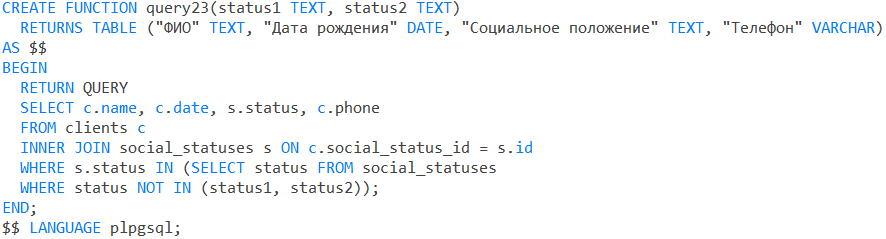


Рисунок 4.70 – Код запроса

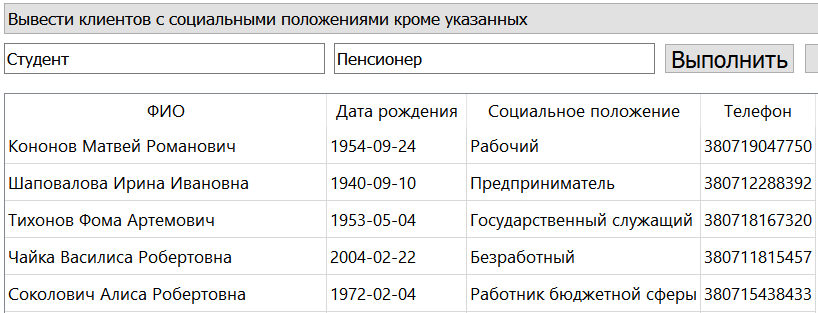


Рисунок 4.71 – Результат запроса

– запрос с подзапросом с использованием case: вывести контракты и ценовую категорию суммы их страхования (рис. 4.72‑4.73);

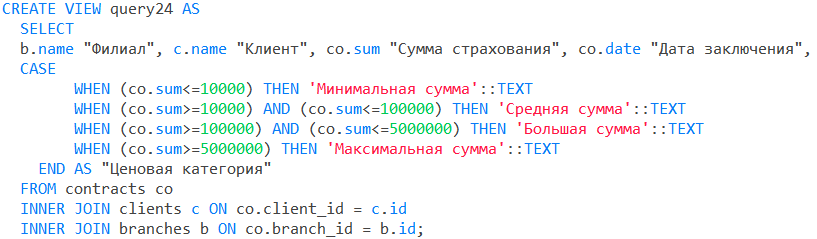


Рисунок 4.72 – Код запроса

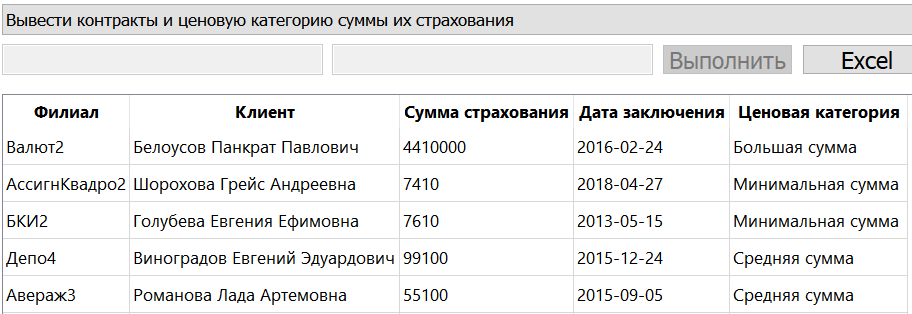


Рисунок 4.73 – Результат запроса

– запрос с подзапросом с операцией над итоговыми данными: вывести филиалы, сумма страхования контрактов которых больше средней суммы страхования контрактов всех филиалов (рис. 4.74‑4.75);

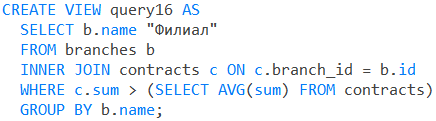


Рисунок 4.74 – Код запроса

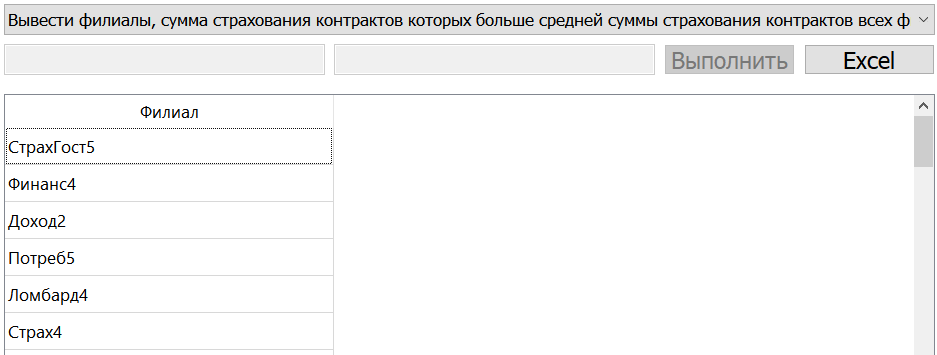


Рисунок 4.75 – Результат запроса

## 4.8 Разработка модифицируемого представления

Для курсового проекта было создано модифицируемое представление companies\_view (рис. 4.76). Понятие **модифицируемого представления** означает, что при изменении данных в самом представлении, эти данные изменятся и в таблицах, которые эти данные хранят. То есть при использовании оператора UPDATE/INSERT/DELETE к представлению, данные обновятся и в таблицах [5].

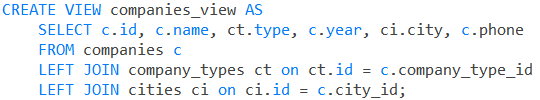


Рисунок 4.76 – SQL-запрос создания представления

Модифицируемость этого представления обеспечивают триггеры типа INSTEAD изображённые на рисунках 4.77-4.79.

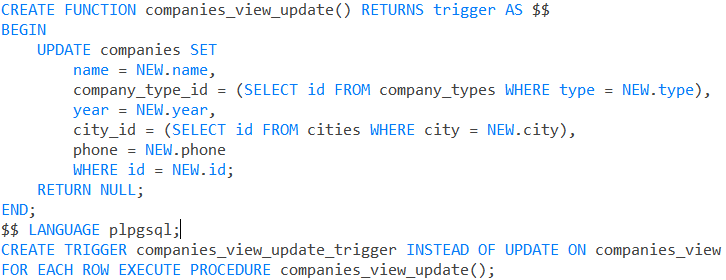


Рисунок 4.77 – Триггер типа INSTEAD OF UPDATE для модифицируемого представления

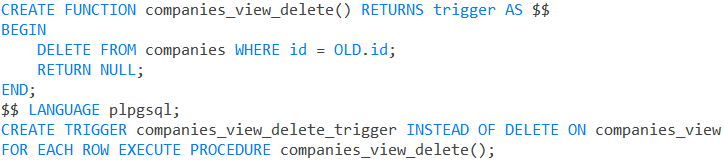


Рисунок 4.78 – Триггер типа INSTEAD OF DELETE для модифицируемого представления

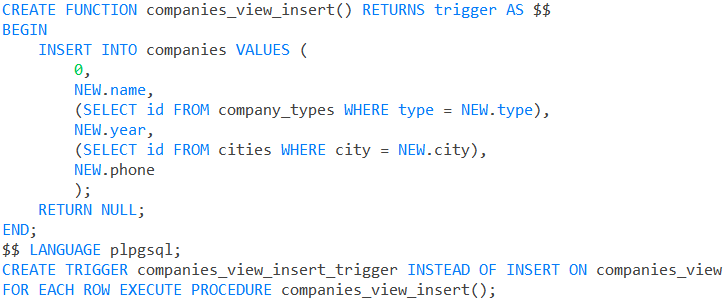


Рисунок 4.79 – Триггер типа INSTEAD OF INSERT для модифицируемого представления

Данные триггеры выполняют операции UPDATE, DELETE и INSERT над основной таблицей «Компании».

# 5 РАЗРАБОТКА ПРИЛОЖЕНИЯ

## 5.1 Формы и компоненты для работы роли «Сотрудник филиала»

Формы и компоненты для работы роли «Сотрудник филиала» изображены на рисунках 5.1-5.3.

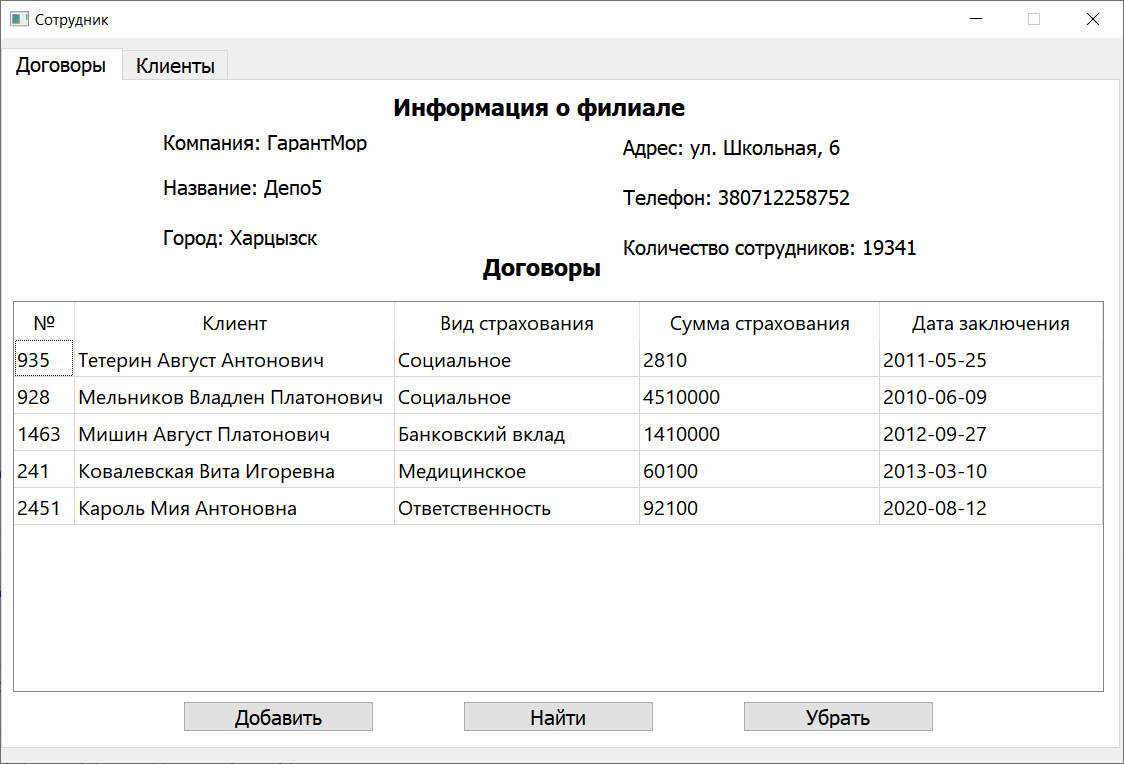


Рисунок 5.1 – Форма для работы роли «Сотрудник филиала» (вкладка «Договоры»)

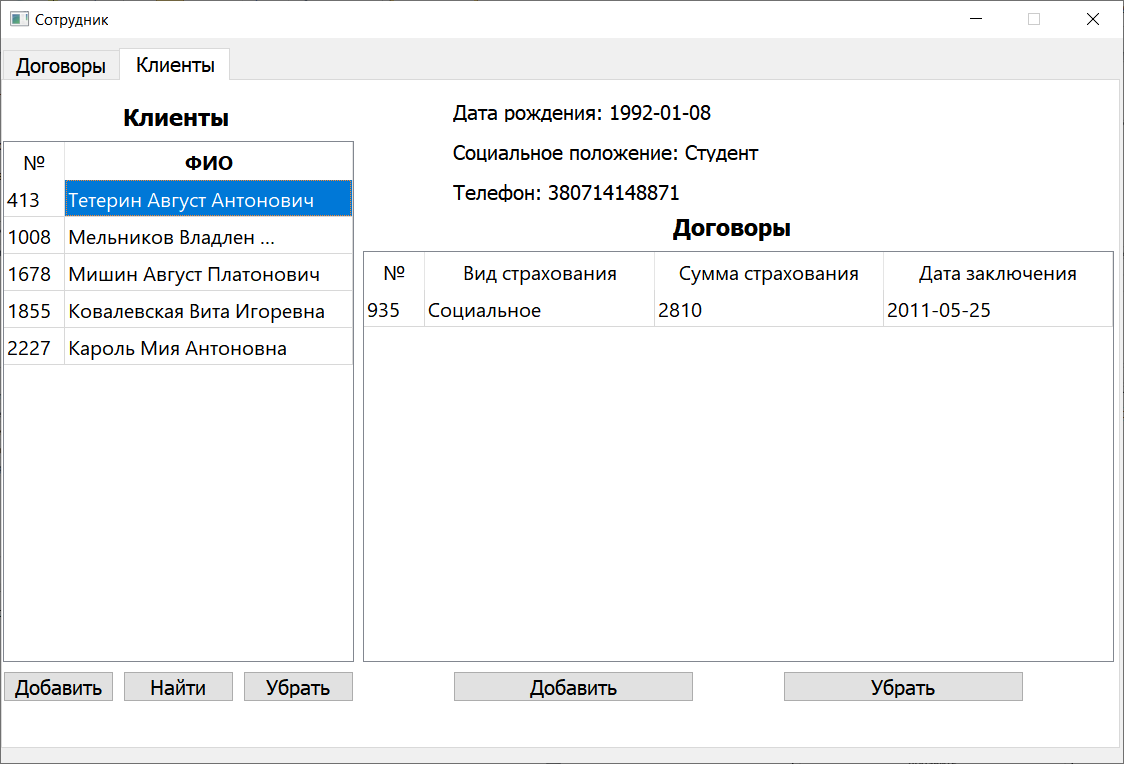


Рисунок 5.2 – Форма для работы роли «Сотрудник филиала» (вкладка «Клиенты»)

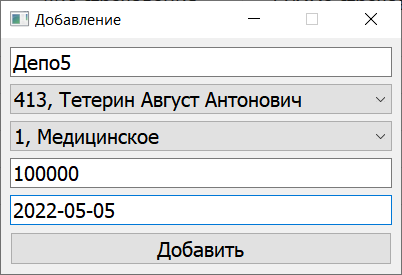


Рисунок 5.3 – Форма добавления договора

## 5.2 Формы и компоненты для работы роли «Клиент»

Формы и компоненты для работы роли «Клиент» изображены на рисунках 5.4-5.5.

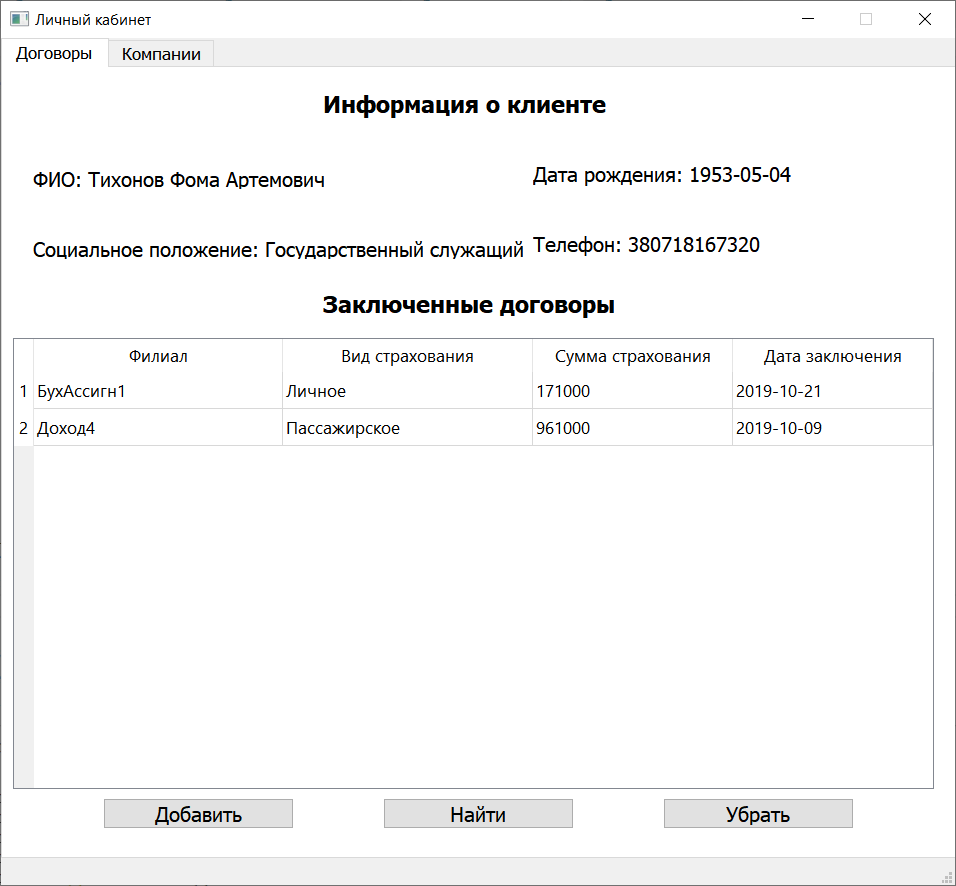


Рисунок 5.4 – Форма для работы роли «Клиент» (вкладка «Договоры»)

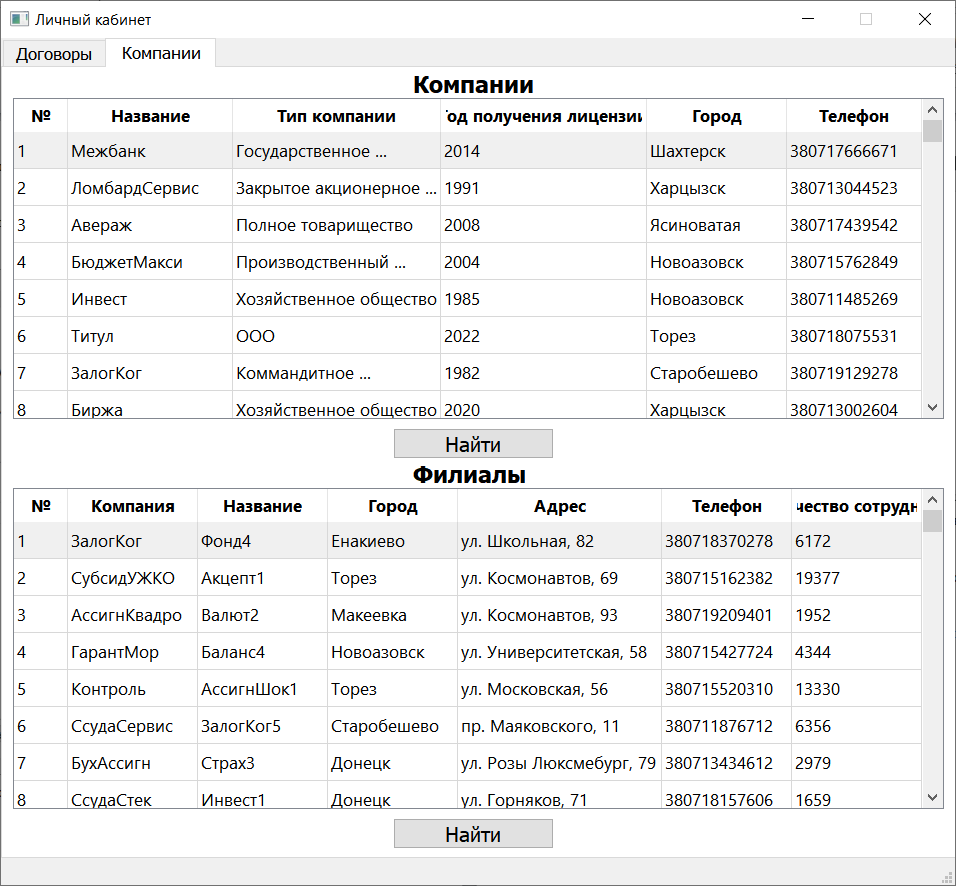


Рисунок 5.5 – Форма для работы роли «Клиент» (вкладка «Компании»)

## 

## 5.3 Формы и компоненты для работы роли «Администратор»

Формы и компоненты для работы роли «Администратор» изображены на рисунках 5.6-5.7.

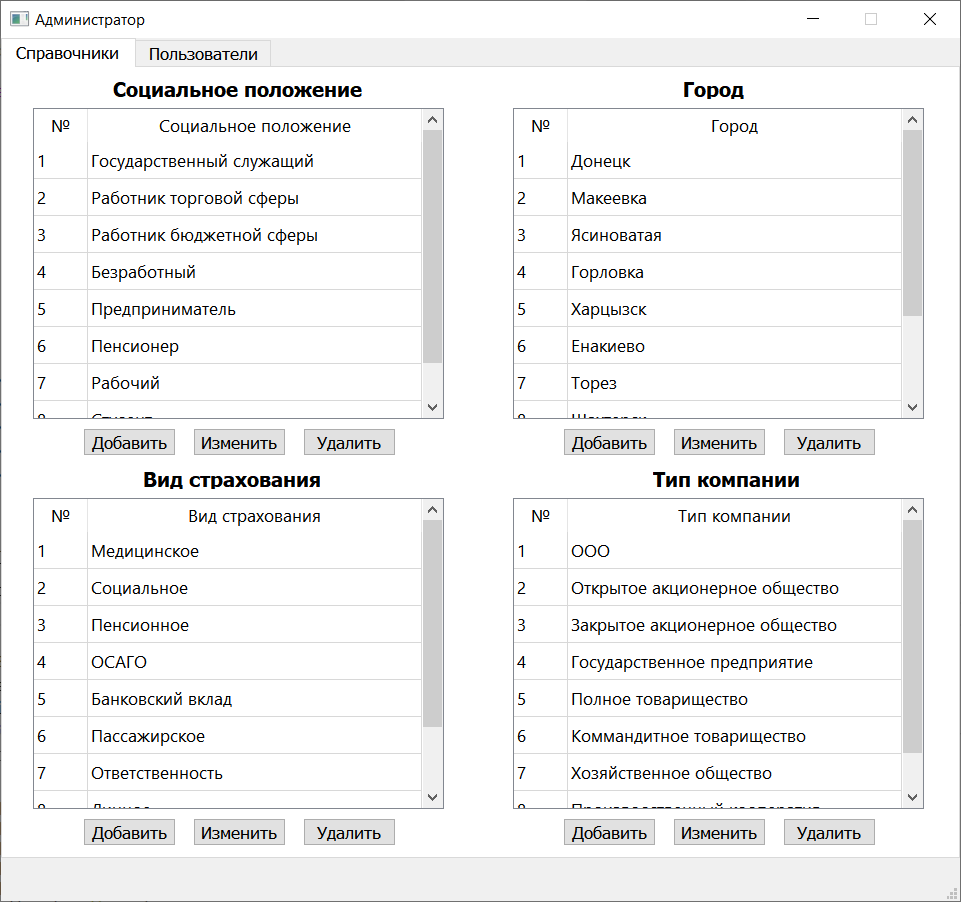


Рисунок 5.6 – Форма для работы роли «Администратор» (вкладка «Справочники»)

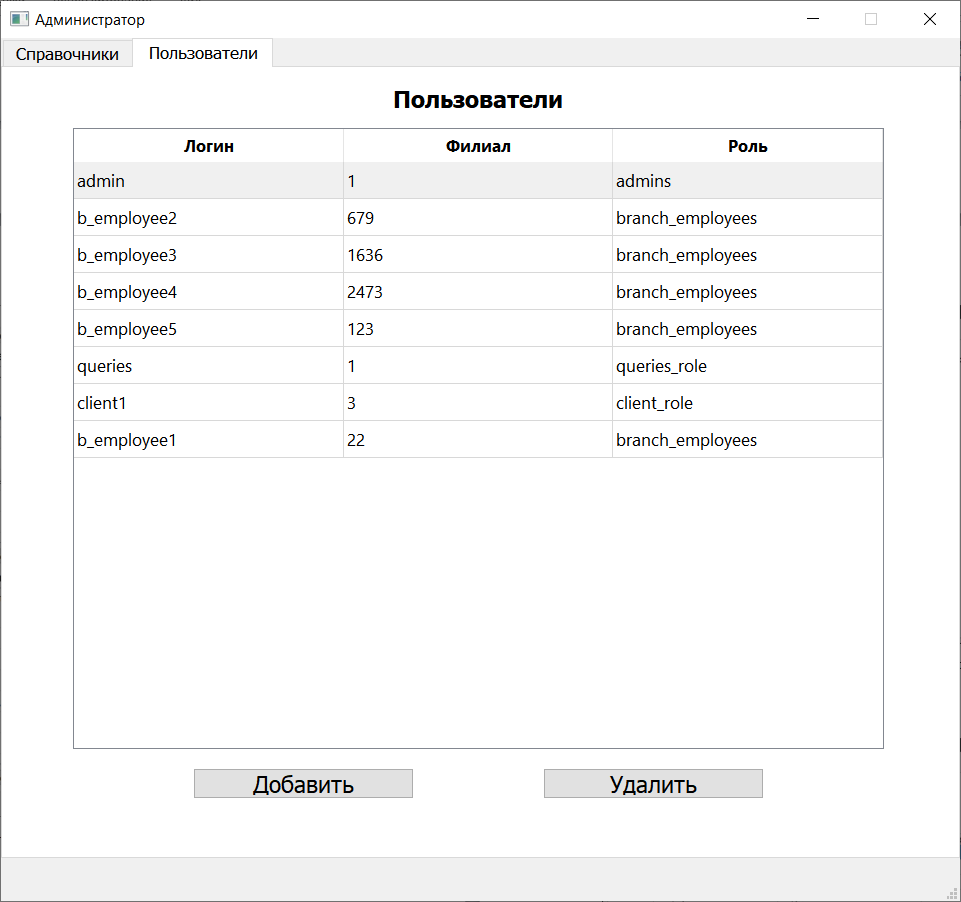


Рисунок 5.7 – Форма для работы роли «Администратор» (вкладка «Пользователи»)

## 5.4 Формы и компоненты для работы роли «Запросы»

Формы и компоненты для работы роли «Запросы» изображены на рисунке 5.8.

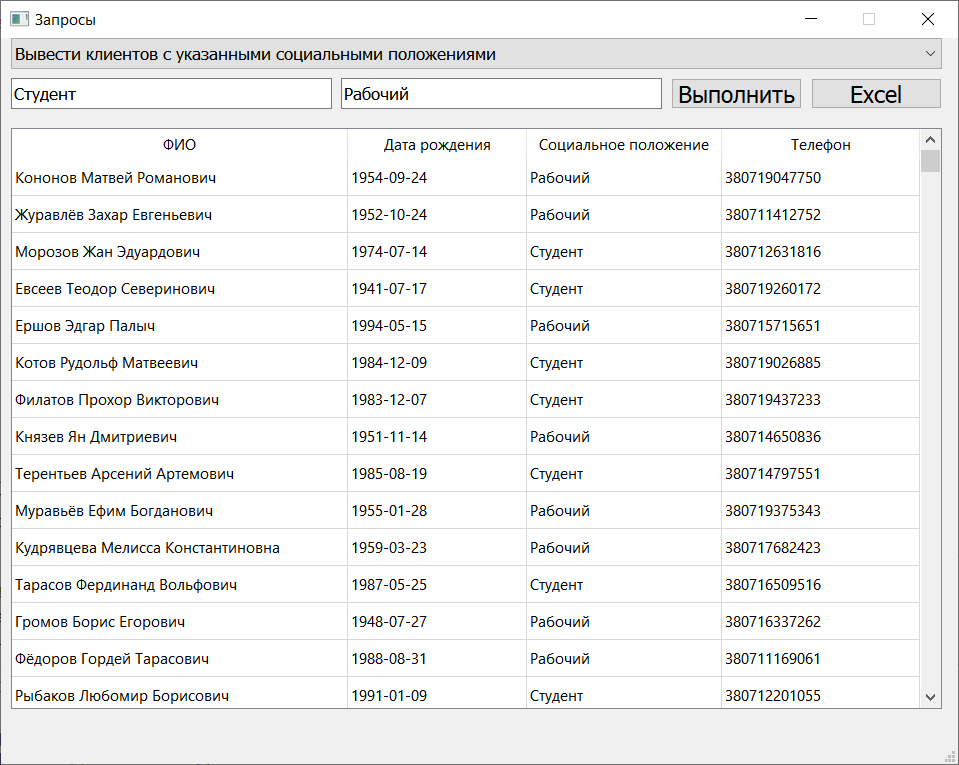


Рисунок 5.8 – Форма для работы роли «Запросы»

## 5.5 Экспортирование результата запроса в Excel

Для запроса «Вывести количество компаний в каждом городе» имеется возможность сгенерировать Excel диаграмму, изображённую на рисунке 5.9.

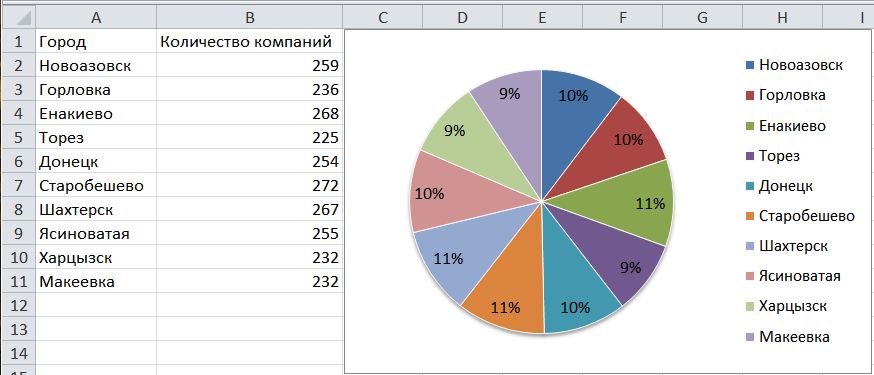


Рисунок 5.9 – Excel диаграмма

# 6 ТЕСТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ

В информационной системе предусмотрены обработки исключительных ситуаций (рис. 6.1-6.6), которые уведомляют пользователя о возникших проблемах, с помощью всплывающих окон.

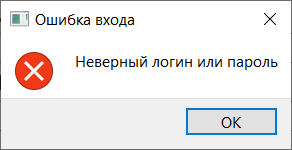


Рисунок 6.1 – Всплывающее окно при вводе неверного логина или пароля при входе в систему

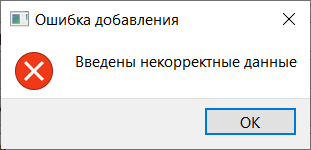


Рисунок 6.2 – Всплывающее окно при вводе пустых значений или некорректных данных

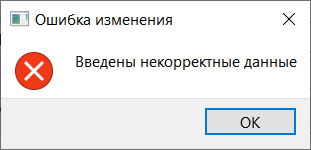


Рисунок 6.3 – Всплывающее окно при изменении на пустые значения или некорректные данные

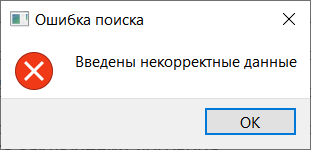


Рисунок 6.4 – Всплывающее окно при поиске пустых значений или некорректных данных

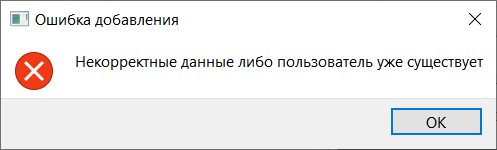


Рисунок 6.5 – Всплывающее окно при вводе администратором некорректных данных при создании пользователя (либо пользователь с введённым логином уже существует)

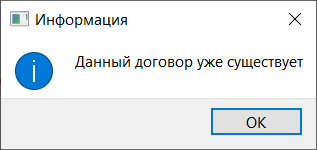


Рисунок 6.6 – Всплывающее окно, если пользователь ввел договор, который уже существует в системе

# ВЫВОДЫ

В ходе выполнения курсовой работы были закреплены практические навыки работы с системами управления базами данных, программирования информационных систем, методов автоматизации работы с базами данных.

В результате выполнения курсового проекта была изучена предметная область проекта, разработана концептуальная модель БД, проведено проектирование реляционной базы данных, создан проект информационной системы, созданы таблицы базы данных, сгенерированы случайные данные для заполнения таблиц, созданы представления, триггеры, домены, индексы, роли, выполнено партицирование таблицы, добавлена защита на уровне строк. Был спроектирован, разработан и протестирован графический пользовательский интерфейс, были составлены запросы в базу данных в соответствии с требованиями к курсовому проекту и произведен экспорт диаграммы в Excel.

В результате создания данной системы, требования, изложенные в постановке задачи, выполнены.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Что такое база данных // ORACLE [Электронный ресурс]. – Режим доступа: https://www.oracle.com/cis/database/what-is-database/

2. Домен (базы данных) // Wikipedia [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Домен\_(базы\_данных)

3. Что такое индекс базы данных и как он работает? // Русские блоги [Электронный ресурс]. – Режим доступа: https://russianblogs.com/article/7093315755/

4. Партицирование таблиц // EZcode [Электронный ресурс]. – Режим доступа: http://easy-code.ru/lesson/partitioning-mysql

5. Представления в SQL // CodeTown [Электронный ресурс]. – Режим доступа: https://codetown.ru/sql/predstavleniya/

# ПРИЛОЖЕНИЕ А ТЕХНИЧЕСКОЕ ЗАДАНИЕ

ГОУ ВПО «Донецкий национальный технический университет»

Факультет Интеллектуальных систем и программирования

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

Утверждаю

Зори С. А.

\_\_\_\_\_\_\_\_\_\_\_

08.02.2022 г.

**ТЕХНИЧЕСКОЕ ЗАДАНИЕ**

**на курсовую работу по дисциплине**

**«Программирование систем с серверами баз данных»**

выдано студентке группы ПИ-19а Сувориной Анне Владимировне

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

**Описание предметной области:** Система обработки информации о деятельности страховых компаний Украины использует сведения о компаниях (название компании, тип (частная, государственная, акционерная,…), год получения лицензии, город, в котором находится центральное отделение, телефон, филиалы (название филиала, город, адрес, телефон, количество сотрудников)) и заключенных с филиалами договорах (клиент (ФИО, дата рождения, социальное положение (служащий, предприниматель, учащийся, рабочий, пенсионер,…), телефон), вид страхования (недвижимость, имущество, от несчастного случая, к бракосочетанию, до совершеннолетия,…), сумма страхования, дата заключения договора).

Донецк – 2022

**Задание на курсовую работу**

1. Спроектировать концептуальную модель базы данных (БД) для заданной предметной области и представить ее 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 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).

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:

* многопользовательский режим работы (одна программа для всех ролей – ситуативный доступ к интерфейсу)
* наличие нескольких ролей пользователя (администратор – добавление/удаление/редактирование пользователей, их прав/ролей; пользователи\_1 – …, пользователи \_2 – ...)
* просмотр содержимого таблиц и представлений (здесь и далее – с учетом прав пользователей);
* 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, сделать защиту на уровне строк, выполнить партицирование одной из основных таблиц

**Рекомендуемое содержание пояснительной записки**

Титульный лист

Реферат

Содержание

Введение

1. Описание предметной области, постановка задачи

2. Обоснование выбора СУБД, описание возможностей СУБД

3. Обоснование выбора инструментальные средств для написания клиентской части, проектирование структуры ПО

3.1 Невизуальные компоненты для работы с данными

3.2 Визуальные компоненты отображения данных

3.3 Разработка шаблонов приложений для работы c таблицами базы данных

4. Проектирование базы данных в выбранной СУБД

4.1 Проектирование концептуальной модели БД

4.2 Создание таблиц, доменов, индексов, сиквенсов

4.3 Разработка триггеров

4.4 Организация многоролевого доступа к данным

4.5 Разграничение доступа к данным на уровне строк (в зависимости от роли и логина)

4.6 Партицирование одной из основных таблиц БД

4.7 Проектирование запросов к базе данных

4.8 Создание представлений и хранимых процедур, функций

5. Разработка клиентского приложения

5.1 Формы и компоненты для работы в «роли 1»

5.2 Формы и компоненты для работы в «роли 2»

5.3 …

5.4 Генерация результатов не менее трех итоговых запросов (диаграммы, экспорт в Excel)

6. Тестирование разработанной информационной системы (в т.ч. включая защиту от несанкционированного доступа, одновременную работы с данными, каскадное удаление)

Заключение/выводы и предложения

Список литературы

Приложение А. Техническое задание

Приложение Б. Листинг шаблонов

Приложение В. Листинг серверного приложения

Приложение Д. Листинг клиентского приложения

Приложение Е. Руководство пользователя

Приложение Ж. Руководство суперпользователя

Приложение З. Руководство администратора

**График выполнения курсовой работы**

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

Дата выдачи задания 08.02.2022

Студентка Суворина А.В.

Руководители проекта Щедрин С.В.

Ногтев Е. А.

Филипишин Д. А.

# ПРИЛОЖЕНИЕ Б ЛИСТИНГ ШАБЛОНОВ

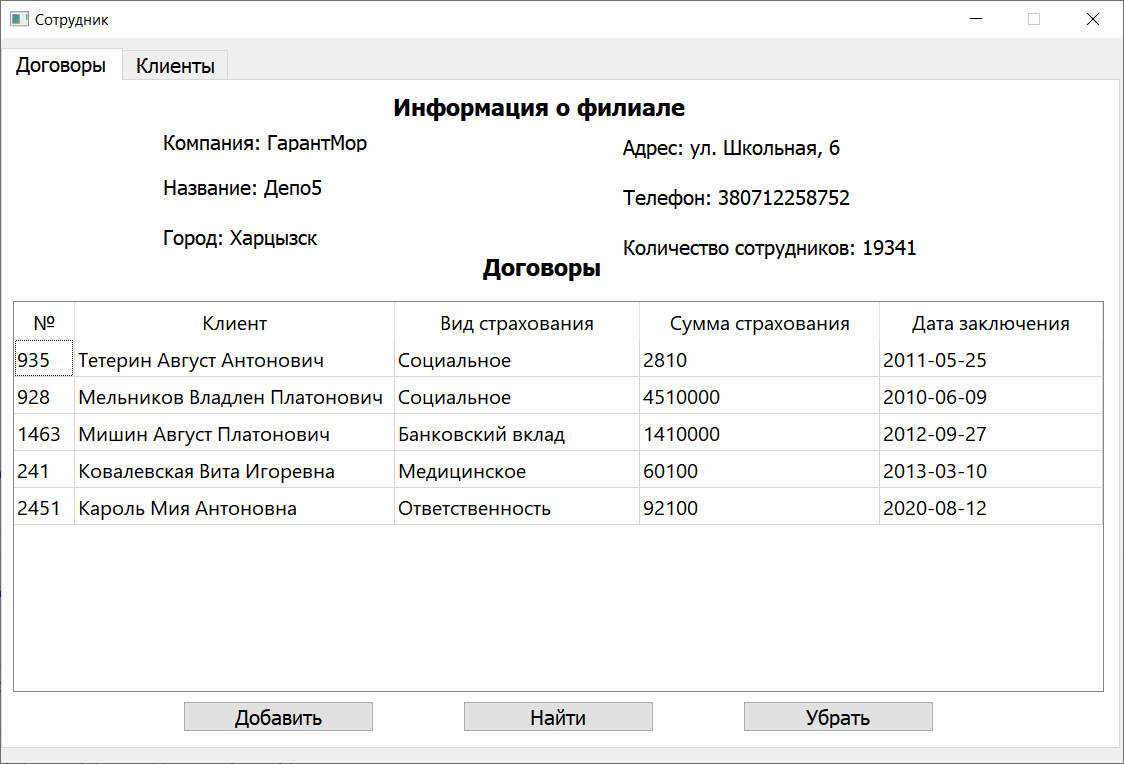


Рисунок Б.1 – Главная форма для работы роли «Сотрудник»

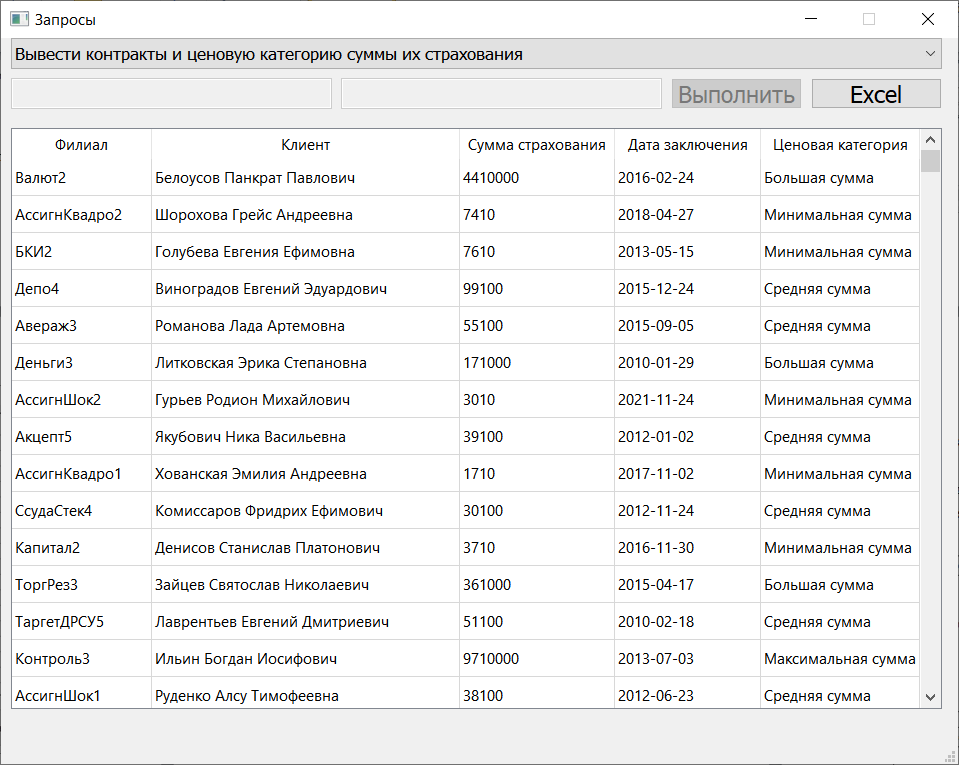


Рисунок Б.2 – Форма запросов

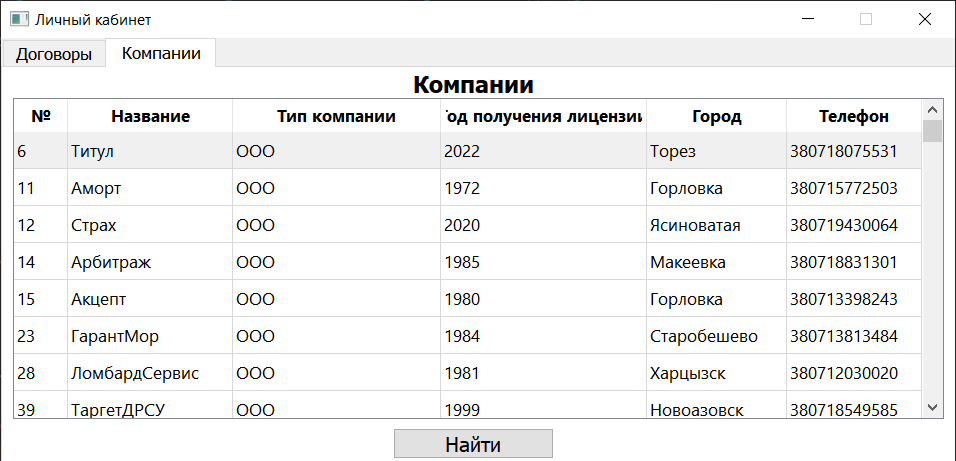


Рисунок Б.3 – Фильтрация данных

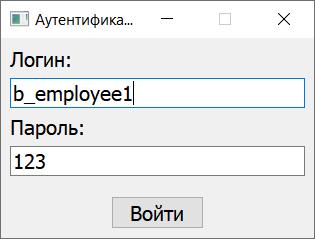


Рисунок Б.4 – Форма авторизации

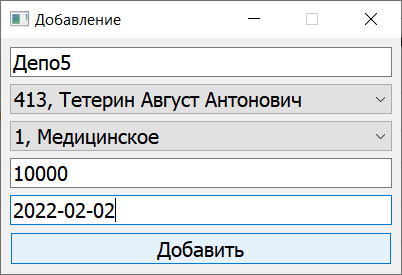


Рисунок Б.5 – Добавление данных

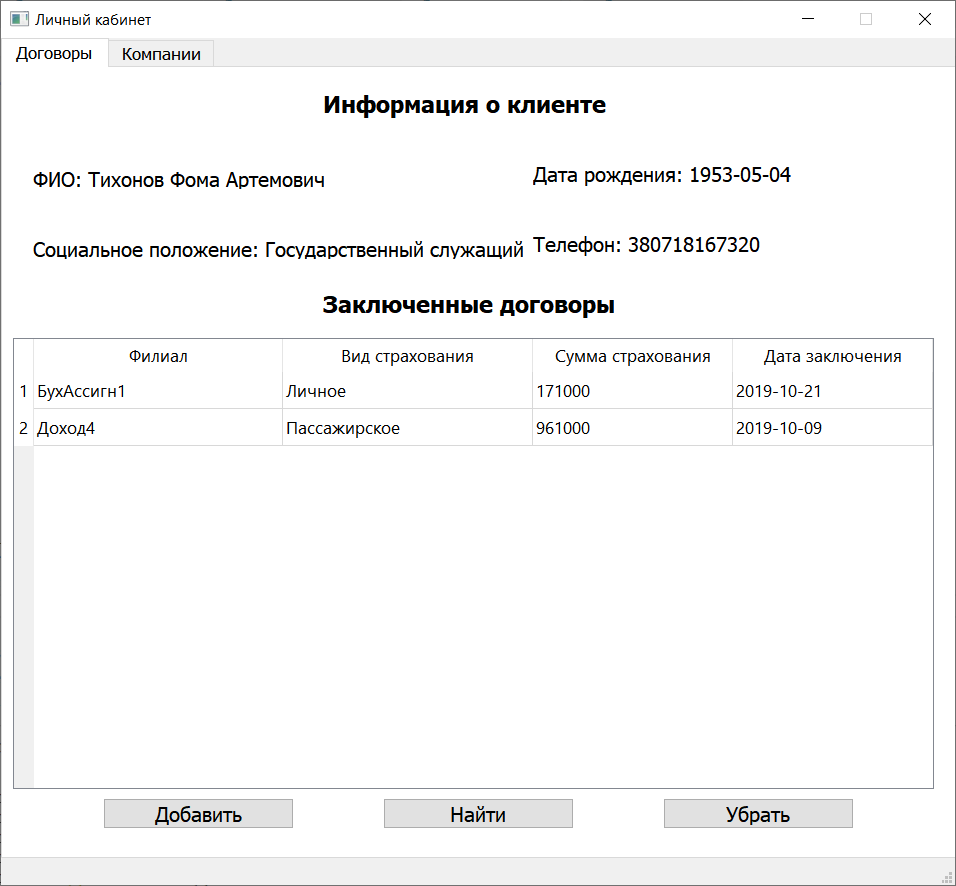


Рисунок Б.6 – Главная форма для работы роли «Клиент»

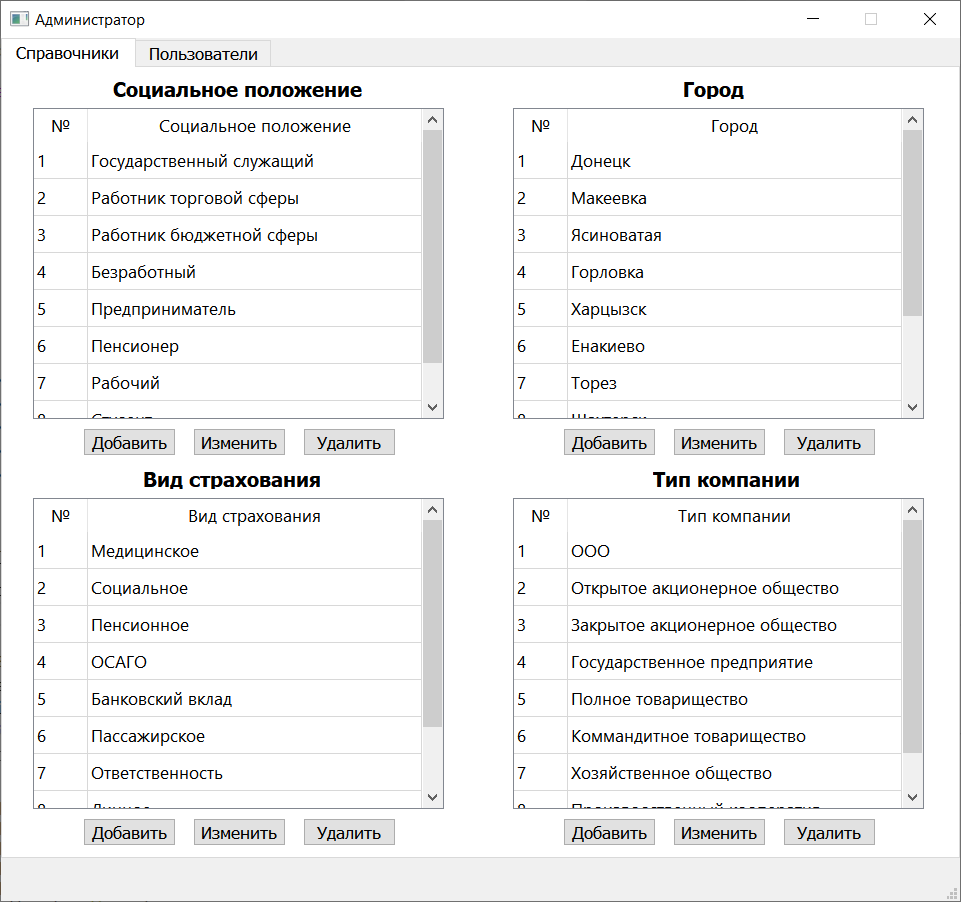


Рисунок Б.7 – Главная форма для работы роли «Администратор»

# ПРИЛОЖЕНИЕ В ЛИСТИНГ СЕРВЕРНОГО ПРИЛОЖЕНИЯ

CREATE TABLE social\_statuses (

    id SERIAL NOT NULL PRIMARY KEY,

    status TEXT NOT NULL

);

CREATE TABLE insurance\_types (

    id SERIAL NOT NULL PRIMARY KEY,

    type TEXT NOT NULL

);

CREATE TABLE cities (

    id SERIAL NOT NULL PRIMARY KEY,

    city TEXT NOT NULL

);

CREATE TABLE company\_types (

    id SERIAL NOT NULL PRIMARY KEY,

    type TEXT NOT NULL

);

CREATE TABLE clients (

    id SERIAL NOT NULL PRIMARY KEY,

    name TEXT NOT NULL,

    date DATE NOT NULL,

    social\_status\_id SERIAL REFERENCES social\_statuses (id) ON DELETE CASCADE NOT NULL,

    phone phone NOT NULL

);

CREATE TABLE companies (

    id SERIAL NOT NULL PRIMARY KEY,

    name TEXT NOT NULL,

    company\_type\_id SERIAL REFERENCES company\_types (id) ON DELETE CASCADE NOT NULL,

    year INT NOT NULL,

    city\_id SERIAL REFERENCES cities (id) ON DELETE CASCADE NOT NULL,

    phone phone NOT NULL

);

CREATE TABLE branches (

    id SERIAL NOT NULL PRIMARY KEY,

    company\_id SERIAL REFERENCES companies (id) ON DELETE CASCADE NOT NULL,

    name TEXT NOT NULL,

    city\_id SERIAL REFERENCES cities (id) ON DELETE CASCADE NOT NULL,

    address TEXT NOT NULL,

    phone phone NOT NULL,

    workers\_count INT NOT NULL

);

CREATE TABLE contracts (

    id SERIAL NOT NULL PRIMARY KEY,

    branch\_id SERIAL REFERENCES branches (id) ON DELETE CASCADE NOT NULL,

    client\_id SERIAL REFERENCES clients (id) ON DELETE CASCADE NOT NULL,

    insurance\_type\_id SERIAL REFERENCES insurance\_types (id) ON DELETE CASCADE NOT NULL,

    sum INT NOT NULL,

    date DATE NOT NULL

);

CREATE TABLE users (

    login TEXT NOT NULL PRIMARY KEY,

    id INT NOT NULL,

    role TEXT NOT NULL

);

ALTER TABLE branches ENABLE ROW LEVEL SECURITY;

ALTER TABLE users ENABLE ROW LEVEL SECURITY;

ALTER TABLE companies ENABLE ROW LEVEL SECURITY;

ALTER TABLE contracts ENABLE ROW LEVEL SECURITY;

ALTER TABLE clients ENABLE ROW LEVEL SECURITY;

CREATE ROLE branch\_employees;

CREATE ROLE clients\_role;

CREATE ROLE admins;

CREATE ROLE queries\_role;

CREATE USER b\_employee1 PASSWORD '123';

GRANT branch\_employees TO b\_employee1;

CREATE USER client1 PASSWORD '123';

GRANT clients\_role TO client1;

CREATE USER admin PASSWORD '123';

GRANT admins TO admin;

CREATE USER queries PASSWORD '123';

GRANT queries\_role TO queries;

GRANT SELECT ON branches TO branch\_employees;

GRANT INSERT, UPDATE, DELETE ON clients TO branch\_employees;

GRANT INSERT, UPDATE, DELETE ON contracts TO branch\_employees;

GRANT SELECT ON social\_statuses TO branch\_employees;

GRANT SELECT ON insurance\_types TO branch\_employees;

GRANT SELECT ON cities TO branch\_employees;

GRANT SELECT ON companies TO branch\_employees;

GRANT SELECT ON users TO branch\_employees;

GRANT SELECT, INSERT, UPDATE, DELETE ON branches TO admins;

GRANT SELECT, INSERT, UPDATE, DELETE ON clients TO admins;

GRANT SELECT, INSERT, UPDATE, DELETE ON contracts TO admins;

GRANT SELECT, INSERT, UPDATE, DELETE ON social\_statuses TO admins;

GRANT SELECT, INSERT, UPDATE, DELETE ON insurance\_types TO admins;

GRANT SELECT, INSERT, UPDATE, DELETE ON cities TO admins;

GRANT SELECT, INSERT, UPDATE, DELETE ON companies TO admins;

GRANT SELECT, INSERT, UPDATE, DELETE ON company\_types TO admins;

GRANT SELECT, INSERT, UPDATE, DELETE ON users TO admins;

GRANT SELECT ON branches TO clients\_role;

GRANT SELECT ON clients TO clients\_role;

GRANT SELECT ON contracts TO clients\_role;

GRANT SELECT ON social\_statuses TO clients\_role;

GRANT SELECT ON insurance\_types TO clients\_role;

GRANT SELECT ON users TO clients\_role;

GRANT SELECT ON companies TO clients\_role;

GRANT SELECT ON cities TO clients\_role;

GRANT SELECT ON company\_types TO clients\_role;

GRANT SELECT ON branches TO queries\_role;

GRANT SELECT ON clients TO queries\_role;

GRANT SELECT ON contracts TO queries\_role;

GRANT SELECT ON social\_statuses TO queries\_role;

GRANT SELECT ON insurance\_types TO queries\_role;

GRANT SELECT ON cities TO queries\_role;

GRANT SELECT ON companies TO queries\_role;

GRANT SELECT ON company\_types TO queries\_role;

GRANT SELECT ON users TO queries\_role;

CREATE POLICY users\_branch\_employees ON users TO branch\_employees USING (login = CURRENT\_USER);

CREATE POLICY branches\_branch\_employees ON branches TO branch\_employees USING (id = (SELECT id FROM users WHERE login = CURRENT\_USER));

CREATE POLICY companies\_branch\_employees ON companies TO branch\_employees USING (id = (SELECT company\_id FROM branches));

CREATE POLICY contracts\_branch\_employees ON contracts TO branch\_employees USING (branch\_id = (SELECT id FROM branches));

CREATE POLICY clients\_branch\_employees ON clients TO branch\_employees USING (id IN (SELECT client\_id FROM contracts));

CREATE POLICY users\_clients\_role ON users TO clients\_role USING (login = CURRENT\_USER);

CREATE POLICY contracts\_clients\_role ON contracts TO clients\_role USING (client\_id = (SELECT id FROM clients));

CREATE POLICY branches\_clients\_role ON branches TO clients\_role USING (true);

CREATE POLICY companies\_clients\_role ON companies TO clients\_role USING (true);

CREATE POLICY clients\_clients\_role ON clients TO clients\_role USING (id = (SELECT id FROM users WHERE login = CURRENT\_USER));

--DROP POLICY companies\_clients\_role ON companies;

CREATE POLICY users\_admins ON users TO admins USING (true);

CREATE POLICY branches\_admins ON branches TO admins USING (true);

CREATE POLICY contracts\_admins ON contracts TO admins USING (true);

CREATE POLICY clients\_admins ON clients TO admins USING (true);

CREATE POLICY companies\_admins ON companies TO admins USING (true);

CREATE POLICY users\_queries ON users TO queries\_role USING (login = CURRENT\_USER);

CREATE POLICY branches\_queries ON branches TO queries\_role USING (true);

CREATE POLICY contracts\_queries ON contracts TO queries\_role USING (true);

CREATE POLICY clients\_queries ON clients TO queries\_role USING (true);

CREATE POLICY companies\_queries ON companies TO queries\_role USING (true);

INSERT INTO users VALUES ('b\_employee1', 1, 'branch\_employees');

INSERT INTO users VALUES ('client1', 1, 'client\_role');

INSERT INTO users VALUES ('admin', 1, 'admins');

INSERT INTO users VALUES ('queries', 1, 'queries\_role');

CREATE FUNCTION users\_after\_insert() RETURNS trigger AS $$

BEGIN

    IF NEW.role = 'clients\_role' THEN

        EXECUTE 'GRANT clients\_role TO ' || NEW.login;

    END IF;

    IF NEW.role = 'branch\_employees' THEN

        EXECUTE 'GRANT branch\_employees TO ' || NEW.login;

    END IF;

    RETURN NULL;

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE TRIGGER users\_after\_insert\_trigger AFTER INSERT ON users

FOR EACH ROW EXECUTE PROCEDURE users\_after\_insert();

CREATE FUNCTION users\_after\_delete() RETURNS trigger AS $$

BEGIN

    EXECUTE 'DROP USER ' || OLD.login;

    RETURN NULL;

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE TRIGGER users\_after\_delete\_trigger AFTER DELETE ON users

FOR EACH ROW EXECUTE PROCEDURE users\_after\_delete();

CREATE FUNCTION users\_after\_update() RETURNS trigger AS $$

BEGIN

    IF OLD.role <> NEW.role THEN

        IF OLD.role IS 'branch\_employees' THEN

            EXECUTE 'ALTER GROUP branch\_employees DROP USER ' || NEW.login;

            EXECUTE 'GRANT clients\_role TO ' || NEW.login;

        END IF;

        IF OLD.role IS 'clients\_role' THEN

            EXECUTE 'ALTER GROUP clients\_role DROP USER ' || NEW.login;

            EXECUTE 'GRANT branch\_employees TO ' || NEW.login;

        END IF;

    END IF;

    RETURN NEW;

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE TRIGGER users\_after\_update\_trigger AFTER UPDATE ON users

FOR EACH ROW EXECUTE PROCEDURE users\_after\_update();

CREATE FUNCTION branches\_after\_delete() RETURNS trigger AS $$

DECLARE

    user\_row users%ROWTYPE;

BEGIN

    FOR user\_row IN SELECT \* FROM users WHERE role = 'branch\_employees' AND id = OLD.id LOOP

        EXECUTE 'DROP USER ' || user\_row.login;

        DELETE FROM users WHERE login = user\_row.login;

    END LOOP;

    RETURN NULL;

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE TRIGGER branches\_after\_delete\_trigger AFTER DELETE ON branches

FOR EACH ROW EXECUTE PROCEDURE branches\_after\_delete();

CREATE FUNCTION clients\_after\_delete() RETURNS trigger AS $$

DECLARE

    user\_row users%ROWTYPE;

BEGIN

    FOR user\_row IN SELECT \* FROM users WHERE role = 'clients\_role' AND id = OLD.id LOOP

        EXECUTE 'DROP USER ' || user\_row.login;

        DELETE FROM users WHERE login = user\_row.login;

    END LOOP;

    RETURN NULL;

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE TRIGGER clients\_after\_delete\_trigger AFTER DELETE ON clients

FOR EACH ROW EXECUTE PROCEDURE clients\_after\_delete();

CREATE FUNCTION contracts\_before\_insert() RETURNS trigger AS $$

BEGIN

    IF array\_length(NEW.contracts, 1) >= 2 THEN

        IF (SELECT count(\*) FROM (SELECT DISTINCT unnest(NEW.contracts)) c) <> array\_length(NEW.contracts, 1) THEN

            RAISE EXCEPTION 'Данный договор уже существует';

        END IF;

    END IF;

    RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER contracts\_before\_insert\_trigger BEFORE INSERT ON contracts

FOR EACH ROW EXECUTE PROCEDURE contracts\_before\_insert();

CREATE FUNCTION contracts\_before\_update() RETURNS trigger AS $$

BEGIN

    IF array\_length(NEW.contracts, 1) >= 2 THEN

        IF (SELECT count(\*) FROM (SELECT DISTINCT unnest(NEW.contracts)) c) <> array\_length(NEW.contracts, 1) THEN

            RAISE EXCEPTION 'Данный договор уже существует';

        END IF;

    END IF;

    RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER contracts\_before\_update\_trigger BEFORE INSERT ON contracts

FOR EACH ROW EXECUTE PROCEDURE contracts\_before\_update();

CREATE TABLE contracts (

    id SERIAL NOT NULL PRIMARY KEY,

    branch\_id SERIAL REFERENCES branches (id) ON DELETE CASCADE NOT NULL,

    client\_id SERIAL REFERENCES clients (id) ON DELETE CASCADE NOT NULL,

        insurance\_type\_id SERIAL REFERENCES insurance\_types (id) ON DELETE CASCADE NOT NULL,

        sum INT NOT NULL,

    date DATE NOT NULL

) PARTITION BY RANGE (id);

CREATE TABLE contracts\_default PARTITION OF contracts DEFAULT;

CREATE TABLE contracts1 PARTITION OF contracts FOR VALUES FROM (1) TO (1000);

CREATE TABLE contracts2 PARTITION OF contracts FOR VALUES FROM (1001) TO (2000);

CREATE TABLE contracts3 PARTITION OF contracts FOR VALUES FROM (2001) TO (2500);

GRANT USAGE, SELECT ON SEQUENCE companies\_id\_seq TO clients\_role;

GRANT SELECT ON query5 TO queries\_role;

GRANT SELECT ON query6 TO queries\_role;

GRANT SELECT ON query7 TO queries\_role;

GRANT SELECT ON query8 TO queries\_role;

GRANT SELECT ON query9 TO queries\_role;

GRANT SELECT ON query11 TO queries\_role;

GRANT SELECT ON query16 TO queries\_role;

GRANT SELECT ON query17 TO queries\_role;

GRANT SELECT ON query21 TO queries\_role;

GRANT SELECT ON query24 TO queries\_role;

GRANT SELECT ON companies\_view TO queries\_role;

CREATE DOMAIN phone AS VARCHAR(13)

CHECK(

        VALUE ~ '^\d{13}$'

);

CREATE INDEX comanies\_index ON companies (id, name, company\_type\_id, year, city\_id, phone);

CREATE INDEX branches\_index ON branches (id, company\_id, name, city\_id, address, phone, workers\_count);

CREATE INDEX clients\_index ON clients (id, name, date, social\_status\_id, phone);

CREATE INDEX contracts\_index ON contracts (id, branch\_id, client\_id, insurance\_type\_id, sum, date);

INSERT INTO users VALUES ('client2', 1, 'client\_role');

CREATE USER client2 PASSWORD '123';

DROP USER client2;

CREATE FUNCTION query1(city\_name TEXT)

    RETURNS TABLE ("Филиал" TEXT)

AS $$

BEGIN

    RETURN QUERY

    SELECT b.name

    FROM branches b

    INNER JOIN cities c ON c.id = b.city\_id

    WHERE c.city = city\_name

    ORDER BY b.name;

END;

$$ LANGUAGE plpgsql;

CREATE FUNCTION query2(social\_status TEXT)

    RETURNS TABLE ("Клиент" TEXT)

AS $$

BEGIN

    RETURN QUERY

    SELECT c.name

    FROM clients c

    INNER JOIN social\_statuses s ON s.id = c.social\_status\_id

    WHERE s.status = social\_status

    ORDER BY c.name;

END;

$$ LANGUAGE plpgsql;

CREATE FUNCTION query3(dates DATE)

    RETURNS TABLE ("Клиент" TEXT)

AS $$

BEGIN

    RETURN QUERY

    SELECT c.name

    FROM clients c

    INNER JOIN contracts co ON co.client\_id = c.id

    WHERE co.date = dates

    ORDER BY c.name;

END;

$$ LANGUAGE plpgsql;

CREATE FUNCTION query4(dates DATE)

    RETURNS TABLE ("Филиал" TEXT)

AS $$

BEGIN

    RETURN QUERY

    SELECT b.name

    FROM branches b

    INNER JOIN contracts co ON co.branch\_id = b.id

    WHERE co.date = dates

    ORDER BY b.name;

END;

$$ LANGUAGE plpgsql;

CREATE VIEW query5 AS

    SELECT c.name "Компания", b.name "Филиал"

    FROM companies c

    INNER JOIN branches b ON b.company\_id = c.id

    ORDER BY c.name, b.name;

CREATE VIEW query6 AS

    SELECT c.name "Компания", ct.type "Тип компании"

    FROM companies c

    INNER JOIN company\_types ct ON c.company\_type\_id = ct.id

    ORDER BY c.name, ct.type;

CREATE VIEW query7 AS

    SELECT c.name "Клиент", co.sum "Старая сумма", co.sum\*1.5 "Новая сумма"

    FROM contracts co

    INNER JOIN clients c ON co.client\_id = c.id

    ORDER BY c.name, co.sum;

CREATE VIEW query8 AS

    SELECT c.name "Компания"

    FROM companies c

    LEFT JOIN branches b ON b.company\_id = c.id

    WHERE b.company\_id IS NULL

    ORDER BY c.name;

CREATE VIEW query9 AS

    SELECT b.name "Филиал", u.login "Сотрудник"

    FROM branches b

    RIGHT JOIN users u ON u.id = b.id

    WHERE u.role = 'branch\_employees'

    ORDER BY b.name;

CREATE FUNCTION query10(year1 INT, year2 INT)

    RETURNS TABLE ("Компания" TEXT, "Год получения лицензии" INT)

AS $$

BEGIN

    RETURN QUERY

    SELECT c.name, c.year

    FROM companies c

    LEFT JOIN branches b ON b.company\_id = c.id

    WHERE b.company\_id IS NULL AND c.year BETWEEN year1 AND year2

    ORDER BY c.name;

END;

$$ LANGUAGE plpgsql;

CREATE VIEW query11 AS

    SELECT c.city "Город", COUNT(c.id) "Количество компаний"

    FROM companies co

    INNER JOIN cities c ON co.city\_id = c.id

    GROUP BY c.city;

CREATE FUNCTION query12(ctype TEXT)

    RETURNS TABLE ("Количество компаний" BIGINT)

AS $$

BEGIN

    RETURN QUERY

    SELECT COUNT(co.id)

    FROM companies co

    INNER JOIN company\_types ct ON co.company\_type\_id = ct.id

    WHERE ct.type = ctype

    GROUP BY ct.type;

END;

$$ LANGUAGE plpgsql;

CREATE FUNCTION query13(ssum INT)

    RETURNS TABLE ("Клиент" TEXT, "Сумма страхования" BIGINT)

AS $$

BEGIN

    RETURN QUERY

    SELECT c.name, SUM(co.sum)

    FROM clients c

    INNER JOIN contracts co ON co.client\_id = c.id

    GROUP BY c.name

    HAVING SUM(co.sum) > ssum;

END;

$$ LANGUAGE plpgsql;

CREATE FUNCTION query14(ssum INT, bday DATE)

    RETURNS TABLE ("Клиент" TEXT, "Сумма страхования" BIGINT)

AS $$

BEGIN

    RETURN QUERY

    SELECT c.name, SUM(co.sum)

    FROM clients c

    INNER JOIN contracts co ON co.client\_id = c.id

    WHERE c.date = bday

    GROUP BY c.name

    HAVING SUM(co.sum) > ssum;

END;

$$ LANGUAGE plpgsql;

CREATE FUNCTION query15(date1 DATE, date2 DATE)

    RETURNS TABLE ("Филиал" TEXT, "Сумма страхования" BIGINT)

AS $$

BEGIN

    RETURN QUERY

    SELECT b.name, SUM(co.sum)

    FROM (SELECT branch\_id, sum FROM contracts

    WHERE date BETWEEN date1 AND date2) co

    LEFT JOIN branches b ON co.branch\_id = b.id

    GROUP BY b.name;

END;

$$ LANGUAGE plpgsql;

CREATE VIEW query16 AS

    SELECT b.name "Филиал"

    FROM branches b

    INNER JOIN contracts c ON c.branch\_id = b.id

    WHERE c.sum > (SELECT AVG(sum) FROM contracts)

    GROUP BY b.name;

CREATE VIEW query17 AS

    SELECT SUM(b.workers\_count) "Общее количество работников",

    AVG(b.workers\_count)::INT "Среднее количество работников"

    FROM branches b;

CREATE FUNCTION query18(year1 INT)

    RETURNS TABLE ("Город" TEXT, "Количество компаний" BIGINT)

AS $$

BEGIN

    RETURN QUERY

    SELECT c.city, COUNT(co.id)

    FROM companies co

    INNER JOIN cities c ON co.city\_id = c.id

    WHERE co.year = year1

    GROUP BY c.city;

END;

$$ LANGUAGE plpgsql;

CREATE FUNCTION query19(letter TEXT)

    RETURNS TABLE ("Количество клиентов" BIGINT)

AS $$

BEGIN

    RETURN QUERY

    SELECT COUNT(c.name)

    FROM clients c

    WHERE c.name LIKE '%'||letter||'%';

END;

$$ LANGUAGE plpgsql;

CREATE FUNCTION query20(ctype\_id INT)

    RETURNS TABLE ("Количество компаний" BIGINT)

AS $$

BEGIN

    RETURN QUERY

    SELECT COUNT(co.id)

    FROM companies co

    INNER JOIN company\_types ct ON co.company\_type\_id = ct.id

    WHERE ct.id = ctype\_id

    GROUP BY ct.type;

END;

$$ LANGUAGE plpgsql;

CREATE VIEW query21 AS

    SELECT c.name "Название"

    FROM companies c

    UNION

    SELECT b.name "Название"

    FROM branches b;

CREATE FUNCTION query22(status1 TEXT, status2 TEXT)

    RETURNS TABLE ("ФИО" TEXT, "Дата рождения" DATE, "Социальное положение" TEXT, "Телефон" VARCHAR)

AS $$

BEGIN

    RETURN QUERY

    SELECT c.name, c.date, s.status, c.phone

    FROM clients c

    INNER JOIN social\_statuses s ON c.social\_status\_id = s.id

    WHERE s.status IN (status1, status2);

END;

$$ LANGUAGE plpgsql;

CREATE FUNCTION query23(status1 TEXT, status2 TEXT)

    RETURNS TABLE ("ФИО" TEXT, "Дата рождения" DATE, "Социальное положение" TEXT, "Телефон" VARCHAR)

AS $$

BEGIN

    RETURN QUERY

    SELECT c.name, c.date, s.status, c.phone

    FROM clients c

    INNER JOIN social\_statuses s ON c.social\_status\_id = s.id

    WHERE s.status IN (SELECT status FROM social\_statuses

    WHERE status NOT IN (status1, status2));

END;

$$ LANGUAGE plpgsql;

CREATE VIEW query24 AS

    SELECT

    b.name "Филиал", c.name "Клиент", co.sum "Сумма страхования", co.date "Дата заключения",

    CASE

        WHEN (co.sum<=10000) THEN 'Минимальная сумма'::TEXT

        WHEN (co.sum>=10000) AND (co.sum<=100000) THEN 'Средняя сумма'::TEXT

        WHEN (co.sum>=100000) AND (co.sum<=5000000) THEN 'Большая сумма'::TEXT

        WHEN (co.sum>=5000000) THEN 'Максимальная сумма'::TEXT

    END AS "Ценовая категория"

    FROM contracts co

    INNER JOIN clients c ON co.client\_id = c.id

    INNER JOIN branches b ON co.branch\_id = b.id;

CREATE VIEW companies\_view AS

        SELECT c.id, c.name, ct.type, c.year, ci.city, c.phone

        FROM companies c

        LEFT JOIN company\_types ct on ct.id = c.company\_type\_id

        LEFT JOIN cities ci on ci.id = c.city\_id;

CREATE FUNCTION companies\_view\_update() RETURNS trigger AS $$

BEGIN

        UPDATE companies SET

                name = NEW.name,

                company\_type\_id = (SELECT id FROM company\_types WHERE type = NEW.type),

                year = NEW.year,

                city\_id = (SELECT id FROM cities WHERE city = NEW.city),

                phone = NEW.phone

                WHERE id = NEW.id;

        RETURN NULL;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER companies\_view\_update\_trigger INSTEAD OF UPDATE ON companies\_view

FOR EACH ROW EXECUTE PROCEDURE companies\_view\_update();

CREATE FUNCTION companies\_view\_delete() RETURNS trigger AS $$

BEGIN

        DELETE FROM companies WHERE id = OLD.id;

        RETURN NULL;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER companies\_view\_delete\_trigger INSTEAD OF DELETE ON companies\_view

FOR EACH ROW EXECUTE PROCEDURE companies\_view\_delete();

CREATE FUNCTION companies\_view\_insert() RETURNS trigger AS $$

BEGIN

        INSERT INTO companies VALUES (

                0,

                NEW.name,

                (SELECT id FROM company\_types WHERE type = NEW.type),

                NEW.year,

                (SELECT id FROM cities WHERE city = NEW.city),

                NEW.phone

                );

        RETURN NULL;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER companies\_view\_insert\_trigger INSTEAD OF INSERT ON companies\_view

FOR EACH ROW EXECUTE PROCEDURE companies\_view\_insert();

# ПРИЛОЖЕНИЕ Г ЛИСТИНГ КЛИЕНТСКОГО ПРИЛОЖЕНИЯ

//admin\_window.py

from PyQt5 import QtWidgets, uic, QtGui

from PyQt5.QtWidgets import QTableWidget, QTableWidgetItem

from . import qt, cru\_window, cu\_users\_window

class AdminWindow(QtWidgets.QMainWindow):

    def \_\_init\_\_(self, app):

        super(AdminWindow, self).\_\_init\_\_()

        self.app = app

        uic.loadUi(self.app.ui\_path.joinpath('admin.ui'), self)

        self.socialstatusesTable.setColumnWidth(0, 5)

        self.socialstatusesTable.setColumnWidth(1, 334)

        self.citiesTable.setColumnWidth(0, 5)

        self.citiesTable.setColumnWidth(1, 334)

        self.insurancetypesTable.setColumnWidth(0, 5)

        self.insurancetypesTable.setColumnWidth(1, 334)

        self.companytypesTable.setColumnWidth(0, 5)

        self.companytypesTable.setColumnWidth(1, 334)

        self.usersTable.setColumnWidth(0, 270)

        self.usersTable.setColumnWidth(1, 269)

        self.usersTable.setColumnWidth(2, 270)

        self.adduserButton.clicked.connect(self.adduser)

        self.addsocialstatusButton.clicked.connect(self.addsocialstatus)

        self.addcityButton.clicked.connect(self.addcity)

        self.addinsurancetypeButton.clicked.connect(self.addinsurancetype)

        self.addcompanytypeButton.clicked.connect(self.addcompanytype)

        self.deleteuserButton.clicked.connect(self.deleteuser)

        self.deletesocialstatusButton.clicked.connect(self.deletesocialstatus)

        self.deletecityButton.clicked.connect(self.deletecity)

        self.deleteinsurancetypeButton.clicked.connect(self.deleteinsurancetype)

        self.deletecompanytypeButton.clicked.connect(self.deletecompanytype)

        self.editsocialstatusButton.clicked.connect(self.editsocialstatus)

        self.editcityButton.clicked.connect(self.editcity)

        self.editinsurancetypeButton.clicked.connect(self.editinsurancetype)

        self.editcompanytypeButton.clicked.connect(self.editcompanytype)

        self.usersTable.cellClicked.connect(self.usersTable\_cellClicked)

        self.socialstatusesTable.cellClicked.connect(self.socialstatusesTable\_cellClicked)

        self.citiesTable.cellClicked.connect(self.citiesTable\_cellClicked)

        self.insurancetypesTable.cellClicked.connect(self.insurancetypesTable\_cellClicked)

        self.companytypesTable.cellClicked.connect(self.companytypesTable\_cellClicked)

        self.centralwidget.setContentsMargins(9, 9, 9, 9)

        self.setFixedSize(self.size())

        self.show()

        self.tabWidget.setCurrentIndex(0)

        self.tabWidget.tabBarClicked.connect(self.tab\_changed)

    def tab\_changed(self, index):

        if index == 0:

            self.update\_socialstatuses()

            self.update\_cities()

            self.update\_insurancetypes()

            self.update\_companytypes()

        elif index == 1:

            self.update\_users()

    def update\_socialstatuses(self):

        self.socialstatusesTable.clearContents()

        self.socialstatusesTable.setRowCount(0)

        rows = self.app.pg.get\_socialstatuses()

        for names in rows:

            qt.add\_table\_row(self.socialstatusesTable, names)

    def update\_cities(self):

        self.citiesTable.clearContents()

        self.citiesTable.setRowCount(0)

        rows = self.app.pg.get\_cities()

        for names in rows:

            qt.add\_table\_row(self.citiesTable, names)

    def update\_insurancetypes(self):

        self.insurancetypesTable.clearContents()

        self.insurancetypesTable.setRowCount(0)

        rows = self.app.pg.get\_insurancetypes()

        for names in rows:

            qt.add\_table\_row(self.insurancetypesTable, names)

    def update\_companytypes(self):

        self.companytypesTable.clearContents()

        self.companytypesTable.setRowCount(0)

        rows = self.app.pg.get\_companytypes()

        for names in rows:

            qt.add\_table\_row(self.companytypesTable, names)

    def update\_users(self):

        self.usersTable.clearContents()

        self.usersTable.setRowCount(0)

        rows = self.app.pg.get\_users()

        for names in rows:

            qt.add\_table\_row(self.usersTable, names)

        if len(rows) > 0:

            self.usersTable.selectRow(0)

            self.usersTable\_cellClicked(0, 0)

    def adduser(self):

        self.users\_add\_window = cu\_users\_window.CUUsersWindow(

            self.app,

            self.update\_users

        )

        self.users\_add\_window.show()

    def addsocialstatus(self):

        self.socialstatuses\_add\_window = cru\_window.CRUWindow(

            self.app,

            self.update\_socialstatuses,

            'social\_statuseses',

            ('add',),

            (('Социальное положение', 'line'),)

        )

        self.socialstatuses\_add\_window.show()

    def addcity(self):

        self.cities\_add\_window = cru\_window.CRUWindow(

            self.app,

            self.update\_cities,

            'cities',

            ('add',),

            (('Город', 'line'),)

        )

        self.cities\_add\_window.show()

    def addinsurancetype(self):

        self.insurancetypes\_add\_window = cru\_window.CRUWindow(

            self.app,

            self.update\_insurancetypes,

            'insurance\_types',

            ('add',),

            (('Вид страхования', 'line'),)

        )

        self.insurancetypes\_add\_window.show()

    def addcompanytype(self):

        self.companytypes\_add\_window = cru\_window.CRUWindow(

            self.app,

            self.update\_companytypes,

            'company\_types',

            ('add',),

            (('Тип компании', 'line'),)

        )

        self.companytypes\_add\_window.show()

    def deleteuser(self):

        self.app.pg.drop\_user(self.usersTable.item(self.users\_table\_row, 0).text())

        self.update\_users()

    def deletesocialstatus(self):

        self.app.pg.delete\_by\_id(self.socialstatusesTable.item(self.socialstatuses\_table\_row, 0).text(), 'social\_statuses')

        self.update\_socialstatuses()

    def deletecity(self):

        self.app.pg.delete\_by\_id(self.citiesTable.item(self.cities\_table\_row, 0).text(), 'cities')

        self.update\_cities()

    def deleteinsurancetype(self):

        self.app.pg.delete\_by\_id(self.insurancetypesTable.item(self.insurancetypes\_table\_row, 0).text(), 'insurance\_types')

        self.update\_insurancetypes()

    def deletecompanytype(self):

        self.app.pg.delete\_by\_id(self.companytypesTable.item(self.companytypes\_table\_row, 0).text(), 'company\_types')

        self.update\_companytypes()

    def editcity(self):

        if self.citiesTable.rowCount() == 0: return

        self.cities\_edit\_window = cru\_window.CRUWindow(

            self.app,

            self.update\_cities,

            'cities',

            ('edit', self.citiesTable.item(self.cities\_table\_row, 0).text()),

            (('Сумма страхования', 'line', 'city'),)

        )

        self.cities\_edit\_window.show()

    def editsocialstatus(self):

        if self.socialstatusesTable.rowCount() == 0: return

        self.socialstatuses\_edit\_window = cru\_window.CRUWindow(

            self.app,

            self.update\_socialstatuses,

            'social\_statuseses',

            ('edit', self.socialstatusesTable.item(self.socialstatuses\_table\_row, 0).text()),

            (('Социальное положение', 'line', 'status'),)

        )

        self.socialstatuses\_edit\_window.show()

    def editinsurancetype(self):

        if self.insurancetypesTable.rowCount() == 0: return

        self.insurancetypes\_edit\_window = cru\_window.CRUWindow(

            self.app,

            self.update\_insurancetypes,

            'insurance\_types',

            ('edit', self.insurancetypesTable.item(self.insurancetypes\_table\_row, 0).text()),

            (('Вид страхования', 'line', 'type'),)

        )

        self.insurancetypes\_edit\_window.show()

    def editcompanytype(self):

        if self.companytypesTable.rowCount() == 0: return

        self.companytypes\_edit\_window = cru\_window.CRUWindow(

            self.app,

            self.update\_companytypes,

            'company\_types',

            ('edit', self.companytypesTable.item(self.companytypes\_table\_row, 0).text()),

            (('Тип компании', 'line', 'type'),)

        )

        self.companytypes\_edit\_window.show()

    def usersTable\_cellClicked(self, row, column):

            self.users\_table\_row = row

    def socialstatusesTable\_cellClicked(self, row, column):

            self.socialstatuses\_table\_row = row

    def citiesTable\_cellClicked(self, row, column):

            self.cities\_table\_row = row

    def insurancetypesTable\_cellClicked(self, row, column):

            self.insurancetypes\_table\_row = row

    def companytypesTable\_cellClicked(self, row, column):

            self.companytypes\_table\_row = row

//client\_window.py

from PyQt5 import QtWidgets, uic, QtGui

from PyQt5.QtWidgets import QTableWidget, QTableWidgetItem

from . import qt, cru\_window, query\_window

class ClientWindow(QtWidgets.QMainWindow):

    def \_\_init\_\_(self, app):

        super(ClientWindow, self).\_\_init\_\_()

        self.app = app

        uic.loadUi(self.app.ui\_path.joinpath('client.ui'), self)

        self.contractsTable.setColumnWidth(0, 5)

        self.contractsTable.setColumnWidth(1, 229)

        self.contractsTable.setColumnWidth(2, 230)

        self.contractsTable.setColumnWidth(3, 195)

        self.contractsTable.setColumnWidth(4, 191)

        self.companiesTable.setColumnWidth(0, 5)

        self.companiesTable.setColumnWidth(1, 165)

        self.companiesTable.setColumnWidth(2, 208)

        self.companiesTable.setColumnWidth(3, 206)

        self.companiesTable.setColumnWidth(4, 140)

        self.companiesTable.setColumnWidth(5, 135)

        self.branchesTable.setColumnWidth(0, 5)

        self.branchesTable.setColumnWidth(1, 130)

        self.branchesTable.setColumnWidth(2, 130)

        self.branchesTable.setColumnWidth(3, 130)

        self.branchesTable.setColumnWidth(4, 204)

        self.branchesTable.setColumnWidth(5, 130)

        self.branchesTable.setColumnWidth(6, 130)

        self.contractsTable.cellClicked.connect(self.contractsTable\_cellClicked)

        self.addcontractButton.clicked.connect(self.addcontract)

        self.deletecontractButton.clicked.connect(self.deletecontract)

        self.searchcontractButton.clicked.connect(self.searchcontract)

        self.searchcompanyButton.clicked.connect(self.searchcompany)

        self.searchbranchButton.clicked.connect(self.searchbranch)

        self.client = self.app.pg.get\_all\_clients()[0]

        self.label\_3.setText('ФИО: ' + self.client[1])

        self.label\_4.setText('Дата рождения: ' + str(self.client[2]))

        self.label\_5.setText('Социальное положение: ' + self.client[3])

        self.label\_6.setText('Телефон: ' + self.client[4])

        self.tabWidget.setCurrentIndex(0)

        self.centralwidget.setContentsMargins(9, 9, 9, 9)

        self.setFixedSize(self.size())

        self.show()

        self.tabWidget.tabBarClicked.connect(self.tab\_changed)

    def tab\_changed(self, index):

        if index == 0:

            self.update\_contracts()

        elif index == 1:

            self.update\_companies()

            self.update\_branches()

    def update\_contracts(self, rows=None):

        self.contractsTable.clearContents()

        self.contractsTable.setRowCount(0)

        if rows==None:

            rows = self.app.pg.get\_contracts\_by\_client\_id2(self.client[0])

        for names in rows:

            qt.add\_table\_row(self.contractsTable, names)

    def update\_companies(self, rows=None):

        self.companiesTable.clearContents()

        self.companiesTable.setRowCount(0)

        if rows==None:

            rows = self.app.pg.get\_companies()

        for names in rows:

            qt.add\_table\_row(self.companiesTable, names)

        if len(rows) > 0:

            self.companiesTable.selectRow(0)

            self.companiesTable\_cellClicked(0, 0)

    def update\_branches(self, rows=None):

        self.branchesTable.clearContents()

        self.branchesTable.setRowCount(0)

        if rows==None:

            rows = self.app.pg.get\_branches()

        for names in rows:

            qt.add\_table\_row(self.branchesTable, names)

        if len(rows) > 0:

            self.branchesTable.selectRow(0)

            self.branchesTable\_cellClicked(0, 0)

    def searchcontract(self):

        if self.contractsTable.rowCount() == 0: return

        self.contracts\_search\_window = cru\_window.CRUWindow(

            self.app,

            self.update\_contracts,

            'contracts',

            ('search',),

            (('Филиал', 'line'),

            ('Вид страхования', 'line'),

            ('Сумма страхования', 'line'),

            ('Дата заключения договора', 'line')),

            self.app.pg.select\_contracts\_where2

        )

        self.contracts\_search\_window.show()

    def addcontract(self):

        self.contracts\_add\_window = cru\_window.CRUWindow(

            self.app,

            self.update\_contracts,

            'contracts',

            ('add',),

            (('Филиал', 'combo', 'name', 'branches'),

            ('Клиент', 'noeditline', self.client[1], self.client[0]),

            ('Вид страхования', 'combo', 'type', 'insurance\_types'),

            ('Сумма страхования', 'line'),

            ('Дата заключения договора', 'line'))

        )

        self.contracts\_add\_window.show()

    def deletecontract(self):

        self.app.pg.delete\_by\_id(self.contractsTable.item(self.contracts\_table\_row, 0).text(), 'contracts')

        self.update\_contracts()

    def searchcompany(self):

        if self.companiesTable.rowCount() == 0: return

        self.companies\_search\_window = cru\_window.CRUWindow(

            self.app,

            self.update\_companies,

            'companies',

            ('search',),

            (('Компания', 'line'),

            ('Тип компании', 'line'),

            ('Год получения лицензии', 'line'),

            ('Город', 'line'),

            ('Телефон', 'line')),

            self.app.pg.select\_companies\_where

        )

        self.companies\_search\_window.show()

    def searchbranch(self):

        if self.branchesTable.rowCount() == 0: return

        self.branches\_search\_window = cru\_window.CRUWindow(

            self.app,

            self.update\_branches,

            'branches',

            ('search',),

            (('Компания', 'line'),

            ('Филиал', 'line'),

            ('Город', 'line'),

            ('Адрес', 'line'),

            ('Телефон', 'line'),

            ('Количество сотрудников', 'line')),

            self.app.pg.select\_branches\_where

        )

        self.branches\_search\_window.show()

    def contractsTable\_cellClicked(self, row, column):

            self.contracts\_table\_row = row

    def companiesTable\_cellClicked(self, row, column):

            self.companies\_table\_row = row

    def branchesTable\_cellClicked(self, row, column):

            self.branches\_table\_row = row

//branch\_window.py

from PyQt5 import QtWidgets, uic, QtGui

from PyQt5.QtWidgets import QTableWidget, QTableWidgetItem

from . import qt, cru\_window

class BranchWindow(QtWidgets.QMainWindow):

    def \_\_init\_\_(self, app):

        super(BranchWindow, self).\_\_init\_\_()

        self.app = app

        uic.loadUi(self.app.ui\_path.joinpath('branch\_employee.ui'), self)

        self.contractsTable.setColumnWidth(0, 5)

        self.contractsTable.setColumnWidth(1, 320)

        self.contractsTable.setColumnWidth(2, 245)

        self.contractsTable.setColumnWidth(3, 240)

        self.contractsTable.setColumnWidth(4, 223)

        self.clientsTable.setColumnWidth(0, 5)

        self.clientsTable.setColumnWidth(1, 288)

        self.contractsTable\_2.setColumnWidth(0, 5)

        self.contractsTable\_2.setColumnWidth(1, 230)

        self.contractsTable\_2.setColumnWidth(2, 229)

        self.contractsTable\_2.setColumnWidth(3, 229)

        self.addcontractButton.clicked.connect(self.addcontract)

        self.addclientButton.clicked.connect(self.addclient)

        self.addclientcontractButton.clicked.connect(self.addclientcontract)

        self.deletecontractButton.clicked.connect(self.deletecontract)

        self.deleteclientButton.clicked.connect(self.deleteclient)

        self.deleteclientcontractButton.clicked.connect(self.deletecontract)

        self.searchcontractButton.clicked.connect(self.searchcontract)

        self.searchclientButton.clicked.connect(self.searchclient)

        self.contractsTable.cellClicked.connect(self.contractsTable\_cellClicked)

        self.clientsTable.cellClicked.connect(self.clientsTable\_cellClicked)

        self.branch = self.app.pg.get\_branch()

        self.label\_3.setText('Компания: ' + self.branch[0])

        self.label\_4.setText('Название: ' + self.branch[1])

        self.label\_5.setText('Город: ' + self.branch[2])

        self.label\_6.setText('Адрес: ' + self.branch[3])

        self.label\_7.setText('Телефон: ' + self.branch[4])

        self.label\_8.setText('Количество сотрудников: ' + str(self.branch[5]))

        self.centralwidget.setContentsMargins(9, 9, 9, 9)

        self.setFixedSize(self.size())

        self.show()

        self.tabWidget.setCurrentIndex(0)

        self.tabWidget.tabBarClicked.connect(self.tab\_changed)

    def tab\_changed(self, index):

        if index == 0:

            self.update\_contracts()

        elif index == 1:

            self.update\_clients()

    def update\_contracts(self, rows=None):

        self.contractsTable.clearContents()

        self.contractsTable.setRowCount(0)

        if rows==None:

            rows = self.app.pg.get\_contracts()

        for names in rows:

            qt.add\_table\_row(self.contractsTable, names)

        if len(rows) > 0:

            self.contractsTable.selectRow(0)

    def update\_clients(self, rows=None):

        self.clientsTable.clearContents()

        self.clientsTable.setRowCount(0)

        self.contractsTable\_2.clearContents()

        self.contractsTable\_2.setRowCount(0)

        if rows==None:

            rows = self.app.pg.get\_all\_clients()

        for names in rows:

            qt.add\_table\_row(self.clientsTable, names)

        if len(rows) > 0:

            self.clientsTable.selectRow(0)

            self.clientsTable\_cellClicked(0, 0)

    def clientsTable\_cellClicked(self, row, column):

        self.clients\_table\_row = row

        self.contractsTable\_2.clearContents()

        self.contractsTable\_2.setRowCount(0)

        self.client\_id = self.clientsTable.item(row, 0).text()

        client = self.app.pg.get\_client\_by\_id(self.client\_id)

        self.label\_11.setText('Дата рождения: ' + str(client[1]))

        self.label\_12.setText('Социальное положение: ' + client[2])

        self.label\_13.setText('Телефон: ' + client[3])

        for contract in self.app.pg.get\_contracts\_by\_client\_id(self.client\_id):

            qt.add\_table\_row(self.contractsTable\_2, contract)

    def contractsTable\_cellClicked(self, row, column):

        self.contracts\_table\_row = row

    def addcontract(self):

        self.contracts\_add\_window = cru\_window.CRUWindow(

            self.app,

            self.update\_contracts,

            'contracts',

            ('add',),

            (('Филиал', 'noeditline', self.branch[1], self.app.pg.get\_branches()[0][0]),

            ('Клиент', 'combo', 'name', 'clients'),

            ('Вид страхования', 'combo', 'type', 'insurance\_types'),

            ('Сумма страхования', 'line'),

            ('Дата заключения договора', 'line'))

        )

        self.contracts\_add\_window.show()

    def addclient(self):

        self.clients\_add\_window = cru\_window.CRUWindow(

            self.app,

            self.update\_clients,

            'clients',

            ('add',),

            (('ФИО', 'line'),

            ('Дата рождения', 'line'),

            ('Социальное положение', 'combo', 'status', 'social\_statuses'),

            ('Телефон', 'line'))

        )

        self.clients\_add\_window.show()

    def addclientcontract(self):

        self.contracts\_add\_window = cru\_window.CRUWindow(

            self.app,

            self.update\_contracts,

            'contracts',

            ('add',),

            (('Филиал', 'noeditline', self.branch[1], self.app.pg.get\_branches()[0][0]),

            ('Клиент', 'noeditline', self.clientsTable.item(self.clients\_table\_row, 1).text(), self.client\_id),

            ('Вид страхования', 'combo', 'type', 'insurance\_types'),

            ('Сумма страхования', 'line'),

            ('Дата заключения договора', 'line'))

        )

        self.contracts\_add\_window.show()

    def deletecontract(self):

        self.app.pg.delete\_by\_id(self.contractsTable.item(self.contracts\_table\_row, 0).text(), 'contracts')

        self.update\_contracts()

    def deleteclient(self):

        self.app.pg.delete\_by\_id(self.clientsTable.item(self.clients\_table\_row, 0).text(), 'clients')

        self.update\_clients()

    def searchcontract(self):

        if self.contractsTable.rowCount() == 0: return

        self.contracts\_search\_window = cru\_window.CRUWindow(

            self.app,

            self.update\_contracts,

            'contracts',

            ('search',),

            (('Клиент', 'line'),

            ('Вид страхования', 'line'),

            ('Сумма страхования', 'line'),

            ('Дата заключения договора', 'line')),

            self.app.pg.select\_contracts\_where3

        )

        self.contracts\_search\_window.show()

    def searchclient(self):

        if self.clientsTable.rowCount() == 0: return

        self.clients\_search\_window = cru\_window.CRUWindow(

            self.app,

            self.update\_clients,

            'clients',

            ('search',),

            (('ФИО', 'line'),

            ('Дата рождения', 'line'),

            ('Социальное положение', 'line'),

            ('Телефон', 'line')),

            self.app.pg.select\_clients\_where

        )

        self.clients\_search\_window.show()

//postgres.py

import psycopg

class Postgres:

    def get\_socialstatuses(self):

        with self.connection.cursor() as cur:

            cur.execute('''

                SELECT \* FROM social\_statuses

            ''')

            return cur.fetchall()

    def get\_cities(self):

        with self.connection.cursor() as cur:

            cur.execute('''

                SELECT \* FROM cities

            ''')

            return cur.fetchall()

    def get\_insurancetypes(self):

        with self.connection.cursor() as cur:

            cur.execute('''

                SELECT \* FROM insurance\_types

            ''')

            return cur.fetchall()

    def get\_companytypes(self):

        with self.connection.cursor() as cur:

            cur.execute('''

                SELECT \* FROM company\_types

            ''')

            return cur.fetchall()

    def get\_companies(self):

        with self.connection.cursor() as cur:

            cur.execute('''

                SELECT co.id, co.name, ct.type, co.year, c.city, co.phone

                FROM companies co

                LEFT JOIN cities c ON c.id=co.city\_id

                LEFT JOIN company\_types ct ON co.company\_type\_id=ct.id

            ''')

            return cur.fetchall()

    def get\_branches(self):

        with self.connection.cursor() as cur:

            cur.execute('''

                SELECT b.id, co.name, b.name, c.city, b.address, b.phone, b.workers\_count

                FROM branches b

                LEFT JOIN cities c ON c.id=b.city\_id

                LEFT JOIN companies co ON b.company\_id=co.id

            ''')

            return cur.fetchall()

    def get\_all\_clients(self):

        with self.connection.cursor() as cur:

            cur.execute('''

                SELECT c.id, c.name, c.date, s.status, c.phone

                FROM clients c

                LEFT JOIN social\_statuses s ON s.id=c.social\_status\_id

            ''')

            return cur.fetchall()

    def get\_all\_contracts(self):

        with self.connection.cursor() as cur:

            cur.execute('''

                SELECT co.id, b.name, c.name, i.type, co.sum, co.date

                FROM contracts co

                LEFT JOIN branches b ON co.branch\_id=b.id

                LEFT JOIN clients c ON co.client\_id=c.id

                LEFT JOIN insurance\_types i ON co.insurance\_type\_id=i.id

            ''')

            return cur.fetchall()

    def get\_users(self):

        with self.connection.cursor() as cur:

            cur.execute('''

                SELECT \* FROM users

            ''')

            return cur.fetchall()

    def get\_branch(self):

        with self.connection.cursor() as cur:

            cur.execute('''

                SELECT co.name, b.name, c.city, b.address, b.phone, b.workers\_count

                FROM branches b

                LEFT JOIN cities c ON c.id=b.city\_id

                LEFT JOIN companies co ON b.company\_id=co.id

            ''')

            return cur.fetchone()

    def get\_contracts(self):

        with self.connection.cursor() as cur:

            cur.execute('''

                SELECT co.id, c.name, i.type, co.sum, co.date

                FROM contracts co

                LEFT JOIN clients c ON co.client\_id=c.id

                LEFT JOIN insurance\_types i ON co.insurance\_type\_id=i.id

            ''')

            return cur.fetchall()

    def get\_client\_by\_id(self, client\_id):

        with self.connection.cursor() as cur:

            cur.execute(f'''

                SELECT c.name, c.date, s.status, c.phone

                FROM clients c

                LEFT JOIN social\_statuses s ON s.id=c.social\_status\_id

                WHERE c.id={client\_id}

            ''')

            return cur.fetchone()

    def get\_contracts\_by\_client\_id2(self, client\_id):

        with self.connection.cursor() as cur:

            cur.execute(f'''

                SELECT co.id, b.name, i.type, co.sum, co.date

                FROM contracts co

                LEFT JOIN branches b ON co.branch\_id=b.id

                LEFT JOIN insurance\_types i ON co.insurance\_type\_id=i.id

                WHERE co.client\_id={client\_id}

            ''')

            return cur.fetchall()

    def get\_contracts\_by\_client\_id(self, client\_id):

        with self.connection.cursor() as cur:

            cur.execute(f'''

                SELECT co.id, i.type, co.sum, co.date

                FROM contracts co

                LEFT JOIN insurance\_types i ON co.insurance\_type\_id=i.id

                WHERE co.client\_id={client\_id}

            ''')

            return cur.fetchall()

    def connect(self, login, password):

        self.connection = psycopg.connect(f'host=127.0.0.1 dbname=postgres user={login} password={password}')

    def check\_role(self):

        with self.connection.cursor() as cur:

            try:

                role = 'branch\_employees'

                cur.execute(f'SET ROLE {role}')

                return role

            except:

                self.connection.rollback()

            try:

                role = 'clients\_role'

                cur.execute(f'SET ROLE {role}')

                return role

            except:

                self.connection.rollback()

            try:

                role = 'queries\_role'

                cur.execute(f'SET ROLE {role}')

                return role

            except:

                self.connection.rollback()

            role = 'admins'

            cur.execute(f'SET ROLE {role}')

            return role

    def reset\_role(self):

        with self.connection.cursor() as cur:

            cur.execute('RESET ROLE')

    def drop\_user(self, user):

        with self.connection.cursor() as cur:

            cur.execute(f"DELETE FROM users WHERE login='{user}'")

            self.connection.commit()

    def get\_user\_role(self, login):

        with self.connection.cursor() as cur:

            cur.execute(f"SELECT role FROM users WHERE login='{login}'")

            return cur.fetchone()[0]

    def create\_user(self, login, password, role, branch\_id):

        with self.connection.cursor() as cur:

            cur.execute(f"CREATE USER {login} PASSWORD '{password}'")

            cur.execute(f'''

                INSERT INTO users

                VALUES ('{login}', {branch\_id}, '{role}')

            ''')

            self.connection.commit()

    def get\_field\_by\_id(self, id, field, table):

        with self.connection.cursor() as cur:

            cur.execute(f'SELECT {field} FROM {table} WHERE id={id}')

            return cur.fetchone()[0]

    def update\_values(self, values, id, table):

        with self.connection.cursor() as cur:

            cur.execute(f'UPDATE {table} SET {values} WHERE id={id}')

            self.connection.commit()

    def select\_contracts\_where(self, branch=None, client=None, insurance\_type=None, sum=None, date=None):

        values = []

        if branch: values.append(f"b.name LIKE '%{branch}%'")

        if client: values.append(f"c.name LIKE '%{client}%'")

        if insurance\_type: values.append(f"i.type='{insurance\_type}'")

        if sum: values.append(f'co.sum={sum}')

        if date: values.append(f'co.date={date}')

        where = ' AND '.join(values)

        with self.connection.cursor() as cur:

            cur.execute(f'''

                SELECT co.id, b.name, c.name, i.type, co.sum, co.date

                FROM contracts co

                LEFT JOIN branches b ON b.id=co.branch\_id

                LEFT JOIN clients c ON c.id=co.client\_id

                LEFT JOIN insurance\_types i ON i.id=co.insurance\_type\_id

                WHERE {where}

            ''')

            return cur.fetchall()

    def select\_contracts\_where2(self, branch=None, insurance\_type=None, sum=None, date=None):

        values = []

        if branch: values.append(f"b.name LIKE '%{branch}%'")

        if insurance\_type: values.append(f"i.type='{insurance\_type}'")

        if sum: values.append(f'co.sum={sum}')

        if date: values.append(f'co.date={date}')

        where = ' AND '.join(values)

        print(where)

        with self.connection.cursor() as cur:

            cur.execute(f'''

                SELECT b.name, i.type, co.sum, co.date

                FROM contracts co

                LEFT JOIN branches b ON b.id=co.branch\_id

                LEFT JOIN insurance\_types i ON i.id=co.insurance\_type\_id

                WHERE {where}

            ''')

            return cur.fetchall()

    def select\_contracts\_where3(self, client=None, insurance\_type=None, sum=None, date=None):

            values = []

            if client: values.append(f"c.name LIKE '%{client}%'")

            if insurance\_type: values.append(f"i.type='{insurance\_type}'")

            if sum: values.append(f'co.sum={sum}')

            if date: values.append(f'co.date={date}')

            where = ' AND '.join(values)

            with self.connection.cursor() as cur:

                cur.execute(f'''

                    SELECT co.id, c.name, i.type, co.sum, co.date

                    FROM contracts co

                    LEFT JOIN clients c ON c.id=co.client\_id

                    LEFT JOIN insurance\_types i ON i.id=co.insurance\_type\_id

                    WHERE {where}

                ''')

                return cur.fetchall()

    def select\_clients\_where(self, name=None, date=None, social\_status=None, phone=None):

        values = []

        if name: values.append(f"c.name LIKE '%{name}%'")

        if date: values.append(f'c.date={date}')

        if social\_status: values.append(f"s.status LIKE '%{social\_status}%'")

        if phone: values.append(f"c.phone='{phone}'")

        where = ' AND '.join(values)

        with self.connection.cursor() as cur:

            cur.execute(f'''

                SELECT c.id, c.name, c.date, s.status, c.phone

                FROM clients c

                LEFT JOIN social\_statuses s ON s.id=c.social\_status\_id

                WHERE {where}

            ''')

            return cur.fetchall()

    def select\_companies\_where(self, name=None, company\_type=None, year=None, city=None, phone=None):

        values = []

        if name: values.append(f"co.name LIKE '%{name}%'")

        if company\_type: values.append(f"ct.type LIKE '%{company\_type}%'")

        if year: values.append(f"co.year='{year}'")

        if city: values.append(f'c.city={city}')

        if phone: values.append(f'co.phone={phone}')

        where = ' AND '.join(values)

        with self.connection.cursor() as cur:

            cur.execute(f'''

                SELECT co.id, co.name, ct.type, co.year, c.city, co.phone

                FROM companies co

                LEFT JOIN company\_types ct ON ct.id=co.company\_type\_id

                LEFT JOIN cities c ON c.id=co.city\_id

                WHERE {where}

            ''')

            return cur.fetchall()

    def select\_branches\_where(self, company=None, name=None, city=None, address=None, phone=None, workers\_count=None):

        values = []

        if company: values.append(f"co.name LIKE '%{company}%'")

        if name: values.append(f"b.name LIKE '%{name}%'")

        if city: values.append(f"c.city='{city}'")

        if address: values.append(f'b.address={address}')

        if phone: values.append(f'b.phone={phone}')

        if workers\_count: values.append(f'b.workers\_count={workers\_count}')

        where = ' AND '.join(values)

        with self.connection.cursor() as cur:

            cur.execute(f'''

                SELECT b.id, co.name, b.name, c.city, b.address, b.phone, b.workers\_count

                FROM branches b

                LEFT JOIN companies co ON co.id=b.company\_id

                LEFT JOIN cities c ON c.id=b.city\_id

                WHERE {where}

            ''')

            return cur.fetchall()

    def get\_id\_field(self, field, table):

        with self.connection.cursor() as cur:

            cur.execute(f'SELECT id, {field} FROM {table}')

            return cur.fetchall()

    def insert\_values(self, values, table):

        if table == 'exhibits':

            values += f', {self.get\_company\_id()}'

        with self.connection.cursor() as cur:

            cur.execute(f'INSERT INTO {table} VALUES (DEFAULT, {values})')

            self.connection.commit()

    def delete\_by\_id(self, id, table):

        with self.connection.cursor() as cur:

            cur.execute(f'DELETE FROM {table} WHERE id={id}')

            self.connection.commit()

    def rollback(self):

        self.connection.rollback()

    def get\_all(self, from\_):

        with self.connection.cursor() as cur:

            cur.execute(f'''

                SELECT \* FROM {from\_}

            ''')

            return cur.fetchall()

    def get\_column\_names(self, from\_):

        with self.connection.cursor() as cur:

            cur.execute(f'SELECT \* FROM {from\_} WHERE FALSE')

            return [desc[0] for desc in cur.description]

    def get\_id(self, field, value, table):

            with self.connection.cursor() as cur:

                cur.execute(f"SELECT id FROM {table} WHERE {field} LIKE '%{value}%'")

                return cur.fetchone()

//auth.py

from PyQt5 import QtWidgets, uic

from . import postgres, branch\_window, admin\_window, client\_window, qt, query\_window

import psycopg

class Auth(QtWidgets.QMainWindow):

    def \_\_init\_\_(self, app):

        super(Auth, self).\_\_init\_\_()

        self.app = app

        uic.loadUi(self.app.ui\_path.joinpath('auth.ui'), self)

        self.gridLayout.setContentsMargins(9, 9, 9, 9)

        self.setFixedSize(self.size())

        self.show()

        self.main\_window = None

        self.pushButton.clicked.connect(self.pushButton\_clicked)

    def pushButton\_clicked(self):

        self.app.pg = postgres.Postgres()

        try:

            self.app.pg.connect(self.loginEdit.text(), self.passwordEdit.text())

            role = self.app.pg.check\_role()

            self.app.pg.reset\_role()

        except:

            qt.message('Ошибка входа', 'Неверный логин или пароль', 'error')

            return

        if role == 'branch\_employees':

            self.main\_window = branch\_window.BranchWindow(self.app)

        elif role == 'clients\_role':

            self.main\_window = client\_window.ClientWindow(self.app)

        elif role == 'admins':

            self.main\_window = admin\_window.AdminWindow(self.app)

        elif role == 'queries\_role':

            self.main\_window = query\_window.QueryWindow(self.app)

        self.hide()

//qt.py

from PyQt5.QtWidgets import QMessageBox, QTableWidgetItem

def message(title, text, msg\_type):

    msg = QMessageBox()

    if msg\_type == 'info':

        msg.setIcon(QMessageBox.Information)

    elif msg\_type == 'error':

        msg.setIcon(QMessageBox.Critical)

    elif msg\_type == 'warn':

        msg.setIcon(QMessageBox.Warning)

    else:

        msg.setIcon(QMessageBox.Information)

    msg.setWindowTitle(title)

    msg.setText(text)

    msg.exec\_()

def add\_table\_row(table, row\_data):

    row = table.rowCount()

    table.setRowCount(row + 1)

    col = 0

    for item in row\_data:

        cell = QTableWidgetItem(str(item))

        table.setItem(row, col, cell)

        col += 1

# ПРИЛОЖЕНИЕ Д РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

При запуске программы открывается окно авторизации. Если ввести верные данные авторизации (логин и пароль), программа откроет личный кабинет пользователя с таблицами и информацией.

Для того, чтобы добавить запись в таблицу, необходимо нажать на кнопку «Добавить» и появится окно добавления. Необходимо заполнить все поля и выбрать нужные данные в справочниках. Если данные будут введены корректно, запись добавится в базу данных.

Для того, чтобы удалить запись в таблице, необходимо выбрать её нажатием левой кнопкой мыши и нажать на кнопку «Удалить».

Для того, чтобы изменить запись в таблице, необходимо выбрать её нажатием левой кнопкой мыши и нажать на кнопку «Изменить». Появится окно изменения. Можно изменить любые данные из доступных полей и справочников. Если данные будут изменены корректно, запись отредактируется в базе данных.

Для того, чтобы найти запись в таблице, необходимо нажать на кнопку «Найти» и появится окно поиска. Можно ввести любые данные в доступные поля и справочники, и по ним будет произведен поиск. В таблице останутся только те записи, которые удовлетворяют условию.

Для того, чтобы выполнить запрос, необходимо выбрать нужный запрос вверху окна. Если это запрос с параметрами, необходимо ввести их в поле под запросом и нажать на кнопку «Выполнить». Если это запрос без параметров, он выполнится автоматически. В таблице появятся те записи, которые соответствуют запросу.

Для того, чтобы экспортировать результат запроса в Excel, необходимо зайти на форму с запросами, выполнить запрос и нажать «Excel». Excel-файл с диаграммой появится в папке с программой.

# ПРИЛОЖЕНИЕ Е РУКОВОДСТВО АДМИНИСТРАТОРА

При запуске программы открывается окно авторизации. Если ввести верные данные авторизации (логин и пароль), программа откроет личный кабинет администратора со справочниками и таблицей пользователей.

Для того, чтобы добавить запись в таблицу, необходимо нажать на кнопку «Добавить» и появится окно добавления. Необходимо заполнить все поля и если данные будут введены корректно, запись добавится в базу данных.

Для того, чтобы удалить запись в таблице, необходимо выбрать её нажатием левой кнопкой мыши и нажать на кнопку «Удалить».

Для того, чтобы изменить запись в таблице, необходимо выбрать её нажатием левой кнопкой мыши и нажать на кнопку «Изменить». Появится окно изменения. Можно изменить любые данные из доступных полей. Если данные будут изменены корректно, запись отредактируется в базе данных.

Для того, чтобы найти запись в таблице, необходимо нажать на кнопку «Найти» и появится окно поиска. Можно ввести любые данные в доступные поля, и по ним будет произведен поиск. В таблице останутся только те записи, которые удовлетворяют условию.

Для того, чтобы выполнить запрос, необходимо выбрать нужный запрос вверху окна. Если это запрос с параметрами, необходимо ввести их в поле под запросом и нажать на кнопку «Выполнить». Если это запрос без параметров, он выполнится автоматически. В таблице появятся те записи, которые соответствуют запросу.

Для того, чтобы экспортировать результат запроса в Excel, необходимо зайти на форму с запросами, выполнить запрос и нажать «Excel». Excel-файл с диаграммой появится в папке с программой.

# ПРИЛОЖЕНИЕ Ж ОТЧЕТ О ПРОВЕРКЕ НА ЗАИМСТВОВАНИЯ

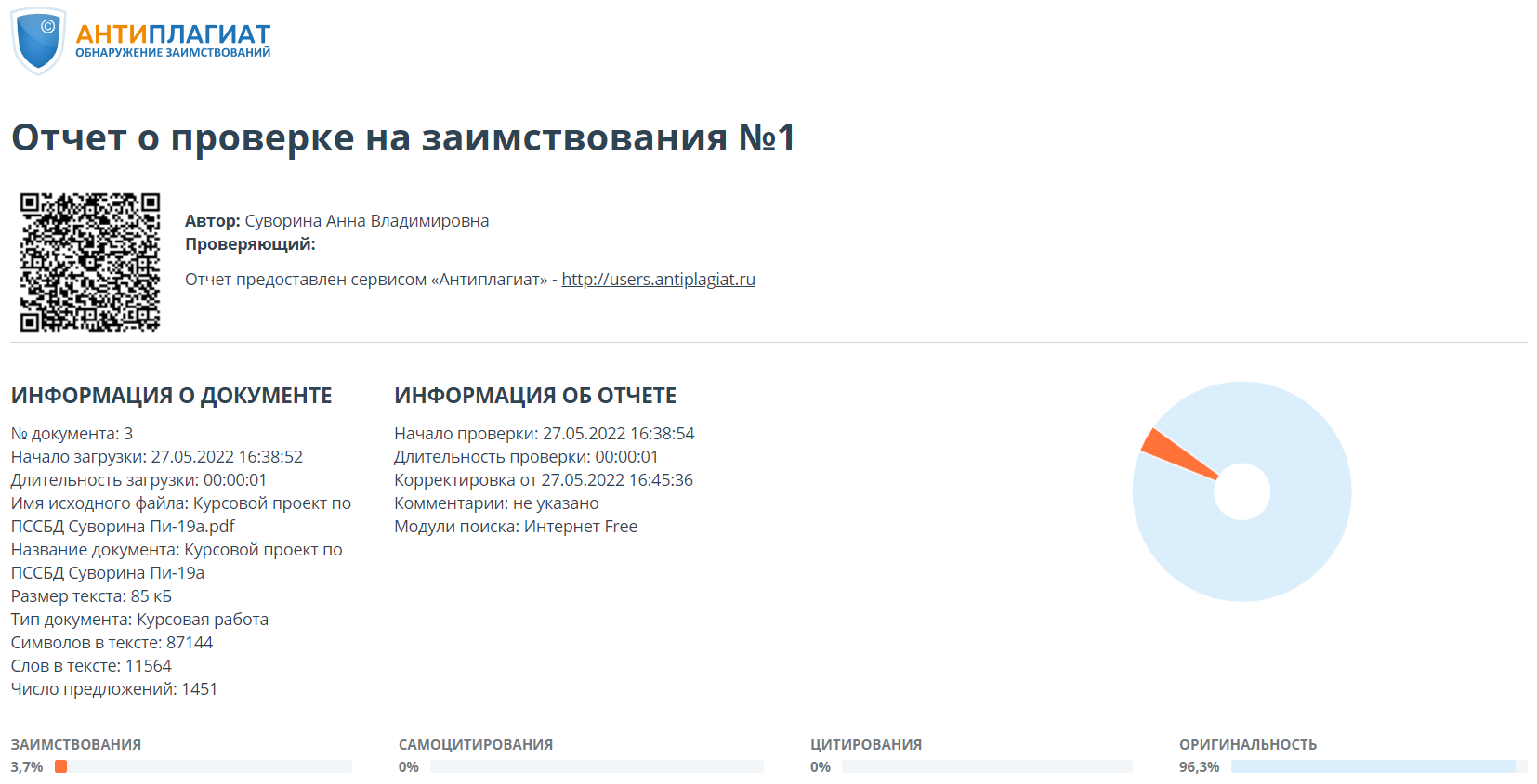


Рисунок Ж.1 – Отчет о проверке на заимствования

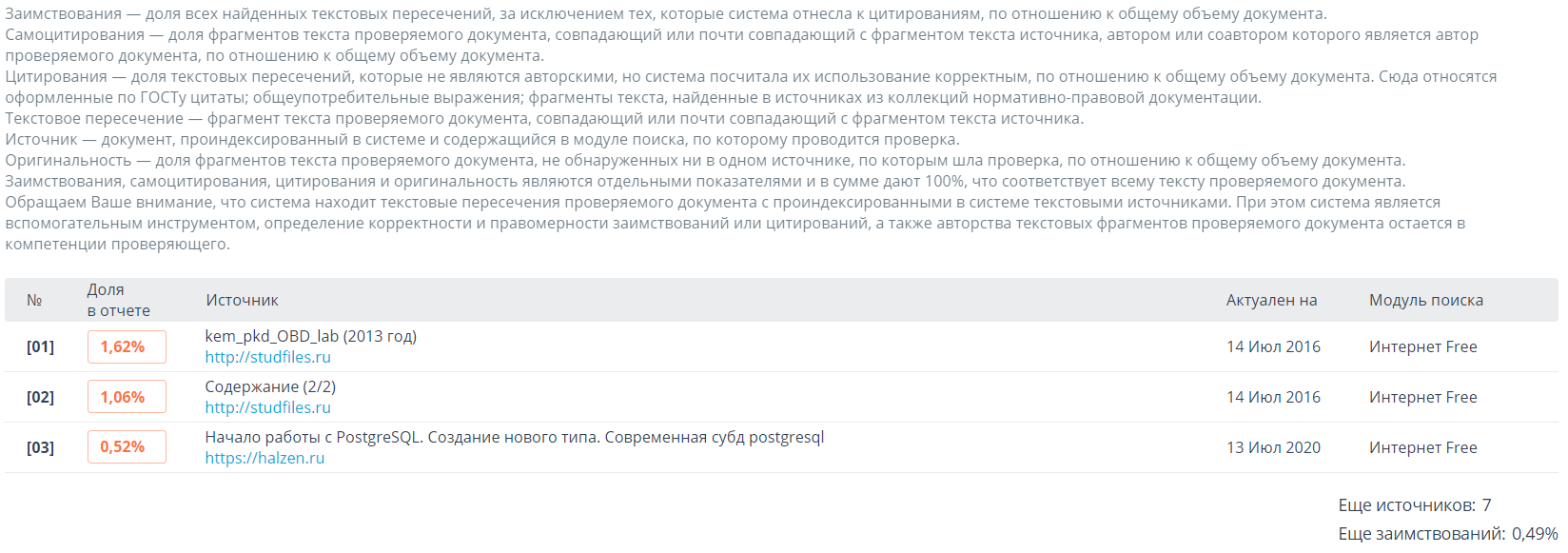


Рисунок Ж.2 – Заимствования