ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

"ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ"

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе по курсу “Базы данных”

Тема работы:

“Разработка информационной системы для автоматизации деятельности компаний мобильных операторов”

Руководители: Щедрин С.В.

Рычка О.В.

Незамова Л.В.

(подпись) (дата)

Разработал: Носаченко А. А.

ст.гр. ПИ-19б

(подпись) (дата)

Донецк - 2021

РЕФЕРАТ

Пояснительная записка содержит 66 с., 64 рис., 12 табл., 5 источников, 5 приложений.

Объектом исследования данной работы является обработка сервисных данных компаний мобильных операторов и способы её оптимизации.

Цель работы – разработать приложение для оптимизации работы мобильных операторов и учёта сервисных транзакций, получить практической опыт и знания для работы с системами управления базами данных, методами моделирования баз данных, составления запросов на языке структурированных запросов SQL.

Результатом работы является реляционная база данных, предназначенная для хранения сервисной информации мобильных операторов, разработанная в соответствии требованиям к курсовой работе. Разработанная база данных написана в СУБД PostgreSQL 12, запросы реализованы средствами языка SQL.

Для разработки системы использована СУБД PostgreSQL 12 и оболочка для неё pgAdmin4; пользовательский интерфейс реализован средствами языка программирования высокого уровня Java.

БАЗЫ ДАННЫХ, МОДЕЛИРОВАНИЕ БАЗ ДАННЫХ, ЗАПРОСЫ, SQL, ТАБЛИЦЫ, JAVA, POSTGRESQL 12, МОБИЛЬНЫЙ ОПЕРАТОР

# СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 3](#_Toc73653849)

[ВВЕДЕНИЕ 5](#_Toc73653850)

[1 ПРЕДПРОЕКТНОЕ ИССЛЕДОВАНИЕ 6](#_Toc73653851)

[1.1 Проблемы автоматизации деятельности мобильных операторов 6](#_Toc73653852)

[1.2 Разработка информационной системы «Сервисные данные мобильного оператора» 7](#_Toc73653853)

[1.2.1 Анализ предметной области 7](#_Toc73653854)

[1.2.2 Описание входных документов 7](#_Toc73653855)

[1.2.3 Описание выходных документов 8](#_Toc73653856)

[1.2.4 Список ограничений 8](#_Toc73653857)

[1.3 Назначение и функции системы 9](#_Toc73653858)

[2 ТЕХНИЧЕСКОЕ ПРОЕКТИРОВАНИЕ 10](#_Toc73653859)

[2.1 Инфологическое проектирование 10](#_Toc73653860)

[2.2 Даталогическое проектирование 11](#_Toc73653861)

[2.3 Физическое проектирование 12](#_Toc73653862)

[3 РАЗРАБОТКА ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ 15](#_Toc73653863)

[3.1 Создание проекта и базы данных 15](#_Toc73653864)

[3.2 Создание таблиц 15](#_Toc73653865)

[3.3 Схема базы данных (поддержка целостности данных) 19](#_Toc73653866)

[3.4 Разработка внешних представлений данных - меню, форм, запросов, отчетов 19](#_Toc73653867)

[3.5 Разработка процедур обработки данных (модулей) 26](#_Toc73653868)

[3.5.1 Симметричное внутреннее соединение с условием (два запроса с условием отбора по внешнему ключу, два – по дате) 26](#_Toc73653869)

[3.5.2 Симметричное внутреннее соединение без условия (три запроса) 28](#_Toc73653870)

[3.5.3 Левое внешнее соединение 30](#_Toc73653871)

[3.5.4 Правое внешнее соединение 30](#_Toc73653872)

[3.5.5 Запрос на запросе по принципу левого соединения 31](#_Toc73653873)

[3.5.6 Итоговый запрос без условия 31](#_Toc73653874)

[3.5.7 Итоговый запрос с условием на данные 32](#_Toc73653875)

[3.5.8 Итоговый запрос с условием на группы 32](#_Toc73653876)

[3.5.9 Итоговый запрос с условием на данные и на группы 33](#_Toc73653877)

[3.5.10 Запрос на запросе по принципу итогового запроса 33](#_Toc73653878)

[3.5.11 Запрос с подзапросом 34](#_Toc73653879)

[3.5.12 Запросы из курсового проекта 35](#_Toc73653880)

[3.6 Тестирование и отладка БД 37](#_Toc73653881)

[3.7 Разработка эксплуатационной документации БД 37](#_Toc73653882)

[ЗАКЛЮЧЕНИЕ 38](#_Toc73653883)

[СПИСОК ЛИТЕРАТУРЫ 39](#_Toc73653884)

[ПРИЛОЖЕНИЕ А 40](#_Toc73653885)

[ПРИЛОЖЕНИЕ Б 44](#_Toc73653886)

[ПРИЛОЖЕНИЕ В 45](#_Toc73653887)

[ПРИЛОЖЕНИЕ Г 62](#_Toc73653888)

[ПРИЛОЖЕНИЕ Д 63](#_Toc73653889)

# ВВЕДЕНИЕ

Целями данной курсовой работы являются проектирование и создание реляционной базы данных на основе свободной объектно-реляционной системы управления базами данных PostgreSQL для оптимизации работы мобильных операторов и закрепление практических навыков разработки реляционных баз данных. Данная система управления базами данных выбрана благодаря её надёжности. Согласно результатам автоматизированного исследования различного ПО на предмет ошибок, проведённом в 2005 году, в исходном коде PostgreSQL было найдено 20 проблемных мест на 775 000 строк исходного кода (в среднем, одна ошибка на 39 000 строк кода). Для сравнения: MySQL — 97 проблем, одна ошибка на 8 000 строк кода. [1]

Разработанная информационная система может быть применена в реальных компаниях мобильных операторов для оптимизации работы их сотрудников.

Оптимальным способом взаимодействия с базой данных является разработанный графический пользовательский интерфейс, созданный при помощи языка программирования высокого уровня Java.

Данный язык был выбран потому, что предоставляет гибкие инструменты создания виртуального окружения. В ходе разработки использовалась библиотека классов, ориентированная на создание графического пользовательского интерфейса – JavaFX. Также следует отметить важную особенность языка Java – его кроссплатформенность, основанную на том, что код языка обрабатывается виртуальной машиной-транслятором, которая может быть подстроена под любую операционную систему, позволяя не изменять при этом код разработанного приложения.

# 1 ПРЕДПРОЕКТНОЕ ИССЛЕДОВАНИЕ

## 1.1 Проблемы автоматизации деятельности мобильных операторов

В процессе своей работы, сотрудники компаний взаимодействуют с большими объемами данных, которые необходимо хранить, упорядочивать, модифицировать. Одним из вариантов решения этой задачи является использование бумажных носителей данных. Недостатком такого способа является сложность поиска определённой финансовой операции, к примеру, по дате или по цене, сортировка контрактов по какому-то признаку.

Частично данную проблему решает установка специального ПО. Примером такого программного обеспечения может служить файловая система персонального компьютера, где все группы данных хранятся в отдельных файлах. У данного подхода имеется ряд недостатков: сложность поиска информации в хранилище, сложность обеспечения одинаковой структуры данных, сложность реализации многопользовательского доступа к хранилищу данных.

Для решения всех проблем разрабатываются автоматизированные информационные системы - специальное ПО, работающее с централизованным хранилищем данных. Преимущество такого подхода очевидно: вся информация хранится в единой базе данных, а персонал обращается к данным через терминалы с графическим интерфейсом и получает конкретную информацию. Данный подход позволяет разделить предшествующие сущности (группы данных) на целостные логические структуры. Это значительно ускорит взаимодействие с данными, позволит одновременную работу нескольких пользователей с базой, также такой подход облегчит отчетную деятельность, т.к. каждый сотрудник сможет составлять необходимые отчеты автоматически.

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

## 1.2 Разработка информационной системы «Сервисные данные мобильного оператора»

### 1.2.1 Анализ предметной области

Сервисные данные компании необходимы сотрудникам, которые контролируют финансовые операции, связанные с клиентами мобильного оператора, и вносят информацию о заключаемых и растогаемых контрактах.

Пользователями системы являются клиенты – лица, желающие получить некоторую услугу. Они вносят заявки на заключение контрактов, а также на оплату включённых в них услуг или на пополнение личного денежного счёта.

Для поиска нужного контракта требуется выполнить запрос в базу данных, находящий соответствия между списком контрактов и клиентами, заключившими их.

Таким образом, для обеспечения работы мобильного оператора, сотрудникам должна быть известна полная информация обо всех пользователях, которые заключали контракты с команией, услугах, которые имеют определённую цену и периодичность их оплаты, и краткую информацию о проводимых финансовых операциях. В связи с этим можно дать характеристику входных данных разрабатываемой системы.

### 1.2.2 Описание входных документов

Информация об оборудовании клиента включает в себя мощность модулей памяти, серверный идентификатор, код порта подключения, максимальная скоростная нагрузка интернет кабеля, пользовательный идентификатор оборудования, наличие схемы передачи данных, тип обработки передаваемых пакетов данных. Одинаковое оборудование может быть у нескольких клиентов. О каждом клиенте должна быть известна следующая информация: наименование, техническая информация, город регистрации, адрес, контактная информация. Также каждый клиент может името от одного личного счёта, главно информацией о которых является денежный баланс.

Клиенты проводят финансовые операции, осовываясь на одном из своих счетов, по пополнению личного счёта или оплате тарифа услуги, указанного в заключённом кантракте.

### 1.2.3 Описание выходных документов

Выходными документами, ожидаемыми от контроля сервисных данных мобильного оператора, являются диаграммы, используемые для представления статистических показателей.

Помимо диаграмм, выходными документами являются различные отчеты, сформированные по конкретным требованиями пользователя, сохраняемые в форматах .html или Excel и содержащие полную информацию по интересующему запросу в удобном для чтения виде. Выходной формат .html позволяет отправить отчёт на печать без дополнительных правок, а готовая таблица может быть открыта в программе Microsoft Excel для выбора пользователем места, куда будет сохранена данная таблица.

### 1.2.4 Список ограничений

В базе должны быть предусмотрены следующие ограничения:

- сумма денег финансовой операции должна быть в перелах от 10 до 1000000 у.е.;

- дата финансовой операции должна быть не ранее года образования компании и не позже текущего года;

- максимальная скоростная проводимость интернета оборудования пользователя должна быть в пределах от 10 до 500 МБ\с;

- стоимость тарифов должна быть в пределах от 10 до 1500 у.е.;

- дата заключения контракта должна быть не ранее года образования компании и не позже текущего года;

- дата вступления в силу контракта должна быть не ранее года заключения и не позже чем через 5 лет от текущего года.

- дата завершения действия контракта должна быть не ранее даты вступления с силу контракта и не позже чем через 15 лет от текущего года.

## 1.3 Назначение и функции системы

Пользователем разрабатываемой информационной системы является сотрудник компании. Назначением программы является упрощение работы пользователя за счет ускорения получения и редактирования данных и улучшения отображения информации.

Основные функции разрабатываемой системы:

- просмотр, добавление, редактирование, удаление записей;

- поиск по конкретному значению;

- группировка по различным критериям;

- генерация диаграмм для текущего состояния базы данных;

- генерация отчётов (в HTML и Excel виде) по избранным запросам.

# 2 ТЕХНИЧЕСКОЕ ПРОЕКТИРОВАНИЕ

## 2.1 Инфологическое проектирование

На основании результатов анализа предметной области можно провести инфологическое проектирование информационной системы. В результате данного проектирования получена семантическая модель предметной области. Для этого были выделены сущности, свойства сущностей и связи между сущностями.

Рассмотрим одну из сущностей в предметной области курсовой работы - тарифы. Среди свойств этой сущности можно выделить название тарифа, услуга, включённая в него, тип оплаты, стоимость.

Сущность счета обладает свойствами: принадлежность кленту, баланс.

Другая сущность – клиенты. Она обладает следующими свойствами: наименование, техническая информация оборудования, город, адрес, контактная информация.

Также следует отметить сущность контракты со следующими свойствами: наименование, клиент, заключивший контракт, даты заключения, вступления в силу и окончания, статус контракта.

Одной из ключевых сущностей является сущность финансовые операции, которая обладает свойствами: счёт, тип транзакции, сумма, дата операции, а также необязательные свойства: часть контракта, связанная с данной операцией, и комментарий.

Таким образом, в результате инфологического проектирования получена семантическая модель предметной области – модель наибольшего уровня абстракции. В ней выделены сущности модели данных, их свойства и отношения. Построение этой модели позволяет перейти к датологическому проектированию модели базы данных.

## 2.2 Даталогическое проектирование

В процессе даталогического проектирования создаётся схема данных из таблиц и отношений между ними на основании построенной ранее семантической модели, схема данных приводится к третьей нормальной форме. [2]

Основными таблицами в модели данных являются таблицы «Клиенты», «Счета», «Контракты» , «Тарифы», «Услуги» и «Финансовые операции».

Как отдельные таблицы вынесены справочники «Техническая информация», «Города клиентов», «Статусы контрактов», «Типы оплаты», «Типы транзакций», «Части контрактов», содержащие редко изменяемые записи и перекрёстные данные. В результате даталогического проектирования была построена реляционная модель данных (рисунок 2.1).

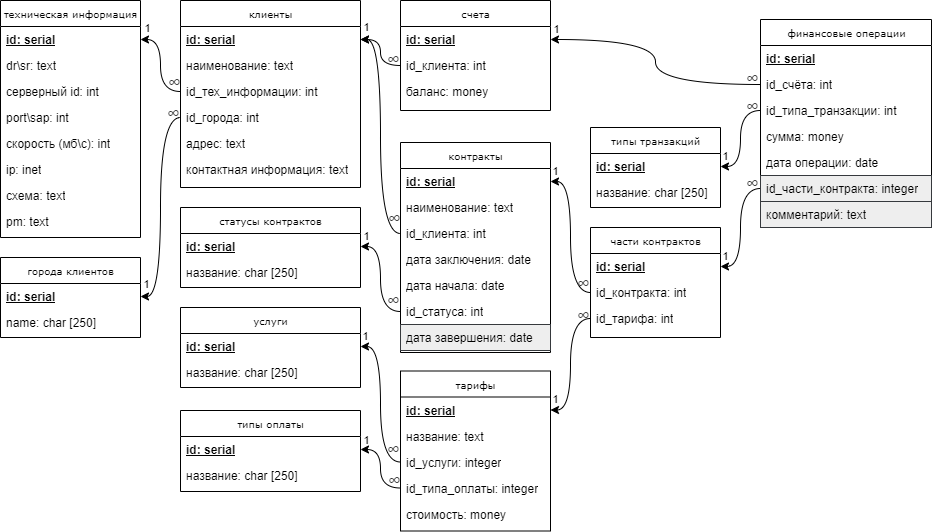


Рисунок 2.1 – Реляционная модель данных

На основании этой модели можно приступать к формированию физической модели базы данных с учётом особенностей ПО – PostgreSQL 12.

## 2.3 Физическое проектирование

На этапе физического проектирования выделяется общая структура базы данных и определяется, какие типы данных будут заложены в полях таблицы. СУБД PostgreSQL располагает большим количеством доступных типов данных. Для упрощения восприятия выбранные типы и общая структура сущностей представлена в виде таблиц (таблицы 2.1 – 2.12). В представленных таблицах все внешние ключи начинаются на «id\_».

Таблица 2.1 – Техническая информация

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица | Поле | Назначение поля | Тип данных |
| technical\_information | id | Код записи | serial |
| dr/sr | Мощность модулей памяти | text |
| server\_id | Серверный идентификатор | integer |
| port/sap | Код порта подключения | integer |
| speed(mb/s) | Максимальная скоростная нагрузка интернет кабеля | integer |
| ip | Пользовательный идентификатор оборудования | inet |
| scheme | Наличие схемы передачи данных | text |
| pm | Тип обработки передаваемых пакетов данных | text |

Таблица 2.2 – Клиенты

|  |  |  |  |
| --- | --- | --- | --- |
| clients | id | Код записи | serial |
| name | Наименование клиента | text |
| id\_technical\_information | Код технической информации | integer |
| id\_city | Код города | integer |
| address | Адрес | text |
| contact\_information | Контактная информация | text |

Таблица 2.3 – Города клиентов

|  |  |  |  |
| --- | --- | --- | --- |
| client\_cities | id | Код записи | serial |
| name | Название города | text |

Таблица 2.4 – Статусы контрактов

|  |  |  |  |
| --- | --- | --- | --- |
| contract\_statuses | id | Код записи | serial |
| name | Название статуса контрактов | character varying(250) |

Таблица 2.5 – Услуги

|  |  |  |  |
| --- | --- | --- | --- |
| services | id | Код записи | serial |
| name | Название услуги | character varying(250) |

Таблица 2.6 – Типы оплаты

|  |  |  |  |
| --- | --- | --- | --- |
| charge\_types | id | Код записи | serial |
| name | Название типа оплаты | character varying(250) |

Таблица 2.7 – Счета

|  |  |  |  |
| --- | --- | --- | --- |
| bills | id | Код записи | serial |
| id\_client | Код клиента | integer |
| balance | Баланс | money |

Таблица 2.8 – Контракты

|  |  |  |  |
| --- | --- | --- | --- |
| contracts | id | Код записи | serial |
| name | Наименование контракта | text |
| id\_client | Код клиента | integer |
| preparation\_date | Дата заключения контракта | date |
| start\_date | Дата вступления в силу контракта | date |
| id\_status | Код статуса контракта | integer |
| end\_date | Дата окончания действия контракта | date |

Таблица 2.9 – Тарифы

|  |  |  |  |
| --- | --- | --- | --- |
| tariffs | id | Код записи | serial |
| name | Название тарифа | text |
| id\_service | Код услуги | integer |
| id\_charge\_type | Код типа оплаты | integer |
| cost | Стоимость | money |

Таблица 2.10 – Типы транзакций

|  |  |  |  |
| --- | --- | --- | --- |
| transaction\_types | id | Код записи | serial |
| name | Название типа транзакций | character varying(250) |

Таблица 2.11 – Части контрактов

|  |  |  |  |
| --- | --- | --- | --- |
| contract\_parts | id | Код записи | serial |
| id\_contract | Код контракта | integer |
| id\_tariff | Код тарифа | integer |

Таблица 2.12 – Финансовые операции

|  |  |  |  |
| --- | --- | --- | --- |
| financial\_operations | id | Код записи | serial |
| id\_bill | Код счёта | integer |
| id\_transaction\_type | Код типа транзакции | integer |
| sum | Сумма | money |
| operation\_date | Дата операции | date |
| id\_contract\_part | Код части контракта | integer |
| comment | Комментарий | text |

Разработанная модель будет применена при фактическом создании структуры базы данных при помощи СУБД PostgreSQL.

В результате полученная база данных будет включать в себя 12 взаимосвязанных таблиц, описывающих основные сущности данных, необходимых для поддержки работы сервисной части компаний мобильных операторов.

# 3 РАЗРАБОТКА ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ

## 3.1 Создание проекта и базы данных

Для разработки приложения был выбран язык объекто-ориентированного программирования Java. Это обеспечит гибкость процесса разработки, дальнейшей поддержки и сопровождения данного программного продукта. Выбранная среда разработки – IntelliJ IDEA Community.

Выбранная графическая оболочка pgAdmin4 для PostgreSQL 12 позволяет создавать базы данных и таблицы с помощью встроенного конструктора с возможностью дальнейшего редактирования SQL-скриптов, а также позволяет прописывать вручную код запросов на языке SQL и применять его к базе данных [3]. На рисунке 3.1 представлен скрипт создания базы данных для курсового проекта.

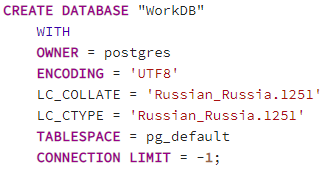


Рисунок 3.1 – Скрипт создания базы данных

## 3.2 Создание таблиц

С помощью конструктора были созданы таблицы. На рисунках 3.2-3.13 представлены скриншоты кода, отвечабщего за создание и настройку таблиц в базе данных.

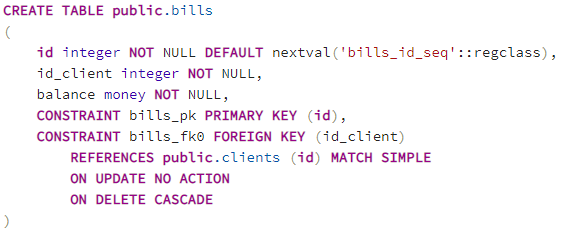


Рисунок 3.2 – Скрипт создания таблицы «Счета»

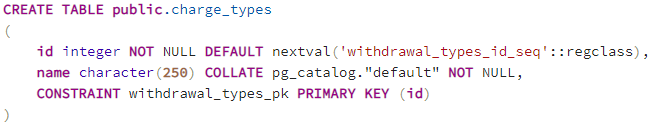


Рисунок 3.3 – Скрипт создания таблицы «Типы оплаты»

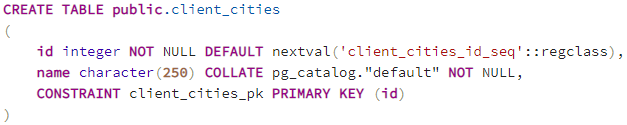


Рисунок 3.4 – Скрипт создания таблицы «Города клиентов»

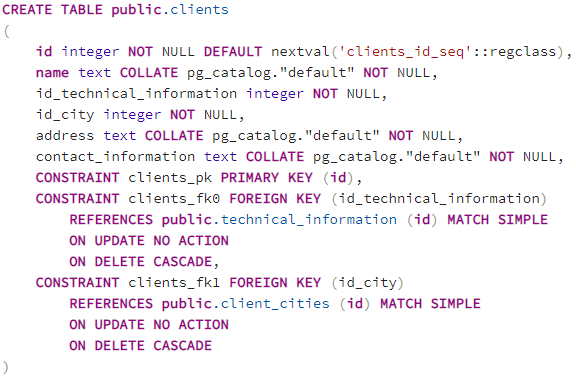


Рисунок 3.5 – Скрипт создания таблицы «Клиенты»

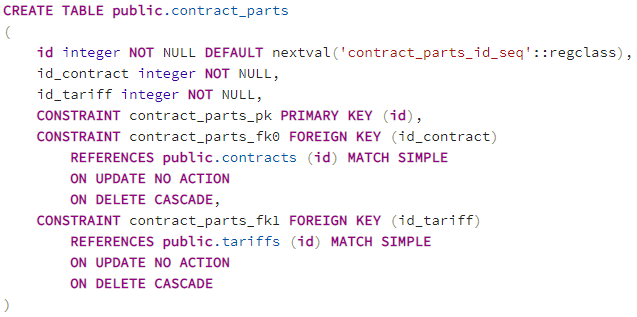


Рисунок 3.6 – Скрипт создания таблицы «Части контрактов»

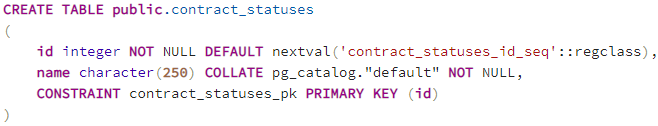


Рисунок 3.7 – Скрипт создания таблицы «Статусы контрактов»

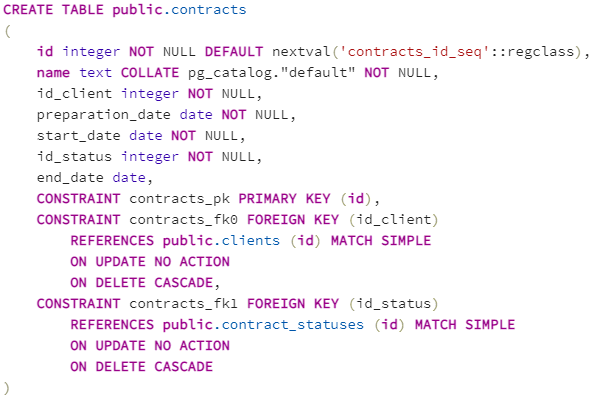


Рисунок 3.8 – Скрипт создания таблицы «Контракты»

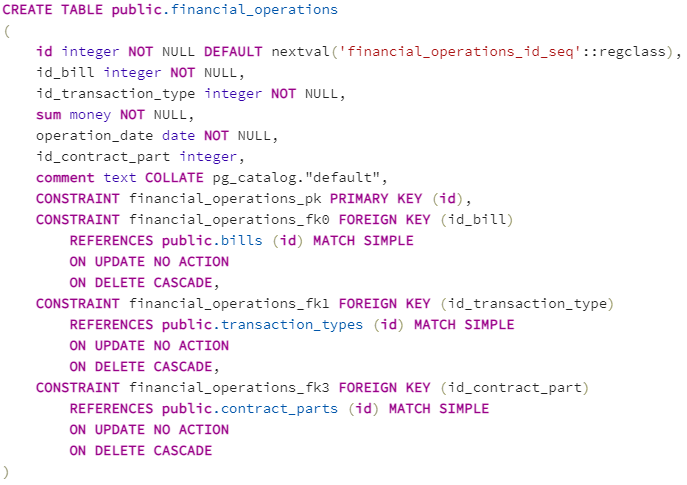


Рисунок 3.9 – Скрипт создания таблицы «Финансовые операции»

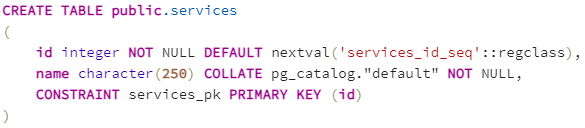


Рисунок 3.10 – Скрипт создания таблицы «Услуги»

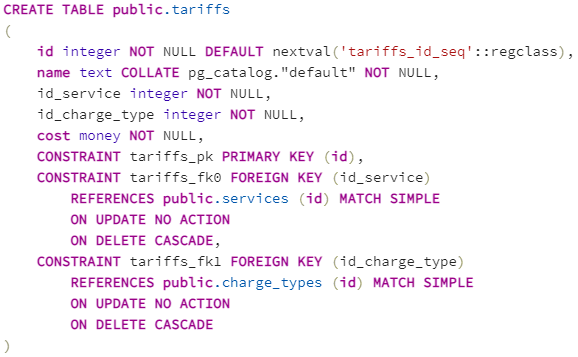


Рисунок 3.11 – Скрипт создания таблицы «Тарифы»

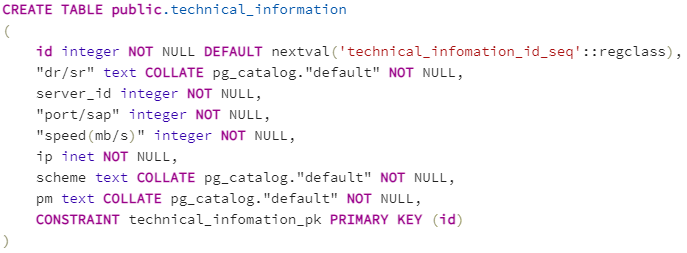


Рисунок 3.12 – Скрипт создания таблицы «Техническая информация»

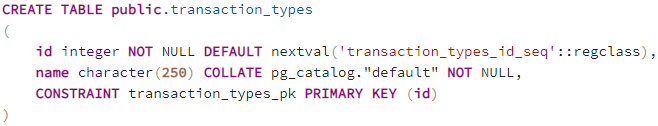


Рисунок 3.13 – Скрипт создания таблицы «Типы транзакций»

## 3.3 Схема базы данных (поддержка целостности данных)

Для поддержания ссылочной целостности на все внешние ключи в базе было наложено правило CASCADE. Это значит, что при удалении и обновлении родительской записи будут удалены или обновлены и все её дочерние элементы. Это позволяет пользователю автоматизировать некоторые процессы.

## 3.4 Разработка внешних представлений данных - меню, форм, запросов, отчетов

Для разработки эффективной модели пользовательского интерфейса сначала следует определить, какие задачи должен выполнять этот интерфейс. Для наглядного представления функциональных требований к интерфейсу была составлена диаграмма прецедентов (Use-case) (рисунок 3.14) [4].

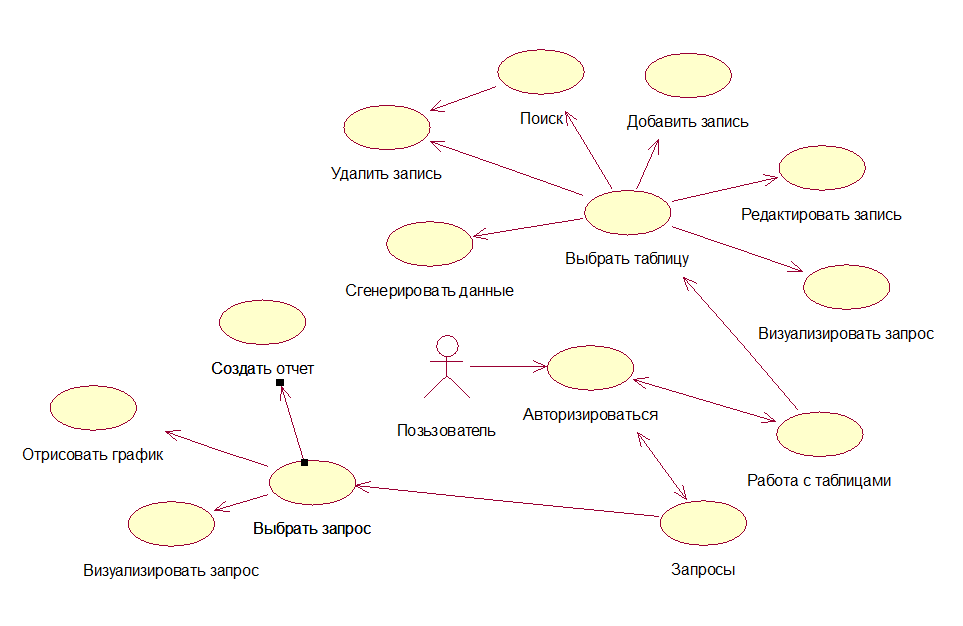


Рисунок 3.14 – Диаграмма прецедентов для разрабатываемой системы

При запуске программы, следуя требованиям проекта, открывается окно авторизации. Для продолжения работы пользователь должен ввести заранее определенные логин и пароль (рисунок 3.15).

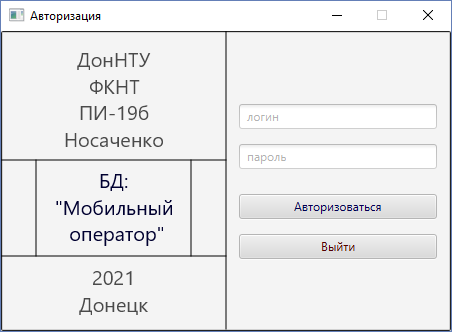


Рисунок 3.15 – Скриншот формы авторизации

Далее перед пользователем открывается главный экран с приветсвенным окном и панелью меню, в которм он может выбрать работу с таблицами или запросами. (рисунок 3.16).

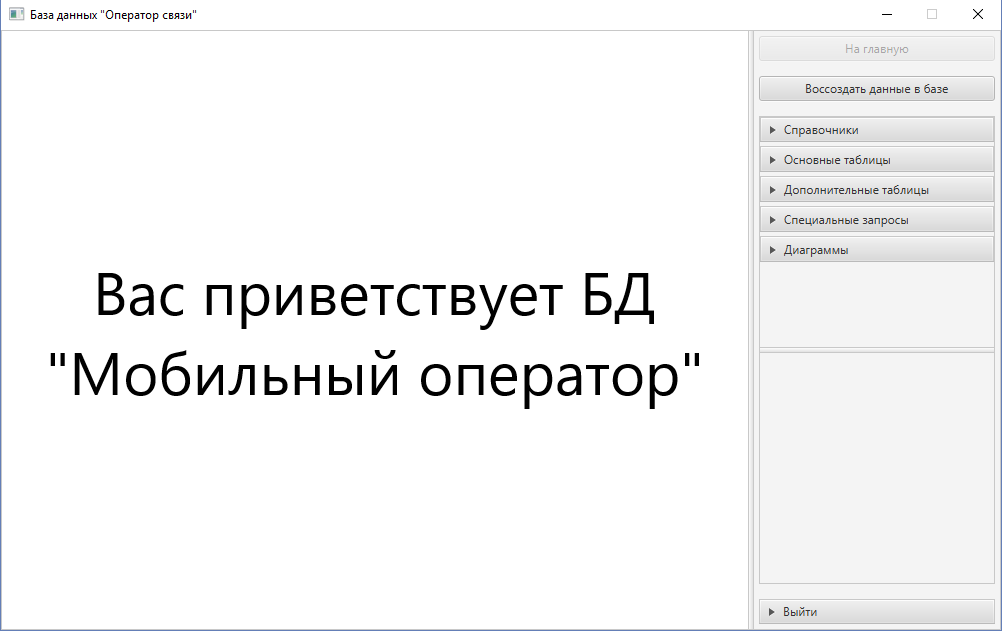


Рисунок 3.16 – Скриншот формы «Главная»

После нажатия на кнопку «Справочники», «Основные таблицы», «Дополнительные таблицы», «Специальные запросы» или «Диаграммы», откроется соответсвующее выпадающее меню переходов (рисунок 3.17).

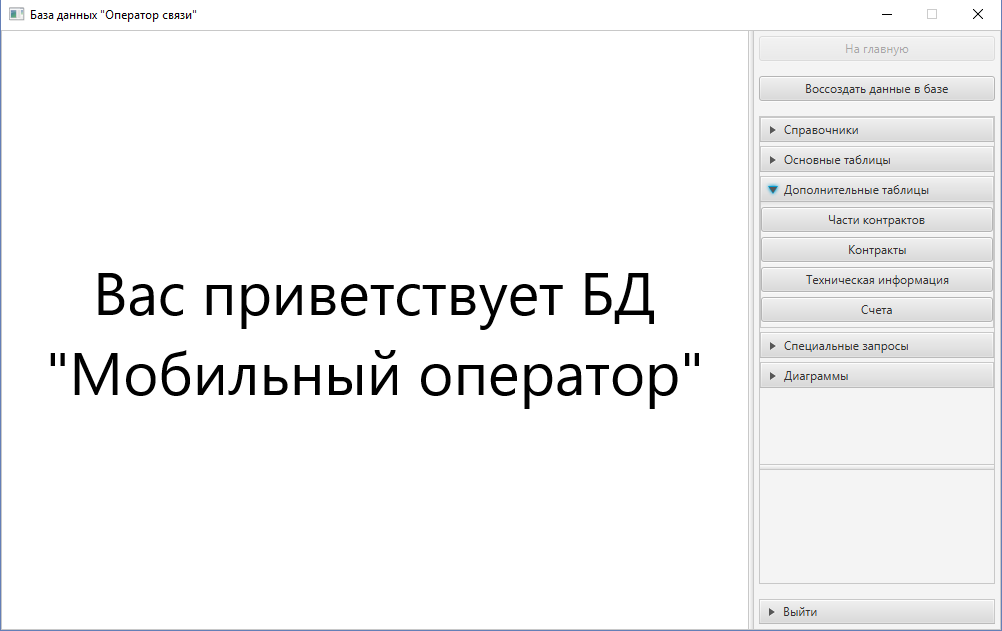


Рисунок 3.17 – Скриншот изменения меню

Когда пользователь выберет в выпадающем списке таблицу и нажмёт на неё, то сможет просмотреть все записи данной таблицы. (рисунок 3.18).

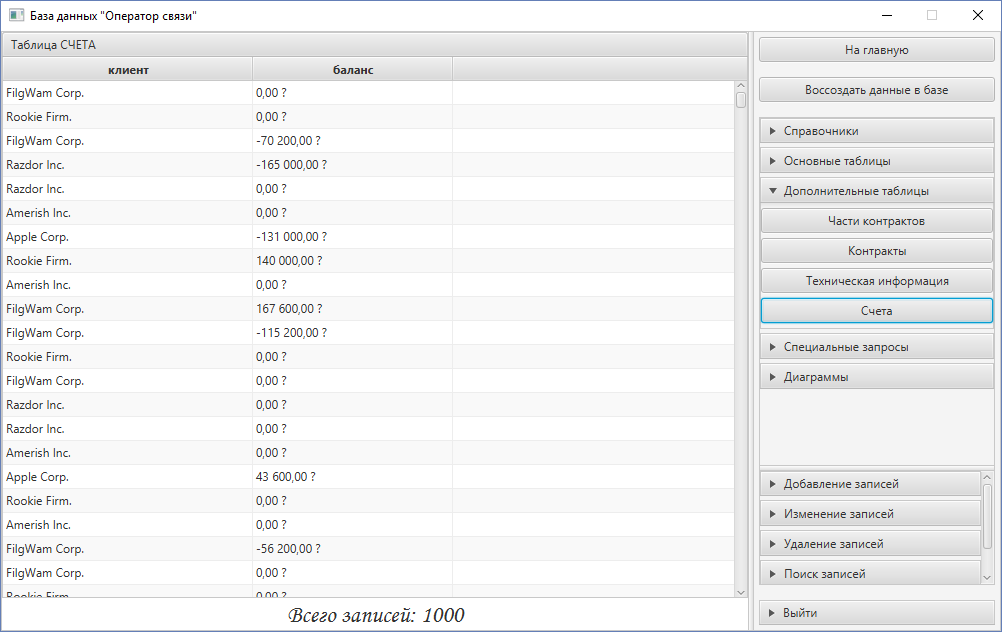


Рисунок 3.18 – Скриншот отображения таблицы «Счета»

Отображение таблицы активирует подменю, включающее кнопки добавления, изменения, удаления, поиска записей и изменения представления данных. Нажатие кнопки «Добавить» или «Редактирование» откроет форму добавления/редактирования записей (рисунок 3.19).

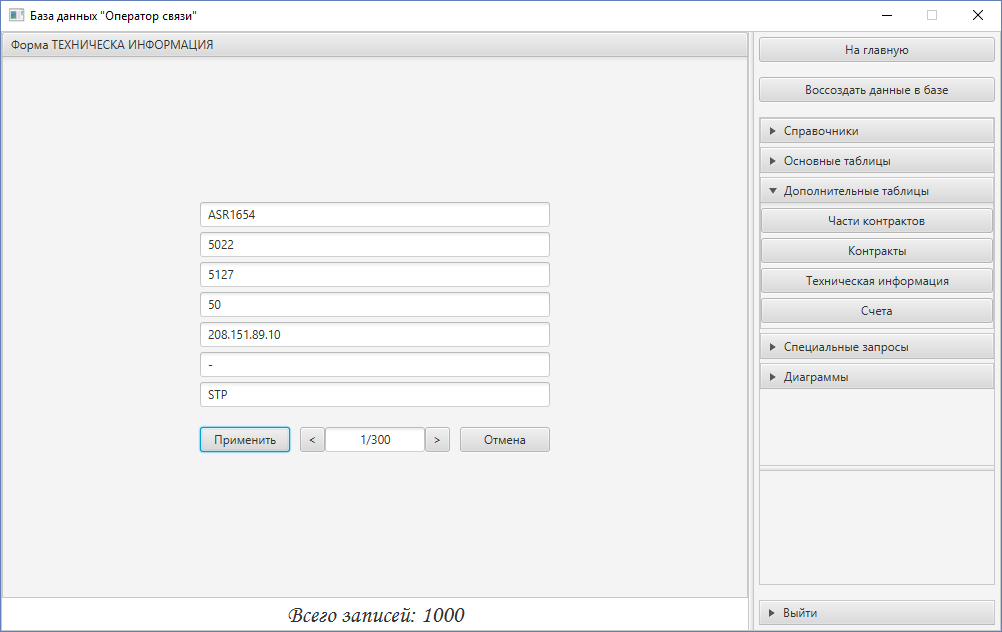


Рисунок 3.19 – Редактирование записей таблицы «Техническая информация»

При нажатии на кнопку «На главную» в меню, пользователь вернётся к главному экрану, при этом подменю скрывается.

Выбор одной из 3-х кнопок диаграмм откроет соответствующую форму с визуализацией данных в отдельном окне (рисунки 3.20 – 3.22).

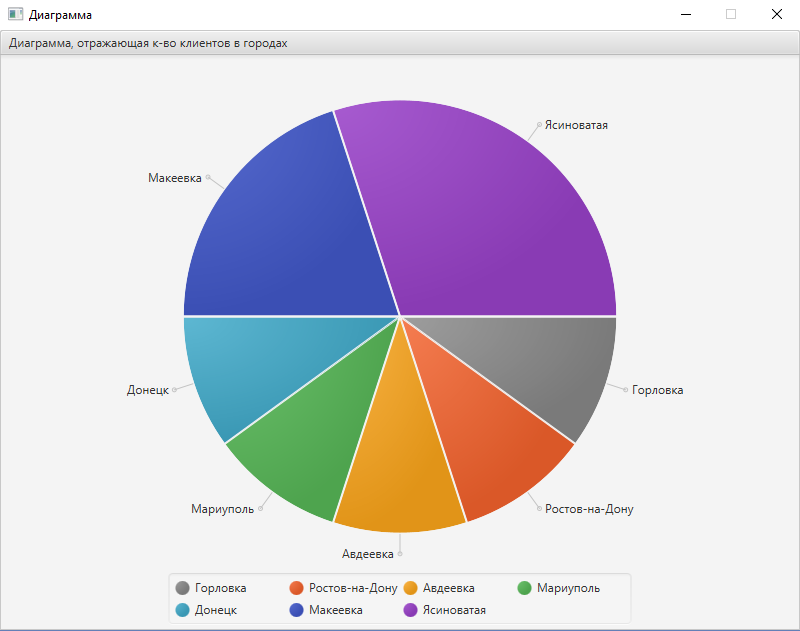


Рисунок 3.20 – Одномерная круговая диаграмма

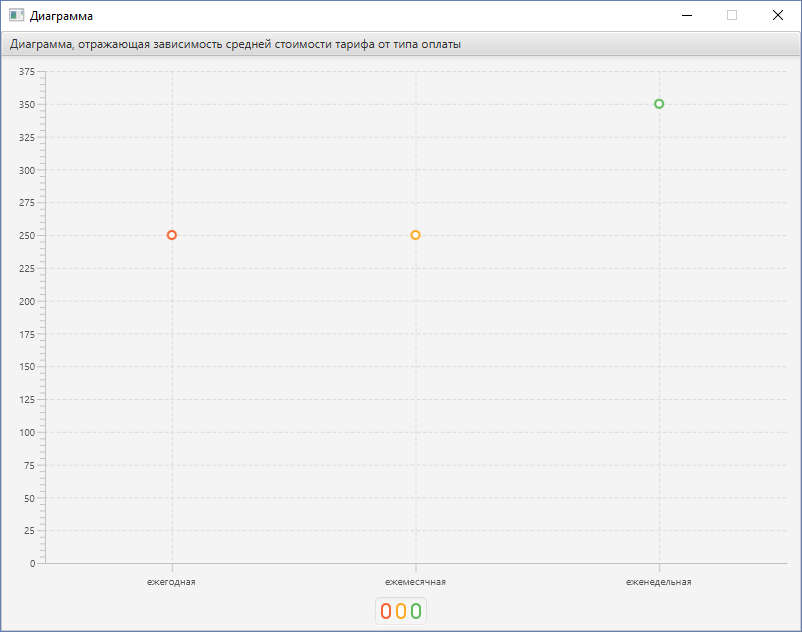


Рисунок 3.21 – Двумерная линейная диаграмма

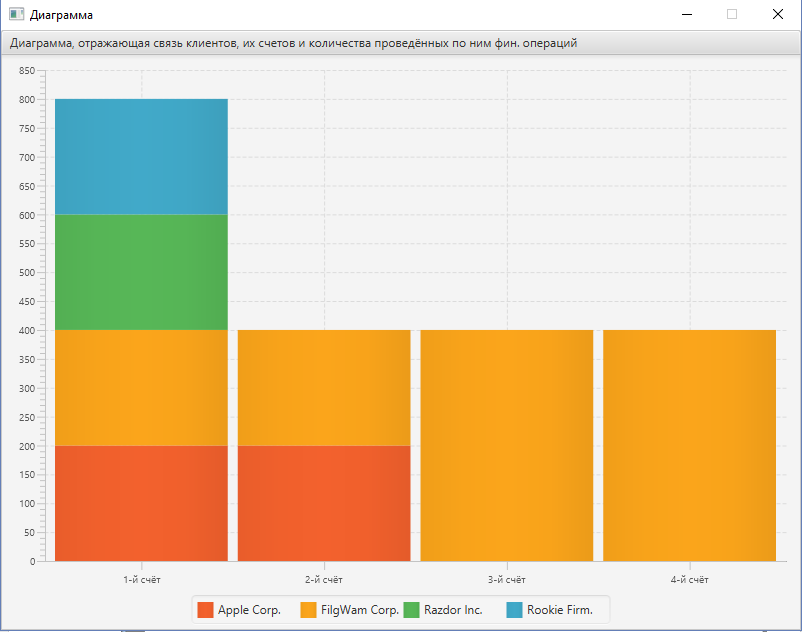


Рисунок 3.22 – Трёхмерная столбчатая диаграмма с дополнением

Когда пользователь выберет нужный запрос и нажмёт на соответствующую кнопку меню, то в новом окне откроется таблица с данными от выполненного запроса. (рисунок 3.23)

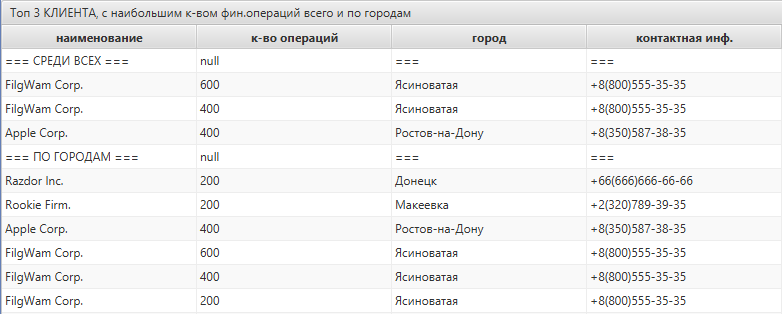


Рисунок 3.23 – Результат выполнения первого специального запроса

Пользователь может сохранить запрос в виде отчёта в формате .html или таблицах Excel. (рисунки 3.24, 3.25)



Рисунок 3.24 – Результат выполнения первого специального запроса

в форме .html-отчёта

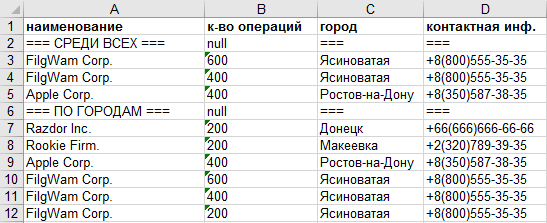


Рисунок 3.25 – Результат выполнения первого специального запроса

в форме Excel-таблицы.

Подробней об алгоритмах составления отчета, создания веб-представления таблицы и построения всех типов диаграмм написано в приложении Д.

## 3.5 Разработка процедур обработки данных (модулей)

Многие события отправляют на сервер запрос и читают данные о результатах выполнения этого запроса, выводя полученную информацию на экран [5]. В требованиях к курсовому проекту указана необходимость реализовать несколько выделенных запросов на выборку.

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

1) отбор по внешнему ключу – информация о тарифах по оказываемым услугам (рисунки 3.26-3.27).

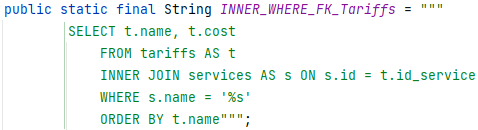


Рисунок 3.26 – Симметричное внутреннее соединение (запрос 1)

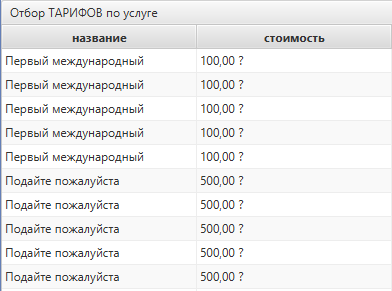


Рисунок 3.27 – Пример выполнения запроса на симметричное

внутреннее объединение (запрос 1)

2) отбор по внешнему ключу – информация о контрактах по клиентам (рисунки 3.28 – 3.29).

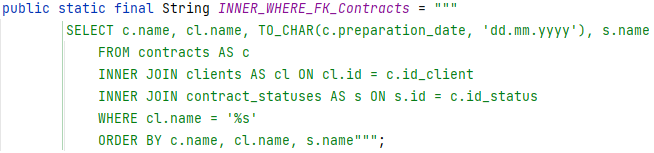


Рисунок 3.28 – Симметричное внутреннее соединение (запрос 2)

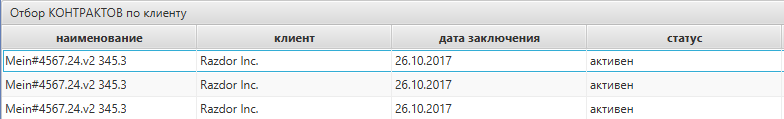


Рисунок 3.29 – Пример выполнения запроса на симметричное

внутреннее объединение (запрос 2)

3) отбор по дате – информация о финансовых операциях по каждому типу транзакций, проведённых в указанны день (рисунки 3.30-3.31).

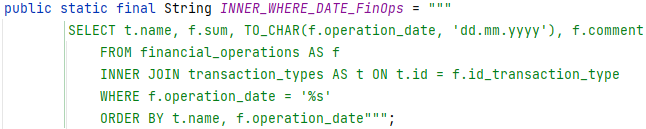


Рисунок 3.30 – Симметричное внутреннее соединение (запрос 3)

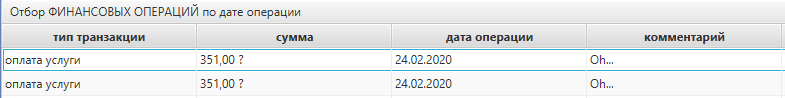


Рисунок 3.31 – Пример выполнения запроса на симметричное

внутреннее соединение (запрос 3)

4) отбор по дате – информация о контрактах по статусам, действие которых прекращается в указанный год (рисунки 3.32 – 3.33).

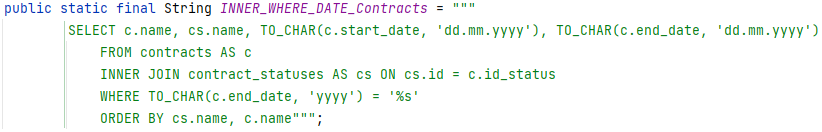


Рисунок 3.32 – Симметричное внутреннее соединение (запрос 4)

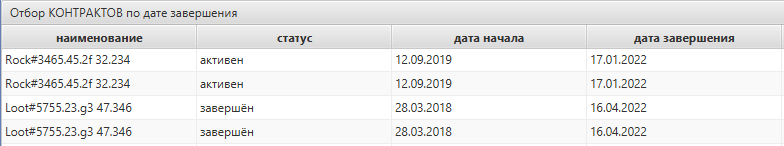


Рисунок 3.33 – Пример выполнения запроса на симметричное

внутреннее соединение (запрос 4)

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

1) Информация о клиентах в различных городах (рисунки 3.34-3.35).

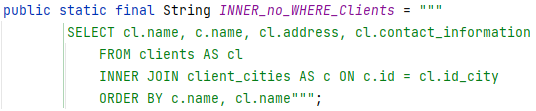


Рисунок 3.34 – Симметричное внутреннее соединение без условия (запрос 1)

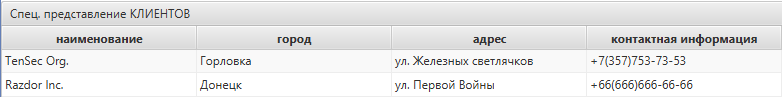


Рисунок 3.35 – Пример выполнения запроса на симметричное

внутреннее соединение без условия (запрос 1)

2) Информация о тарифах с различными типами оплаты (рисунки 3.36 - 3.37).

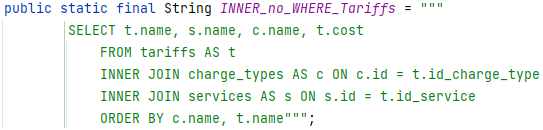


Рисунок 3.36 – Симметричное внутреннее соединение без условия (запрос 2)

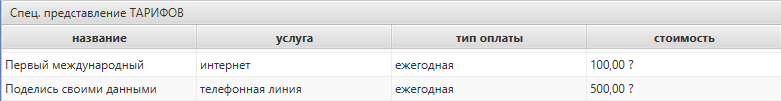


Рисунок 3.37 – Пример выполнения запроса на симметричное

внутреннее соединение без условия (запрос 2)

3) Информация о частях контрактов с различными контрактами и тарифами (рисунки 3.38-3.39).

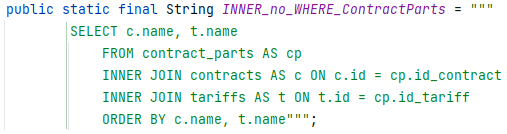


Рисунок 3.38 – Симметричное внутреннее соединение без условия (запрос 3)

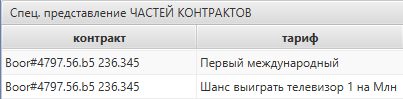


Рисунок 3.39 – Пример выполнения запроса на симметричное

внутреннее соединение без условия (запрос 3)

### 3.5.3 Левое внешнее соединение

1) Информация о финансовых операциях без частей контрактов (рисунки 3.40-3.41).

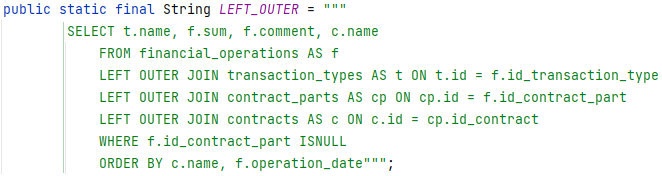


Рисунок 3.40 – Левое внешнее соединение

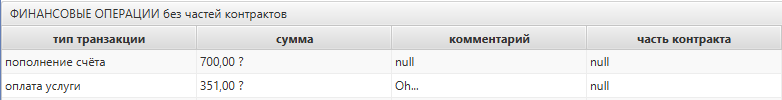


Рисунок 3.41 – Пример выполнения запроса на левое внешнее соединение

### 3.5.4 Правое внешнее соединение

1) Краткая информация о клиентах. Дополнительно вывести счета указанного клиента (рисунки 3.42-3.43).

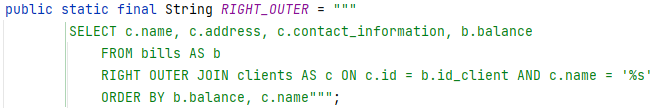


Рисунок 3.42 – Правое внешнее соединение

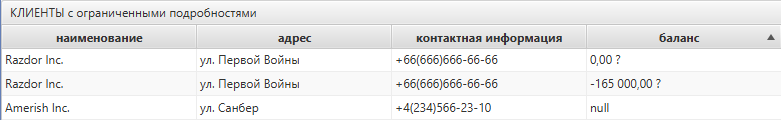


Рисунок 3.43 – Пример выполнения запроса на правое внешнее соединение

### 3.5.5 Запрос на запросе по принципу левого соединения

1) Информация о типах транзакций и денежных суммах проведённых финансовых операций (рисунки 3.44-3.45).

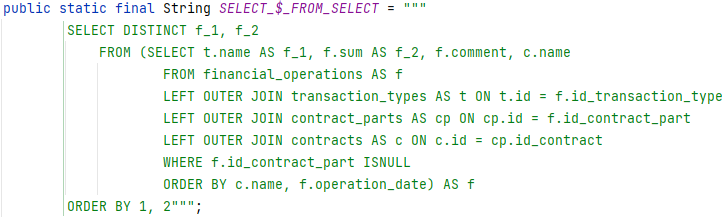


Рисунок 3.44 – Запрос на запросе по принципу левого соединения

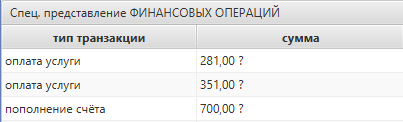


Рисунок 3.45 – Пример выполнения запроса на запросе

по принципу левого соединения

### 3.5.6 Итоговый запрос без условия

1) Информация о количестве клиентов с разной скоростью интернета (рисунки 3.46-3.47).

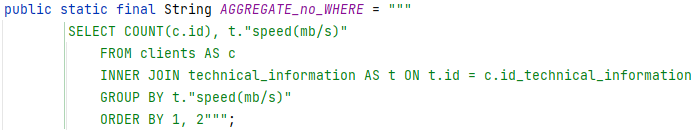


Рисунок 3.46 – Итоговый запрос без условия

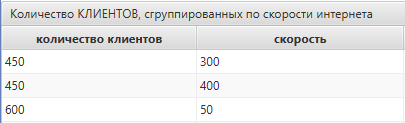


Рисунок 3.47 – Пример выполнения итогового запроса без условия

### 3.5.7 Итоговый запрос с условием на данные

1) Информация о средней стоимости тарифов с указанной услугой (рисунки 3.48-3.49).

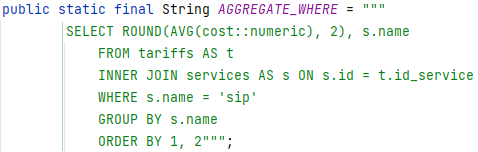


Рисунок 3.48 – Итоговый запрос с условием на данные

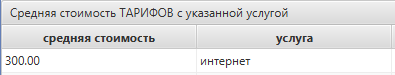


Рисунок 3.49 – Пример выполнения итогового запроса с условием на данные

### 3.5.8 Итоговый запрос с условием на группы

1) Количество финансовых операций с проходящей суммой более 500 у.е. (рисунки 3.50-3.51).

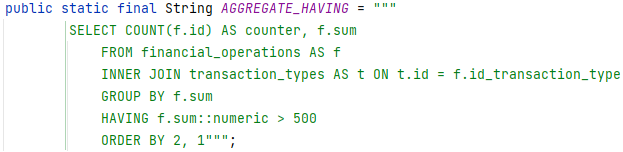


Рисунок 3.50 – Итоговый запрос с условием на группы

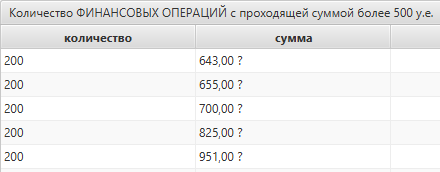


Рисунок 3.51 – Пример выполнения итогового запроса с условием на группы

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

1) Информация об отрицательных счетах указанного клиента (рисунки 3.52-3.53).

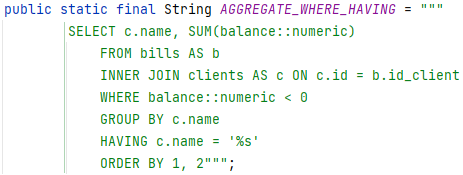


Рисунок 3.52 – Итоговый запрос с условием на данные и на группы

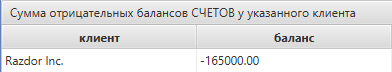


Рисунок 3.53 – Пример выполнения итогового запроса

с условием на данные и на группы

### 3.5.10 Запрос на запросе по принципу итогового запроса

1) Информация о количестве клиентов с различной скоростью интернетоборудования (рисунки 3.54-3.55).

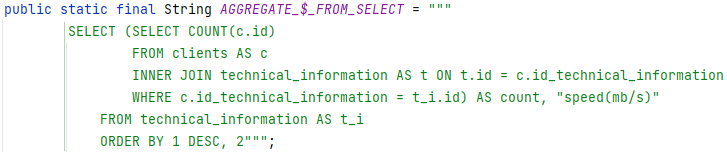


Рисунок 3.54 – Запрос на запросе по принципу итогового запроса

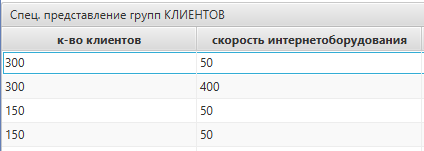


Рисунок 3.55 – Пример выполнения запроса на запросе

по принципу итогового запроса

### 3.5.11 Запрос с подзапросом

1) Информация о счетах клиентов, наименования которых начинаются на указанный ряд символов (рисунки 3.56-3.57).

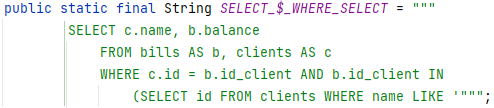


Рисунок 3.56 – Запрос с подзапросом

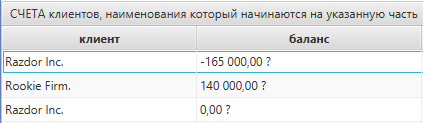


Рисунок 3.57 – Пример выполнения запроса с подзапросом

### 3.5.12 Запросы из курсового проекта

1) Вывести трёх клиентов с наибольшим количеством финансовых операций среди всех и общий список по городам (рисунки 3.58-3.59).

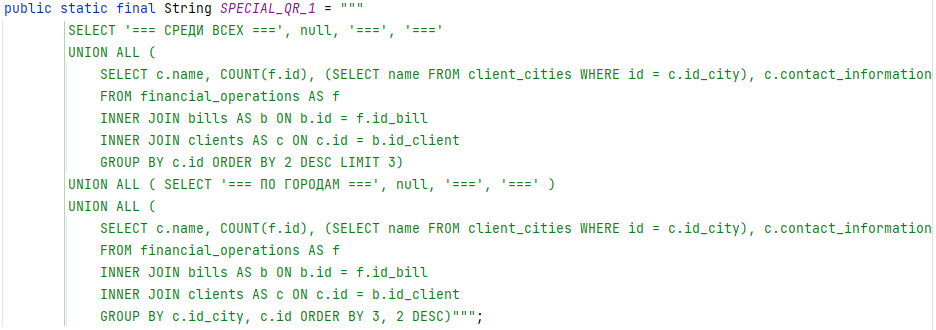


Рисунок 3.58 – Первый специальный запрос

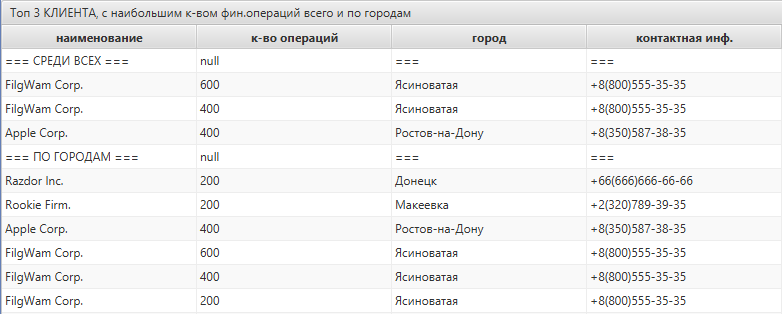


Рисунок 3.59 – Пример выполнения первого специального запроса.

2) Информация о среднем количестве финансовых операций для клиентов и для городов (рисунки 3.60-3.61).

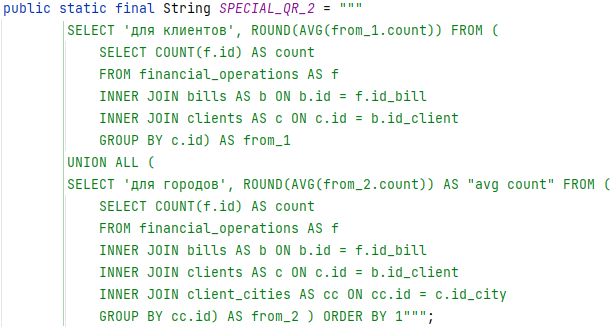


Рисунок 3.60 – Второй специальный запрос

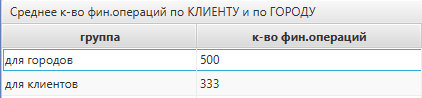


Рисунок 3.61 – Пример выполнения второго специального запроса

3) Информация о количестве клиентов, совершивших финансовые операции за указанный период, и сумму денег, прошедших через эти операции (рисунки 3.62-3.63).

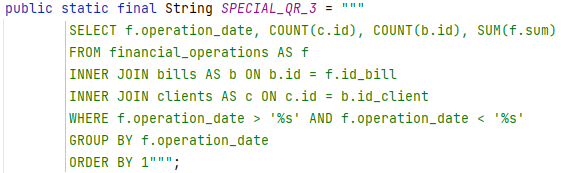


Рисунок 3.62 – Третий специальный запрос

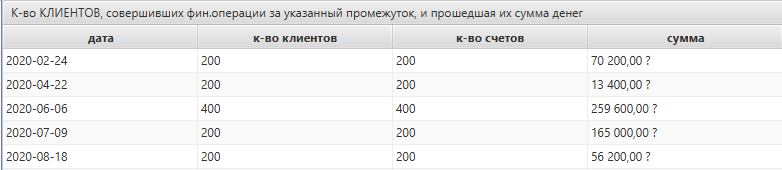


Рисунок 3.63 – Пример выполнения третьего специального запроса

## 3.6 Тестирование и отладка БД

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

Для корректной работы генератора необходимо, чтобы таблицы-справочники базы данных были заполнены.

## 3.7 Разработка эксплуатационной документации БД

По завершению разработки и отладки программного продукта было написано краткое руководство пользователя, в котором описаны основные положения и рекомендации по использованию программы, а также постледовательности выполнения действия для получения различных результатов применения разработанного программного продукта. Руководство пользователя приведено в приложении Б.

# ЗАКЛЮЧЕНИЕ

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

В результате выполнения курсовой работы было проведено инфологическое, даталогическое и физическое проектирование модели базы данных по заданному варианту (симуляция работы контроля сервисных данных мобильного оператора), был создан проект базы данных, созданы и заполнены соответствующим образом все необходимые таблицы (таблицы-справочники, вспомогательные таблицы и основные таблицы). Также были разработаны предоставленные требованиями к курсовой работе сложные запросы на выборку. Результатами некоторых запросов стали диаграммы статистических показателей. В программе реализована возможность сохранения результатов запросов в формат html и Excel.

Было разработано клиентское приложение на языке программирования Java.

Разработанная система может быть использована для обслуживания реальных мобильных операторов. Для пользователей разработанной системы было написано соответствующее руководство использования.

# СПИСОК ЛИТЕРАТУРЫ

1. Ганс-Юрген Шениг – PostgreSQL 11. Мастерство разработки – Москва: ДМК Пресс, 2019. – 353 с.

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

3. Джуба С., Волков А. – Изучаем PostgreSQL 10 – Москва: ДМК Пресс, 2019. – 401 с.

4. Скотт К. UML. Основные концепции. – М.: Вильямс, 2002. – 144 с.

5. Бьюли А. Изучаем SQL. – М.: Вильямс, 2010. – 100 с.

# ПРИЛОЖЕНИЕ А

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

# ПРИЛОЖЕНИЕ Б

РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Для корректной работы программы необходима персональный компьютер под управлением операционной системы Windows 7 или выше.

Для открытия программы необходимо запустить исполняемый файл CourseW.exe. При открытии программы на форме авторизации необходимо будет ввести правильный логин и пароль. В случае верного ввода логина и пароля форма авторизации закроется, вместо нее откроется меню, на котором вы выбираете работать вам с таблицами или запросами. После выбора вы переходите на форму в зависимости от вашего выбора. Если вы нажали кнопку «Работа с таблицами», то перейдёте на форму, которая позволяет добавлять изменять и удалять поля из таблиц.

Справа в верхнем углу находится выпадающий список со всеми таблицами. Ниже находятся кнопки «Добавить запись», «Изменить запись», «Удалить запись», «Поиск записей». Прежде чем удалить выбранные вами записи программа повторно спросит вашего согласия на удаление данных. Для поиска выбираете поле и пишите текст, который должен быть включён в искомые записи, после чего нажимаете кнопку «Применить». Над списком таблиц надодится кнопка «Воссоздать данные в базе», при нажатии на которую будут сгенерированы тестовые данные, программа повторно спросит вашего согласия на удаление существующих данных в таблице. Если пользователь нажмет на кнопку «На главную», то текущее отображение таблицы будет скрыто.

Все запросы, также как таблицы и диаграммы, находятся в соответствующем им выпадающем меню. Нажатие на элемент такого меню отображает необходимые пользователю данные.

# ПРИЛОЖЕНИЕ В

ИСХОДНЫЙ КОД ПРОГРАММЫ БАЗЫ ДАННЫХ

// AppCore.java

package work;

import java.util.\*;

import java.sql.Connection;

import java.sql.Statement;

import java.sql.ResultSet;

import java.sql.DriverManager;

import javafx.application.Application;

import javafx.application.Platform;

import javafx.fxml.FXMLLoader;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.Alert;

import javafx.stage.Stage;

import javafx.stage.StageStyle;

public class AppCore extends Application

{

public static Main\_Controller mainController;

public static Authorization\_Controller authorizationController;

public static Queries\_Controller queriesController;

public static Diagrams\_Controller diagramsController;

public static Dictionary<String, Stage> stages = new Hashtable<>();

public static String login\_password = "";

public static Connection connection;

public static Statement statement;

public static ResultSet resultSet;

//Functions

public static void showAlert(String alertMessage)

{

Alert alert = new Alert(Alert.AlertType.INFORMATION);

alert.initStyle(StageStyle.UNDECORATED);

alert.setGraphic(null);

alert.setHeaderText(null);

alert.setContentText(alertMessage);

alert.show();

Thread thread = new Thread(() -> {

try

{

Thread.sleep(3000);

if (alert.isShowing())

Platform.runLater(alert::close);

} catch (Exception exp) { exp.printStackTrace(); }

});

thread.setDaemon(true); thread.start();

}

public static void setConnection()

{

try

{

Class.forName("org.postgresql.Driver");

connection = DriverManager.

getConnection("jdbc:postgresql://localhost:5432/PracticeDB", "A.T.A.N", "123");

connection.setAutoCommit(false);

System.out.println("-- DB opening successful");

}

catch (Exception e) { e.printStackTrace(); System.exit(0); }

}

public static void createEndDaemon()

{

Thread thread = new Thread(() -> {

while(true)

{

if (Thread.currentThread().isInterrupted())

{

try { resultSet.close(); statement.close(); connection.commit(); connection.close(); }

catch (Exception e) { e.printStackTrace(); System.exit(0); }

}

}

});

thread.setDaemon(true); thread.start();

}

@Override

public void start(Stage mainForm) throws Exception

{

setConnection();

createEndDaemon();

// Main form load

Parent root = FXMLLoader.load(getClass().getResource("Main\_Form.fxml"));

mainForm.setTitle("База данных \"Оператор связи\"");

mainForm.setScene(new Scene(root, 1000, 600));

mainForm.setResizable(false);

stages.put("Main", mainForm);

// Queries form load

root = FXMLLoader.load(getClass().getResource("Queries\_Form.fxml"));

Stage queriesForm = new Stage();

queriesForm.setTitle("Результаты специализированного запроса");

queriesForm.setScene(new Scene(root, 800, 600));

queriesForm.setResizable(false);

stages.put("Queries", queriesForm);

// Queries form load

root = FXMLLoader.load(getClass().getResource("Diagrams\_Form.fxml"));

Stage diagramsForm = new Stage();

diagramsForm.setTitle("Диаграмма");

diagramsForm.setScene(new Scene(root, 800, 600));

diagramsForm.setResizable(false);

stages.put("Diagrams", diagramsForm);

// Authorization form load

root = FXMLLoader.load(getClass().getResource("Authorization\_Form.fxml"));

Stage authorizationForm = new Stage();

authorizationForm.setTitle("Авторизация");

authorizationForm.setScene(new Scene(root, 450, 300));

authorizationForm.setResizable(false);

stages.put("Authorization", authorizationForm);

stages.get("Authorization").show();

}

public static void main(String[] args) { launch(args); }

}

// Part of Main\_Contraller.java

package work;

import DBTables.\*;

import java.net.URL;

import java.util.\*;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.fxml.FXML;

import javafx.application.Platform;

import javafx.scene.layout.\*;

import javafx.stage.StageStyle;

import javafx.scene.control.\*;

import javafx.scene.control.cell.PropertyValueFactory;

public class Main\_Controller

{

@FXML

private ResourceBundle resources;

@FXML

private URL location;

// region CODE segment

// VARIABLES

private String curTable;

private ArrayList<ArrayList<String>> records;

private int recordsCount;

private String actionType;

private int curRecord;

private boolean curRecord\_changed = false;

// FUNCTIONS

private void show\_Base\_Easier(String query, String boxTitle)

{

ObservableList<FullTableRow> rows = FXCollections.observableArrayList();

try {

AppCore.statement = AppCore.connection.createStatement();

AppCore.resultSet = AppCore.statement.executeQuery(query);

recordsCount = 0;

while (AppCore.resultSet.next()) {

FullTableRow row = new FullTableRow();

row.setField\_01(AppCore.resultSet.getString(1).trim());

rows.add(row);

recordsCount++;

}

recordsCounter\_FIELD.setText("Всего записей: " + recordsCount);

} catch (Exception e) {

e.printStackTrace();

}

try {

AppCore.resultSet.close();

AppCore.statement.close();

AppCore.connection.commit();

} catch (Exception e) {

e.printStackTrace();

}

name\_CommonCatalogT\_COLUMN.setCellValueFactory(new PropertyValueFactory<>("field\_01"));

commonCatalog\_TABLE.setItems(rows);

toMain\_BTN.setDisable(false);

subMenu\_BOX.setVisible(true);

show\_BOX.toFront();

tables\_BOX.toFront();

commonCatalogT\_BOX.toFront();

commonCatalogT\_BOX.setText(boxTitle);

}

private void show\_CommonCatalog(String tableName)

{

showType\_BOX.setDisable(true);

fieldsChange\_BTN.setDisable(true);

fieldsDelete\_BTN.setDisable(true);

choseChange\_BTN.setDisable(false);

choseDelete\_BTN.setDisable(false);

switch (tableName)

{

case "contract\_parts" -> {

ObservableList<FullTableRow> rows = FXCollections.observableArrayList();

try {

AppCore.statement = AppCore.connection.createStatement();

AppCore.resultSet = AppCore.statement.executeQuery(QrsCommonCatalogs.SELECT\_CONTRACT\_PARTS);

recordsCount = 0;

while (AppCore.resultSet.next()) {

FullTableRow row = new FullTableRow();

row.setField\_01(AppCore.resultSet.getString(1).trim());

row.setField\_02(AppCore.resultSet.getString(2).trim());

rows.add(row);

recordsCount++;

}

recordsCounter\_FIELD.setText("Всего записей: " + recordsCount);

} catch (Exception e) {

e.printStackTrace();

}

try {

AppCore.resultSet.close();

AppCore.statement.close();

AppCore.connection.commit();

} catch (Exception e) {

e.printStackTrace();

}

contract\_ContractPartsT\_COLUMN.setCellValueFactory(new PropertyValueFactory<>("field\_01"));

tariff\_ContractPartsT\_COLUMN.setCellValueFactory(new PropertyValueFactory<>("field\_02"));

contractParts\_TABLE.setItems(rows);

toMain\_BTN.setDisable(false);

subMenu\_BOX.setVisible(true);

show\_BOX.toFront();

tables\_BOX.toFront();

contractPartsT\_BOX.toFront();

}

case "bills" -> {

ObservableList<FullTableRow> rows = FXCollections.observableArrayList();

try {

AppCore.statement = AppCore.connection.createStatement();

AppCore.resultSet = AppCore.statement.executeQuery(QrsCommonCatalogs.SELECT\_BILLS);

recordsCount = 0;

while (AppCore.resultSet.next()) {

FullTableRow row = new FullTableRow();

row.setField\_01(AppCore.resultSet.getString(1).trim());

row.setField\_02(AppCore.resultSet.getString(2).trim());

rows.add(row);

recordsCount++;

}

recordsCounter\_FIELD.setText("Всего записей: " + recordsCount);

} catch (Exception e) {

e.printStackTrace();

}

try {

AppCore.resultSet.close();

AppCore.statement.close();

AppCore.connection.commit();

} catch (Exception e) {

e.printStackTrace();

}

client\_BillsT\_COLUMN.setCellValueFactory(new PropertyValueFactory<>("field\_01"));

balance\_BillsT\_COLUMN.setCellValueFactory(new PropertyValueFactory<>("field\_02"));

bills\_TABLE.setItems(rows);

toMain\_BTN.setDisable(false);

subMenu\_BOX.setVisible(true);

show\_BOX.toFront();

tables\_BOX.toFront();

billsT\_BOX.toFront();

}

case "tariffs" -> {

ObservableList<FullTableRow> rows = FXCollections.observableArrayList();

try {

AppCore.statement = AppCore.connection.createStatement();

AppCore.resultSet = AppCore.statement.executeQuery(QrsCommonCatalogs.SELECT\_TARIFFS);

recordsCount = 0;

while (AppCore.resultSet.next()) {

FullTableRow row = new FullTableRow();

row.setField\_01(AppCore.resultSet.getString(1).trim());

row.setField\_02(AppCore.resultSet.getString(2).trim());

row.setField\_03(AppCore.resultSet.getString(3).trim());

row.setField\_04(AppCore.resultSet.getString(4).trim());

rows.add(row);

recordsCount++;

}

recordsCounter\_FIELD.setText("Всего записей: " + recordsCount);

} catch (Exception e) {

e.printStackTrace();

}

try {

AppCore.resultSet.close();

AppCore.statement.close();

AppCore.connection.commit();

} catch (Exception e) {

e.printStackTrace();

}

name\_TariffsT\_COLUMN.setCellValueFactory(new PropertyValueFactory<>("field\_01"));

service\_TariffsT\_COLUMN.setCellValueFactory(new PropertyValueFactory<>("field\_02"));

chargeType\_TariffsT\_COLUMN.setCellValueFactory(new PropertyValueFactory<>("field\_03"));

cost\_TariffsT\_COLUMN.setCellValueFactory(new PropertyValueFactory<>("field\_04"));

tariffs\_TABLE.setItems(rows);

toMain\_BTN.setDisable(false);

subMenu\_BOX.setVisible(true);

show\_BOX.toFront();

tables\_BOX.toFront();

tariffsT\_BOX.toFront();

}

// easier

case "client\_cities" -> show\_Base\_Easier(QrsCommonCatalogs.SELECT\_CLIENT\_CITIES, "Таблица ГОРОДА КЛИЕНТОВ");

case "transaction\_types" -> show\_Base\_Easier(QrsCommonCatalogs.SELECT\_TRANSACTION\_TYPES, "Таблица ТИПЫ ТРАНЗАКЦИЙ");

case "services" -> show\_Base\_Easier(QrsCommonCatalogs.SELECT\_SERVICES, "Таблица УСЛУГИ");

case "charge\_types" -> show\_Base\_Easier(QrsCommonCatalogs.SELECT\_CHARGE\_TYPES, "Таблица ТИПЫ ОПЛАТЫ");

case "contract\_statuses" -> show\_Base\_Easier(QrsCommonCatalogs.SELECT\_CONTRACT\_STATUSES, "Таблица СТАТУСЫ КОНТРАКТОВ");

}

}

// region HARD

// Clients =====================================

private void use\_Record\_ClientsF(String alert\_msg, String action\_type)

{

AppCore.showAlert(alert\_msg);

name\_ClientsF\_FIELD.setEditable(true); name\_ClientsF\_FIELD.setText("");

techInf\_ClientsF\_CHOICE.setEditable(true);

city\_ClientsF\_CHOICE.setEditable(true);

address\_ClientsF\_FIELD.setEditable(true); address\_ClientsF\_FIELD.setText("");

contactInf\_ClientsF\_FIELD.setEditable(true); contactInf\_ClientsF\_FIELD.setText("");

{

try {

ObservableList<String> items = FXCollections.observableArrayList();

items.add(null);

AppCore.statement = AppCore.connection.createStatement();

AppCore.resultSet = AppCore.statement.executeQuery(QrsTechInf.SELECT\_4CLIENTS);

while (AppCore.resultSet.next())

{

items.add(

"[" + AppCore.resultSet.getString(1).trim() + "] " +

"Серв.ID: " + AppCore.resultSet.getString(2).trim() + "; " +

"Port/sap: " + AppCore.resultSet.getString(3).trim() + "; " +

"Скорость: " + AppCore.resultSet.getString(4).trim()

);

}

techInf\_ClientsF\_CHOICE.setItems(items);

techInf\_ClientsF\_CHOICE.setValue(null);

} catch (Exception e) { e.printStackTrace(); }

try { AppCore.resultSet.close(); AppCore.statement.close(); AppCore.connection.commit(); }

catch (Exception e) { e.printStackTrace(); }

try {

ObservableList<String> items = FXCollections.observableArrayList();

items.add(null);

AppCore.statement = AppCore.connection.createStatement();

AppCore.resultSet = AppCore.statement.executeQuery(QrsCommonCatalogs.SELECT\_CLIENT\_CITIES);

while (AppCore.resultSet.next())

{ items.add(AppCore.resultSet.getString(1).trim()); }

city\_ClientsF\_CHOICE.setItems(items);

city\_ClientsF\_CHOICE.setValue(null);

} catch (Exception e) { e.printStackTrace(); }

try { AppCore.resultSet.close(); AppCore.statement.close(); AppCore.connection.commit(); }

catch (Exception e) { e.printStackTrace(); }

}

use\_ClientsF\_BTN.setVisible(true);

prev\_ClientsF\_BTN.setVisible(false);

cur\_ClientsF\_FIELD.setVisible(false);

next\_ClientsF\_BTN.setVisible(false);

cancel\_ClientsF\_BTN.setVisible(true);

actionType = action\_type;

toMain\_BTN.setDisable(false);

subMenu\_BOX.setVisible(false);

show\_BOX.toFront();

forms\_BOX.toFront();

clientsF\_BOX.toFront();

}

// show

private void show\_Clients(String showType)

{

showType\_BOX.setDisable(false);

fieldsChange\_BTN.setDisable(false);

fieldsDelete\_BTN.setDisable(false);

choseChange\_BTN.setDisable(true);

choseDelete\_BTN.setDisable(true);

if (showType.equals("asTable"))

{

ObservableList<FullTableRow> rows = FXCollections.observableArrayList();

try

{

AppCore.statement = AppCore.connection.createStatement();

AppCore.resultSet = AppCore.statement.executeQuery(QrsClients.SELECT\_TABLE);

recordsCount = 0;

while (AppCore.resultSet.next())

{

FullTableRow row = new FullTableRow();

row.setField\_01(AppCore.resultSet.getString(1).trim());

row.setField\_02(AppCore.resultSet.getString(2).trim());

row.setField\_03(AppCore.resultSet.getString(3).trim());

row.setField\_04(AppCore.resultSet.getString(4).trim());

row.setField\_05(AppCore.resultSet.getString(5).trim());

row.setField\_06(AppCore.resultSet.getString(6).trim());

row.setField\_07(AppCore.resultSet.getString(7).trim());

rows.add(row);

recordsCount++;

}

recordsCounter\_FIELD.setText("Всего записей: " + recordsCount);

} catch (Exception e) { e.printStackTrace(); }

try { AppCore.resultSet.close(); AppCore.statement.close(); AppCore.connection.commit(); }

catch (Exception e) { e.printStackTrace(); }

name\_ClientsT\_COLUMN.setCellValueFactory(new PropertyValueFactory<> ("field\_01"));

serverID\_ClientsT\_COLUMN.setCellValueFactory(new PropertyValueFactory<> ("field\_02"));

portSAP\_ClientsT\_COLUMN.setCellValueFactory(new PropertyValueFactory<> ("field\_03"));

speed\_ClientsT\_COLUMN.setCellValueFactory(new PropertyValueFactory<> ("field\_04"));

city\_ClientsT\_COLUMN.setCellValueFactory(new PropertyValueFactory<> ("field\_05"));

address\_ClientsT\_COLUMN.setCellValueFactory(new PropertyValueFactory<> ("field\_06"));

contactInf\_ClientsT\_COLUMN.setCellValueFactory(new PropertyValueFactory<>("field\_07"));

clients\_TABLE.setItems(rows);

toMain\_BTN.setDisable(false);

subMenu\_BOX.setVisible(true);

tableShowType\_BTN.setSelected(true);

recordsShowType\_BTN.setSelected(false);

show\_BOX.toFront();

tables\_BOX.toFront();

clientsT\_BOX.toFront();

}

else if (showType.equals("byRecords"))

{

records = new ArrayList<>();

try

{

AppCore.statement = AppCore.connection.createStatement();

AppCore.resultSet = AppCore.statement.executeQuery(QrsClients.SELECT\_TABLE);

recordsCount = 0;

while (AppCore.resultSet.next())

{

ArrayList<String> row = new ArrayList<>();

row.add(AppCore.resultSet.getString(1).trim());

row.add(

"[" + AppCore.resultSet.getString(8).trim() + "] " +

"Серв.ID: " + AppCore.resultSet.getString(2).trim() + "; " +

"Port/sap: " + AppCore.resultSet.getString(3).trim() + "; " +

"Скорость: " + AppCore.resultSet.getString(4).trim()

);

row.add(AppCore.resultSet.getString(5).trim());

row.add(AppCore.resultSet.getString(6).trim());

row.add(AppCore.resultSet.getString(7).trim());

records.add(row);

recordsCount++;

}

recordsCounter\_FIELD.setText("Всего записей: " + recordsCount);

} catch (Exception e) { e.printStackTrace(); }

try { AppCore.resultSet.close(); AppCore.statement.close(); AppCore.connection.commit(); }

catch (Exception e) { e.printStackTrace(); }

curRecord = 1;

use\_ClientsF\_BTN.setVisible(false);

prev\_ClientsF\_BTN.setVisible(true);

next\_ClientsF\_BTN.setVisible(true);

techInf\_ClientsF\_CHOICE.setItems(null);

city\_ClientsF\_CHOICE.setItems(null);

showRecord\_ClientsF();

name\_ClientsF\_FIELD.setEditable(false);

techInf\_ClientsF\_CHOICE.setEditable(false);

city\_ClientsF\_CHOICE.setEditable(false);

address\_ClientsF\_FIELD.setEditable(false);

contactInf\_ClientsF\_FIELD.setEditable(false);

cur\_ClientsF\_FIELD.setVisible(true);

cancel\_ClientsF\_BTN.setVisible(false);

toMain\_BTN.setDisable(false);

subMenu\_BOX.setVisible(true);

tableShowType\_BTN.setSelected(false);

recordsShowType\_BTN.setSelected(true);

show\_BOX.toFront();

forms\_BOX.toFront();

clientsF\_BOX.toFront();

}

}

private void showRecord\_ClientsF()

{

curRecord\_changed = false;

if (records.size() != 0)

{

cur\_ClientsF\_FIELD.setText(curRecord + "/" + records.size());

name\_ClientsF\_FIELD.setText(records.get(curRecord - 1).get(0));

techInf\_ClientsF\_CHOICE.setValue(records.get(curRecord - 1).get(1));

city\_ClientsF\_CHOICE.setValue(records.get(curRecord - 1).get(2));

address\_ClientsF\_FIELD.setText(records.get(curRecord - 1).get(3));

contactInf\_ClientsF\_FIELD.setText(records.get(curRecord - 1).get(4));

}

else

{

cur\_ClientsF\_FIELD.setText("0/0");

name\_ClientsF\_FIELD.setText("");

techInf\_ClientsF\_CHOICE.setValue(null);

city\_ClientsF\_CHOICE.setValue(null);

address\_ClientsF\_FIELD.setText("");

contactInf\_ClientsF\_FIELD.setText("");

use\_ClientsF\_BTN.setVisible(false);

prev\_ClientsF\_BTN.setVisible(false);

next\_ClientsF\_BTN.setVisible(false);

}

}

// wherePart

private String getWherePart\_ClientsF()

{

StringBuilder wherePart = new StringBuilder();

if (!name\_ClientsF\_FIELD.getText().equals(""))

wherePart.append(" AND cl.name = '").append(name\_ClientsF\_FIELD.getText()).append("'");

if (techInf\_ClientsF\_CHOICE.getValue() != null)

{

String tempStr = techInf\_ClientsF\_CHOICE.getValue();

wherePart.append(" AND cl.id\_technical\_information = ")

.append(tempStr, tempStr.indexOf('[') + 1, tempStr.indexOf(']'));

}

if (city\_ClientsF\_CHOICE.getValue() != null)

wherePart.append(" AND cl.id\_city = (SELECT id FROM client\_cities WHERE name = '")

.append(city\_ClientsF\_CHOICE.getValue()).append("')");

if (!address\_ClientsF\_FIELD.getText().equals(""))

wherePart.append(" AND cl.address = '").append(address\_ClientsF\_FIELD.getText()).append("'");

if (!contactInf\_ClientsF\_FIELD.getText().equals(""))

wherePart.append(" AND cl.contact\_information = '").append(contactInf\_ClientsF\_FIELD.getText()).append("'");

return wherePart.toString();

}

// search

private void searchRecord\_Clients()

{

ObservableList<FullTableRow> rows = FXCollections.observableArrayList();

try

{

AppCore.statement = AppCore.connection.createStatement();

AppCore.resultSet = AppCore.statement.executeQuery(String.format(QrsClients.SELECT\_2CHANGE, getWherePart\_ClientsF()));

recordsCount = 0;

while (AppCore.resultSet.next())

{

FullTableRow row = new FullTableRow();

row.setField\_01(AppCore.resultSet.getString(1).trim());

row.setField\_02(AppCore.resultSet.getString(2).trim());

row.setField\_03(AppCore.resultSet.getString(3).trim());

row.setField\_04(AppCore.resultSet.getString(4).trim());

row.setField\_05(AppCore.resultSet.getString(5).trim());

row.setField\_06(AppCore.resultSet.getString(6).trim());

row.setField\_07(AppCore.resultSet.getString(7).trim());

rows.add(row);

recordsCount++;

}

clients\_TABLE.setItems(rows);

recordsCounter\_FIELD.setText("Всего записей: " + recordsCount);

} catch (Exception e) { e.printStackTrace(); }

try { AppCore.resultSet.close(); AppCore.statement.close(); AppCore.connection.commit(); }

catch (Exception e) { e.printStackTrace(); }

toMain\_BTN.setDisable(false);

subMenu\_BOX.setVisible(true);

showType\_BOX.setDisable(true);

tableShowType\_BTN.setSelected(false);

recordsShowType\_BTN.setSelected(false);

show\_BOX.toFront();

tables\_BOX.toFront();

clientsT\_BOX.toFront();

}

// change

private void changeRecord\_ClientsF()

{

records = new ArrayList<>();

try

{

AppCore.statement = AppCore.connection.createStatement();

AppCore.resultSet = AppCore.statement.executeQuery(

String.format(QrsClients.SELECT\_2CHANGE, getWherePart\_ClientsF()));

while (AppCore.resultSet.next())

{

ArrayList<String> row = new ArrayList<>();

row.add(AppCore.resultSet.getString(1).trim());

row.add(

"[" + AppCore.resultSet.getString(8).trim() + "] " +

"Серв.ID: " + AppCore.resultSet.getString(2).trim() + "; " +

"Port/sap: " + AppCore.resultSet.getString(3).trim() + "; " +

"Скорость: " + AppCore.resultSet.getString(4).trim()

);

row.add(AppCore.resultSet.getString(5).trim());

row.add(AppCore.resultSet.getString(6).trim());

row.add(AppCore.resultSet.getString(7).trim());

records.add(row);

}

} catch (Exception e) { e.printStackTrace(); }

try { AppCore.resultSet.close(); AppCore.statement.close(); AppCore.connection.commit(); }

catch (Exception e) { e.printStackTrace(); }

curRecord = 1;

actionType = "change";

AppCore.showAlert("Приступайте к изменениям");

use\_ClientsF\_BTN.setVisible(true);

prev\_ClientsF\_BTN.setVisible(true);

next\_ClientsF\_BTN.setVisible(true);

showRecord\_ClientsF();

name\_ClientsF\_FIELD.setEditable(true);

techInf\_ClientsF\_CHOICE.setEditable(true);

city\_ClientsF\_CHOICE.setEditable(true);

address\_ClientsF\_FIELD.setEditable(true);

contactInf\_ClientsF\_FIELD.setEditable(true);

cur\_ClientsF\_FIELD.setVisible(true);

cancel\_ClientsF\_BTN.setVisible(true);

toMain\_BTN.setDisable(false);

subMenu\_BOX.setVisible(false);

show\_BOX.toFront();

forms\_BOX.toFront();

clientsF\_BOX.toFront();

}

// delete

private void deleteRecord\_ClientsF()

{

int rowsCount = 0;

try

{

AppCore.statement = AppCore.connection.createStatement();

AppCore.resultSet = AppCore.statement.executeQuery(

String.format(QrsClients.SELECT\_2DELETE, getWherePart\_ClientsF()));

AppCore.resultSet.next();

rowsCount = AppCore.resultSet.getInt(1);

} catch (Exception e) { e.printStackTrace(); }

try { AppCore.resultSet.close(); AppCore.statement.close(); AppCore.connection.commit(); }

catch (Exception e) { e.printStackTrace(); }

Alert alert = new Alert(Alert.AlertType.CONFIRMATION);

alert.initStyle(StageStyle.UNDECORATED);

alert.setGraphic(null);

alert.setHeaderText(null);

alert.setContentText("Найдено " + rowsCount +

" подходящих записей\n\nУдалить все?\n(Также будут удалены все связанные записи)");

Optional<ButtonType> option = alert.showAndWait();

if (option.get() == ButtonType.OK)

{

try {

AppCore.statement = AppCore.connection.createStatement();

int executeCount = AppCore.statement.executeUpdate(

"DELETE FROM clients AS cl WHERE id = id" + getWherePart\_ClientsF());

recordsCounter\_FIELD.setText("Всего записей: " + (recordsCount - rowsCount));

AppCore.showAlert("Удалённых записей: " + executeCount);

}

catch (Exception e)

{

e.printStackTrace();

AppCore.showAlert("Не удалось провести удаление");

}

try { AppCore.resultSet.close(); AppCore.statement.close(); AppCore.connection.commit(); }

catch (Exception e) { e.printStackTrace(); }

}

}

// FinOps =====================================

private void use\_Record\_FinOpsF(String alert\_msg, String action\_type)

{

AppCore.showAlert(alert\_msg);

bill\_FinOpsF\_CHOICE.setEditable(true);

transactionType\_FinOpsF\_CHOICE.setEditable(true);

sum\_FinOpsF\_FIELD.setEditable(true); sum\_FinOpsF\_FIELD.setText("");

operationDate\_FinOpsF\_FIELD.setEditable(true); operationDate\_FinOpsF\_FIELD.setText("");

contractPart\_FinOpsF\_CHOICE.setEditable(true);

comment\_FinOpsF\_FIELD.setEditable(true); comment\_FinOpsF\_FIELD.setText("");

{

try {

ObservableList<String> items = FXCollections.observableArrayList();

items.add(null);

AppCore.statement = AppCore.connection.createStatement();

AppCore.resultSet = AppCore.statement.executeQuery(QrsCommonCatalogs.SELECT\_BILLS);

while (AppCore.resultSet.next())

{

items.add(

"[" + AppCore.resultSet.getString(3).trim() + "] " +

"Клиент: " + AppCore.resultSet.getString(1).trim() +

" , баланс: " + AppCore.resultSet.getString(2).trim()

);

}

bill\_FinOpsF\_CHOICE.setItems(items);

bill\_FinOpsF\_CHOICE.setValue(null);

} catch (Exception e) { e.printStackTrace(); }

try { AppCore.resultSet.close(); AppCore.statement.close(); AppCore.connection.commit(); }

catch (Exception e) { e.printStackTrace(); }

try {

ObservableList<String> items = FXCollections.observableArrayList();

items.add(null);

AppCore.statement = AppCore.connection.createStatement();

AppCore.resultSet = AppCore.statement.executeQuery(QrsCommonCatalogs.SELECT\_TRANSACTION\_TYPES);

while (AppCore.resultSet.next())

{ items.add(AppCore.resultSet.getString(1).trim()); }

transactionType\_FinOpsF\_CHOICE.setItems(items);

transactionType\_FinOpsF\_CHOICE.setValue(null);

} catch (Exception e) { e.printStackTrace(); }

try { AppCore.resultSet.close(); AppCore.statement.close(); AppCore.connection.commit(); }

catch (Exception e) { e.printStackTrace(); }

try { AppCore.resultSet.close(); AppCore.statement.close(); AppCore.connection.commit(); }

catch (Exception e) { e.printStackTrace(); }

try {

ObservableList<String> items = FXCollections.observableArrayList();

items.add(null);

AppCore.statement = AppCore.connection.createStatement();

AppCore.resultSet = AppCore.statement.executeQuery(QrsCommonCatalogs.SELECT\_CONTRACT\_PARTS);

while (AppCore.resultSet.next())

{

items.add(

"[" + AppCore.resultSet.getString(3).trim() + "] " +

"Контракт: " + AppCore.resultSet.getString(1).trim() +

", тариф: " + AppCore.resultSet.getString(2).trim()

);

}

contractPart\_FinOpsF\_CHOICE.setItems(items);

contractPart\_FinOpsF\_CHOICE.setValue(null);

} catch (Exception e) { e.printStackTrace(); }

try { AppCore.resultSet.close(); AppCore.statement.close(); AppCore.connection.commit(); }

catch (Exception e) { e.printStackTrace(); }

}

use\_FinOpsF\_BTN.setVisible(true);

prev\_FinOpsF\_BTN.setVisible(false);

cur\_FinOpsF\_FIELD.setVisible(false);

next\_FinOpsF\_BTN.setVisible(false);

cancel\_FinOpsF\_BTN.setVisible(true);

actionType = action\_type;

toMain\_BTN.setDisable(false);

subMenu\_BOX.setVisible(false);

show\_BOX.toFront();

forms\_BOX.toFront();

finOpsF\_BOX.toFront();

}

// show

private void show\_FinOps(String showType)

{

showType\_BOX.setDisable(false);

fieldsChange\_BTN.setDisable(false);

fieldsDelete\_BTN.setDisable(false);

choseChange\_BTN.setDisable(true);

choseDelete\_BTN.setDisable(true);

if (showType.equals("asTable"))

{

ObservableList<FullTableRow> rows = FXCollections.observableArrayList();

try

{

AppCore.statement = AppCore.connection.createStatement();

AppCore.resultSet = AppCore.statement.executeQuery(QrsFinOps.SELECT\_TABLE);

recordsCount = 0;

while (AppCore.resultSet.next())

{

FullTableRow row = new FullTableRow();

row.setField\_01(AppCore.resultSet.getString(1).trim());

row.setField\_02(AppCore.resultSet.getString(2).trim());

row.setField\_03(AppCore.resultSet.getString(3).trim());

row.setField\_04(AppCore.resultSet.getString(4).trim());

row.setField\_05(AppCore.resultSet.getString(5).trim());

String temp = AppCore.resultSet.getString(6);

row.setField\_06(temp != null ? temp.trim() : "null");

temp = AppCore.resultSet.getString(7);

row.setField\_07(temp != null ? temp.trim() : "null");

temp = AppCore.resultSet.getString(8);

row.setField\_08(temp != null ? temp.trim() : "null");

rows.add(row);

recordsCount++;

}

recordsCounter\_FIELD.setText("Всего записей: " + recordsCount);

} catch (Exception e) { e.printStackTrace(); }

try { AppCore.resultSet.close(); AppCore.statement.close(); AppCore.connection.commit(); }

catch (Exception e) { e.printStackTrace(); }

client\_FinOpsT\_COLUMN.setCellValueFactory(new PropertyValueFactory<> ("field\_01"));

balance\_FinOpsT\_COLUMN.setCellValueFactory(new PropertyValueFactory<> ("field\_02"));

transactionType\_FinOpsT\_COLUMN.setCellValueFactory(new PropertyValueFactory<> ("field\_03"));

sum\_FinOpsT\_COLUMN.setCellValueFactory(new PropertyValueFactory<> ("field\_04"));

operationDate\_FinOpsT\_COLUMN.setCellValueFactory(new PropertyValueFactory<> ("field\_05"));

contract\_FinOpsT\_COLUMN.setCellValueFactory(new PropertyValueFactory<> ("field\_06"));

tariff\_FinOpsT\_COLUMN.setCellValueFactory(new PropertyValueFactory<> ("field\_07"));

comment\_FinOpsT\_COLUMN.setCellValueFactory(new PropertyValueFactory<> ("field\_08"));

finOps\_TABLE.setItems(rows);

toMain\_BTN.setDisable(false);

subMenu\_BOX.setVisible(true);

tableShowType\_BTN.setSelected(true);

recordsShowType\_BTN.setSelected(false);

show\_BOX.toFront();

tables\_BOX.toFront();

finOpsT\_BOX.toFront();

}

else if (showType.equals("byRecords"))

{

records = new ArrayList<>();

try

{

AppCore.statement = AppCore.connection.createStatement();

AppCore.resultSet = AppCore.statement.executeQuery(QrsFinOps.SELECT\_TABLE);

recordsCount = 0;

while (AppCore.resultSet.next())

{

ArrayList<String> row = new ArrayList<>();

row.add(

"[" + AppCore.resultSet.getString(9).trim() + "] " +

"Клиент: " + AppCore.resultSet.getString(1).trim() +

" , баланс: " + AppCore.resultSet.getString(2).trim()

);

row.add(AppCore.resultSet.getString(3).trim());

row.add(AppCore.resultSet.getString(4).trim());

row.add(AppCore.resultSet.getString(5).trim());

String temp = AppCore.resultSet.getString(10);

row.add(temp != null ?

"[" + AppCore.resultSet.getString(10).trim() + "] " +

"Контракт: " + AppCore.resultSet.getString(6).trim() +

", тариф: " + AppCore.resultSet.getString(7).trim()

: "null"

);

temp = AppCore.resultSet.getString(8);

row.add(temp != null ? temp.trim() : "null");

records.add(row);

recordsCount++;

}

recordsCounter\_FIELD.setText("Всего записей: " + recordsCount);

} catch (Exception e) { e.printStackTrace(); }

try { AppCore.resultSet.close(); AppCore.statement.close(); AppCore.connection.commit(); }

catch (Exception e) { e.printStackTrace(); }

curRecord = 1;

use\_FinOpsF\_BTN .setVisible(false);

prev\_FinOpsF\_BTN.setVisible(true);

next\_FinOpsF\_BTN.setVisible(true);

bill\_FinOpsF\_CHOICE.setItems(null);

transactionType\_FinOpsF\_CHOICE.setItems(null);

contractPart\_FinOpsF\_CHOICE.setItems(null);

showRecord\_FinOpsF();

bill\_FinOpsF\_CHOICE.setEditable(false);

transactionType\_FinOpsF\_CHOICE.setEditable(false);

sum\_FinOpsF\_FIELD.setEditable(false);

operationDate\_FinOpsF\_FIELD.setEditable(false);

contractPart\_FinOpsF\_CHOICE.setEditable(false);

comment\_FinOpsF\_FIELD.setEditable(false);

cur\_FinOpsF\_FIELD .setVisible(true);

cancel\_FinOpsF\_BTN.setVisible(false);

toMain\_BTN.setDisable(false);

subMenu\_BOX.setVisible(true);

tableShowType\_BTN.setSelected(false);

recordsShowType\_BTN.setSelected(true);

show\_BOX.toFront();

forms\_BOX.toFront();

finOpsF\_BOX.toFront();

}

}

private void showRecord\_FinOpsF()

{

curRecord\_changed = false;

if (records.size() != 0)

{

cur\_FinOpsF\_FIELD.setText(curRecord + "/" + records.size());

bill\_FinOpsF\_CHOICE.setValue(records.get(curRecord - 1).get(0));

transactionType\_FinOpsF\_CHOICE.setValue(records.get(curRecord - 1).get(1));

sum\_FinOpsF\_FIELD.setText (records.get(curRecord - 1).get(2));

operationDate\_FinOpsF\_FIELD.setText (records.get(curRecord - 1).get(3));

contractPart\_FinOpsF\_CHOICE.setValue(records.get(curRecord - 1).get(4));

comment\_FinOpsF\_FIELD.setText (records.get(curRecord - 1).get(5));

}

else

{

cur\_FinOpsF\_FIELD.setText("0/0");

bill\_FinOpsF\_CHOICE.setValue(null);

transactionType\_FinOpsF\_CHOICE.setValue(null);

sum\_FinOpsF\_FIELD.setText ("");

operationDate\_FinOpsF\_FIELD.setText ("");

contractPart\_FinOpsF\_CHOICE.setValue(null);

comment\_FinOpsF\_FIELD.setText ("");

use\_FinOpsF\_BTN .setVisible(false);

prev\_FinOpsF\_BTN.setVisible(false);

next\_FinOpsF\_BTN.setVisible(false);

}

}

// wherePart

private String getWherePart\_FinOpsF()

{

StringBuilder wherePart = new StringBuilder();

if (bill\_FinOpsF\_CHOICE.getValue() != null)

{

String tempStr = bill\_FinOpsF\_CHOICE.getValue();

wherePart.append(" AND f.id\_bill = ")

.append(tempStr, tempStr.indexOf('[') + 1, tempStr.indexOf(']'));

}

if (transactionType\_FinOpsF\_CHOICE.getValue() != null)

wherePart.append(" AND f.id\_transaction\_type = (SELECT id FROM transaction\_types WHERE name = '")

.append(transactionType\_FinOpsF\_CHOICE.getValue()).append("')");

if (!sum\_FinOpsF\_FIELD.getText().equals(""))

wherePart.append(" AND f.sum = '").append(sum\_FinOpsF\_FIELD.getText()).append("'");

if (!operationDate\_FinOpsF\_FIELD.getText().equals(""))

wherePart.append(" AND f.operation\_date = '").append(operationDate\_FinOpsF\_FIELD.getText()).append("'");

if (contractPart\_FinOpsF\_CHOICE.getValue() != null)

{

wherePart.append(" AND f.id\_contract\_part ");

if (contractPart\_FinOpsF\_CHOICE.getValue().equals("null")) wherePart.append("ISNULL");

else { String tempStr = contractPart\_FinOpsF\_CHOICE.getValue();

wherePart.append("= ").append(tempStr, tempStr.indexOf('[') + 1, tempStr.indexOf(']')); }

}

if (!comment\_FinOpsF\_FIELD.getText().equals("")) {

wherePart.append(" AND f.comment ");

if (comment\_FinOpsF\_FIELD.getText().equals("null")) wherePart.append("ISNULL");

else wherePart.append("= '").append(comment\_FinOpsF\_FIELD.getText()).append("'");

}

return wherePart.toString();

}

// search

private void searchRecord\_FinOps()

{

ObservableList<FullTableRow> rows = FXCollections.observableArrayList();

try

{

AppCore.statement = AppCore.connection.createStatement();

AppCore.resultSet = AppCore.statement.executeQuery(String.format(QrsFinOps.SELECT\_2CHANGE, getWherePart\_FinOpsF()));

recordsCount = 0;

while (AppCore.resultSet.next())

{

FullTableRow row = new FullTableRow();

row.setField\_01(AppCore.resultSet.getString(1).trim());

row.setField\_02(AppCore.resultSet.getString(2).trim());

row.setField\_03(AppCore.resultSet.getString(3).trim());

row.setField\_04(AppCore.resultSet.getString(4).trim());

row.setField\_05(AppCore.resultSet.getString(5).trim());

String temp = AppCore.resultSet.getString(6);

row.setField\_06(temp != null ? temp.trim() : "null");

temp = AppCore.resultSet.getString(7);

row.setField\_07(temp != null ? temp.trim() : "null");

temp = AppCore.resultSet.getString(8);

row.setField\_08(temp != null ? temp.trim() : "null");

rows.add(row);

recordsCount++;

}

finOps\_TABLE.setItems(rows);

recordsCounter\_FIELD.setText("Всего записей: " + recordsCount);

} catch (Exception e) { e.printStackTrace(); }

try { AppCore.resultSet.close(); AppCore.statement.close(); AppCore.connection.commit(); }

catch (Exception e) { e.printStackTrace(); }

toMain\_BTN.setDisable(false);

subMenu\_BOX.setVisible(true);

showType\_BOX.setDisable(true);

tableShowType\_BTN.setSelected(false);

recordsShowType\_BTN.setSelected(false);

show\_BOX.toFront();

tables\_BOX.toFront();

finOpsT\_BOX.toFront();

}

// change

private void changeRecord\_FinOpsF()

{

records = new ArrayList<>();

try

{

AppCore.statement = AppCore.connection.createStatement();

AppCore.resultSet = AppCore.statement.executeQuery(

String.format(QrsFinOps.SELECT\_2CHANGE, getWherePart\_FinOpsF()));

while (AppCore.resultSet.next())

{

ArrayList<String> row = new ArrayList<>();

row.add(

"[" + AppCore.resultSet.getString(9).trim() + "] " +

"Клиент: " + AppCore.resultSet.getString(1).trim() +

" , баланс: " + AppCore.resultSet.getString(2).trim()

);

row.add(AppCore.resultSet.getString(3).trim());

row.add(AppCore.resultSet.getString(4).trim());

row.add(AppCore.resultSet.getString(5).trim());

String temp = AppCore.resultSet.getString(10);

if (temp != null) {

row.add(

"[" + temp.trim() + "] " +

"Контракт: " + AppCore.resultSet.getString(6).trim() +

", тариф: " + AppCore.resultSet.getString(7).trim()

);

} else row.add("[null]");

temp = AppCore.resultSet.getString(8);

if (temp != null) { row.add(temp.trim()); } else row.add("null");

records.add(row);

}

} catch (Exception e) { e.printStackTrace(); }

try { AppCore.resultSet.close(); AppCore.statement.close(); AppCore.connection.commit(); }

catch (Exception e) { e.printStackTrace(); }

curRecord = 1;

actionType = "change";

AppCore.showAlert("Приступайте к изменениям");

use\_FinOpsF\_BTN.setVisible(true);

prev\_FinOpsF\_BTN.setVisible(true);

next\_FinOpsF\_BTN.setVisible(true);

showRecord\_FinOpsF();

bill\_FinOpsF\_CHOICE.setEditable(true);

transactionType\_FinOpsF\_CHOICE.setEditable(true);

sum\_FinOpsF\_FIELD.setEditable(true);

operationDate\_FinOpsF\_FIELD.setEditable(true);

contractPart\_FinOpsF\_CHOICE.setEditable(true);

comment\_FinOpsF\_FIELD.setEditable(true);

cur\_FinOpsF\_FIELD.setVisible(true);

cancel\_FinOpsF\_BTN.setVisible(true);

toMain\_BTN.setDisable(false);

subMenu\_BOX.setVisible(false);

show\_BOX.toFront();

forms\_BOX.toFront();

finOpsF\_BOX.toFront();

}

// delete

private void deleteRecord\_FinOpsF()

{

int rowsCount = 0;

try

{

AppCore.statement = AppCore.connection.createStatement();

AppCore.resultSet = AppCore.statement.executeQuery(

String.format(QrsFinOps.SELECT\_2DELETE, getWherePart\_FinOpsF()));

AppCore.resultSet.next();

rowsCount = AppCore.resultSet.getInt(1);

} catch (Exception e) { e.printStackTrace(); }

try { AppCore.resultSet.close(); AppCore.statement.close(); AppCore.connection.commit(); }

catch (Exception e) { e.printStackTrace(); }

Alert alert = new Alert(Alert.AlertType.CONFIRMATION);

alert.initStyle(StageStyle.UNDECORATED);

alert.setGraphic(null);

alert.setHeaderText(null);

alert.setContentText("Найдено " + rowsCount +

" подходящих записей\n\nУдалить все?\n(Также будут удалены все связанные записи)");

Optional<ButtonType> option = alert.showAndWait();

if (option.get() == ButtonType.OK)

{

try {

AppCore.statement = AppCore.connection.createStatement();

int executeCount = AppCore.statement.executeUpdate(

"DELETE FROM financial\_operations AS f WHERE id = id" + getWherePart\_ContractsF());

recordsCounter\_FIELD.setText("Всего записей: " + (recordsCount - rowsCount));

AppCore.showAlert("Удалённых записей: " + executeCount);

}

catch (Exception e)

{

e.printStackTrace();

AppCore.showAlert("Не удалось провести удаление");

}

try { AppCore.resultSet.close(); AppCore.statement.close(); AppCore.connection.commit(); }

catch (Exception e) { e.printStackTrace(); }

}

}

// endregion

// region MORE EASIER

private void search\_Base(String title, String query)

{

TextInputDialog dialog = new TextInputDialog();

dialog.initStyle(StageStyle.UTILITY);

dialog.setGraphic(null);

dialog.setHeaderText(null);

dialog.setTitle(title);

dialog.setContentText("Название:");

Optional<String> result = dialog.showAndWait();

result.ifPresent(value ->

{

ObservableList<FullTableRow> rows = FXCollections.observableArrayList();

try {

AppCore.statement = AppCore.connection.createStatement();

AppCore.resultSet = AppCore.statement.executeQuery(String.format(

query, value

));

recordsCount = 0;

while (AppCore.resultSet.next()) {

FullTableRow row = new FullTableRow();

row.setField\_01(AppCore.resultSet.getString(1).trim());

rows.add(row);

recordsCount++;

}

commonCatalog\_TABLE.setItems(rows);

recordsCounter\_FIELD.setText("Всего записей: " + recordsCount);

} catch (Exception e) {

e.printStackTrace(); AppCore.showAlert("Не удалось найти записи");

}

try {

AppCore.resultSet.close();

AppCore.statement.close();

AppCore.connection.commit();

} catch (Exception e) {

e.printStackTrace();

}

});

}

private void proc\_Base(String title, String[] queryValues, String successText, String failText)

{

TextInputDialog dialog = new TextInputDialog(queryValues[1]);

dialog.initStyle(StageStyle.UTILITY);

dialog.setGraphic(null);

dialog.setHeaderText(null);

dialog.setTitle(title);

dialog.setContentText("Название:");

Optional<String> result = dialog.showAndWait();

result.ifPresent(value ->

{

try {

AppCore.statement = AppCore.connection.createStatement();

AppCore.statement.executeUpdate(

String.format(queryValues[0], value, queryValues[1]));

AppCore.showAlert(successText);

}

catch (Exception e)

{

e.printStackTrace();

AppCore.showAlert(failText);

}

try { AppCore.resultSet.close(); AppCore.statement.close(); AppCore.connection.commit(); }

catch (Exception e) { e.printStackTrace(); }

});

show\_CommonCatalog(curTable);

}

private void delete\_Base(String title, String query)

{

Alert alert = new Alert(Alert.AlertType.CONFIRMATION);

alert.initStyle(StageStyle.UTILITY);

alert.setGraphic(null);

alert.setHeaderText(null);

alert.setTitle(title);

alert.setContentText("Выбранная запись будет удалена.\nПродолжить?");

Optional<ButtonType> option = alert.showAndWait();

if (option.get() == ButtonType.OK)

{

try {

AppCore.statement = AppCore.connection.createStatement();

AppCore.statement.executeUpdate(String.format(

query, commonCatalog\_TABLE.getSelectionModel().getSelectedItem().getField\_01()));

AppCore.showAlert("Запись удалена");

}

catch (Exception e)

{

e.printStackTrace();

AppCore.showAlert("Не удалось провести удаление");

}

try { AppCore.resultSet.close(); AppCore.statement.close(); AppCore.connection.commit(); }

catch (Exception e) { e.printStackTrace(); }

}

show\_CommonCatalog(curTable);

}

// Client cities =====================================

// search

private void searchRecord\_Cities()

{ search\_Base("Поиск города клиентов", QrsCommonCatalogs.SELECT\_CLIENT\_CITIES\_2SEARCH); }

// add

private void addRecord\_Cities()

{

proc\_Base("Добавление нового города клиентов",

new String[] {QrsCommonCatalogs.ADD\_CLIENT\_CITIES, null},

"Новая запись добавлена", "Не удалось добавить запись");

}

// change

private void changeRecord\_Cities()

{

if (commonCatalog\_TABLE.getSelectionModel().getSelectedItem() == null)

{ AppCore.showAlert("Для выполнения действия\nсначала выберите строку"); return; }

proc\_Base("Изменение города клиентов", new String[] {QrsCommonCatalogs.CHANGE\_CLIENT\_CITIES,

commonCatalog\_TABLE.getSelectionModel().getSelectedItem().getField\_01()},

"Запись обновлена", "Не удалось обновить запись");

}

// delete

private void deleteRecord\_Cities()

{

if (commonCatalog\_TABLE.getSelectionModel().getSelectedItem() == null)

{ AppCore.showAlert("Для выполнения действия\nсначала выберите строку"); return; }

delete\_Base("Удаление города клиентов", QrsCommonCatalogs.DELETE\_CLIENT\_CITIES);

}

// Transaction types =====================================

// search

private void searchRecord\_TransactionTypes()

{ search\_Base("Поиск типа транзакций", QrsCommonCatalogs.SELECT\_TRANSACTION\_TYPES\_2SEARCH); }

// add

private void addRecord\_TransactionTypes()

{

proc\_Base("Добавление нового типа транзакций",

new String[]{QrsCommonCatalogs.ADD\_TRANSACTION\_TYPES, null},

"Новая запись добавлена", "Не удалось добавить запись");

}

// change

private void changeRecord\_TransactionTypes()

{

if (commonCatalog\_TABLE.getSelectionModel().getSelectedItem() == null)

{ AppCore.showAlert("Для выполнения действия\nсначала выберите строку"); return; }

proc\_Base("Изменение типа транзакций", new String[] {QrsCommonCatalogs.CHANGE\_TRANSACTION\_TYPES,

commonCatalog\_TABLE.getSelectionModel().getSelectedItem().getField\_01()},

"Запись обновлена", "Не удалось обновить запись");

}

// delete

private void deleteRecord\_TransactionTypes()

{

if (commonCatalog\_TABLE.getSelectionModel().getSelectedItem() == null)

{ AppCore.showAlert("Для выполнения действия\nсначала выберите строку"); return; }

delete\_Base("Удаление типа транзакций", QrsCommonCatalogs.DELETE\_TRANSACTION\_TYPES);

}

// Services =====================================

// search

private void searchRecord\_Services()

{ search\_Base("Поиск услуги", QrsCommonCatalogs.SELECT\_SERVICES\_2SEARCH); }

// add

private void addRecord\_Services()

{

proc\_Base("Добавление новой услуги",

new String[]{QrsCommonCatalogs.ADD\_SERVICES, null},

"Новая запись добавлена", "Не удалось добавить запись");

}

// change

private void changeRecord\_Services()

{

if (commonCatalog\_TABLE.getSelectionModel().getSelectedItem() == null)

{ AppCore.showAlert("Для выполнения действия\nсначала выберите строку"); return; }

proc\_Base("Изменение услуги", new String[] {QrsCommonCatalogs.CHANGE\_SERVICES,

commonCatalog\_TABLE.getSelectionModel().getSelectedItem().getField\_01()},

"Запись обновлена", "Не удалось обновить запись");

}

// delete

private void deleteRecord\_Services()

{

if (commonCatalog\_TABLE.getSelectionModel().getSelectedItem() == null)

{ AppCore.showAlert("Для выполнения действия\nсначала выберите строку"); return; }

delete\_Base("Удаление услуги", QrsCommonCatalogs.DELETE\_SERVICES);

}

// Charge types =====================================

// search

private void searchRecord\_ChargeTypes()

{ search\_Base("Поиск типа оплаты", QrsCommonCatalogs.SELECT\_CHARGE\_TYPES\_2SEARCH); }

// add

private void addRecord\_ChargeTypes()

{

proc\_Base("Добавление нового типа оплаты",

new String[]{QrsCommonCatalogs.ADD\_CHARGE\_TYPES, null},

"Новая запись добавлена", "Не удалось добавить запись");

}

// change

private void changeRecord\_ChargeTypes()

{

if (commonCatalog\_TABLE.getSelectionModel().getSelectedItem() == null)

{ AppCore.showAlert("Для выполнения действия\nсначала выберите строку"); return; }

proc\_Base("Изменение типа оплаты", new String[] {QrsCommonCatalogs.CHANGE\_CHARGE\_TYPES,

commonCatalog\_TABLE.getSelectionModel().getSelectedItem().getField\_01()},

"Запись обновлена", "Не удалось обновить запись");

}

// delete

private void deleteRecord\_ChargeTypes()

{

if (commonCatalog\_TABLE.getSelectionModel().getSelectedItem() == null)

{ AppCore.showAlert("Для выполнения действия\nсначала выберите строку"); return; }

delete\_Base("Удаление типа оплаты", QrsCommonCatalogs.DELETE\_CHARGE\_TYPES);

}

// Contract statuses =====================================

// search

private void searchRecord\_ContractStatuses()

{ search\_Base("Поиск статуса контрактов", QrsCommonCatalogs.SELECT\_CONTRACT\_STATUSES\_2SEARCH); }

// add

private void addRecord\_ContractStatuses()

{

proc\_Base("Добавление нового статуса контрактов",

new String[]{QrsCommonCatalogs.ADD\_CONTRACT\_STATUSES, null},

"Новая запись добавлена", "Не удалось добавить запись");

}

// change

private void changeRecord\_ContractStatuses()

{

if (commonCatalog\_TABLE.getSelectionModel().getSelectedItem() == null)

{ AppCore.showAlert("Для выполнения действия\nсначала выберите строку"); return; }

proc\_Base("Изменение статуса контрактов", new String[] {QrsCommonCatalogs.CHANGE\_CONTRACT\_STATUSES,

commonCatalog\_TABLE.getSelectionModel().getSelectedItem().getField\_01()},

"Запись обновлена", "Не удалось обновить запись");

}

// delete

private void deleteRecord\_ContractStatuses()

{

if (commonCatalog\_TABLE.getSelectionModel().getSelectedItem() == null)

{ AppCore.showAlert("Для выполнения действия\nсначала выберите строку"); return; }

delete\_Base("Удаление статуса контрактов", QrsCommonCatalogs.DELETE\_CONTRACT\_STATUSES);

}

// endregion

//////////////////////// BTNs CONTROLS //////////////////////////////

@FXML

void initialize()

{

AppCore.mainController = this;

toMain\_BTN.setDisable(true);

subMenu\_BOX.setVisible(false);

titleText\_BOX.toFront();

toMain\_BTN.setOnAction(event ->

{

toMain\_BTN.setDisable(true);

subMenu\_BOX.setVisible(false);

titleText\_BOX.toFront();

});

//////////////////////////////// QUIT group //////////////////////////////////

quitAccount\_BTN.setOnAction(event ->

{ AppCore.stages.get("Main").hide(); AppCore.stages.get("Authorization").show(); });

quitApp\_BTN.setOnAction(event -> Platform.exit());

}

// endregion

}

// Queries\_Controller.java

// region CODE

// common VARIABLES

private int recordsCount;

// common FUNCTIONS

public void callQuery(String[] queryData)

{

switch (queryData[0])

{

case "INNER\_WHERE\_FK\_Tariffs" -> show\_Columns\_2\_TABLE(String.format(

QrsSpecial.INNER\_WHERE\_FK\_Tariffs, queryData[1]), "Отбор ТАРИФОВ по услуге",

"название", "стоимость");

case "INNER\_WHERE\_FK\_Contracts" -> show\_Columns\_4\_TABLE(String.format(

QrsSpecial.INNER\_WHERE\_FK\_Contracts, queryData[1]), "Отбор КОНТРАКТОВ по клиенту",

"наименование", "клиент", "дата заключения", "статус");

case "INNER\_WHERE\_DATE\_FinOps" -> show\_Columns\_4\_TABLE(String.format(

QrsSpecial.INNER\_WHERE\_DATE\_FinOps, queryData[1]), "Отбор ФИНАНСОВЫХ ОПЕРАЦИЙ по дате операции",

"тип транзакции", "сумма", "дата операции", "комментарий");

case "INNER\_WHERE\_DATE\_Contracts" -> show\_Columns\_4\_TABLE(String.format(

QrsSpecial.INNER\_WHERE\_DATE\_Contracts, queryData[1]), "Отбор КОНТРАКТОВ по дате завершения",

"наименование", "статус", "дата начала", "дата завершения");

//////////////////////////////////////////////////////

case "INNER\_no\_WHERE\_Clients" -> show\_Columns\_4\_TABLE(

QrsSpecial.INNER\_no\_WHERE\_Clients, "Спец. представление КЛИЕНТОВ",

"наименование", "город", "адрес", "контактная информация");

case "INNER\_no\_WHERE\_Tariffs" -> show\_Columns\_4\_TABLE(

QrsSpecial.INNER\_no\_WHERE\_Tariffs, "Спец. представление ТАРИФОВ",

"название", "услуга", "тип оплаты", "стоимость");

case "INNER\_no\_WHERE\_ContractParts" -> show\_Columns\_2\_TABLE(

QrsSpecial.INNER\_no\_WHERE\_ContractParts, "Спец. представление ЧАСТЕЙ КОНТРАКТОВ",

"контракт", "тариф");

//////////////////////////////////////////////////////

case "LEFT\_OUTER" -> show\_Columns\_4\_TABLE(

QrsSpecial.LEFT\_OUTER, "ФИНАНСОВЫЕ ОПЕРАЦИИ без частей контрактов",

"тип транзакции", "сумма", "комментарий", "часть контракта");

case "RIGHT\_OUTER" -> show\_Columns\_4\_TABLE(String.format(

QrsSpecial.RIGHT\_OUTER, queryData[1]), "КЛИЕНТЫ с ограниченными подробностями",

"наименование", "адрес", "контактная информация", "баланс");

case "SELECT\_$\_FROM\_SELECT" -> show\_Columns\_2\_TABLE(

QrsSpecial.SELECT\_$\_FROM\_SELECT, "Спец. представление ФИНАНСОВЫХ ОПЕРАЦИЙ",

"тип транзакции", "сумма");

//////////////////////////////////////////////////////

case "AGGREGATE\_no\_WHERE" -> show\_Columns\_2\_TABLE(

QrsSpecial.AGGREGATE\_no\_WHERE, "Количество КЛИЕНТОВ, сгруппированных по скорости интернета",

"количество клиентов", "скорость");

case "AGGREGATE\_WHERE" -> show\_Columns\_2\_TABLE(String.format(

QrsSpecial.AGGREGATE\_WHERE, queryData[1]), "Средняя стоимость ТАРИФОВ с указанной услугой",

"средняя стоимость", "услуга");

case "AGGREGATE\_HAVING" -> show\_Columns\_2\_TABLE(

QrsSpecial.AGGREGATE\_HAVING, "Количество ФИНАНСОВЫХ ОПЕРАЦИЙ с проходящей суммой более 500 у.е.",

"количество", "сумма");

case "AGGREGATE\_WHERE\_HAVING" -> show\_Columns\_2\_TABLE(String.format(

QrsSpecial.AGGREGATE\_WHERE\_HAVING, queryData[1]), "Сумма отрицательных балансов СЧЕТОВ у указанного клиента",

"клиент", "баланс");

case "AGGREGATE\_$\_FROM\_SELECT" -> show\_Columns\_2\_TABLE(

QrsSpecial.AGGREGATE\_$\_FROM\_SELECT, "Спец. представление групп КЛИЕНТОВ",

"к-во клиентов", "скорость интернетоборудования");

case "SELECT\_$\_WHERE\_SELECT" -> show\_Columns\_2\_TABLE(QrsSpecial.SELECT\_$\_WHERE\_SELECT + queryData[1] + "%') ORDER BY 1, 2",

"СЧЕТА клиентов, наименования который начинаются на указанную часть",

"клиент", "баланс");

//////////////////////////////////////////////////////

case "SPECIAL\_QR\_1" -> show\_Columns\_4\_TABLE(QrsSpecial.SPECIAL\_QR\_1,

"Топ 3 КЛИЕНТА, с наибольшим к-вом фин.операций всего и по городам",

"наименование", "к-во операций", "город", "контактная инф.");

case "SPECIAL\_QR\_2" -> show\_Columns\_2\_TABLE(QrsSpecial.SPECIAL\_QR\_2,

"Среднее к-во фин.операций по КЛИЕНТУ и по ГОРОДУ",

"группа", "к-во фин.операций");

case "SPECIAL\_QR\_3" -> show\_Columns\_4\_TABLE(String.format(QrsSpecial.SPECIAL\_QR\_3, queryData[1], queryData[2]),

"К-во КЛИЕНТОВ, совершивших фин.операции за указанный промежуток, и прошедшая их сумма денег",

"дата", "к-во клиентов", "к-во счетов", "сумма");

}

}

private void show\_Columns\_4\_TABLE(String query, String boxTitle, String c\_name\_1, String c\_name\_2, String c\_name\_3, String c\_name\_4)

{

ObservableList<FullTableRow> rows = FXCollections.observableArrayList();

try {

AppCore.statement = AppCore.connection.createStatement();

AppCore.resultSet = AppCore.statement.executeQuery(query);

recordsCount = 0;

while (AppCore.resultSet.next()) {

FullTableRow row = new FullTableRow();

row.setField\_01(AppCore.resultSet.getString(1).trim());

String temp = AppCore.resultSet.getString(2);

row.setField\_02(temp != null ? temp.trim() : "null");

temp = AppCore.resultSet.getString(3);

row.setField\_03(temp != null ? temp.trim() : "null");

temp = AppCore.resultSet.getString(4);

row.setField\_04(temp != null ? temp.trim() : "null");

rows.add(row);

recordsCount++;

}

recordsCounter\_FIELD.setText("Всего записей: " + recordsCount);

} catch (Exception e) {

e.printStackTrace();

}

try {

AppCore.resultSet.close();

AppCore.statement.close();

AppCore.connection.commit();

} catch (Exception e) {

e.printStackTrace();

}

on4\_1\_COLUMN.setCellValueFactory(new PropertyValueFactory<>("field\_01"));

on4\_1\_COLUMN.setText(c\_name\_1);

on4\_2\_COLUMN.setCellValueFactory(new PropertyValueFactory<>("field\_02"));

on4\_2\_COLUMN.setText(c\_name\_2);

on4\_3\_COLUMN.setCellValueFactory(new PropertyValueFactory<>("field\_03"));

on4\_3\_COLUMN.setText(c\_name\_3);

on4\_4\_COLUMN.setCellValueFactory(new PropertyValueFactory<>("field\_04"));

on4\_4\_COLUMN.setText(c\_name\_4);

columns\_4\_TABLE.setItems(rows);

columns\_4\_BOX.toFront();

columns\_4\_BOX.setText(boxTitle);

}

private void show\_Columns\_2\_TABLE(String query, String boxTitle, String c\_name\_1, String c\_name\_2)

{

ObservableList<FullTableRow> rows = FXCollections.observableArrayList();

try {

AppCore.statement = AppCore.connection.createStatement();

AppCore.resultSet = AppCore.statement.executeQuery(query);

recordsCount = 0;

while (AppCore.resultSet.next()) {

FullTableRow row = new FullTableRow();

row.setField\_01(AppCore.resultSet.getString(1).trim());

row.setField\_02(AppCore.resultSet.getString(2).trim());

rows.add(row);

recordsCount++;

}

recordsCounter\_FIELD.setText("Всего записей: " + recordsCount);

} catch (Exception e) {

e.printStackTrace();

}

try {

AppCore.resultSet.close();

AppCore.statement.close();

AppCore.connection.commit();

} catch (Exception e) {

e.printStackTrace();

}

on2\_1\_COLUMN.setCellValueFactory(new PropertyValueFactory<>("field\_01"));

on2\_1\_COLUMN.setText(c\_name\_1);

on2\_2\_COLUMN.setCellValueFactory(new PropertyValueFactory<>("field\_02"));

on2\_2\_COLUMN.setText(c\_name\_2);

columns\_2\_TABLE.setItems(rows);

columns\_2\_BOX.toFront();

columns\_2\_BOX.setText(boxTitle);

}

// for FXML CODE

@FXML

void initialize() { AppCore.queriesController = this; }

// endregion

}

// Diagrams\_Controller.java

package work;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.fxml.FXML;

import javafx.scene.chart.\*;

import javafx.scene.control.TitledPane;

import java.net.URL;

import java.util.ResourceBundle;

public class Diagrams\_Controller {

@FXML

private ResourceBundle resources;

@FXML

private URL location;

// region FXML

@FXML

private TitledPane dimension\_1\_BOX;

@FXML // к-во клиентов в городах

private PieChart dimension\_1\_DIAGRAM;

@FXML

private TitledPane dimension\_2\_BOX;

@FXML // зависимость средней стоимости тарифа от типа оплаты

private LineChart<String, Double> dimension\_2\_DIAGRAM;

@FXML

private CategoryAxis dimension\_2\_CATEGORY;

@FXML

private NumberAxis dimension\_2\_NUMBER;

@FXML // связь клиентов, их счетов и количества проведённых по ним фин. операций

private TitledPane dimension\_3\_BOX;

@FXML

private StackedBarChart<String, Double> dimension\_3\_DIAGRAM;

@FXML

private CategoryAxis dimension\_3\_CATEGORY;

@FXML

private NumberAxis dimension\_3\_NUMBER;

// endregion

// region CODE

// common FUNCTIONS

public void callDiagram(String diagramType)

{

switch (diagramType)

{

case "dimension\_1" -> show\_dimension\_1\_DIAGRAM();

case "dimension\_2" -> show\_dimension\_2\_DIAGRAM();

case "dimension\_3" -> show\_dimension\_3\_DIAGRAM();

}

}

private void show\_dimension\_1\_DIAGRAM()

{

ObservableList<PieChart.Data> data = FXCollections.observableArrayList();

try {

AppCore.statement = AppCore.connection.createStatement();

AppCore.resultSet = AppCore.statement.executeQuery("""

SELECT COUNT(c.id), ci.name FROM clients AS c

INNER JOIN client\_cities AS ci ON ci.id = c.id\_city

GROUP BY ci.name ORDER BY 1

""");

while (AppCore.resultSet.next()) {

data.add(new PieChart.Data(

AppCore.resultSet.getString(2).trim(),

Double.parseDouble(AppCore.resultSet.getString(1).trim())

));

}

} catch (Exception e) { e.printStackTrace(); }

try { AppCore.resultSet.close(); AppCore.statement.close(); AppCore.connection.commit(); }

catch (Exception e) { e.printStackTrace(); }

dimension\_1\_DIAGRAM.setData(data);

dimension\_1\_BOX.toFront();

dimension\_1\_BOX.setText("Диаграмма, отражающая к-во клиентов в городах");

}

private void show\_dimension\_2\_DIAGRAM()

{

ObservableList<XYChart.Series<String, Double>> data = FXCollections.observableArrayList();

try {

AppCore.statement = AppCore.connection.createStatement();

AppCore.resultSet = AppCore.statement.executeQuery("""

SELECT ROUND(AVG(t.cost::numeric), 2), ch.name FROM tariffs AS t

INNER JOIN charge\_types AS ch ON ch.id = t.id\_charge\_type

GROUP BY ch.name ORDER BY 2

""");

while (AppCore.resultSet.next()) {

XYChart.Series<String, Double> data\_part = new XYChart.Series<>();

data\_part.getData().add(new XYChart.Data<>(

AppCore.resultSet.getString(2).trim(),

Double.parseDouble(AppCore.resultSet.getString(1).trim())

));

data.add(data\_part);

}

} catch (Exception e) { e.printStackTrace(); }

try { AppCore.resultSet.close(); AppCore.statement.close(); AppCore.connection.commit(); }

catch (Exception e) { e.printStackTrace(); }

dimension\_2\_DIAGRAM.setData(data);

dimension\_2\_BOX.toFront();

dimension\_2\_BOX.setText("Диаграмма, отражающая зависимость средней стоимости тарифа от типа оплаты");

}

private void show\_dimension\_3\_DIAGRAM()

{

ObservableList<XYChart.Series<String, Double>> data = FXCollections.observableArrayList();

try {

AppCore.statement = AppCore.connection.createStatement();

AppCore.resultSet = AppCore.statement.executeQuery("""

SELECT COUNT(f.id), b.id, c.name FROM financial\_operations AS f

INNER JOIN bills AS b ON b.id = f.id\_bill

INNER JOIN clients AS c ON c.id = b.id\_client

GROUP BY b.id, c.name ORDER BY 3, 1, 2

""");

XYChart.Series<String, Double> data\_holder = new XYChart.Series<>();

String client = ""; int counter = 1;

while (AppCore.resultSet.next()) {

if (!client.equals(AppCore.resultSet.getString(3)))

{

if (!data\_holder.getData().isEmpty())

{

data.add(data\_holder);

data\_holder = new XYChart.Series<>();

}

counter = 1;

client = AppCore.resultSet.getString(3);

data\_holder.setName(client);

}

data\_holder.getData().add(new XYChart.Data<>(

counter + "-й счёт",

Double.parseDouble(AppCore.resultSet.getString(1).trim())

));

counter++;

}

data.add(data\_holder);

} catch (Exception e) { e.printStackTrace(); }

try { AppCore.resultSet.close(); AppCore.statement.close(); AppCore.connection.commit(); }

catch (Exception e) { e.printStackTrace(); }

dimension\_3\_DIAGRAM.setData(data);

dimension\_3\_BOX.toFront();

dimension\_3\_BOX.setText("Диаграмма, отражающая связь клиентов, их счетов и количества проведённых по ним фин. операций");

}

// for FXML CODE

@FXML

void initialize() { AppCore.diagramsController = this; }

// endregion

}

// Authorization\_Controller.java

package work;

import java.net.URL;

import java.util.ResourceBundle;

import javafx.application.Platform;

import javafx.fxml.FXML;

import javafx.scene.control.Button;

import javafx.scene.control.PasswordField;

import javafx.scene.control.TextField;

public class Authorization\_Controller {

@FXML

private ResourceBundle resources;

@FXML

private URL location;

//==================================================

@FXML

private TextField login\_FIELD;

@FXML

private PasswordField password\_FIELD;

@FXML

private Button authorize\_BTN;

@FXML

private Button close\_BTN;

@FXML

void initialize()

{

AppCore.authorizationController = this;

// BTNs actions

authorize\_BTN.setOnAction(event ->

{

if (login\_FIELD.getText().equals("admin") && password\_FIELD.getText().equals("password"))

{

AppCore.login\_password = "admin\_password";

AppCore.stages.get("Authorization").hide();

AppCore.stages.get("Main").show();

}

else { AppCore.showAlert("Неверные логин и/или пароль\nПожалуйста, попробуйте ещё раз"); }

});

close\_BTN.setOnAction(event -> Platform.exit());

}

}

# ПРИЛОЖЕНИЕ Г

ИСХОДНЫЙ КОД ФУНКЦИИ-ГЕНЕРАТОРА

Код генератора находится в файле «Main\_Controller.java»

generation\_BTN.setOnAction(event ->

{

Alert alert = new Alert(Alert.AlertType.CONFIRMATION);

alert.initStyle(StageStyle.UTILITY);

alert.setGraphic(null);

alert.setHeaderText(null);

alert.setTitle("Воссоздание данных базы");

alert.setContentText("""

Все данные основных таблиц из базы

будут безвозвратно удалены.

После чего база будет наполнена

стортовым набором значений.

ПРОДОЛЖИТЬ?""");

Optional<ButtonType> option = alert.showAndWait();

if (option.get() == ButtonType.OK)

{

try {

AppCore.statement = AppCore.connection.createStatement();

StringBuilder query = new StringBuilder();

String divide = ";\n";

query .append(GenerateQrsData.truncate);

query.append(divide).append(GenerateQrsData.insertHeader\_TechInf);

query.append(GenerateQrsData.insertData\_TechInf. repeat(100)); query.delete(query.length() - 1, query.length());

query.append(divide).append(GenerateQrsData.insertHeader\_Clients);

query.append(GenerateQrsData.insertData\_Clients. repeat(150)); query.delete(query.length() - 1, query.length());

query.append(divide).append(GenerateQrsData.insertHeader\_Bills);

query.append(GenerateQrsData.insertData\_Bills. repeat(100)); query.delete(query.length() - 1, query.length());

query.append(divide).append(GenerateQrsData.insertHeader\_Contracts);

query.append(GenerateQrsData.insertData\_Contracts. repeat(200)); query.delete(query.length() - 1, query.length());

query.append(divide).append(GenerateQrsData.insertHeader\_Tariffs);

query.append(GenerateQrsData.insertData\_Tariffs. repeat(5)); query.delete(query.length() - 1, query.length());

query.append(divide).append(GenerateQrsData.insertHeader\_ContractParts);

query.append(GenerateQrsData.insertData\_ContractParts.repeat(250)); query.delete(query.length() - 1, query.length());

query.append(divide).append(GenerateQrsData.insertHeader\_FinOps);

query.append(GenerateQrsData.insertData\_FinOps. repeat(200)); query.delete(query.length() - 1, query.length());

AppCore.statement.executeUpdate(query.toString());

AppCore.showAlert("База данных воссоздана");

}

catch (Exception e) { e.printStackTrace(); AppCore.showAlert("Не удалось воссоздать базу данных"); }

try { AppCore.statement.close(); AppCore.connection.commit(); }

catch (Exception e) { e.printStackTrace(); }

}

});

# ПРИЛОЖЕНИЕ Д

ПОСТРОЕНИЕ ДИАГРАММ И ВЫГРУЗКА ДАННЫХ

Для построения и отображения диаграмм используются элементы JavaFX – PieChart, LineChart и StackedBarChart. Код, который строит и отображает диаграммы, написан в файле «Diagrams\_Controller.java» (рисунки Д.1-Д.3).

По условию курсовой работы надо было организовать три типа диаграмм: одномерная(круговая), двумерная и трёхмерная. Данные, используемые в диаграммах, программа получает благодаря запросам на языке SQL.

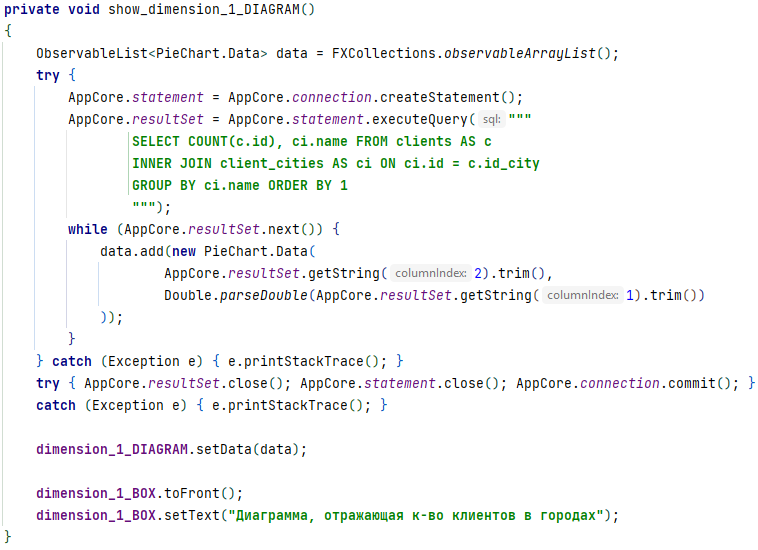


Рисунок Д.1 – Код создания и отрисовки круговой диаграммы



Рисунок Д.2 – Код создания и отрисовки двухмерной диаграммы

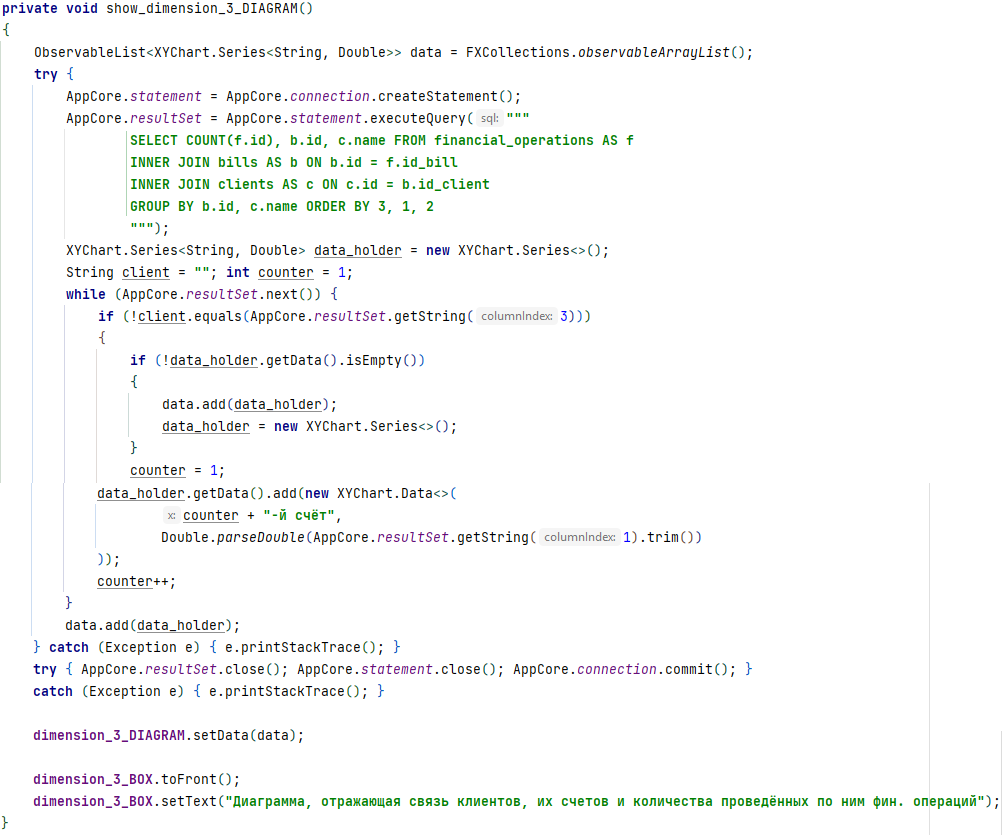


Рисунок Д.3 – Код создания и отрисовки трехмерной диаграммы

Пользователь может выгружать данные из запроса в виде отчёта в формате .html и Excel. Код данных действий прописан в файле «Queries\_Contraller.java». (рисунки Д.4-Д.5)

Для выгрузки отчёта в формате .html код создает новый документ формата .html. И потом программа автоматически прописывает нужные теги на языке HTML 5.



Рисунок Д.4 – Код создания и выгрузки отчёта в формате .html

В ходе выполнения кода программа создаст файл Microsoft Exsel, в котором откроет лист и задаст ему имя, после чего заполнит ячейки выбранными данными и в конце выделит таблицу по контуру.

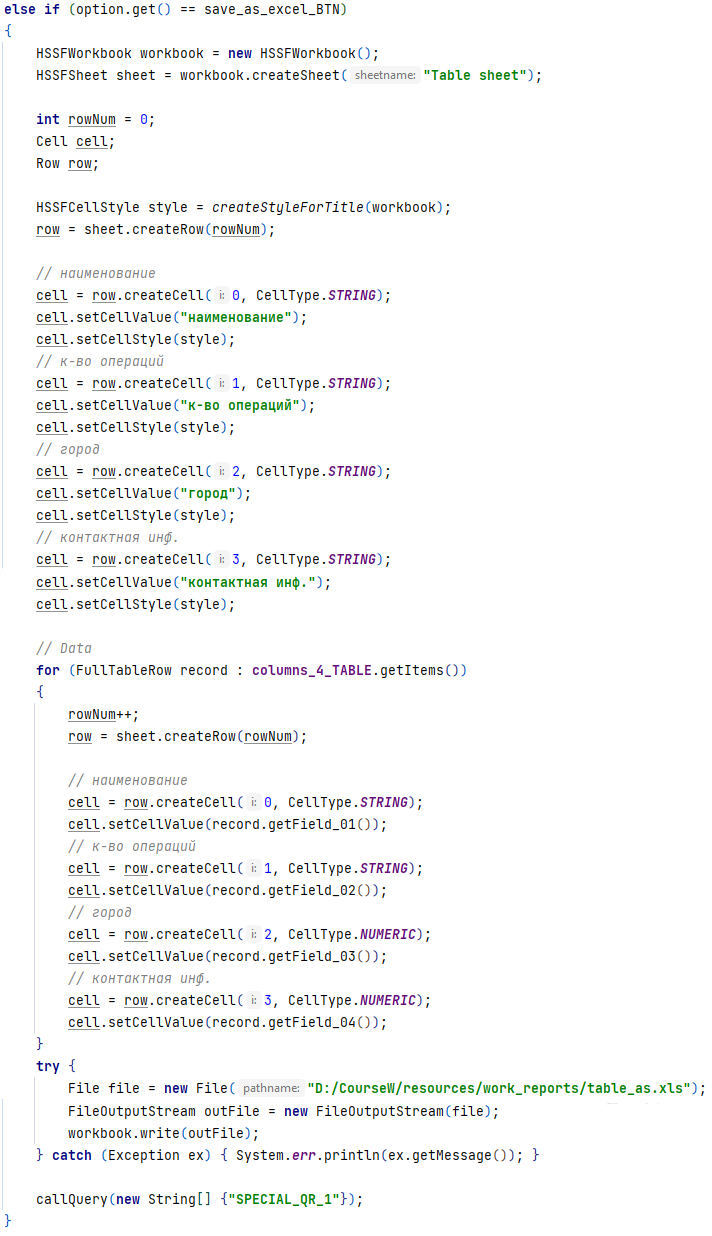


Рисунок Д.5 – Код создания и выгрузки отчёта в формате Excel