ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

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

Тема работы:

«Разработка информационной системы “Фирмы-провайдеры”»

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

|  |  |
| --- | --- |
| Разработал: | Саевский О. В. |
| ст. гр. ПИ-19а (подпись)  (дата) | |

Донецк – 2022

РЕФЕРАТ

Пояснительная записка к курсовому проекту содержит: 86 страниц, 98 рисунков, 1 таблица, 5 источников, 7 приложений.

Объект исследования – информационная система «Фирмы-провайдеры».

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

Результат выполнения проекта – разработанная информационная система «Фирмы-провайдеры», использующая вышеперечисленные механизмы СУБД.

БАЗА ДАННЫХ, EF, СУБД, POSTGRESQL, ЗАЩИТА НА УРОВНЕ СТРОК, РОЛЬ, ТРИГГЕР, ЗАПРОС, ДОМЕН, ПАРТИЦИРОВАНИЕ, C#

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc107948508)

[1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 6](#_Toc107948509)

[2 ОБОСНОВАНИЕ ВЫБОРА СУБД, ОПИСАНИЕ ВОЗМОЖНОСТЕЙ СУБД 8](#_Toc107948510)

[3 ОБОСНОВАНИЕ ВЫБОРА ИНСТРУМЕНТАЛЬНЫХ СРЕДСТВ ДЛЯ НАПИСАНИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ 9](#_Toc107948511)

[3.1 Невизуальные компоненты для работы с данными 9](#_Toc107948512)

[3.2 Визуальные компоненты для работы с данными 9](#_Toc107948513)

[3.3 Разработка шаблонов приложений для работы с шаблонами базы данных 11](#_Toc107948514)

[4 ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ 14](#_Toc107948515)

[4.1 Проектирование концептуальной модели БД 14](#_Toc107948516)

[4.2 Создание таблиц, доменов, индексов, последовательностей 15](#_Toc107948517)

[4.3 Разработка триггеров 19](#_Toc107948518)

[4.4 Организация многоролевого доступа к данным 22](#_Toc107948519)

[4.5 Разграничение доступа к данным на уровне строк 23](#_Toc107948520)

[4.6 Партицирование одной из основных таблиц БД 24](#_Toc107948521)

[4.7 Проектирование запросов к базе данных 25](#_Toc107948522)

[4.8 Разработка модифицируемого представления 38](#_Toc107948523)

[5 РАЗРАБОТКА ПРИЛОЖЕНИЯ 40](#_Toc107948524)

[5.1 Формы и компоненты для работы в роли “Сотрудник фирмы” 40](#_Toc107948525)

[5.2 Формы и компоненты для работы в роли “Абонент” 41](#_Toc107948526)

[5.3 Формы и компоненты для работы в роли “Администратор” 42](#_Toc107948527)

[5.4 Экспортирование результата запроса в Excel 45](#_Toc107948528)

[6 ТЕСТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ 46](#_Toc107948529)

[ВЫВОДЫ 48](#_Toc107948530)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 49](#_Toc107948531)

ПРИЛОЖЕНИЕ А [ТЕХНИЧЕСКОЕ ЗАДАНИЕ 50](#_Toc107948533)

ПРИЛОЖЕНИЕ Б [ЛИСТИНГ ШАБЛОНОВ 56](#_Toc107948535)

ПРИЛОЖЕНИЕ В [ЛИСТИНГ СЕРВЕРНОГО ПРИЛОЖЕНИЯ 60](#_Toc107948537)

[ПРИЛОЖЕНИЕ Г ЛИСТИНГ КЛИЕНТСКОГО ПРИЛОЖЕНИЯ 64](#_Toc107948538)

ПРИЛОЖЕНИЕ Д [РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 84](#_Toc107948540)

ПРИЛОЖЕНИЕ Е [РУКОВОДСТВО АДМИНИСТРАТОРА 85](#_Toc107948542)

[ПРИЛОЖЕНИЕ Ж ОТЧЕТ О ПРОВЕРКЕ НА ЗАИМСТВОВАНИЯ 86](#_Toc107948543)

# ВВЕДЕНИЕ

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

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

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

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

Для разработки системы были использованы следующие инструменты: СУБД PostgreSQL, язык программирования C# 8.0, интегрированная среда разработки Visual Studio, редактор SQL и ORM Entity Framework Core.

Разработанная в ходе выполнения курсового проекта информационная система может быть использована в реальных условиях для учета пользователей электронной почты.

# 1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

В задании курсовой работы требуется разработать базу данных для автоматизации учета пользователей электронной почты. Рассмотрим особенности данной системы.

Для автоматизации учета пользователей электронной почты необходима информация о фирмах-провайдерах (название фирмы, тип собственности (государственная, частная , ЗАО, ОАО,…), адрес, телефон, год начала работы), о заключенных с ними договорах (абонент (ФИО или название предприятия, тип (частное лицо, ВУЗ, школа, предприятие,…), физический адрес, адрес электронной почты), дата подключения, стоимость подключения, стоимость пересылки 1 Mb информации) и о предоставленных провайдерами услугах (абонент, дата предоставления, объем сообщения (в Мb)).

В ходе декомпозиции предметной области были выделены следующие таблицы: фирмы-провайдеры, абоненты, виды абонентов, типы собственности, услуги, контракты.

Исходя из условия многопользовательской системы, были выделены три группы пользователей: системный администратор, сотрудник фирмы, абонент.

Основными задачами администратора являются:

* Создание новых пользователей БД;
* Добавление, удаление прав пользователей;
* Добавление, редактирование, удаление фирм-провайдеров;
* Управление справочниками БД.

К основным задачам сотрудника фирмы относятся:

* Просмотр, добавление, редактирование, удаление услуг.
* Просмотр, добавление, редактирование, удаление контрактов.

Основные задачи абонента:

* Просмотр, добавление, удаление контрактов;
* Просмотр информации о себе;

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

# 2 ОБОСНОВАНИЕ ВЫБОРА СУБД, ОПИСАНИЕ ВОЗМОЖНОСТЕЙ СУБД

В качестве системы управления базой данных была выбрана PostgreSQL.

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

– поддержка БД неограниченного размера;

– мощные и надёжные механизмы транзакций и репликации;

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

– наследование;

– легкая расширяемость.

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

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

# 3 ОБОСНОВАНИЕ ВЫБОРА ИНСТРУМЕНТАЛЬНЫХ СРЕДСТВ ДЛЯ НАПИСАНИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ

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

Для разработки пользовательского графического интерфейса был выбран графический фреймворк WPF.

Для взаимодействия с СУБД используется самая популярная ORM система для C# – Entity Framework Core, он обеспечивает доступ ко многим функциям СУБД.

Чтобы генерировать Excel файлы с диаграммами был выбран C# модуль Microsoft.Office.Tools.Excel.

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

## 3.1 Невизуальные компоненты для работы с данными

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

## 3.2 Визуальные компоненты для работы с данными

Для отображения данных из таблиц был выбран элемент управления DataGrid, который является достаточно гибким и позволяет автоматизировать вывод информации пользователю. Внешний вид описывается на языке XAML и привязывается к списку данных. Пример использования данного элемента управления для отображения таблицы из базы данных представлен на рисунках 3.1 и 3.2.

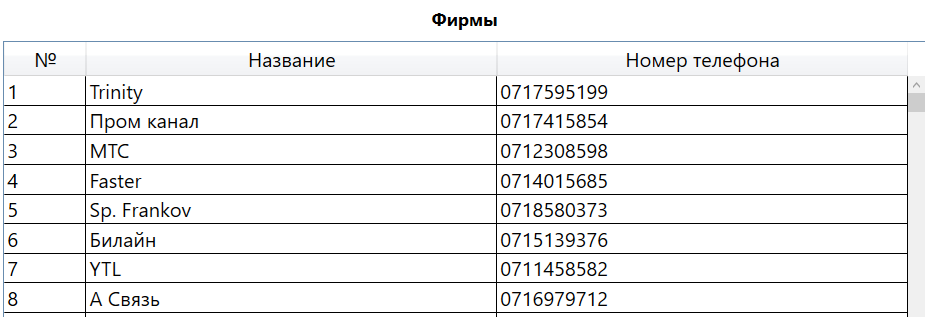


Рисунок 3.1 – Пример использования DataGrid для отображения данных таблицы

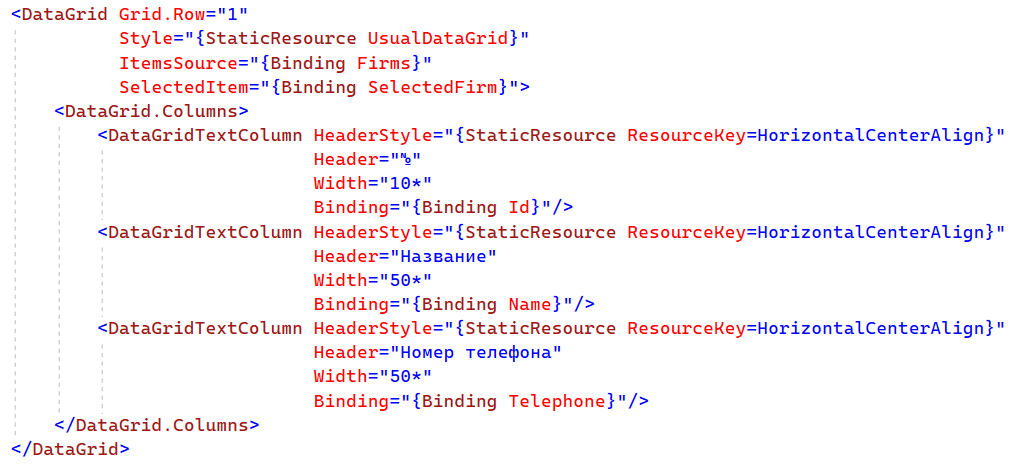


Рисунок 3.2 – Пример использования DataGrid для отображения данных таблицы (код программы)

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

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

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

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

Код формы авторизации, а также одной из таблиц показан на рисунках 3.3 - 3.4.



Рисунок 3.3 – Код формы авторизации

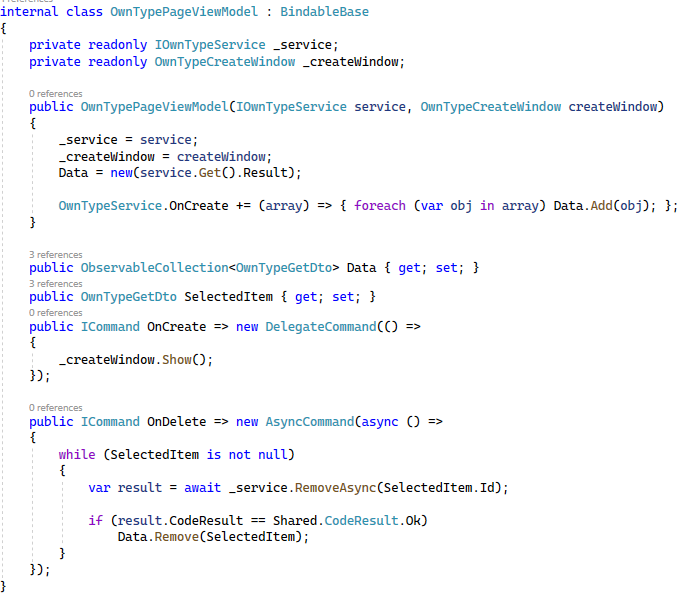


Рисунок 3.4 – Код формы работы с формой таблицы “Типы собственности”

# 4 ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ

## 4.1 Проектирование концептуальной модели БД

После декомпозиции предметной области была разработана концептуальная модель БД, изображённая на рисунке 4.1.

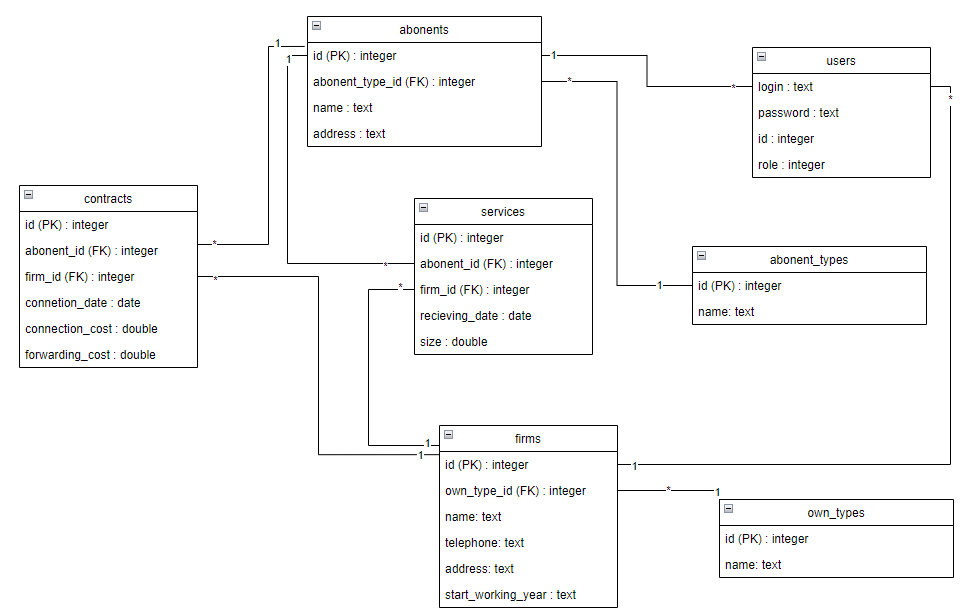


Рисунок 4.1 – Концептуальная модель БД

Концептуальная модель БД содержит следующие основные таблицы: абоненты (abonents), контракты (contracts), услуги (services), фирмы (firms), пользователи (users). Таблицы-справочники: типы собственности (own\_types), виды абонентов (abonent\_types).

## 4.2 Создание таблиц, доменов, индексов, последовательностей

SQL-запрос для создания таблицы “Абоненты” изображён на рисунке 4.2.

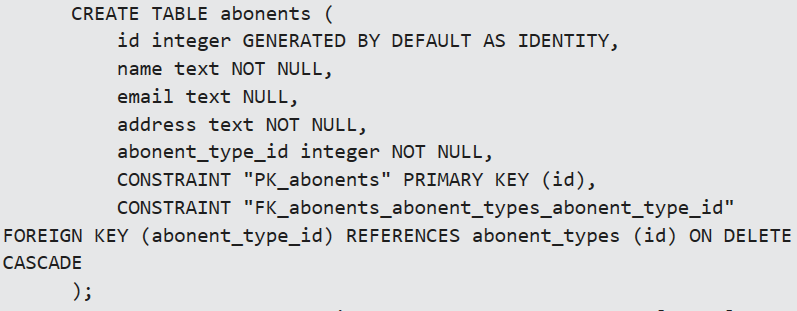


Рисунок 4.2 – SQL-запрос создания таблицы “Абоненты”

SQL-запрос для создания таблицы “Фирмы” изображён на рисунке 4.3.

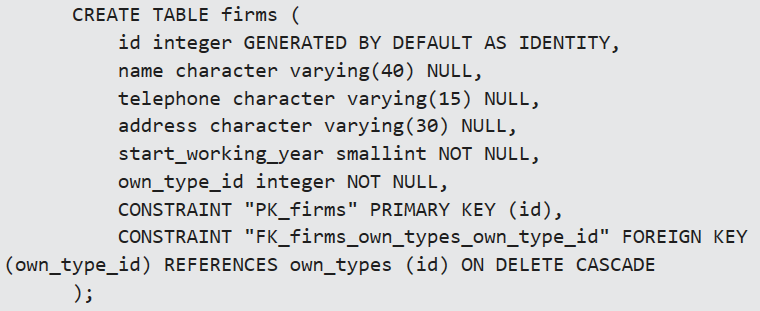


Рисунок 4.3 – SQL-запрос создания таблицы “Фирмы”

SQL-запрос для создания таблицы “Контракты” изображён на рисунке 4.4.

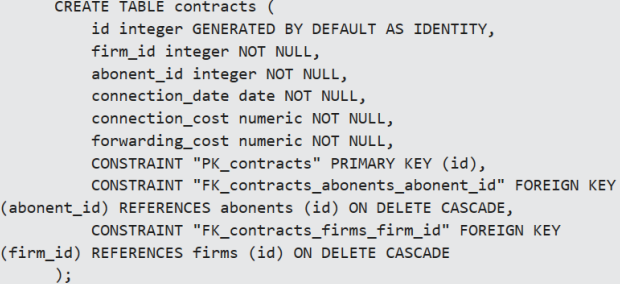


Рисунок 4.4 – SQL-запрос создания таблицы “Контракты”

SQL-запрос для создания таблицы “Услуги” изображён на рисунке 4.5.

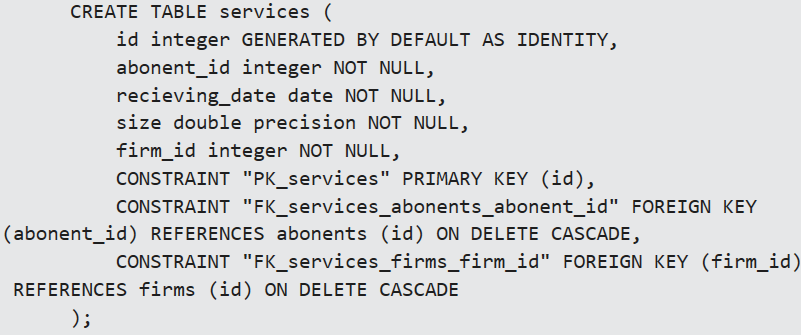


Рисунок 4.5 – SQL-запрос создания таблицы “Услуги”

SQL-запрос для создания таблицы “Пользователи” изображён на рисунке 4.6.

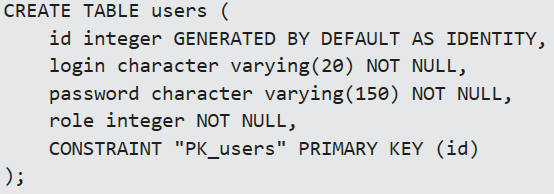


Рисунок 4.6 – SQL-запрос создания таблицы “Пользователи”

SQL-запрос для создания таблицы “Типы собственности” изображён на рисунке 4.7.

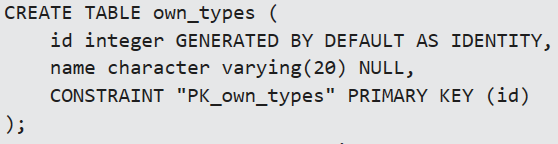


Рисунок 4.7 – SQL-запрос создания таблицы “Типы собственности”

SQL-запрос для создания таблицы “Виды абонентов” изображён на рисунке 4.8.

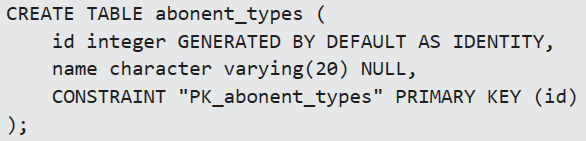


Рисунок 4.8 – SQL-запрос создания таблицы “Виды абонентов”

В созданных таблицах использованы домены, изображённые на рисунке 4.9. Домен по сути представляет собой тип данных с дополнительными условиями (ограничивающими допустимый набор значений).

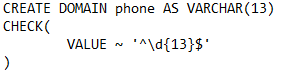


Рисунок 4.9 – Домен “Номер телефона” (13 цифр)

Также для таблиц были созданы индексы (рис. 4.10). Операции поиска выборки (SELECT) данных из таблиц по значениям их полей могут быть существенно ускорены путем использования индексации данных. Индекс содержит упорядоченный (в алфавитном или числовом порядке) список содержимого столбцов или группы столбцов в индексируемой таблице с идентификаторами этих строк.

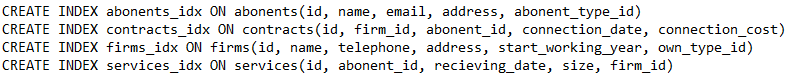


Рисунок 4.10 – SQL-запросы для создания индексов

Согласно техническому заданию разработаны последовательности (рис. 4.11) для таблиц, которые с помощью триггеров задают значения первичного ключа для вновь добавляемой записи.

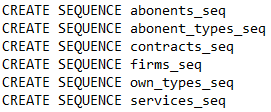


Рисунок 4.11 – SQL-запросы для создания последовательностей

## 4.3 Разработка триггеров

Для всех таблицы были добавлены триггеры типа BEFORE INSERT, которые используют созданные последовательности и генерируют идентификаторы новых записей. SQL-запрос создания одного из этих триггеров представлен на рисунке 4.12.

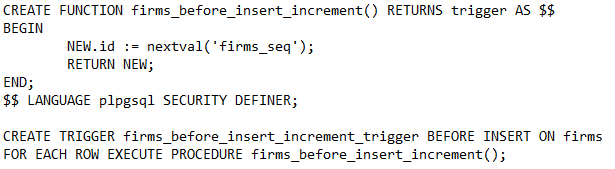


Рисунок 4.12 – SQL-запрос создания триггера BEFORE INSERT для таблицы “Фирмы”

Для таблицы пользователей был создан триггер типа AFTER UPDATE, который удаляет пользователя из старой группы и добавляет в новую (в случае если администратор поменял его роль). SQL-запрос этого триггера изображён на рисунке 4.13.

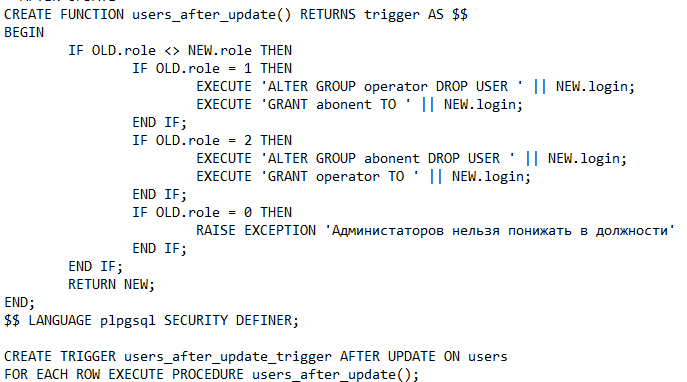


Рисунок 4.13 – SQL-запрос создания триггера AFTER UPDATE для таблицы “Пользователи”

Триггер типа AFTER INSERT для таблицы пользователей (рис. 4.14) создаёт роль для нового пользователя.

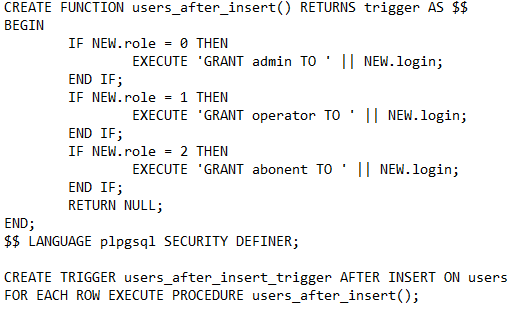


Рисунок 4.14 – SQL-запрос создания триггера AFTER INSERT для таблицы “Пользователи”

Триггер типа AFTER DELETE для таблицы пользователей (рис. 4.15) удаляет роль удаленного пользователя.

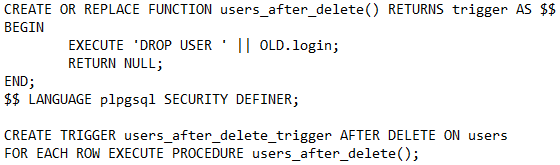


Рисунок 4.15 – SQL-запрос создания триггера AFTER DELETE для таблицы “Пользователи”

Триггер типа BEFORE DELETE для таблицы фирм (рис. 4.16) проверяет есть ли контракты у удаляемой фирмы.

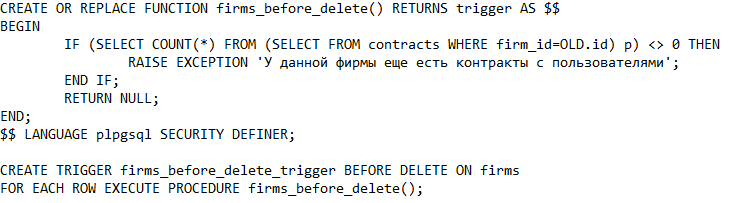


Рисунок 4.16 – SQL-запрос создания триггера BEFORE DELETE для таблицы “Фирмы”

Триггер типа BEFORE INSERT для таблицы фирм (рис. 4.17) проверяет есть ли уже фирма с таким же названием и номером телефона.

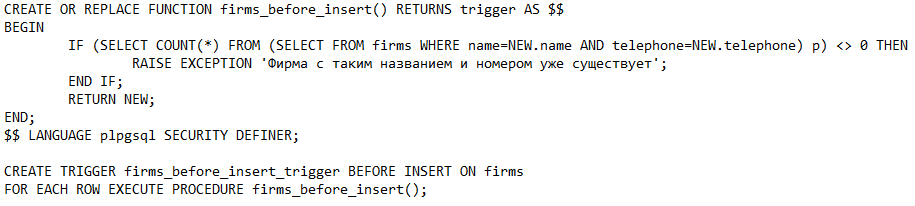


Рисунок 4.17 – SQL-запрос создания триггера BEFORE INSERT для таблицы “Фирмы”

Триггер типа BEFORE UPDATE для таблицы фирм (рис. 4.18) проверяет есть ли уже фирма с таким же названием и номером телефона.

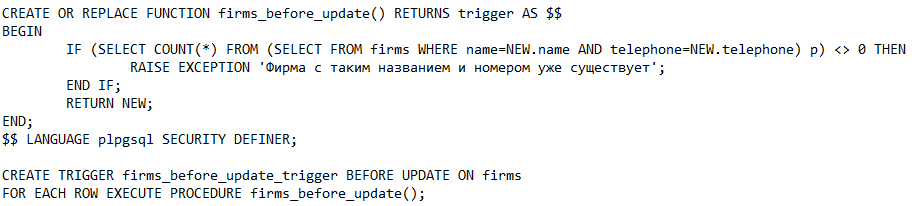


Рисунок 4.18 – SQL-запрос создания триггера BEFORE UPDATE для таблицы “Фирмы”

## 4.4 Организация многоролевого доступа к данным

Для информационной системы были созданы следующие роли: администратор (admin), сотрудник фирмы (operator), абонент (abonent). SQL-запросы для создания этих ролей изображены на рисунке 4.19.



Рисунок 4.19 – Создание ролей

На рисунке 4.20 на примере роли “Сотрудник фирмы” продемонстрированы SQL-запросы создания прав доступа к нужным таблицам.

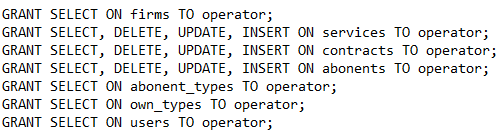


Рисунок 4.20 – Права доступа для роли “Сотрудник фирмы”

## 4.5 Разграничение доступа к данным на уровне строк

Чтобы разграничить данные на уровне строк для начала необходимо включить защиту на уровне строк для нужных таблиц. SQL-запросы изображены на рисунке 4.21.

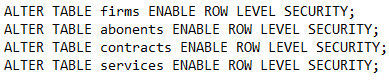


Рисунок 4.21 – SQL-запросы включения защиты на уровне строк для таблиц

Затем для каждой роли пользователей создаются политики защиты, где описано, к каким строкам таблицы роль имеет доступ. На рисунке 4.22 на примере роли “Сотрудник фирмы” продемонстрировано создание таких политик.

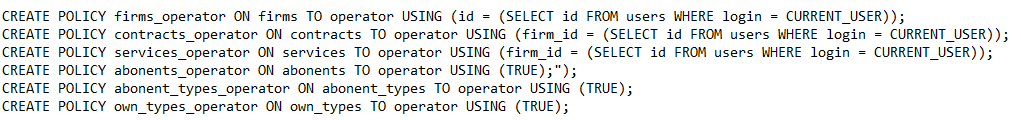


Рисунок 4.22 – Создание политик для роли “Сотрудник фирмы”

В выражении USING прописывается условие доступа к строке.

## 4.6 Партицирование одной из основных таблиц БД

В соответствии с техническим задание было выполнено партицирование (секционирование) по диапазону таблицы “Контракты” (рис. 4.23). Секционированием данных называется разбиение одной большой логической таблицы на несколько меньших физических секций. Секционирование позволяет перенести редко используемые данные на более дешёвые носители и в определённых ситуациях кардинально увеличить производительность.

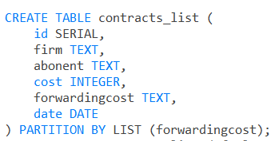


Рисунок 4.23 – SQL-запрос партицирования таблицы «Контракты» по списку

Для этой таблицы также были созданы секции (рис. 4.24), в том числе и секция default.

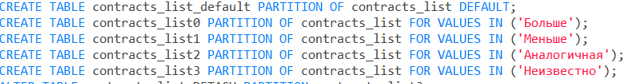


Рисунок 4.24 – SQL-запросы создания секций

В зависимости от идентификатора запись попадает в нужную секцию.

## 4.7 Проектирование запросов к базе данных

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

– симметричное внутренне соединение с условием: вывести абонентов с указанным типом (рис. 4.25-4.26);

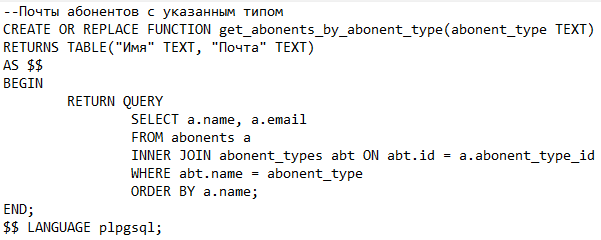


Рисунок 4.25 – Запрос

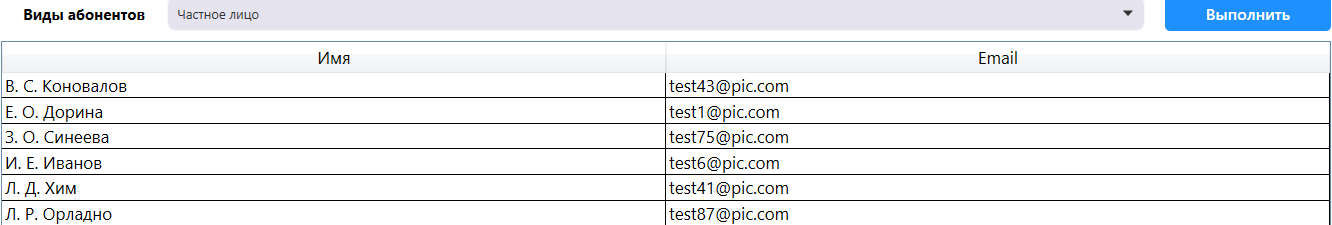


Рисунок 4.26 – Результат запроса

– симметричное внутренне соединение с условием: вывести фирмы с заданным типом собственности (рис. 4.27-4.28);

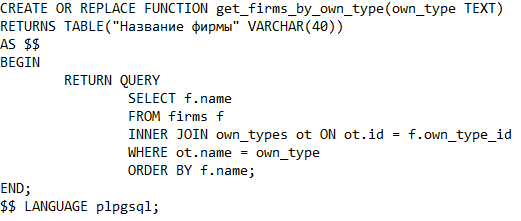


Рисунок 4.27 – Запрос

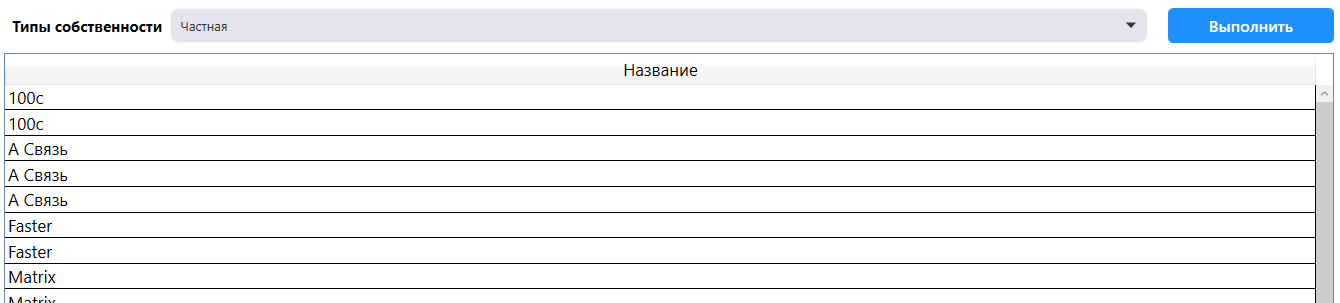


Рисунок 4.28 – Результат запроса

– симметричное внутренне соединение с условием: вывести все фирмы, которые предоставляли услуги в указанную дату (рис. 4.29-4.30);

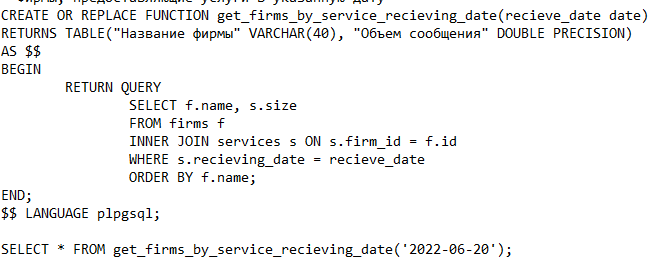


Рисунок 4.29 – Запрос

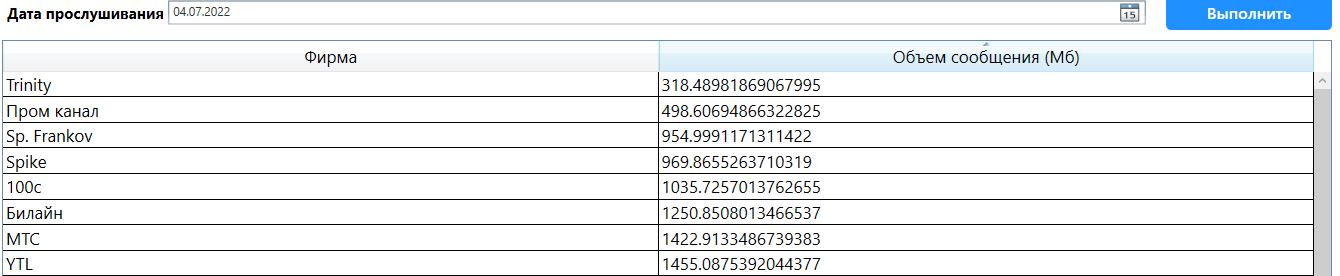


Рисунок 4.30 – Результат запроса

– симметричное внутренне соединение с условием: вывести всех абонентов и объем сообщения услуг, предоставленных в указанную дату (рис. 4.31-4.32);

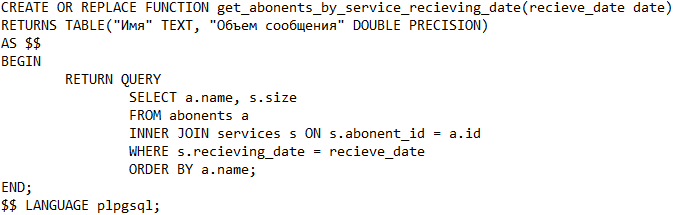


Рисунок 4.31 – Запрос

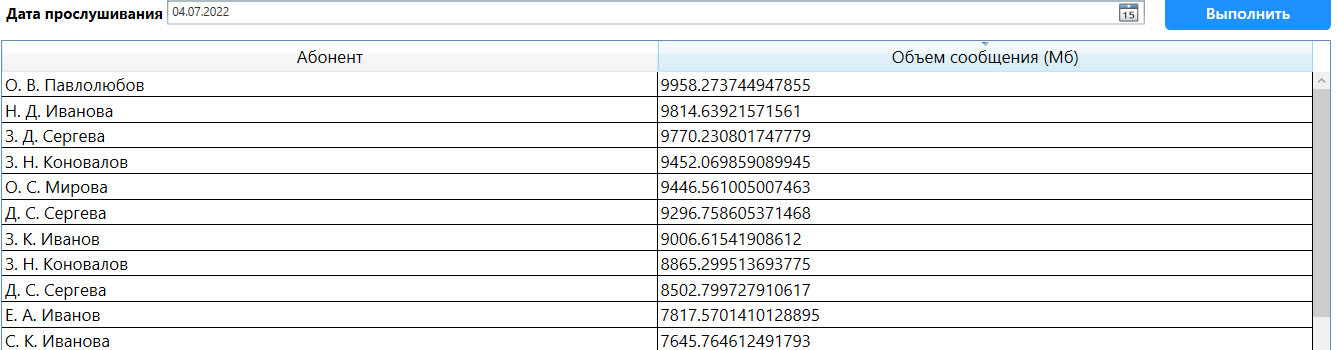


Рисунок 4.32 – Результат запроса

– симметричное внутренне соединение без условия: Вывести абонентов и стоимость подключения их контрактов (рис. 4.33-4.34);

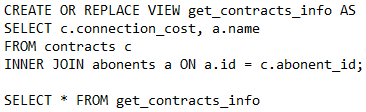


Рисунок 4.33 – Запрос

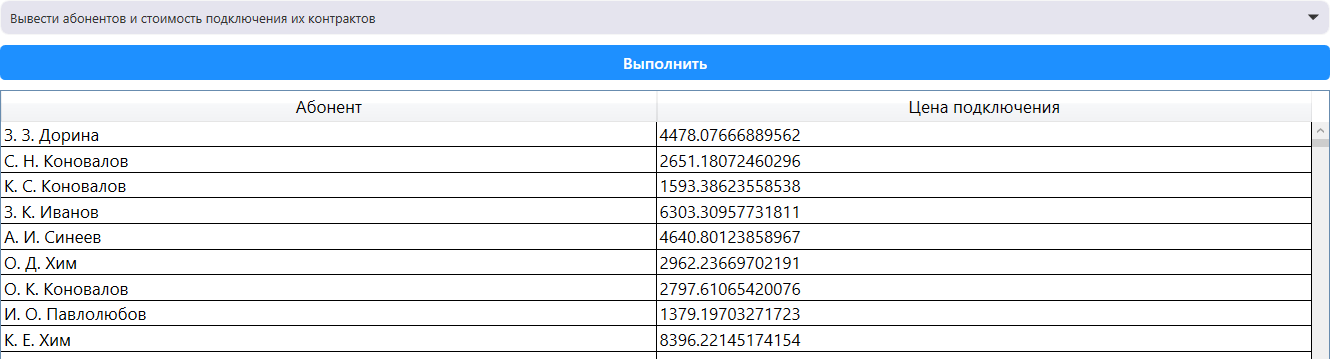


Рисунок 4.34 – Результат запроса

– симметричное внутренне соединение без условия: вывести абонентов и объем сообщения их услуг (рис. 4.35-4.36);

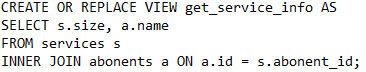


Рисунок 4.35 – Запрос

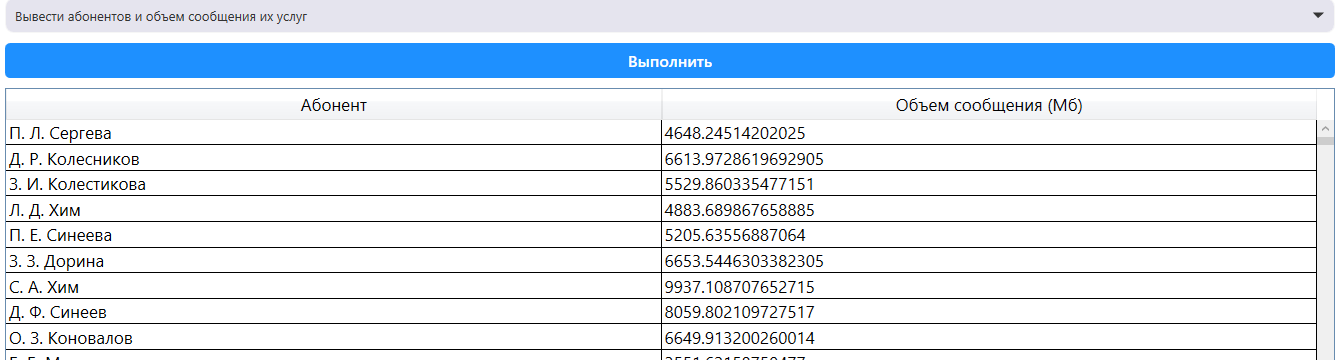


Рисунок 4.36 – Результат запроса

– симметричное внутренне соединение без условия: вывести абонентов и их типы (рис. 4.37-4.38);

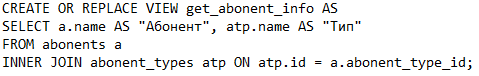


Рисунок 4.37 – Запрос

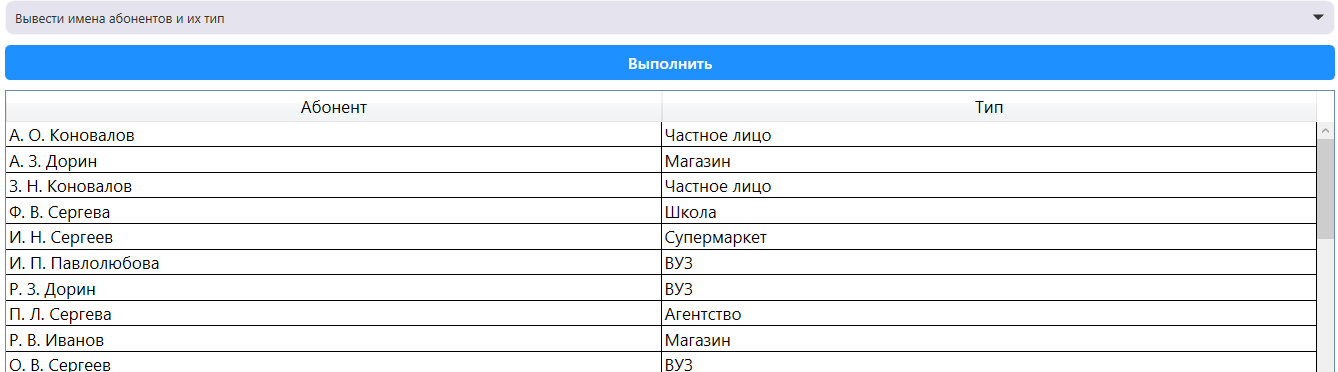


Рисунок 4.38 – Результат запроса

– левое внешнее соединение: вывести все контракты абонентов, у которых указана почта (рис. 4.39-4.40);

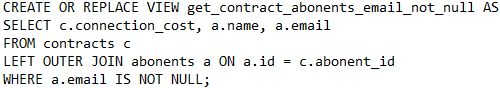


Рисунок 4.39 – Запрос

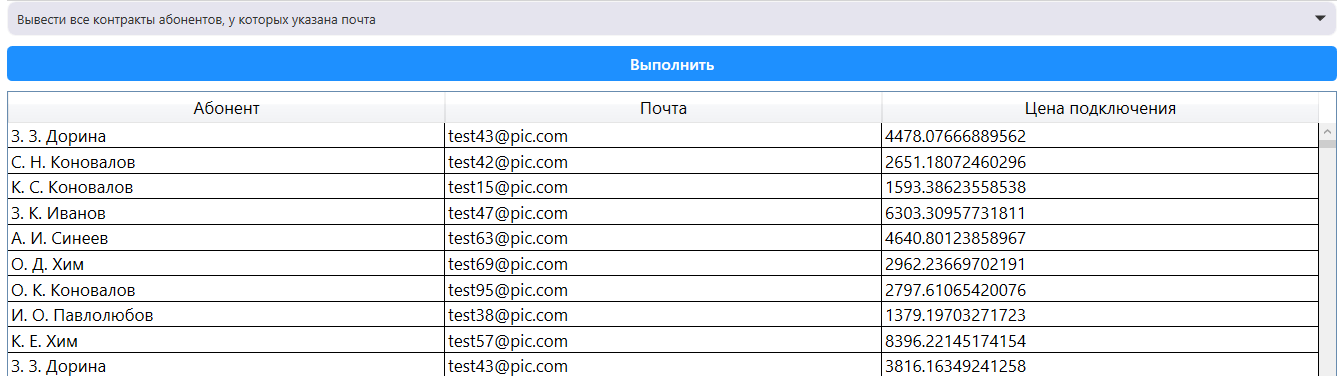


Рисунок 4.40 – Результат запроса

– правое внешнее соединение: вывести все фирмы, которые хоть раз отказывали услуги (рис. 4.41-4.42);

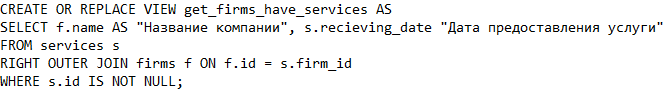


Рисунок 4.41 – Запрос

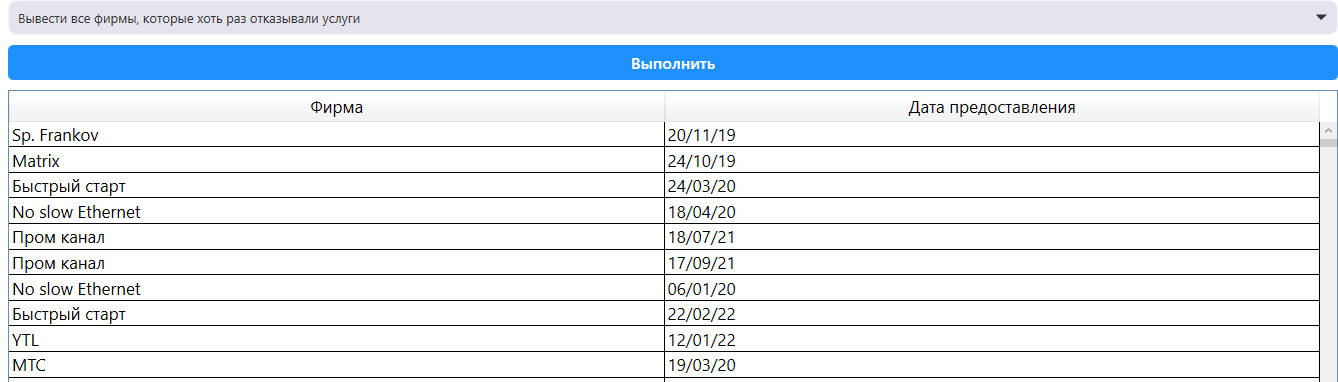


Рисунок 4.42 – Результат запроса

– запрос на запросе: вывести информацию о фирмах, открывшихся в указанный период, и уже предоставлявших услуги пользователям (рис. 4.43-4.44);

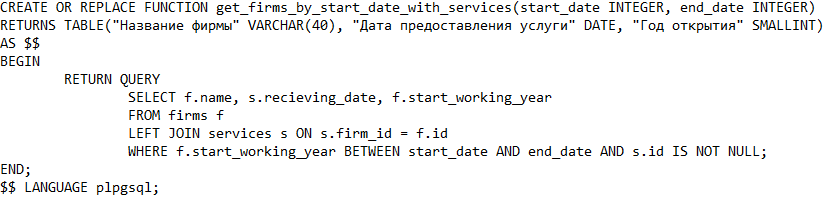


Рисунок 4.43 – Запрос

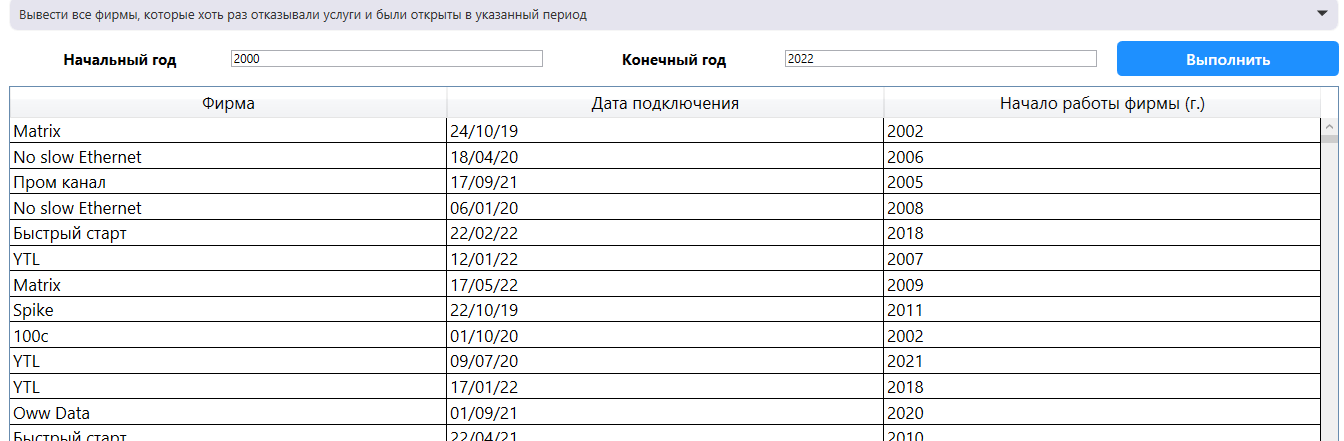


Рисунок 4.44 – Результат запроса

– итоговый запрос без условия: вывести общее число предоставленных мегабайт фирм (рис. 4.45-4.46);

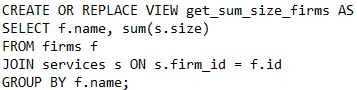


Рисунок 4.45 – Запрос

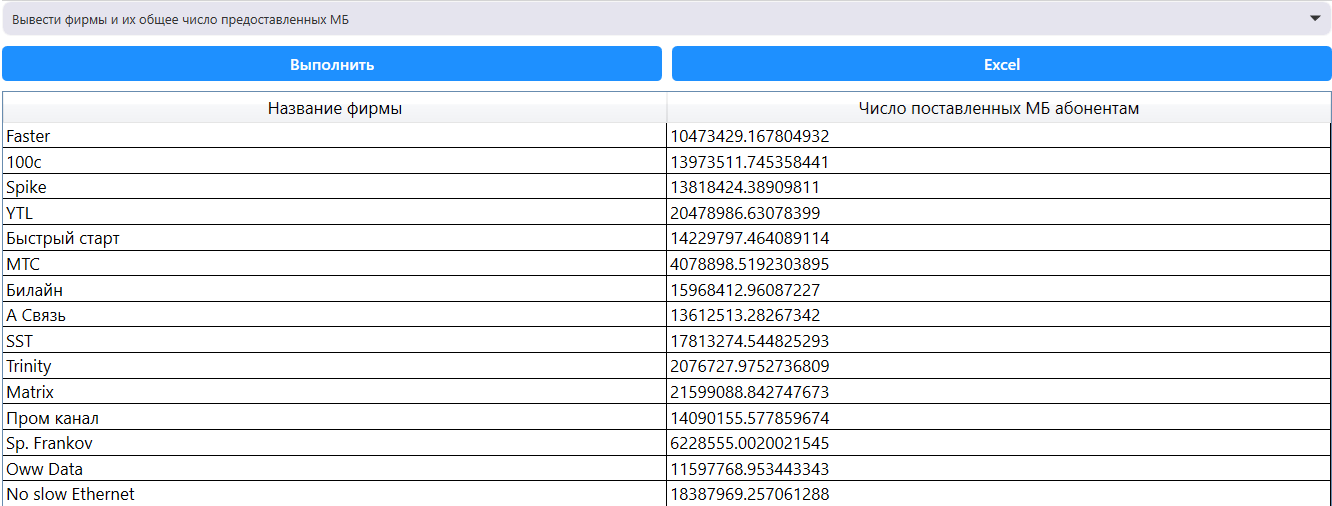


Рисунок 4.46 – Результат запроса

– итоговый запрос с условием на данные: вывести количество фирм с указанным типом собственности (рис. 4.47-4.48);

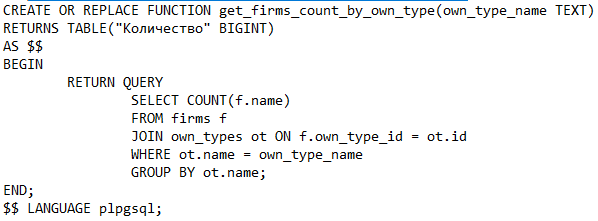


Рисунок 4.47 – Запрос запроса

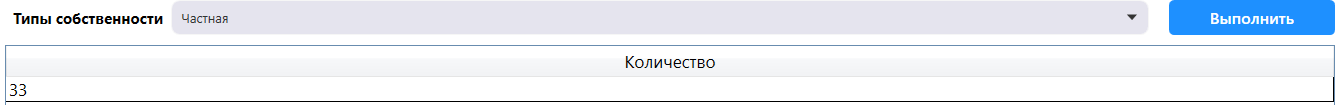


Рисунок 4.48 – Результат

– итоговый запрос с условием на группы: вывести абонентов, которые составили контрактов на подключение в сумме на цену более указанного числа (рис. 4.49-4.50);

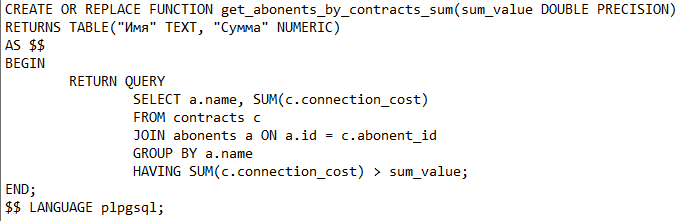


Рисунок 4.49 – Запрос

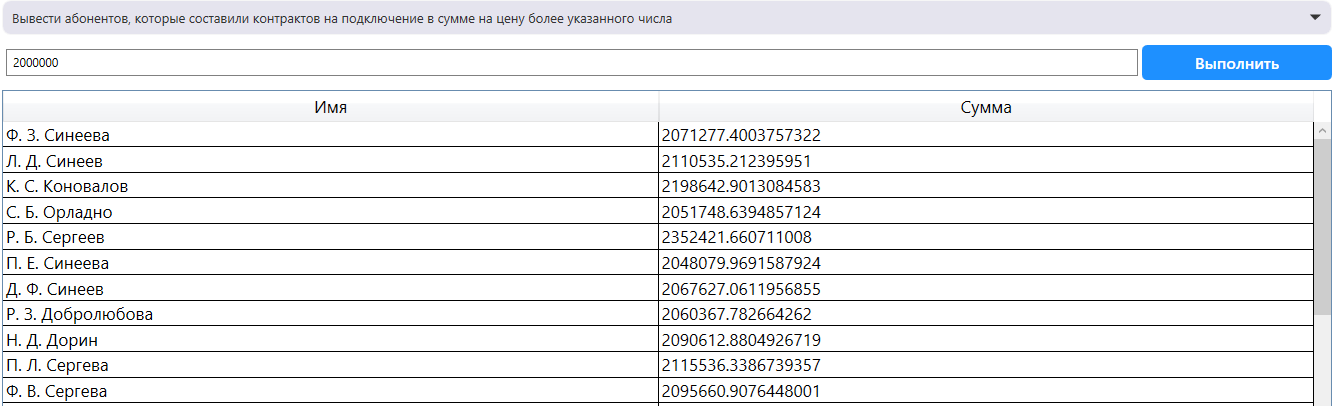


Рисунок 4.50 – Результат запроса

– итоговый запрос с условием на данные и на группы: вывести абонентов, которые составили контрактов на подключение в сумме на цену более указанного числа и которые начинаются в указанную дату (рис. 4.51-4.52);

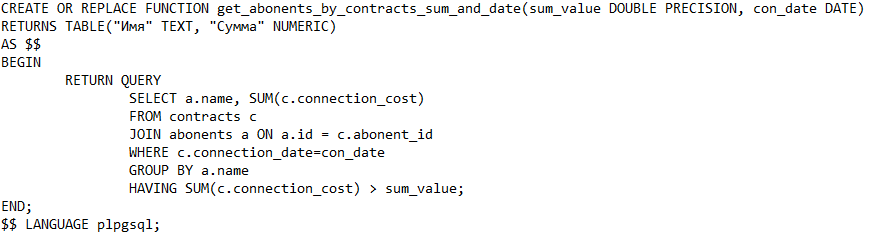


Рисунок 4.51 – Запрос

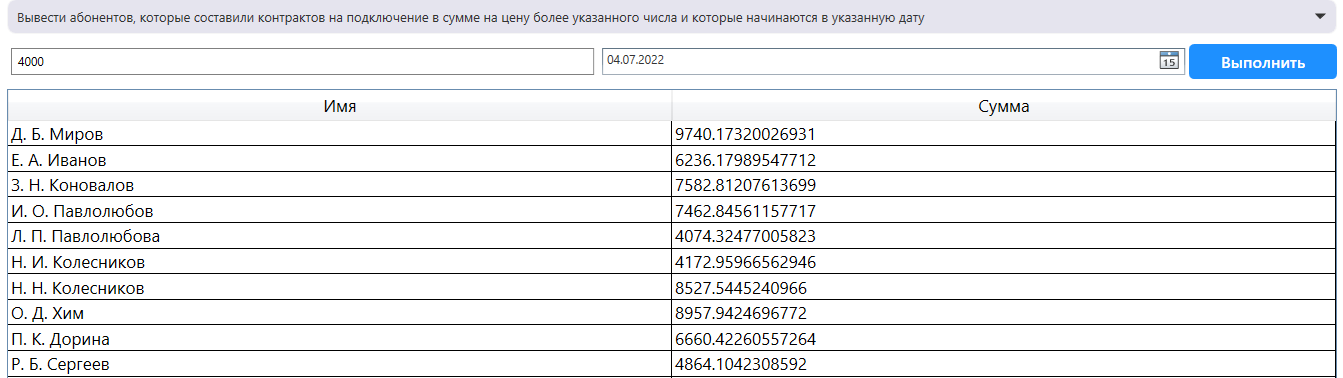


Рисунок 4.52 – Результат запроса

– запрос на запросе по принципу итогового запроса: вывести фирмы и их общую прибыль за подключение до и после инфляции (уменьшение на 30%), в период за между двумя датами (рис. 4.53-4.54);

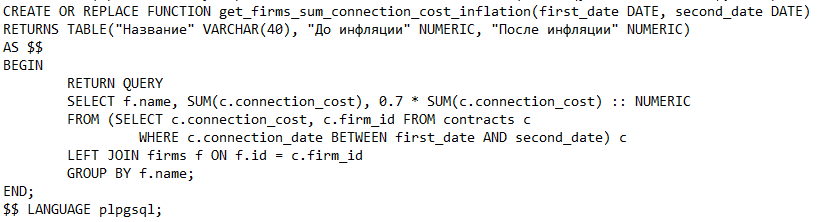


Рисунок 4.53 – Запрос

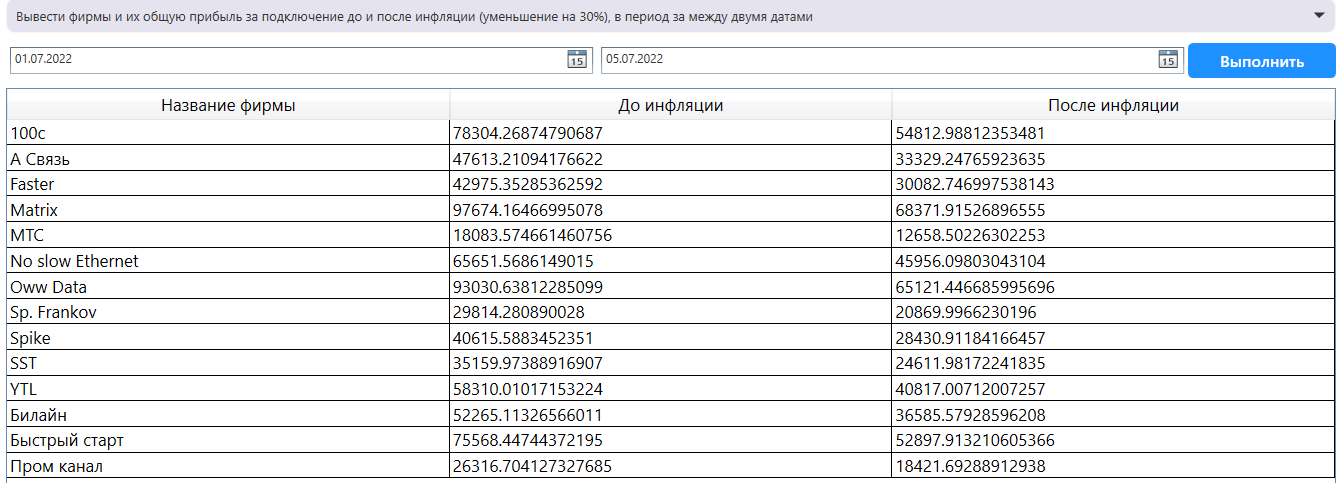


Рисунок 4.54 – Результат запроса

– запрос с подзапросом: вывести фирмы и их общую прибыль за подключения, сумма которой больше средней прибыли за подключения (рис. 4.55-4.56);

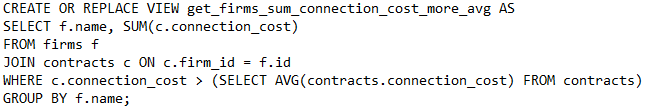


Рисунок 4.55 – Запрос

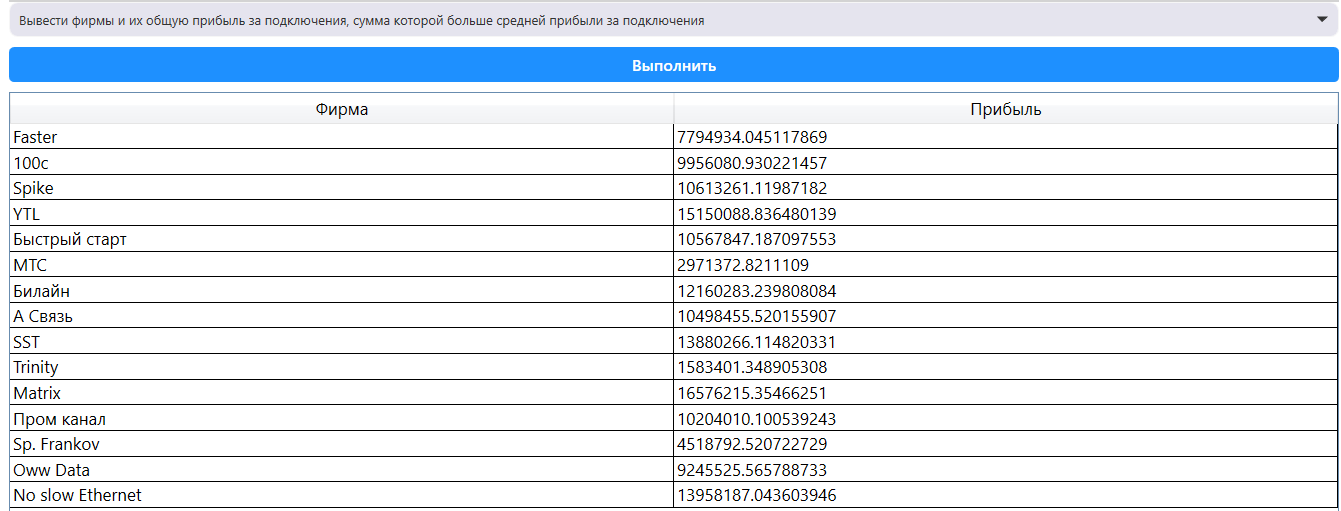


Рисунок 4.56 – Результат запроса

– итоговый запрос с условием на данные по маске: вывести типы собственности и количество фирм, в названии которых встречается введенная подстрока (рис. 4.57-4.58);

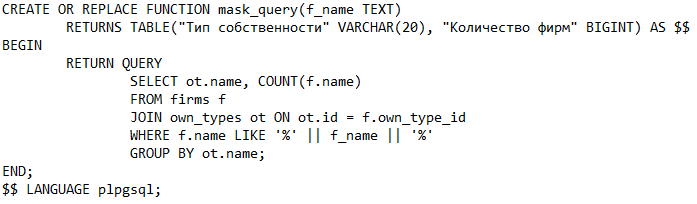


Рисунок 4.57 – Запрос

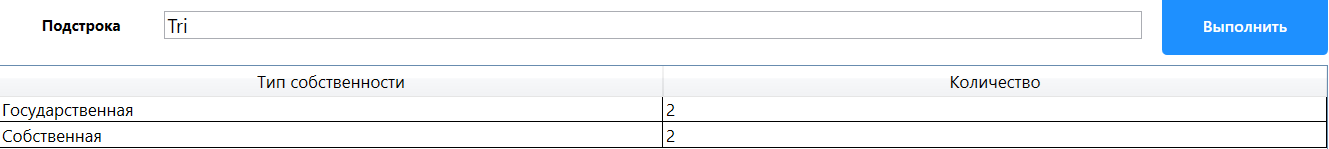


Рисунок 4.58 – Результат запроса

– итоговый запрос с использованием CASE: вывести среднюю стоимость подключения указанной фирмы (рис. 4.59-4.60);

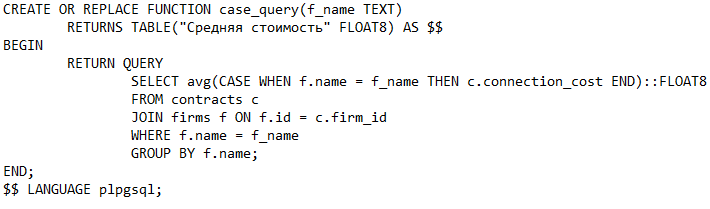


Рисунок 4.59 – Запрос

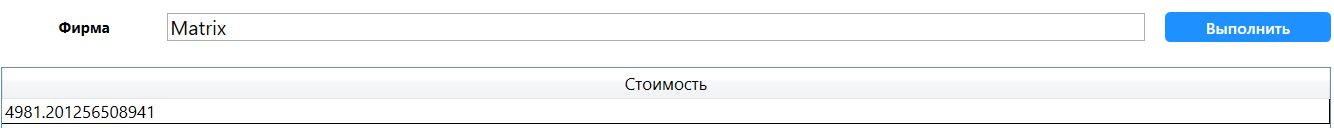


Рисунок 4.60 – Результат запроса

– итоговый запрос с использованием объединения: вывести количество контрактов и предоставляемых услуг (рис. 4.61-4.62);

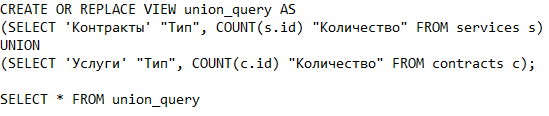


Рисунок 4.61 – Запрос

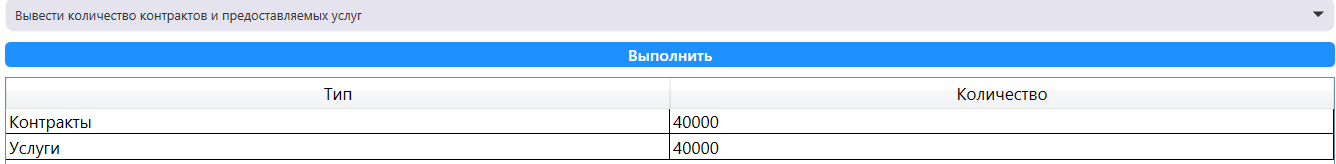


Рисунок 4.62 – Результат запроса

– запрос с подзапросом с использованием IN: вывести все контракты, фирмы которых имеют указанный тип собственности (рис. 4.63-4.64);

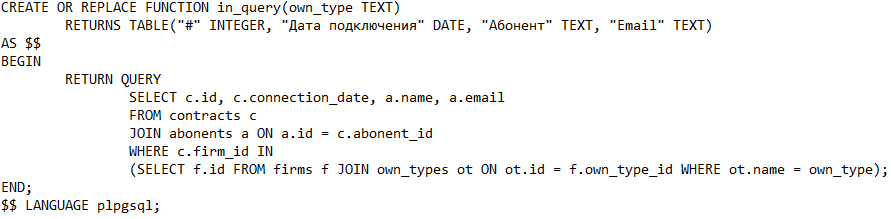


Рисунок 4.63 – Запрос

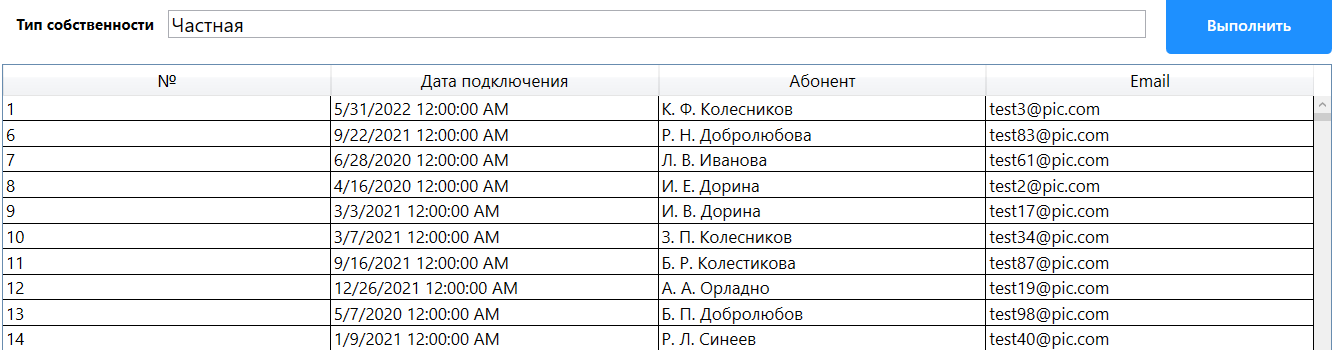


Рисунок 4.64 – Результат запроса

– запрос с подзапросом с использованием NOT IN: вывести все контракты, фирмы которых не имеют указанный тип собственности (рис. 4.65-4.66);

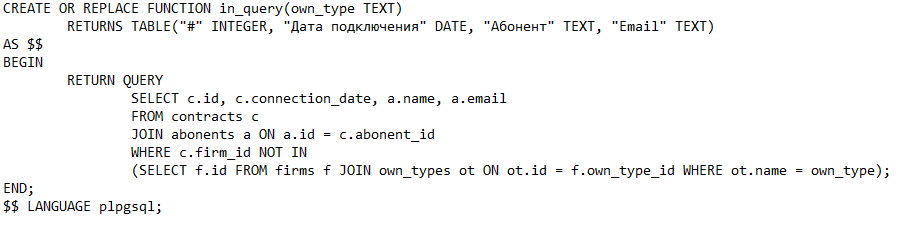


Рисунок 4.65 – Запрос

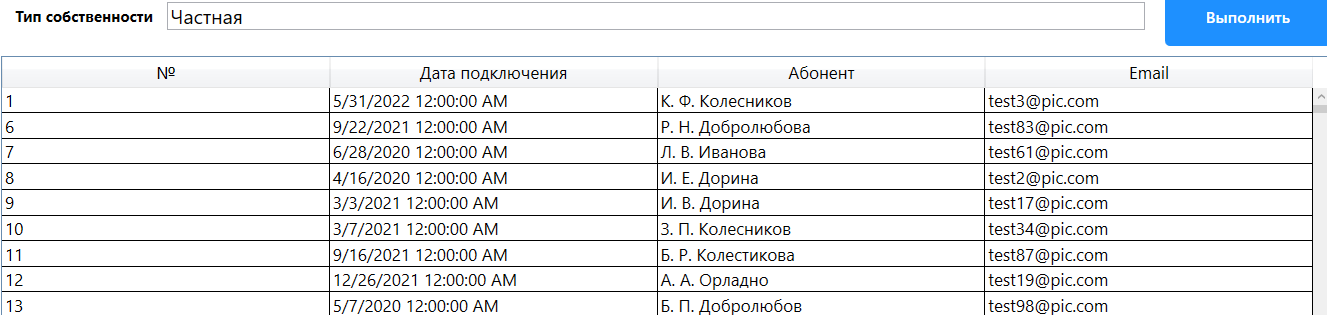


Рисунок 4.66 – Результат запроса

## 4.8 Разработка модифицируемого представления

Для курсового проекта было создано модифицируемое представление firms\_view (рис. 4.67).

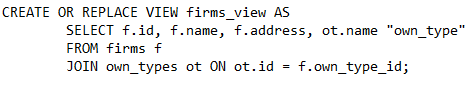


Рисунок 4.67 – SQL-запрос создания представления

Модифицируемость этого представления обеспечивают триггеры типа INSTEAD изображённые на рисунках 4.68-4.70.

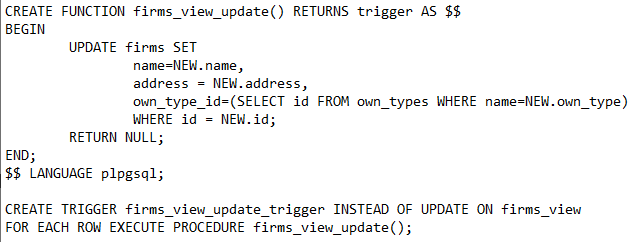


Рисунок 4.68 – Триггер типа INSTEAD OF UPDATE для модифицируемого представления

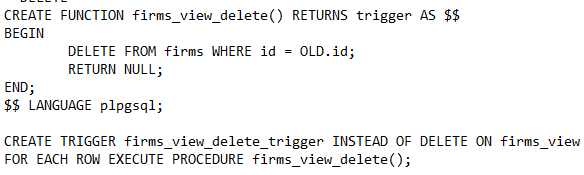


Рисунок 4.69 – Триггер типа INSTEAD OF DELETE для модифицируемого представления

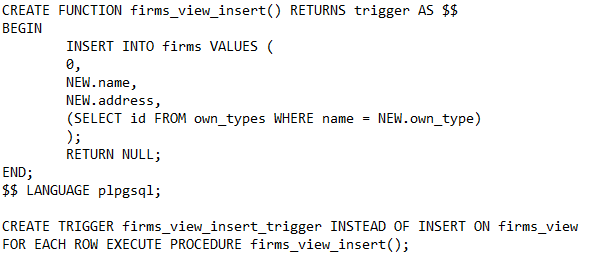


Рисунок 4.70 – Триггер типа INSTEAD OF INSERT для модифицируемого представления

Данные триггеры выполняют операции UPDATE, DELETE и INSERT над основной таблицей “Фирмы”.

# 5 РАЗРАБОТКА ПРИЛОЖЕНИЯ

## 5.1 Формы и компоненты для работы в роли “Сотрудник фирмы”

Формы и компоненты для работы в роли “Сотрудник фирмы” изображены на рисунках 5.1-5.2.

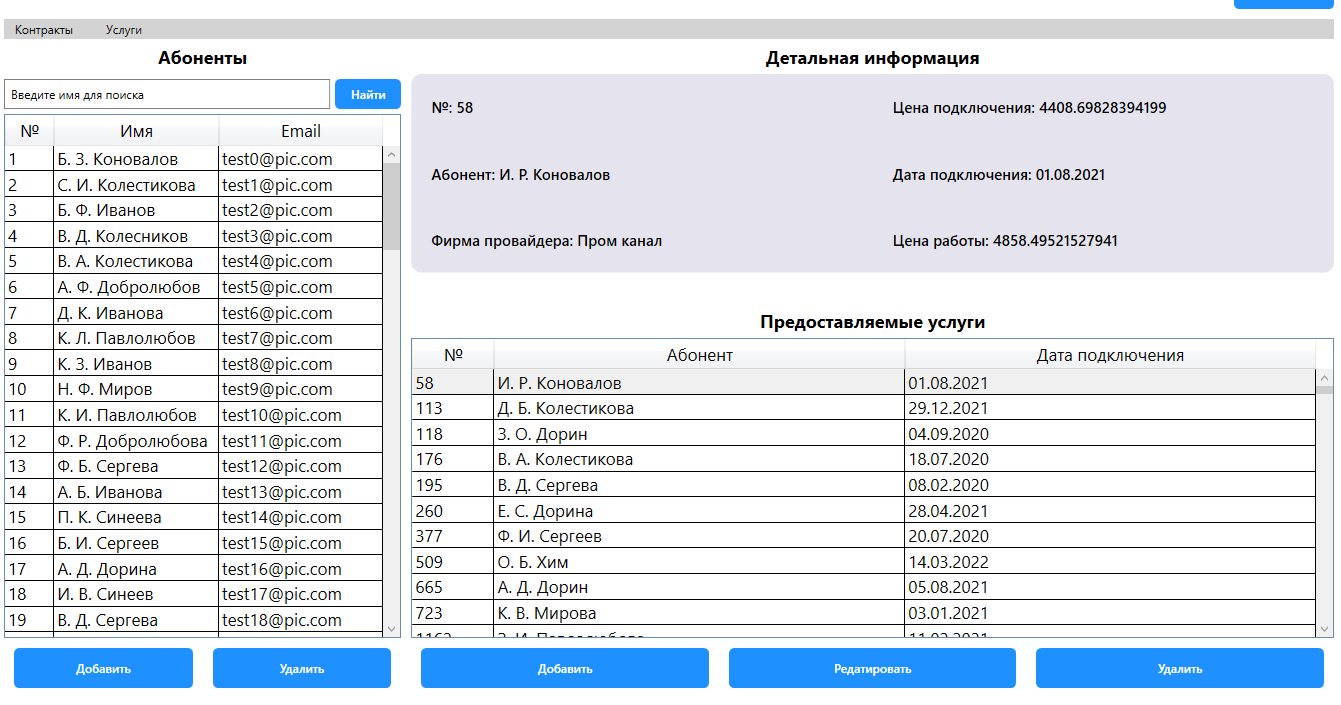


Рисунок 5.1 – Форма для работы в роли “Сотрудник фирмы (вкладка “Контракты”)”

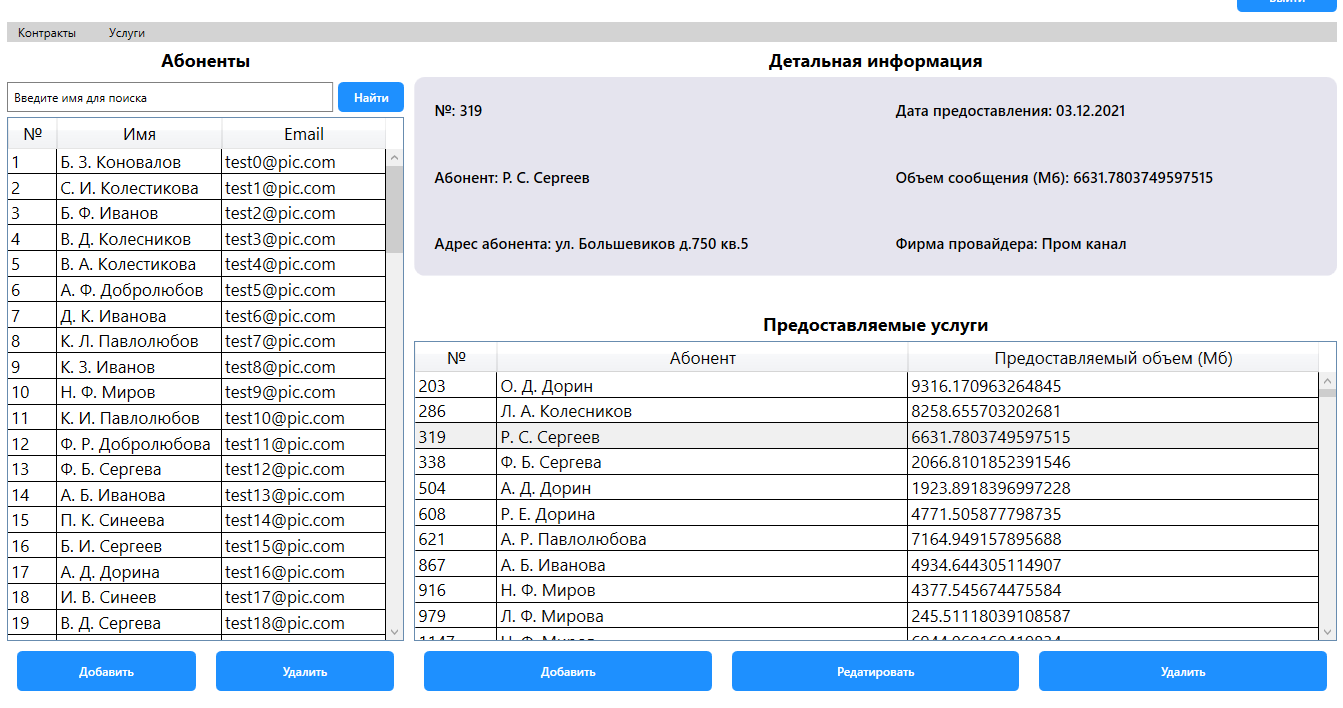


Рисунок 5.2 – Форма для работы в роли “Сотрудник фирмы (вкладка “Услуги”)”

## 5.2 Формы и компоненты для работы в роли “Абонент”

Формы и компоненты для работы в роли “Абонент” 5.3-5.4.

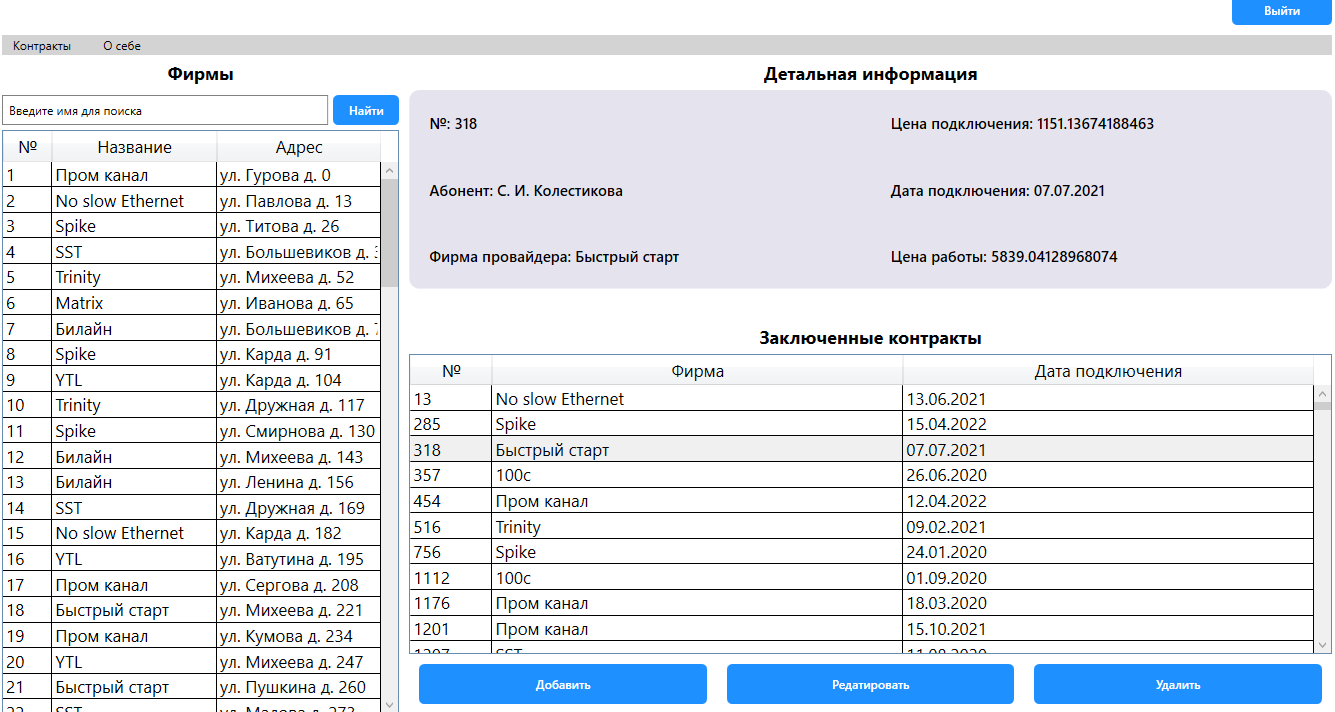


Рисунок 5.3 – Форма для работы в роли “Абонент” (вкладка “Контракты”)

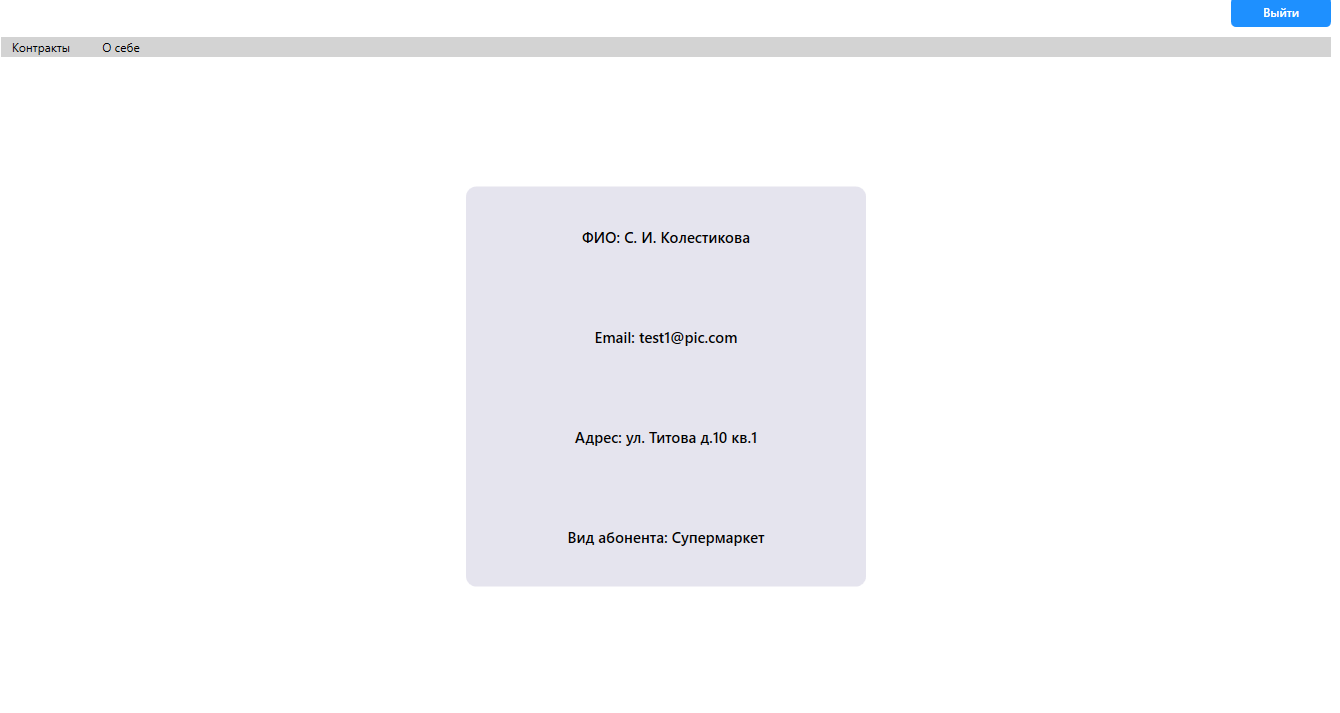


Рисунок 5.4 – Форма для работы в роли “Абонент” (вкладка “О себе”)

## 5.3 Формы и компоненты для работы в роли “Администратор”

Формы и компоненты для работы в роли “Администратор” изображены на рисунках 5.5-5.9.

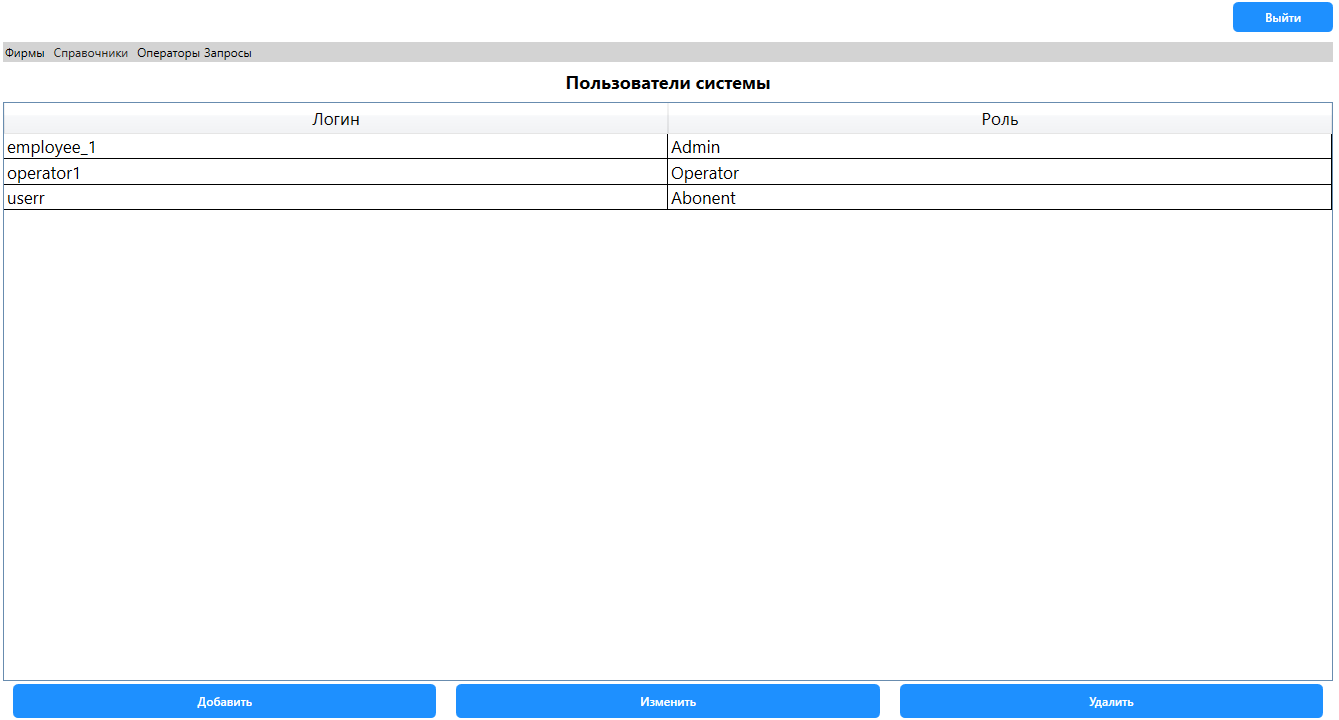


Рисунок 5.5 – Форма для работы в роли “Администратор” (вкладка “Операторы”)

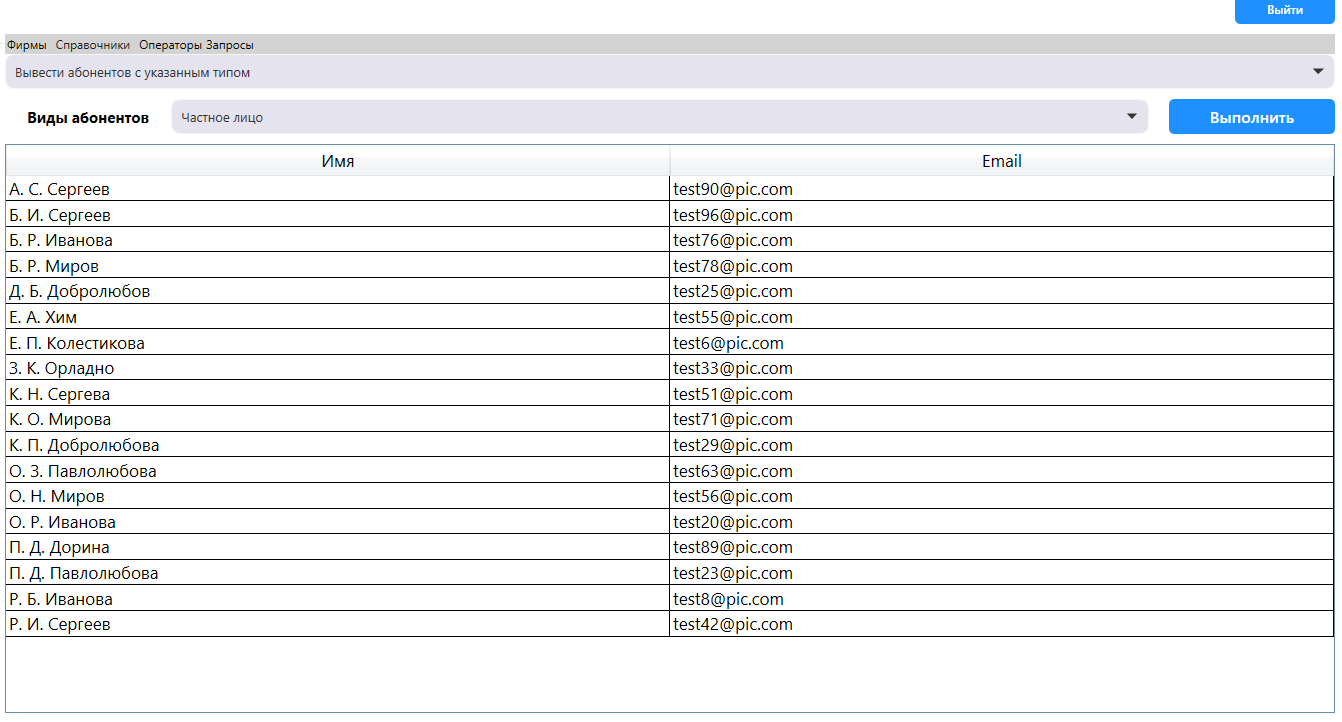


Рисунок 5.6 – Форма для работы в роли “Администратор” (вкладка “Запросы”)

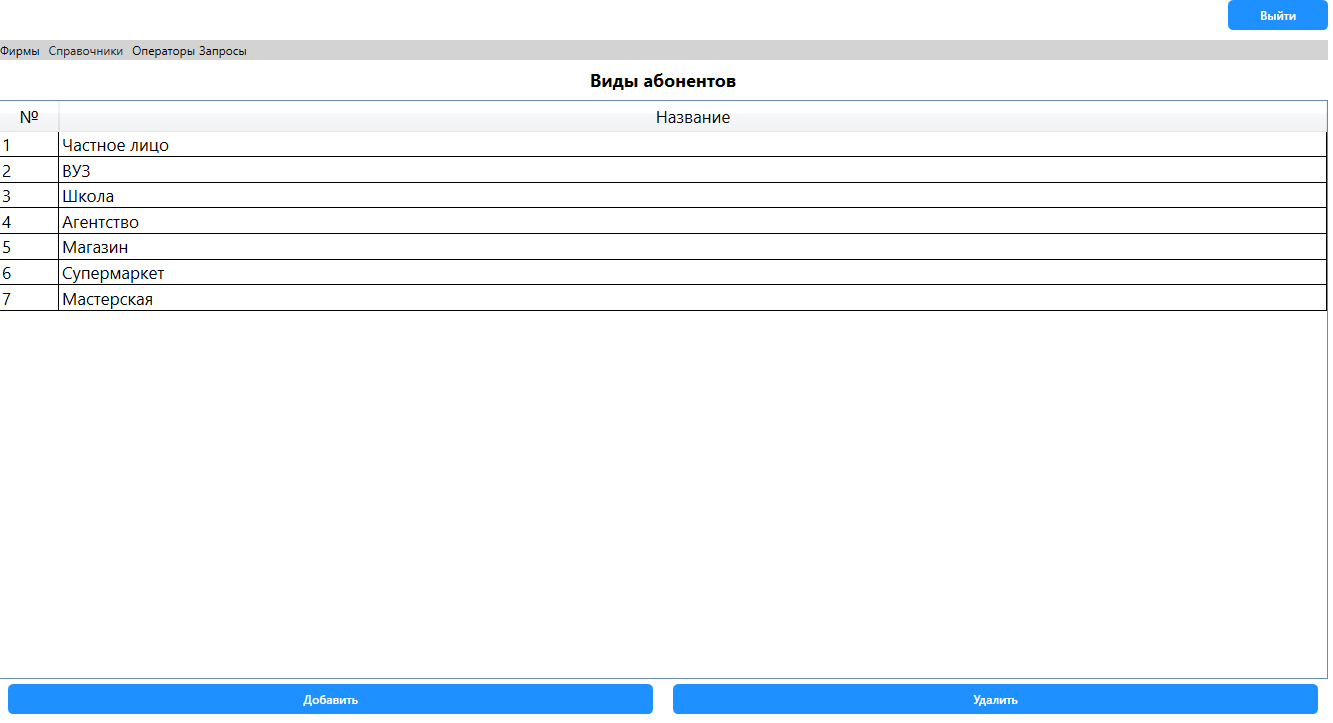


Рисунок 5.7 – Форма для работы в роли “Администратор” (вкладка “Справочник (Виды абонентов)”)

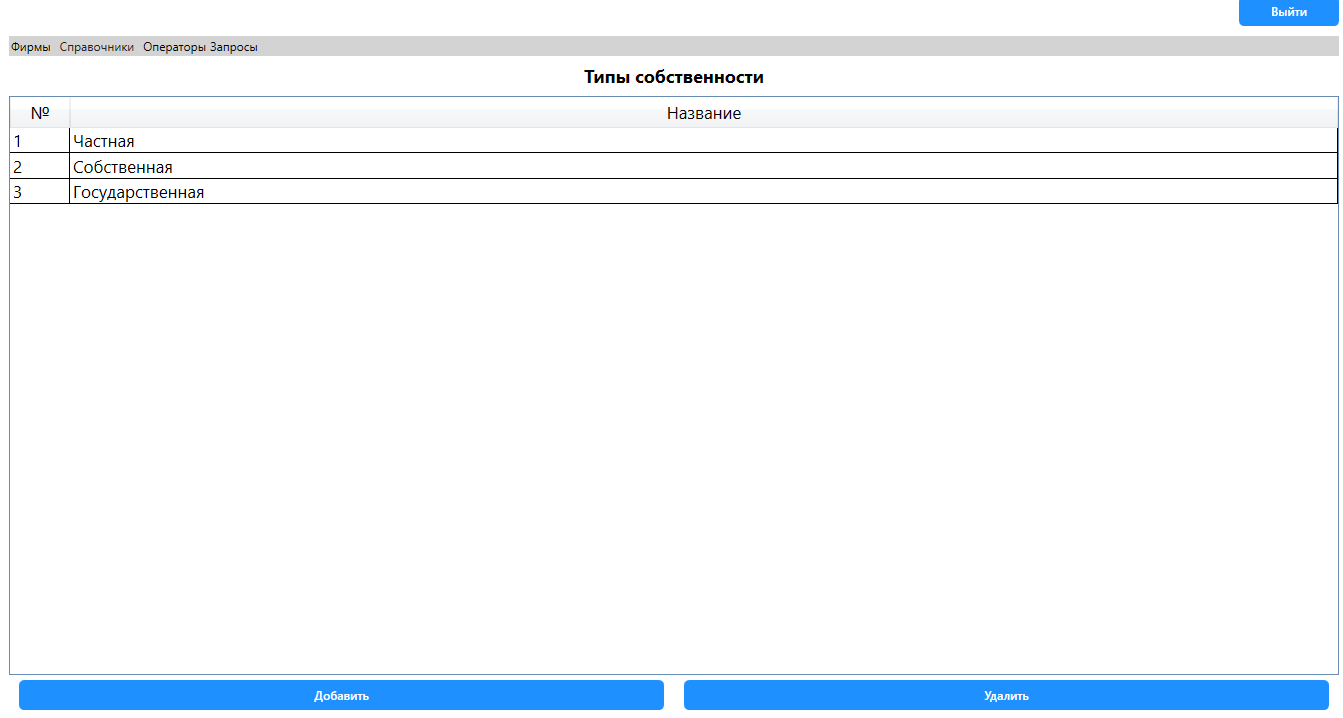


Рисунок 5.8 – Форма для работы в роли “Администратор” (вкладка “Справочник (Типы собственности)”)

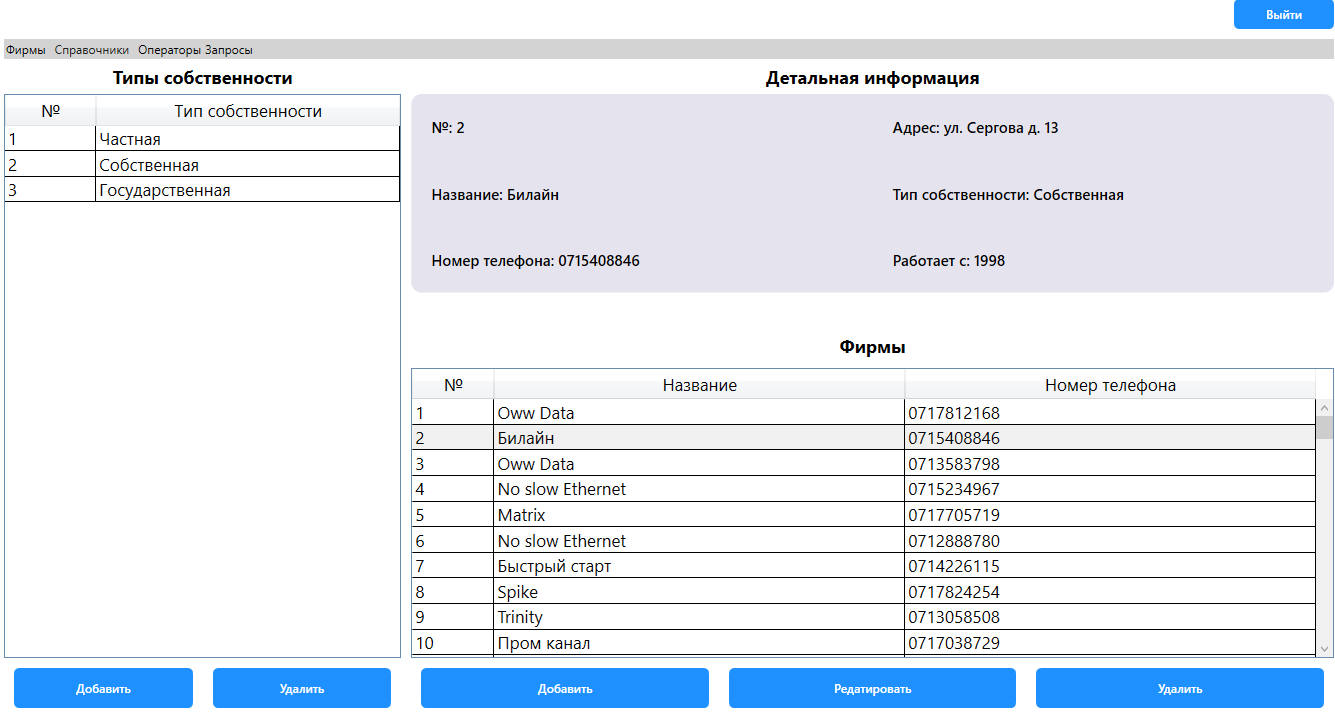


Рисунок 5.9 – Форма для работы в роли “Администратор” (вкладка “Фирмы”)

## 5.4 Экспортирование результата запроса в Excel

Для запроса “Вывести фирмы и их общее число предоставленных мегабайт” имеется возможность сгенерировать Excel диаграмму, изображённую на рисунке 5.10.

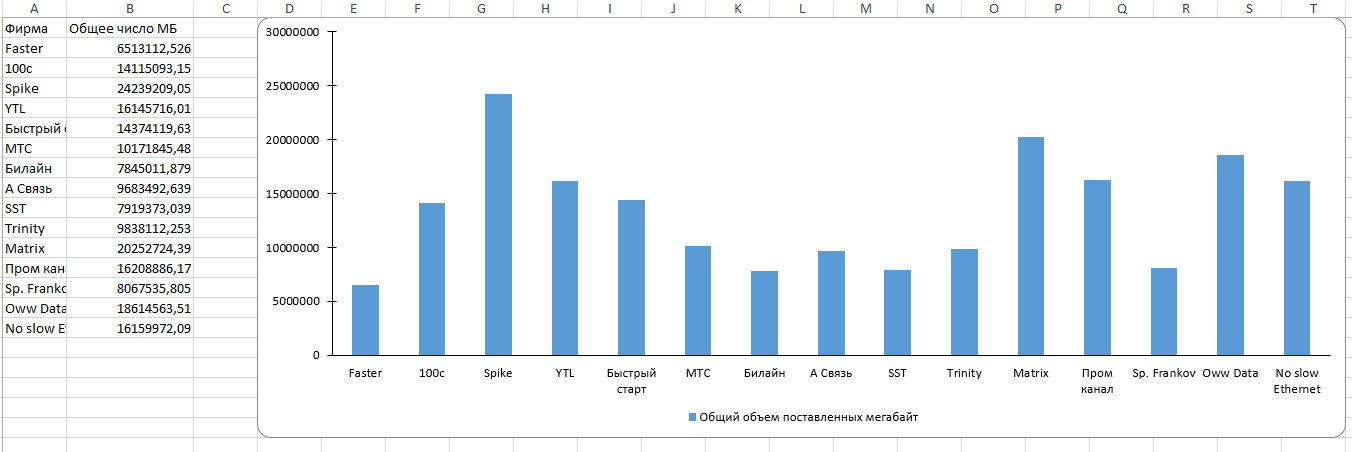


Рисунок 5.10 – Excel диаграмма

# 6 ТЕСТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ

В информационной системе предусмотрены обработки исключительных ситуаций (рис. 6.1-6.5), которые уведомляют пользователя о возникших проблемах, с помощью всплывающих окон.

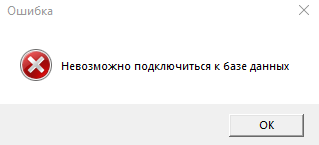


Рисунок 6.1 – Ошибка авторизации

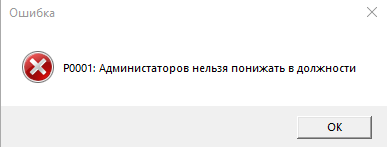


Рисунок 6.2 – Ошибка понижения администратора в должности

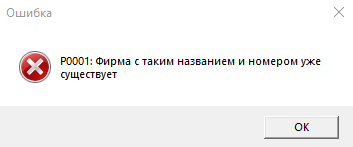


Рисунок 6.3 – Ошибка добавления фирмы

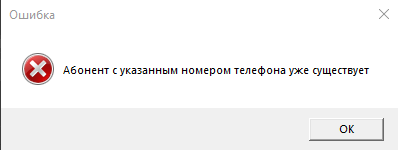


Рисунок 6.4 – Ошибка добавления абонента

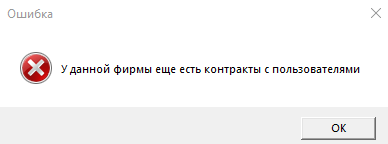


Рисунок 6.5 – Ошибка удаления фирмы

# ВЫВОДЫ

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

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

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Саймон Ригс, Ханну Кросинг. Администрирование PostgreSQL 9. Книга рецептов – М.: ДМК Пресс, 2013. – 21 с.

2. CREATE DOMAIN // PostgresPro [Электронный ресурс]. – Режим доступа: <https://postgrespro.ru/docs/postgresql/9.5/sql-createdomain>

3. Секционирование таблиц // PostgresPro [Электронный ресурс]. – Режим доступа: <https://postgrespro.ru/docs/postgresql/10/ddl-partitioning>

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

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

# 

# ПРИЛОЖЕНИЕ А

# ТЕХНИЧЕСКОЕ ЗАДАНИЕ

ГОУ ВПО «Донецкий национальный технический университет»

Факультет Интеллектуальных систем и программирования

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

Утверждаю

Зори С. А.

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

08.02.2022 г.

**ТЕХНИЧЕСКОЕ ЗАДАНИЕ**

**на курсовую работу по дисциплине**

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

выдано студенту группы ПИ-19а Саевскому Олегу Владиславовичу

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

**Описание предметной области:** Для автоматизации учета пользователей электронной почты необходима информация о фирмах-провайдерах (название фирмы, тип собственности (государственная, частная , ЗАО, ОАО,…), адрес, телефон, год начала работы), о заключенных с ними договорах (абонент (ФИО или название предприятия, тип (частное лицо, ВУЗ, школа, предприятие,…), физический адрес, адрес электронной почты), дата подключения, стоимость подключения, стоимость пересылки 1 Mb информации) и о предоставленных провайдерами услугах (абонент, дата предоставления, объем сообщения (в Мb)).

Донецк – 2022

**Задание на курсовую работу**

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

* симметричное внутреннее соединение с условием (два запроса с условием отбора по внешнему ключу, два – по датам);
* симметричное внутреннее соединение без условия (три запроса);
* левое внешнее соединение;
* правое внешнее соединение;
* запрос на запросе по принципу левого соединения;
* итоговый запрос без условия;
* итоговый запрос без условия c итоговыми данными вида: «всего», «в том числе»;
* итоговые запросы с условием на данные (по значению, по маске, с использованием индекса, без использования индекса);
* итоговый запрос с условием на группы;
* итоговый запрос с условием на данные и на группы;
* запрос на запросе по принципу итогового запроса;
* запрос с использованием объединения
* запросы с подзапросами (с использованием in, not in, case, операциями над итоговыми данными).

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

* многопользовательский режим работы (одна программа для всех ролей – ситуативный доступ к интерфейсу)
* наличие нескольких ролей пользователя (администратор – добавление/удаление/редактирование пользователей, их прав/ролей; пользователи\_1 – …, пользователи \_2 – ...)
* просмотр содержимого таблиц и представлений (здесь и далее – с учетом прав пользователей);
* добавление, редактирование и удаление записей таблиц и модифицируемых представлений;
* работа с наборами данных, находящимися в отношении «один-ко-многим» (создать составную форму для просмотра и редактирования данных родительской и дочерней таблиц);
* поиск и фильтрация данных отображаемых таблиц;
* просмотр результатов выполнения запросов;
* визуализация результатов одного из итоговых запросов (диаграммы, экспорт в Excel).

1. Обеспечить защиту данных, информации от несанкционированного доступа, сделать защиту на уровне строк, выполнить партицирование одной из основных таблиц

**Рекомендуемое содержание пояснительной записки**

Титульный лист

Реферат

Содержание

Введение

1. Описание предметной области, постановка задачи

2. Обоснование выбора СУБД, описание возможностей СУБД

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

3.1 Невизуальные компоненты для работы с данными

3.2 Визуальные компоненты отображения данных

3.3 Разработка шаблонов приложений для работы c таблицами базы данных

4. Проектирование базы данных в выбранной СУБД

4.1 Проектирование концептуальной модели БД

4.2 Создание таблиц, доменов, индексов, сиквенсов

4.3 Разработка триггеров

4.4 Организация многоролевого доступа к данным

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

4.6 Партицирование одной из основных таблиц БД

4.7 Проектирование запросов к базе данных

4.8 Создание представлений и хранимых процедур, функций

5. Разработка клиентского приложения

5.1 Формы и компоненты для работы в «роли 1»

5.2 Формы и компоненты для работы в «роли 2»

5.3 …

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

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

Заключение/выводы и предложения

Список литературы

Приложение А. Техническое задание

Приложение Б. Листинг шаблонов

Приложение В. Листинг серверного приложения

Приложение Д. Листинг клиентского приложения

Приложение Е. Руководство пользователя

Приложение Ж. Отчет о проверке на взаимствования

Приложение З. Руководство администратора

**График выполнения курсовой работы**

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

Дата выдачи задания 08.02.2022

Студент Саевский О.В.

Руководители проекта Щедрин С.В.

Ногтев Е. А.

Филипишин Д. А.

# ПРИЛОЖЕНИЕ Б

# ЛИСТИНГ ШАБЛОНОВ

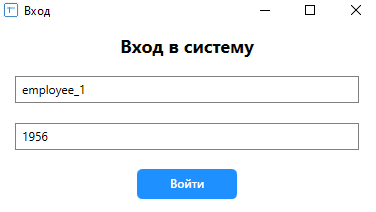


Рисунок Б.1 – Форма авторизации

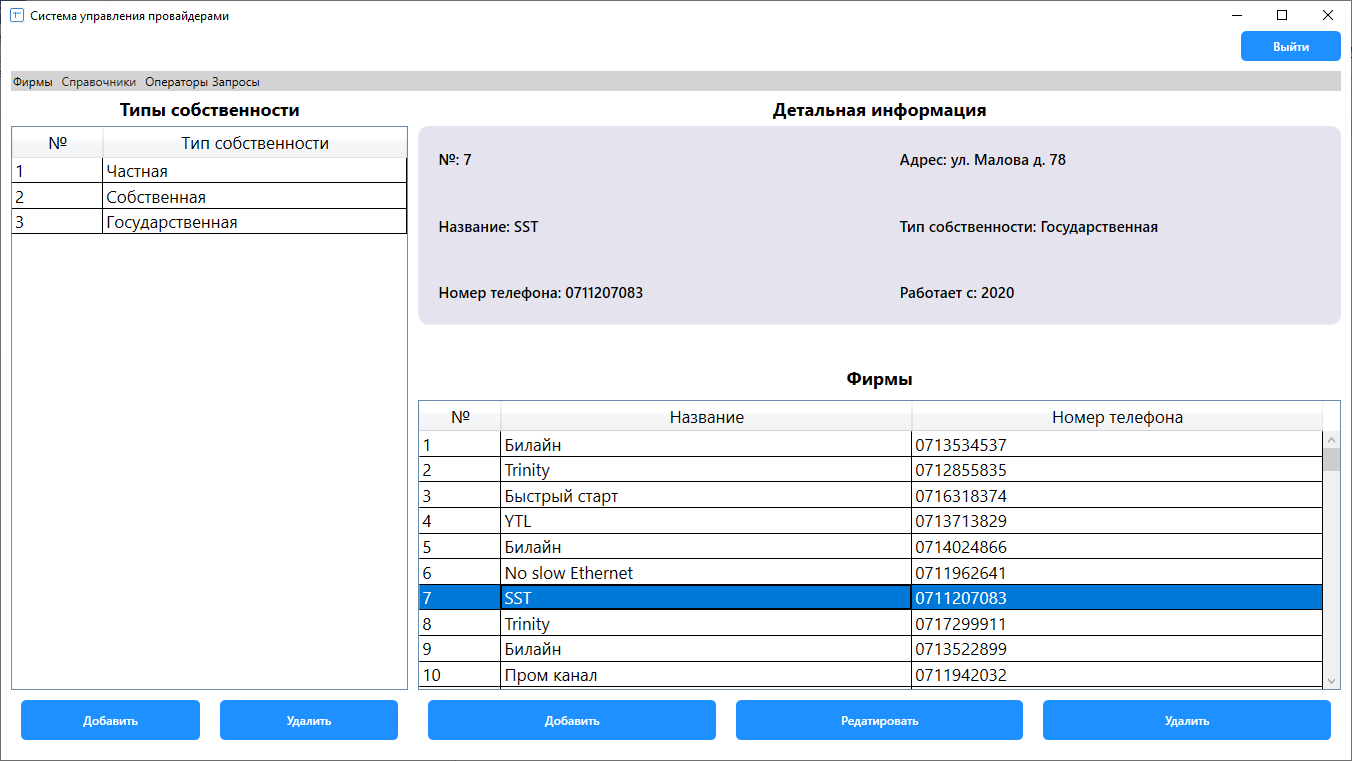


Рисунок Б.2 – Главная форма роли «Администратор»

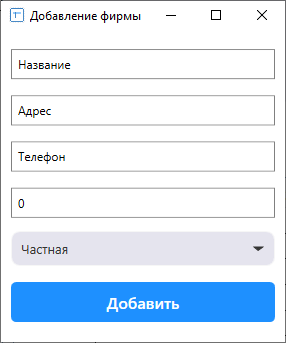


Рисунок Б.3 – Форма добавление фирмы

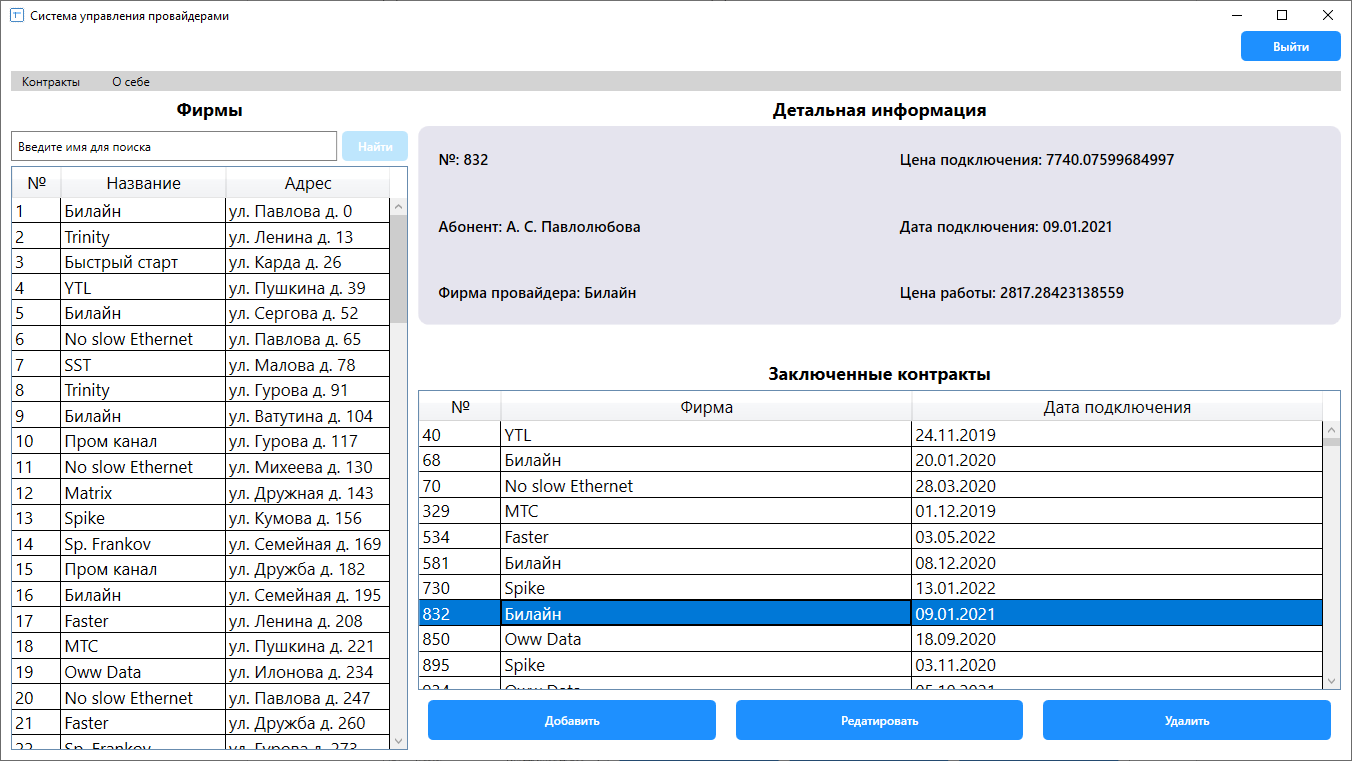


Рисунок Б.4 – Главная форма роли «Абонент»



Рисунок Б.5 –Форма вкладки «О себе» роли «Абонент»

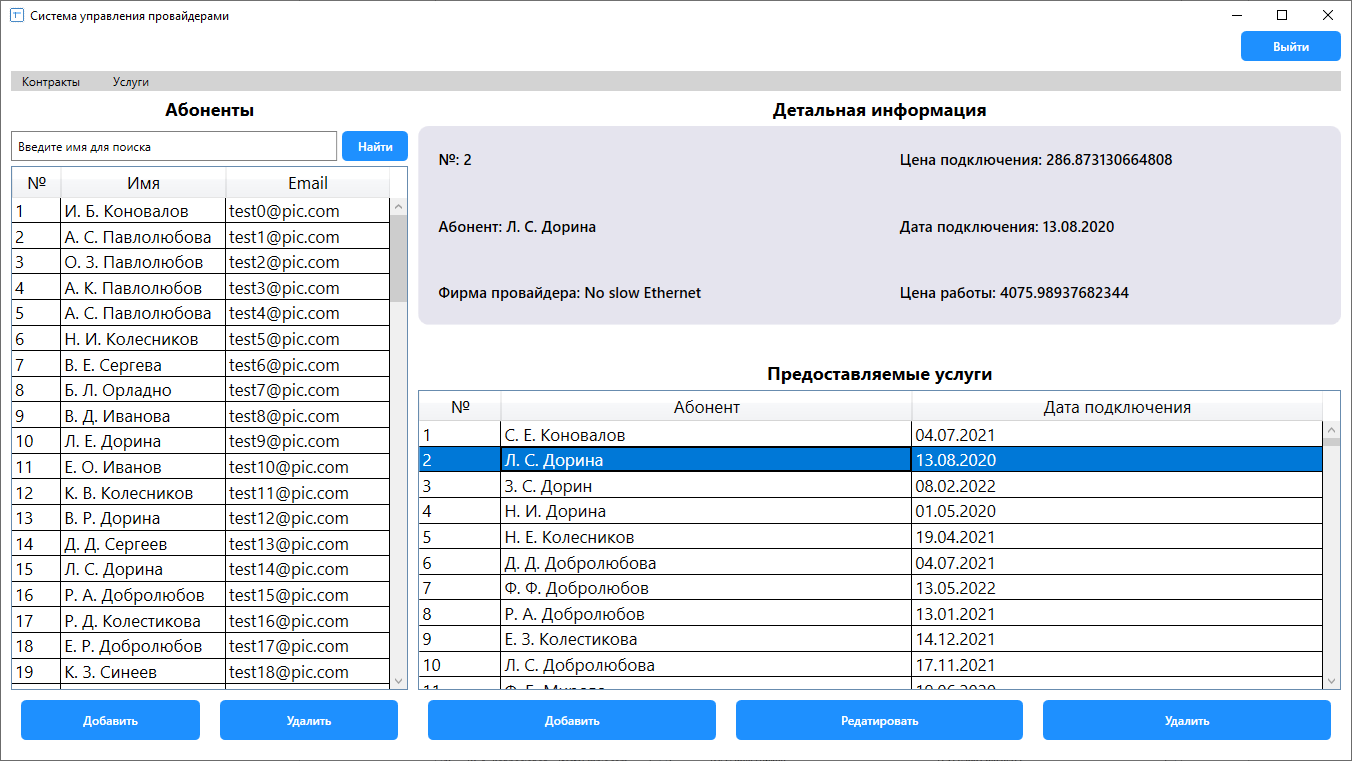


Рисунок Б.6 –Форма вкладки «Контракты» роли «Сотрудник фирмы»

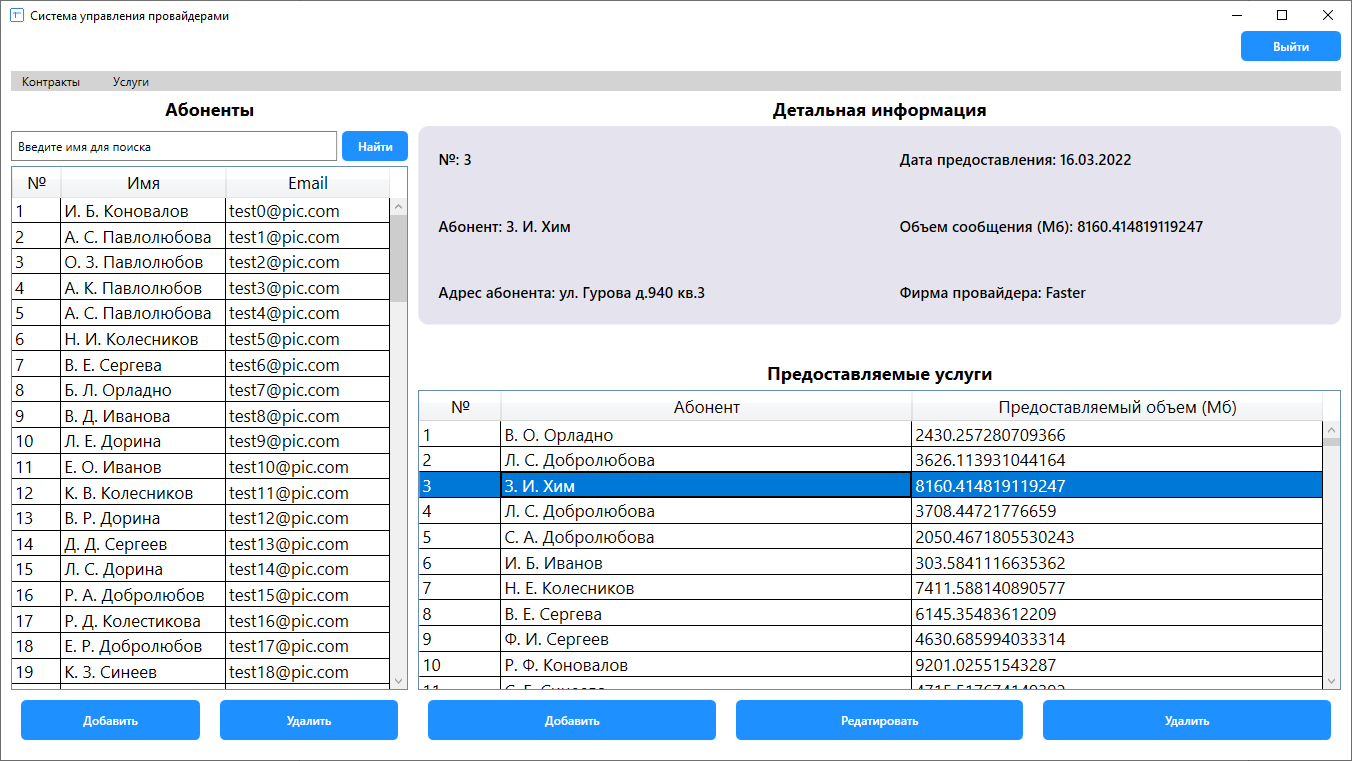


Рисунок Б.7 – Форма вкладки «Услуги» роли «Сотрудник фирмы»

# ПРИЛОЖЕНИЕ В

# ЛИСТИНГ СЕРВЕРНОГО ПРИЛОЖЕНИЯ

using Microsoft.EntityFrameworkCore;

using ProviderSystemManager.DAL.Database;

using ProviderSystemManager.DAL.Queries;

namespace ProviderSystemManager.DAL.QueryCreators

{

internal class QueryCreator

{

public static void Init(ProviderDbContext dbContext)

{

var query = new AbonentsByAbonentTypeQuery(dbContext);

var query2 = new FirmsByOwnType(dbContext);

var query3 = new FirmsByServiceRecievingDate(dbContext);

var query4 = new ContractsInfoQuery(dbContext);

var query5 = new ServiceInfoQuery(dbContext);

var query6 = new ContractAbonentsEmailNotNullQuery(dbContext);

var query7 = new FirmHaveServicesQuery(dbContext);

var query8 = new FirmsByStartDateWithServicesQuery(dbContext);

var query9 = new AbonentsByServiceRecievingDateQuery(dbContext);

var query10 = new AbonentInfoQuery(dbContext);

var query11 = new SumSizeFirmsQuery(dbContext);

var query12 = new FirmsCountByOwnTypeQuery(dbContext);

var query13 = new AbonentsByContractsSumQuery(dbContext);

var query14 = new AbonentsByContractsSumAndDateQuery(dbContext);

var query15 = new FirmsSumConnectionCostInflationQuery(dbContext);

var query16 = new FirmsSumConnectionCostMoreAvgQuery(dbContext);

var query17 = new MaskQuery(dbContext);

var query18 = new СaseQuery(dbContext);

var query19 = new UnionQuery(dbContext);

var query20 = new InQuery(dbContext);

var query21 = new NotInQuery(dbContext);

dbContext.Database.ExecuteSqlRaw(query.CreateQuery);

dbContext.Database.ExecuteSqlRaw(query2.CreateQuery);

dbContext.Database.ExecuteSqlRaw(query3.CreateQuery);

dbContext.Database.ExecuteSqlRaw(query4.CreateQuery);

dbContext.Database.ExecuteSqlRaw(query5.CreateQuery);

dbContext.Database.ExecuteSqlRaw(query6.CreateQuery);

dbContext.Database.ExecuteSqlRaw(query7.CreateQuery);

dbContext.Database.ExecuteSqlRaw(query8.CreateQuery);

dbContext.Database.ExecuteSqlRaw(query9.CreateQuery);

dbContext.Database.ExecuteSqlRaw(query10.CreateQuery);

dbContext.Database.ExecuteSqlRaw(query11.CreateQuery);

dbContext.Database.ExecuteSqlRaw(query12.CreateQuery);

dbContext.Database.ExecuteSqlRaw(query13.CreateQuery);

dbContext.Database.ExecuteSqlRaw(query14.CreateQuery);

dbContext.Database.ExecuteSqlRaw(query15.CreateQuery);

dbContext.Database.ExecuteSqlRaw(query16.CreateQuery);

dbContext.Database.ExecuteSqlRaw(query17.CreateQuery);

dbContext.Database.ExecuteSqlRaw(query18.CreateQuery);

dbContext.Database.ExecuteSqlRaw(query19.CreateQuery);

dbContext.Database.ExecuteSqlRaw(query20.CreateQuery);

dbContext.Database.ExecuteSqlRaw(query21.CreateQuery);

#region SEQUENCES

dbContext.Database.ExecuteSqlRaw("CREATE SEQUENCE firms\_seq");

dbContext.Database.ExecuteSqlRaw("CREATE SEQUENCE abonents\_seq");

dbContext.Database.ExecuteSqlRaw("CREATE SEQUENCE abonent\_types\_seq");

dbContext.Database.ExecuteSqlRaw("CREATE SEQUENCE contracts\_seq");

dbContext.Database.ExecuteSqlRaw("CREATE SEQUENCE own\_types\_seq");

dbContext.Database.ExecuteSqlRaw("CREATE SEQUENCE services\_seq");

#endregion

#region GRANDS

dbContext.Database.ExecuteSqlRaw("GRANT SELECT ON firms TO abonent;");

dbContext.Database.ExecuteSqlRaw("GRANT SELECT ON services TO abonent;");

dbContext.Database.ExecuteSqlRaw("GRANT SELECT, DELETE, UPDATE, INSERT ON contracts TO abonent;");

dbContext.Database.ExecuteSqlRaw("GRANT SELECT ON abonents TO abonent;");

dbContext.Database.ExecuteSqlRaw("GRANT SELECT ON abonent\_types TO abonent;");

dbContext.Database.ExecuteSqlRaw("GRANT SELECT ON own\_types TO abonent;");

dbContext.Database.ExecuteSqlRaw("GRANT SELECT ON users TO abonent;");

dbContext.Database.ExecuteSqlRaw("GRANT SELECT ON firms TO operator;");

dbContext.Database.ExecuteSqlRaw("GRANT SELECT, DELETE, UPDATE, INSERT ON services TO operator;");

dbContext.Database.ExecuteSqlRaw("GRANT SELECT, DELETE, UPDATE, INSERT ON contracts TO operator;");

dbContext.Database.ExecuteSqlRaw("GRANT SELECT, DELETE, UPDATE, INSERT ON abonents TO operator;");

dbContext.Database.ExecuteSqlRaw("GRANT SELECT ON abonent\_types TO operator;");

dbContext.Database.ExecuteSqlRaw("GRANT SELECT ON own\_types TO operator;");

dbContext.Database.ExecuteSqlRaw("GRANT SELECT ON users TO operator;");

dbContext.Database.ExecuteSqlRaw("GRANT SELECT, DELETE, UPDATE, INSERT ON firms TO admin;");

dbContext.Database.ExecuteSqlRaw("GRANT SELECT, DELETE, UPDATE, INSERT ON services TO admin;");

dbContext.Database.ExecuteSqlRaw("GRANT SELECT, DELETE, UPDATE, INSERT ON contracts TO admin;");

dbContext.Database.ExecuteSqlRaw("GRANT SELECT, DELETE, UPDATE, INSERT ON abonents TO admin;");

dbContext.Database.ExecuteSqlRaw("GRANT SELECT, DELETE, UPDATE, INSERT ON abonent\_types TO admin;");

dbContext.Database.ExecuteSqlRaw("GRANT SELECT, DELETE, UPDATE, INSERT ON own\_types TO admin;");

dbContext.Database.ExecuteSqlRaw("GRANT SELECT, DELETE, UPDATE, INSERT ON users TO admin;");

//#endregion

//#region TRIGGERS

dbContext.Database.ExecuteSqlRaw("CREATE OR REPLACE FUNCTION users\_after\_update() RETURNS trigger AS $$ BEGIN IF OLD.role <> NEW.role THEN IF OLD.role = 1 THEN EXECUTE 'ALTER GROUP operator DROP USER ' || NEW.login; EXECUTE 'GRANT abonent TO ' || NEW.login; END IF; IF OLD.role = 2 THEN EXECUTE 'ALTER GROUP abonent DROP USER ' || NEW.login; EXECUTE 'GRANT operator TO ' || NEW.login; END IF; IF OLD.role = 0 THEN RAISE EXCEPTION 'Администаторов нельзя понижать в должности'; END IF; END IF; RETURN NEW; END; $$ LANGUAGE plpgsql SECURITY DEFINER; CREATE TRIGGER users\_after\_update\_trigger AFTER UPDATE ON users FOR EACH ROW EXECUTE PROCEDURE users\_after\_update();");

dbContext.Database.ExecuteSqlRaw("CREATE OR REPLACE FUNCTION users\_after\_insert() RETURNS trigger AS $$ BEGIN IF NEW.role = 0 THEN EXECUTE 'GRANT admin TO ' || NEW.login; END IF; IF NEW.role = 1 THEN EXECUTE 'GRANT operator TO ' || NEW.login; END IF; IF NEW.role = 2 THEN EXECUTE 'GRANT abonent TO ' || NEW.login; END IF; RETURN NULL; END; $$ LANGUAGE plpgsql SECURITY DEFINER; CREATE TRIGGER users\_after\_insert\_trigger AFTER INSERT ON users FOR EACH ROW EXECUTE PROCEDURE users\_after\_insert();");

dbContext.Database.ExecuteSqlRaw("CREATE OR REPLACE FUNCTION users\_after\_delete() RETURNS trigger AS $$ BEGIN EXECUTE 'DROP USER ' || OLD.login; RETURN NULL; END; $$ LANGUAGE plpgsql SECURITY DEFINER; CREATE TRIGGER users\_after\_delete\_trigger AFTER DELETE ON users FOR EACH ROW EXECUTE PROCEDURE users\_after\_delete();");

dbContext.Database.ExecuteSqlRaw("CREATE OR REPLACE FUNCTION firms\_before\_delete() RETURNS trigger AS $$ BEGIN IF (SELECT COUNT(\*) FROM (SELECT FROM contracts WHERE firm\_id=OLD.id) p) <> 0 THEN RAISE EXCEPTION 'У данной фирмы еще есть контракты с пользователями'; END IF; RETURN NULL; END; $$ LANGUAGE plpgsql SECURITY DEFINER; CREATE TRIGGER firms\_before\_delete\_trigger BEFORE DELETE ON firms FOR EACH ROW EXECUTE PROCEDURE firms\_before\_delete();");

dbContext.Database.ExecuteSqlRaw("CREATE OR REPLACE FUNCTION firms\_before\_insert() RETURNS trigger AS $$ BEGIN IF (SELECT COUNT(\*) FROM (SELECT FROM firms WHERE name=NEW.name AND telephone=NEW.telephone) p) <> 0 THEN RAISE EXCEPTION 'Фирма с таким названием и номером уже существует'; END IF; RETURN NEW; END; $$ LANGUAGE plpgsql SECURITY DEFINER; CREATE TRIGGER firms\_before\_insert\_trigger BEFORE INSERT ON firms FOR EACH ROW EXECUTE PROCEDURE firms\_before\_insert();");

dbContext.Database.ExecuteSqlRaw("CREATE OR REPLACE FUNCTION firms\_before\_update() RETURNS trigger AS $$ BEGIN IF (SELECT COUNT(\*) FROM (SELECT FROM firms WHERE name=NEW.name AND telephone=NEW.telephone) p) <> 0 THEN RAISE EXCEPTION 'Фирма с таким названием и номером уже существует'; END IF; RETURN NEW; END; $$ LANGUAGE plpgsql SECURITY DEFINER; CREATE TRIGGER firms\_before\_update\_trigger BEFORE UPDATE ON firms FOR EACH ROW EXECUTE PROCEDURE firms\_before\_update();");

dbContext.Database.ExecuteSqlRaw("CREATE FUNCTION firms\_before\_insert\_increment() RETURNS trigger AS $$ BEGIN NEW.id := nextval('firms\_seq'); RETURN NEW; END; $$ LANGUAGE plpgsql SECURITY DEFINER; CREATE TRIGGER firms\_before\_insert\_increment\_trigger BEFORE INSERT ON firms FOR EACH ROW EXECUTE PROCEDURE firms\_before\_insert\_increment();");

//#endregion

//#region SECURITY

dbContext.Database.ExecuteSqlRaw("ALTER TABLE firms ENABLE ROW LEVEL SECURITY;");

dbContext.Database.ExecuteSqlRaw("ALTER TABLE abonents ENABLE ROW LEVEL SECURITY;");

dbContext.Database.ExecuteSqlRaw("ALTER TABLE contracts ENABLE ROW LEVEL SECURITY;");

dbContext.Database.ExecuteSqlRaw("ALTER TABLE services ENABLE ROW LEVEL SECURITY;");

//#endregion

//#region POLICY

dbContext.Database.ExecuteSqlRaw("CREATE POLICY firms\_abonent ON firms TO abonent USING (TRUE);");

dbContext.Database.ExecuteSqlRaw("CREATE POLICY contracts\_abonent ON contracts TO abonent USING (abonent\_id = (SELECT id FROM users WHERE login = CURRENT\_USER));");

dbContext.Database.ExecuteSqlRaw("CREATE POLICY services\_abonent ON services TO abonent USING (TRUE);");

dbContext.Database.ExecuteSqlRaw("CREATE POLICY abonents\_abonent ON abonents TO abonent USING (TRUE);");

dbContext.Database.ExecuteSqlRaw("CREATE POLICY abonent\_types\_abonent ON abonent\_types TO abonent USING (TRUE);");

dbContext.Database.ExecuteSqlRaw("CREATE POLICY own\_types\_abonent ON own\_types TO abonent USING (TRUE);");

dbContext.Database.ExecuteSqlRaw("CREATE POLICY firms\_operator ON firms TO operator USING (id = (SELECT id FROM users WHERE login = CURRENT\_USER));");

dbContext.Database.ExecuteSqlRaw("CREATE POLICY contracts\_operator ON contracts TO operator USING (firm\_id = (SELECT id FROM users WHERE login = CURRENT\_USER));");

dbContext.Database.ExecuteSqlRaw("CREATE POLICY services\_operator ON services TO operator USING (firm\_id = (SELECT id FROM users WHERE login = CURRENT\_USER));");

dbContext.Database.ExecuteSqlRaw("CREATE POLICY abonents\_operator ON abonents TO operator USING (TRUE);");

dbContext.Database.ExecuteSqlRaw("CREATE POLICY abonent\_types\_operator ON abonent\_types TO operator USING (TRUE);");

dbContext.Database.ExecuteSqlRaw("CREATE POLICY own\_types\_operator ON own\_types TO operator USING (TRUE);");

dbContext.Database.ExecuteSqlRaw("CREATE POLICY firms\_admin ON firms TO operator USING (TRUE);");

dbContext.Database.ExecuteSqlRaw("CREATE POLICY contracts\_admin ON contracts TO operator USING (TRUE);");

dbContext.Database.ExecuteSqlRaw("CREATE POLICY services\_admin ON services TO operator USING (TRUE);");

dbContext.Database.ExecuteSqlRaw("CREATE POLICY abonents\_admin ON abonents TO operator USING (TRUE);");

dbContext.Database.ExecuteSqlRaw("CREATE POLICY abonent\_types\_admin ON abonent\_types TO operator USING (TRUE);");

dbContext.Database.ExecuteSqlRaw("CREATE POLICY own\_types\_admin ON own\_types TO operator USING (TRUE);");

#endregion

}

}

} using ProviderSystemManager.DAL.Database;

using ProviderSystemManager.DAL.Models;

namespace ProviderSystemManager.DAL.TableCreators;

public class AbonentCreator

{

public static int Count => 20000;

public static string[] Initials => new string[]

{

"А.",

"Б.",

"С.",

"Д.",

"Е.",

"П.",

"К.",

"Р.",

"В.",

"З.",

"Ф.",

"Л.",

"Н.",

"О.",

"И.",

"Я."

};

public static string[] SecondNames => new string[]

{

"Сергева",

"Сергеев",

"Иванова",

"Иванов",

"Мирова",

"Миров",

"Добролюбов",

"Добролюбова",

"Синеев",

"Синеева",

"Хим",

"Орладно",

"Дорин",

"Дорина",

"Колесников",

"Колестикова",

"Павлолюбов",

"Павлолюбова",

"Коновалов",

"Коновалова"

};

public static void Init(ProviderDbContext dbContext)

{

var random = new Random();

for(int i = 0; i < Count; i++)

{

var firstInitial = Initials[random.Next(Initials.Length - 1)];

var secondInitial = Initials[random.Next(Initials.Length - 1)];

var secondName = SecondNames[random.Next(SecondNames.Length - 1)];

var street = FirmCreator.StreetNames[random.Next(FirmCreator.StreetNames.Length - 1)];

var abonent = new Abonent { Name = $"{firstInitial} {secondInitial} {secondName}", Address = $"ул. {street} д.{i \* 10} кв.{i % 7}", Email = $"test{i}@pic.com", AbonentTypeId = random.Next(1, AbonentTypeCreator.Count) };

dbContext.Abonents?.Add(abonent);

}

dbContext.SaveChanges();

}

} using ProviderSystemManager.DAL.Database;

using ProviderSystemManager.DAL.Models;

namespace ProviderSystemManager.DAL.TableCreators;

public class AbonentTypeCreator

{

public static int Count => 7;

public static void Init(ProviderDbContext context)

{

var abonentType1 = new AbonentType() {Name = "Частное лицо"};

var abonentType2 = new AbonentType() { Name = "ВУЗ" };

var abonentType3 = new AbonentType() { Name = "Школа" };

var abonentType4 = new AbonentType() { Name = "Агентство" };

var abonentType5 = new AbonentType() { Name = "Магазин" };

var abonentType6 = new AbonentType() { Name = "Супермаркет" };

var abonentType7 = new AbonentType() { Name = "Мастерская" };

context.AbonentTypes?.Add(abonentType1);

context.AbonentTypes?.Add(abonentType2);

context.AbonentTypes?.Add(abonentType3);

context.AbonentTypes?.Add(abonentType4);

context.AbonentTypes?.Add(abonentType5);

context.AbonentTypes?.Add(abonentType6);

context.AbonentTypes?.Add(abonentType7);

context.SaveChanges();

}

} using ProviderSystemManager.DAL.Database;

using ProviderSystemManager.DAL.Models;

namespace ProviderSystemManager.DAL.TableCreators;

public class ContractCreator

{

public static int Count => 40000;

public static void Init(ProviderDbContext dbContext)

{

var random = new Random();

for(int i = 0; i < 40000; i++)

{

var contract = new Contract()

{

FirmId = random.Next(1, FirmCreator.Count),

AbonentId = random.Next(1, AbonentCreator.Count),

ConnectionCost = (decimal)(random.Next(1, 10000) + random.NextDouble()),

ConnectionDate = DateOnly.FromDateTime(DateTime.Now.AddDays(random.Next(-1000, 0))),

ForwardingCost = (decimal)(random.Next(1, 10000) + random.NextDouble())

};

dbContext.Contracts?.Add(contract);

}

dbContext.SaveChanges();

}

} using ProviderSystemManager.DAL.Database;

using ProviderSystemManager.DAL.Models;

namespace ProviderSystemManager.DAL.TableCreators;

public class FirmCreator

{

public static int Count => 20000;

private static string[] ProviderNames = new string[]

{

"Trinity",

"Matrix",

"Билайн",

"MTC",

"A Связь",

"Пром канал",

"100с",

"Быстрый старт",

"Spike",

"Oww Data",

"Faster",

"No slow Ethernet",

"Sp. Frankov",

"YTL",

"SST",

"QWE"

};

public static string[] StreetNames = new string[]

{

"Пушкина",

"Куйбышева",

"Ватутина",

"Титова",

"Гурова",

"Илонова",

"Битова",

"Сергова",

"Синяя",

"Ленина",

"Кирова",

"Малова",

"Большевиков",

"Иванова",

"Павлова",

"Михеева",

"Карда",

"Кумова",

"Семейная",

"Дружба",

"Дружная",

"Веселова",

"Смирнова",

"Каталова"

};

public static void Init(ProviderDbContext dbContext)

{

var ownTypeId = OwnTypeCreator.Count;

var random = new Random();

for (int i = 0; i < Count; i++)

{

var street = StreetNames[random.Next(StreetNames.Length - 1)];

var name = ProviderNames[random.Next(ProviderNames.Length - 1)];

if(ownTypeId == 0)

ownTypeId = OwnTypeCreator.Count;

var firm = new Firm()

{

Name = name,

Address = $"ул. {street} д. {i \* 13}",

Telephone = $"071{random.Next(1000000, 9999999)}",

OwnTypeId = ownTypeId,

StartWorkingYear = (short)random.Next(1995, 2022)

};

ownTypeId--;

dbContext.Firms?.Add(firm);

dbContext.SaveChanges();

dbContext.Entry(firm).State = Microsoft.EntityFrameworkCore.EntityState.Detached;

}

}

} using ProviderSystemManager.DAL.Database;

using ProviderSystemManager.DAL.Models;

namespace ProviderSystemManager.DAL.TableCreators;

public class OwnTypeCreator

{

public static int Count => 3;

public static void Init(ProviderDbContext context)

{

var ownType1 = new OwnType() {Name = "Частная"};

var ownType2 = new OwnType() { Name = "Собственная" };

var ownType3 = new OwnType() { Name = "Государственная" };

context.OwnTypes?.Add(ownType1);

context.OwnTypes?.Add(ownType2);

context.OwnTypes?.Add(ownType3);

context.SaveChanges();

context.Entry(ownType1).State = Microsoft.EntityFrameworkCore.EntityState.Detached;

context.Entry(ownType2).State = Microsoft.EntityFrameworkCore.EntityState.Detached;

context.Entry(ownType3).State = Microsoft.EntityFrameworkCore.EntityState.Detached;

}

} using ProviderSystemManager.DAL.Database;

using ProviderSystemManager.DAL.Models;

namespace ProviderSystemManager.DAL.TableCreators;

public class ServiceCreator

{

public static int Count => 40000;

public static void Init(ProviderDbContext dbContext)

{

var random = new Random();

for(int i = 0; i < Count; i++)

{

var service = new Service { AbonentId = random.Next(1, AbonentCreator.Count), Size = random.Next(1, 10000) + random.NextDouble(), RecievingDate = DateOnly.FromDateTime(DateTime.Now.AddDays(random.Next(-1000, 0))), FirmId = random.Next(1, FirmCreator.Count) };

dbContext.Services?.Add(service);

}

dbContext.SaveChanges();

}

} using Microsoft.EntityFrameworkCore;

using ProviderSystemManager.DAL.Database;

using ProviderSystemManager.DAL.Enums;

using ProviderSystemManager.DAL.Models;

using System.Security.Cryptography;

namespace ProviderSystemManager.DAL.TableCreators;

public class UserCreator

{

public static void Init(ProviderDbContext context)

{

var user1 = new User() { Login = "operator1", Password = "1954", Role = UserRole.Operator, Id = 1 };

var user2 = new User() { Login = "userr", Password = "1955", Role = UserRole.Abonent, Id = 2 };

var user3 = new User() { Login = "employee\_1", Password = "1956", Role = UserRole.Admin, Id = 3 };

context.Users?.Add(user1);

context.Users?.Add(user2);

context.Users?.Add(user3);

context.SaveChanges();

context.Entry(user1).State = EntityState.Detached;

context.Entry(user2).State = EntityState.Detached;

context.Entry(user3).State = EntityState.Detached;

}

private static string HashPassword(string password)

{

byte[] salt;

new RNGCryptoServiceProvider().GetBytes(salt = new byte[16]);

var pbkdf2 = new Rfc2898DeriveBytes(password, salt, 100000);

var hash = pbkdf2.GetBytes(20);

var hashBytes = new byte[36];

Array.Copy(salt, 0, hashBytes, 0, 16);

Array.Copy(hash, 0, hashBytes, 16, 20);

return Convert.ToBase64String(hashBytes);

}}

# ПРИЛОЖЕНИЕ Г

ЛИСТИНГ КЛИЕНТСКОГО ПРИЛОЖЕНИЯ

using DevExpress.Mvvm;

using ProviderSystemManager.BLL.Services.Interfaces;

using ProviderSystemManager.DAL.Enums;

using ProviderSystemManager.Shared;

using ProviderSystemManager.WPF.Session;

using ProviderSystemManager.WPF.Utils;

using System;

using System.Configuration;

using System.Linq;

using System.Windows.Input;

namespace ProviderSystemManager.WPF.ViewModels

{

internal class SignInWindowViewModel : BindableBase

{

private readonly IDbContextService \_service;

private readonly IUserService \_userService;

public SignInWindowViewModel(IDbContextService service, IUserService userService)

{

\_service = service;

\_userService = userService;

}

public Action OnSuccessSignIn { get; set; }

public string Login { get; set; } = "employee\_1";

public string Password { get; set; } = "1956";

public ICommand SignInCommand => new AsyncCommand(async () =>

{

var dbConnection = string.Format(ConfigurationManager.ConnectionStrings["ProviderDB"].ConnectionString, Login, Password);

var connectionResult = await \_service.ChangeConnectionAsync(dbConnection);

if (connectionResult.CodeResult == CodeResult.Bad)

{

MessageBoxManager.ShowError(connectionResult.Errors.First());

return;

}

var userResponse = await \_userService.GetByLoginAsync(Login);

if(userResponse.CodeResult == CodeResult.Bad)

{

MessageBoxManager.ShowError(userResponse.Errors.First());

return;

}

User.Login = userResponse.Result.Login;

User.Password = userResponse.Result.Password;

User.Role = Enum.Parse<UserRole>(userResponse.Result.UserRole);

OnSuccessSignIn?.Invoke();

});

}

}

using DevExpress.Mvvm;

using ProviderSystemManager.DAL.Enums;

using ProviderSystemManager.WPF.Session;

using ProviderSystemManager.WPF.Views.Admin;

using ProviderSystemManager.WPF.Views.Roles;

using System;

using System.Windows.Controls;

using System.Windows.Input;

namespace ProviderSystemManager.WPF.ViewModels

{

internal class MainWindowViewModel : BindableBase

{

public MainWindowViewModel(AdminMainPage adminPage, OperatorMainPage operatorPage, UserMainPage userPage)

{

switch (User.Role)

{

case UserRole.Admin: CurrentPage = adminPage; break;

case UserRole.Operator: CurrentPage = operatorPage; break;

case UserRole.User: CurrentPage = userPage; break;

}

}

public Action OnExitAction { get; set; }

public Page CurrentPage { get; set; }

public ICommand OnExit => new DelegateCommand(() =>

{

OnExitAction?.Invoke();

});

}

}

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.DependencyInjection;

using ProviderSystemManager.BLL.Exporters.Interfaces;

using ProviderSystemManager.BLL.Exporters.QueryExporters;

using ProviderSystemManager.BLL.MappingConfiguration;

using ProviderSystemManager.BLL.Services;

using ProviderSystemManager.BLL.Services.Interfaces;

using ProviderSystemManager.DAL;

using ProviderSystemManager.DAL.Database;

using ProviderSystemManager.DAL.Queries;

using ProviderSystemManager.DAL.Queries.Interfaces;

using ProviderSystemManager.DAL.Repositories;

using ProviderSystemManager.DAL.Repositories.Interfaces;

using ProviderSystemManager.WPF.ViewModels;

using ProviderSystemManager.WPF.ViewModels.Queries;

using ProviderSystemManager.WPF.ViewModels.Roles;

using ProviderSystemManager.WPF.ViewModels.Tables;

using ProviderSystemManager.WPF.ViewModels.Tables.CreateUpdate;

using ProviderSystemManager.WPF.ViewModels.Tables.CreateUpdate.Create;

using ProviderSystemManager.WPF.ViewModels.Tables.CreateUpdate.Update;

using ProviderSystemManager.WPF.Views.Admin;

using ProviderSystemManager.WPF.Views.Queries;

using ProviderSystemManager.WPF.Views.Roles;

using ProviderSystemManager.WPF.Views.Tables;

using ProviderSystemManager.WPF.Views.Tables.CreateUpdate;

using System.Collections.Generic;

using System.Configuration;

namespace ProviderSystemManager.WPF.DI

{

internal static class IoC

{

private static readonly ServiceProvider \_provider;

static IoC()

{

var services = new ServiceCollection();

#region Views

services.AddTransient<AdminMainPage>();

services.AddTransient<OperatorMainPage>();

services.AddTransient<UserMainPage>();

services.AddTransient<UsersPage>();

services.AddTransient<FirmPage>();

services.AddTransient<AbonentPage>();

services.AddTransient<AbonentTypesPage>();

services.AddTransient<OwnTypePage>();

services.AddTransient<ContractPage>();

services.AddTransient<ServicePage>();

services.AddTransient<CommonQueryPage>();

services.AddTransient<GetAbonentsByTypeQueryPage>();

services.AddTransient<FirmNamesByOwnTypeQueryPage>();

services.AddTransient<FirmServiceSizeByDateQueryPage>();

services.AddTransient<ContractsInfoQueryPage>();

services.AddTransient<ServiceInfoQueryPage>();

services.AddTransient<ContractAbonentsEmailNotNullQueryPage>();

services.AddTransient<FirmHaveServicesQueryPage>();

services.AddTransient<FirmsByStartDateWithServicesQueryPage>();

services.AddTransient<AbonentsByServiceRecievingDateQueryPage>();

services.AddTransient<AbonentInfoQueryPage>();

services.AddTransient<SumSizeFirmsQueryPage>();

services.AddTransient<FirmsCountByOwnTypeQueryPage>();

services.AddTransient<AbonentsByContractsSumQueryPage>();

services.AddTransient<AbonentsByContractsSumAndDateQueryPage>();

services.AddTransient<FirmsSumConnectionCostInflationQueryPage>();

services.AddTransient<FirmsSumConnectionCostMoreAvgQueryPage>();

services.AddTransient<AbonentTypeCreateWindow>();

services.AddTransient<OwnTypeCreateWindow>();

#endregion

#region ViewModels

services.AddTransient<ContractPageViewModel>();

services.AddTransient<SignInWindowViewModel>();

services.AddTransient<MainWindowViewModel>();

services.AddTransient<OperatorMainPageViewModel>();

services.AddTransient<AdminMainPageViewModel>();

services.AddTransient<UserMainPageViewModel>();

services.AddTransient<UsersPageViewModel>();

services.AddTransient<FirmPageViewModel>();

services.AddTransient<AbonentPageViewModel>();

services.AddTransient<ServicePageViewModel>();

services.AddSingleton<UserCreateUpdateWindowViewModel>();

services.AddTransient<CommonQueryPageViewModel>();

services.AddTransient<AbonentsByAbonentTypeQueryPageViewModel>();

services.AddTransient<FirmsByOwnTypeQueryPageViewModel>();

services.AddTransient<FirmsByServiceRecievingDateQueryPageViewModel>();

services.AddTransient<ContractsInfoQueryPageViewModel>();

services.AddTransient<ServiceInfoQueryPageViewModel>();

services.AddTransient<ContractAbonentsEmailNotNullQueryPageViewModel>();

services.AddTransient<FirmHaveServicesQueryPageViewModel>();

services.AddTransient<FirmsByStartDateWithServicesQueryPageViewModel>();

services.AddTransient<AbonentsByServiceRecievingDateQueryPageViewModel>();

services.AddTransient<AbonentInfoQueryPageViewModel>();

services.AddTransient<SumSizeFirmsQueryPageViewModel>();

services.AddTransient<FirmsCountByOwnTypeQueryPageViewModel>();

services.AddTransient<AbonentsByContractsSumQueryPageViewModel>();

services.AddTransient<AbonentsByContractsSumAndDateQueryPageViewModel>();

services.AddTransient<FirmsSumConnectionCostInflationQueryPageViewModel>();

services.AddTransient<FirmsSumConnectionCostMoreAvgQueryPageViewModel>();

services.AddTransient<AbonentTypePageViewModel>();

services.AddTransient<OwnTypePageViewModel>();

services.AddTransient<AbonentTypeCreateWindowViewModel>();

services.AddTransient<OwnTypeCreateWindowViewModel>();

services.AddTransient<ContractCreateWindowViewModel>();

services.AddTransient<UserUpdateWindowViewModel>();

services.AddTransient<FirmCreateWindowViewModel>();

services.AddTransient<FirmUpdateWindowViewModel>();

#endregion

#region Database

var dbConnection = string.Format(ConfigurationManager.ConnectionStrings["ProviderDB"].ConnectionString, "postgres", "1956");

services.AddDbContext<ProviderDbContext>(opt => opt.UseNpgsql(dbConnection));

#endregion

#region Repositories

services.AddTransient<IContractRepository, ContractRepository>();

services.AddTransient<IFirmRepository, FirmRepository>();

services.AddTransient<IFirmRepository, FirmRepository>();

services.AddTransient<IUserRepository, UserRepository>();

services.AddTransient<IOwnTypeRepository, OwnTypeRepository>();

services.AddTransient<IAbonentRepository, AbonentRepository>();

services.AddTransient<IServiceRepository, ServiceRepository>();

services.AddTransient<IAbonentTypeRepository, AbonentTypeRepository>();

#endregion

#region Queries

services.AddTransient<IAbonentsByAbonentTypeQuery, AbonentsByAbonentTypeQuery>();

services.AddTransient<IFirmsByOwnType, FirmsByOwnType>();

services.AddTransient<IFirmsByServiceRecievingDate, FirmsByServiceRecievingDate>();

services.AddTransient<IContractsInfoQuery, ContractsInfoQuery>();

services.AddTransient<IServiceInfoQuery, ServiceInfoQuery>();

services.AddTransient<IContractAbonentsEmailNotNullQuery, ContractAbonentsEmailNotNullQuery>();

services.AddTransient<IFirmHaveServicesQuery, FirmHaveServicesQuery>();

services.AddTransient<IFirmsByStartDateWithServicesQuery, FirmsByStartDateWithServicesQuery>();

services.AddTransient<IAbonentsByServiceRecievingDateQuery, AbonentsByServiceRecievingDateQuery>();

services.AddTransient<IAbonentInfoQuery, AbonentInfoQuery>();

services.AddTransient<ISumSizeFirmsQuery, SumSizeFirmsQuery>();

services.AddTransient<IFirmsCountByOwnTypeQuery, FirmsCountByOwnTypeQuery>();

services.AddTransient<IAbonentsByContractsSumQuery, AbonentsByContractsSumQuery>();

services.AddTransient<IAbonentsByContractsSumAndDateQuery, AbonentsByContractsSumAndDateQuery>();

services.AddTransient<IFirmsSumConnectionCostInflationQuery, FirmsSumConnectionCostInflationQuery>();

services.AddTransient<IFirmsSumConnectionCostMoreAvgQuery, FirmsSumConnectionCostMoreAvgQuery>();

#endregion

#region Services

services.AddTransient<IContractService, ContractService>();

services.AddTransient<IFirmService, BLL.Services.FirmService>();

services.AddTransient<IDbContextService, DbContextService>();

services.AddTransient<IFirmService, BLL.Services.FirmService>();

services.AddTransient<IOwnTypeService, OwnTypeService>();

services.AddTransient<IUserService, UserService>();

services.AddTransient<IAbonentService, AbonentService>();

services.AddTransient<IServiceService, ServiceService>();

services.AddTransient<IAbonentTypeService, AbonentTypeService>();

#endregion

#region Exporters

services.AddTransient<IExporter<IEnumerable<SumSizeFirmsModel>>, SumSizeFirmsQueryExporter>();

#endregion

#region AutoMapper

services.AddAutoMapper(typeof(DtoModelProfile));

#endregion

\_provider = services.BuildServiceProvider();

var dbContext = \_provider.GetRequiredService<ProviderDbContext>();

DbInitializer.Initialize(dbContext);

}

public static T Resolve<T>() => \_provider.GetRequiredService<T>();

}

}

using ProviderSystemManager.DAL.Enums;

namespace ProviderSystemManager.WPF.Session

{

internal static class User

{

public static string Login { get; set; }

public static string Password { get; set; }

public static UserRole Role { get; set; }

}

}

using System.Windows;

namespace ProviderSystemManager.WPF.Utils

{

internal static class MessageBoxManager

{

public static MessageBoxResult ShowError(string error) => MessageBox.Show(error, "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

public static MessageBoxResult ShowSuccess(string msg) => MessageBox.Show(msg, "Успех", MessageBoxButton.OK, MessageBoxImage.Information);

}

}

using ProviderSystemManager.WPF.DI;

using ProviderSystemManager.WPF.ViewModels;

using ProviderSystemManager.WPF.ViewModels.Queries;

using ProviderSystemManager.WPF.ViewModels.Roles;

using ProviderSystemManager.WPF.ViewModels.Tables;

using ProviderSystemManager.WPF.ViewModels.Tables.CreateUpdate;

using ProviderSystemManager.WPF.ViewModels.Tables.CreateUpdate.Create;

using ProviderSystemManager.WPF.ViewModels.Tables.CreateUpdate.Update;

namespace ProviderSystemManager.WPF.Utils

{

internal class ViewModelLocator

{

public SignInWindowViewModel SignInWindowViewModel => IoC.Resolve<SignInWindowViewModel>();

public MainWindowViewModel MainWindowViewModel => IoC.Resolve<MainWindowViewModel>();

public AdminMainPageViewModel AdminMainPageViewModel => IoC.Resolve<AdminMainPageViewModel>();

public OperatorMainPageViewModel OperatorMainPageViewModel => IoC.Resolve<OperatorMainPageViewModel>();

public UserMainPageViewModel UserMainPageViewModel => IoC.Resolve<UserMainPageViewModel>();

public UsersPageViewModel UsersPageViewModel => IoC.Resolve<UsersPageViewModel>();

public UserCreateUpdateWindowViewModel UserCreateUpdateWindowViewModel => IoC.Resolve<UserCreateUpdateWindowViewModel>();

public FirmPageViewModel FirmPageViewModel => IoC.Resolve<FirmPageViewModel>();

public AbonentPageViewModel AbonentPageViewModel => IoC.Resolve<AbonentPageViewModel>();

public ContractPageViewModel ContractPageViewModel => IoC.Resolve<ContractPageViewModel>();

public ServicePageViewModel ServicePageViewModel => IoC.Resolve<ServicePageViewModel>();

public CommonQueryPageViewModel CommonQueryPageViewModel => IoC.Resolve<CommonQueryPageViewModel>();

public AbonentsByAbonentTypeQueryPageViewModel GetAbonentsByTypeQueryPageViewModel => IoC.Resolve<AbonentsByAbonentTypeQueryPageViewModel>();

public FirmsByOwnTypeQueryPageViewModel FirmNamesByOwnTypeQueryPageViewModel => IoC.Resolve<FirmsByOwnTypeQueryPageViewModel>();

public FirmsByServiceRecievingDateQueryPageViewModel FirmServiceSizeByDateQueryPageViewModel => IoC.Resolve<FirmsByServiceRecievingDateQueryPageViewModel>();

public ContractsInfoQueryPageViewModel ContractsInfoQueryPageViewModel => IoC.Resolve<ContractsInfoQueryPageViewModel>();

public ServiceInfoQueryPageViewModel ServiceInfoQueryPageViewModel => IoC.Resolve<ServiceInfoQueryPageViewModel>();

public ContractAbonentsEmailNotNullQueryPageViewModel ContractAbonentsEmailNotNullQueryPageViewModel => IoC.Resolve<ContractAbonentsEmailNotNullQueryPageViewModel>();

public FirmHaveServicesQueryPageViewModel FirmHaveServicesQueryPageViewModel => IoC.Resolve<FirmHaveServicesQueryPageViewModel>();

public FirmsByStartDateWithServicesQueryPageViewModel FirmsByStartDateWithServicesQueryPageViewModel => IoC.Resolve<FirmsByStartDateWithServicesQueryPageViewModel>();

public AbonentsByServiceRecievingDateQueryPageViewModel AbonentsByServiceRecievingDateQueryViewModel => IoC.Resolve<AbonentsByServiceRecievingDateQueryPageViewModel>();

public AbonentInfoQueryPageViewModel AbonentInfoQueryPageViewModel => IoC.Resolve<AbonentInfoQueryPageViewModel>();

public SumSizeFirmsQueryPageViewModel SumSizeFirmsQueryPageViewModel => IoC.Resolve<SumSizeFirmsQueryPageViewModel>();

public FirmsCountByOwnTypeQueryPageViewModel FirmsCountByOwnTypeQueryPageViewModel => IoC.Resolve<FirmsCountByOwnTypeQueryPageViewModel>();

public AbonentsByContractsSumQueryPageViewModel AbonentsByContractsSumQueryPageViewModel => IoC.Resolve<AbonentsByContractsSumQueryPageViewModel>();

public AbonentsByContractsSumAndDateQueryPageViewModel AbonentsByContractsSumAndDateQueryPageViewModel => IoC.Resolve<AbonentsByContractsSumAndDateQueryPageViewModel>();

public FirmsSumConnectionCostInflationQueryPageViewModel FirmsSumConnectionCostInflationQueryPageViewModel => IoC.Resolve<FirmsSumConnectionCostInflationQueryPageViewModel>();

public FirmsSumConnectionCostMoreAvgQueryPageViewModel FirmsSumConnectionCostMoreAvgQueryPageViewModel => IoC.Resolve<FirmsSumConnectionCostMoreAvgQueryPageViewModel>();

public AbonentTypePageViewModel AbonentTypePageViewModel => IoC.Resolve<AbonentTypePageViewModel>();

public OwnTypePageViewModel OwnTypePageViewModel => IoC.Resolve<OwnTypePageViewModel>();

public AbonentTypeCreateWindowViewModel AbonentTypeCreateWindowViewModel => IoC.Resolve<AbonentTypeCreateWindowViewModel>();

public OwnTypeCreateWindowViewModel OwnTypeCreateWindowViewModel => IoC.Resolve<OwnTypeCreateWindowViewModel>();

public ContractCreateWindowViewModel ContractCreateWindowViewModel => IoC.Resolve<ContractCreateWindowViewModel>();

public UserUpdateWindowViewModel UserUpdateWindowViewModel => IoC.Resolve<UserUpdateWindowViewModel>();

public FirmCreateWindowViewModel FirmCreateWindowViewModel => IoC.Resolve<FirmCreateWindowViewModel>();

public FirmUpdateWindowViewModel FirmUpdateWindowViewModel => IoC.Resolve<FirmUpdateWindowViewModel>();

}

}

using DevExpress.Mvvm;

using ProviderSystemManager.DAL.Queries.Interfaces;

using System.Collections.ObjectModel;

namespace ProviderSystemManager.WPF.ViewModels.Queries

{

internal class AbonentInfoQueryPageViewModel : BindableBase

{

private readonly IAbonentInfoQuery \_query;

public AbonentInfoQueryPageViewModel(IAbonentInfoQuery query)

{

\_query = query;

Data = new();

}

public ObservableCollection<AbonentInfoModel> Data { get; set; }

public IAsyncCommand Execute => new AsyncCommand(async () =>

{

var result = await \_query.Execute();

Data.Clear();

foreach (var item in result)

{

Data.Add(item);

}

});

}

}

using DevExpress.Mvvm;

using ProviderSystemManager.BLL.Services.Interfaces;

using ProviderSystemManager.DAL.Models.Queries;

using ProviderSystemManager.DAL.Queries.Interfaces;

using ProviderSystemManager.Shared.Dtos.Get;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Linq;

namespace ProviderSystemManager.WPF.ViewModels.Queries

{

internal class AbonentsByAbonentTypeQueryPageViewModel : BindableBase

{

private readonly IAbonentsByAbonentTypeQuery \_query;

public AbonentsByAbonentTypeQueryPageViewModel(IAbonentTypeService abonentTypeService, IAbonentsByAbonentTypeQuery query)

{

\_query = query;

AbonentTypes = new(abonentTypeService.Get().Result);

SelectedAbonentType = AbonentTypes.FirstOrDefault();

Data = new ObservableCollection<EmailsByAbonentTypeModel>();

}

public List<AbonentTypeGetDto> AbonentTypes { get; set; }

public AbonentTypeGetDto SelectedAbonentType { get; set; }

public ObservableCollection<EmailsByAbonentTypeModel> Data { get; set; }

public IAsyncCommand Execute => new AsyncCommand(async () =>

{

var result = await \_query.Execute(SelectedAbonentType.Name);

Data.Clear();

foreach(var item in result)

{

Data.Add(item);

}

});

}

}

using DevExpress.Mvvm;

using ProviderSystemManager.DAL.Queries.Interfaces;

using System;

using System.Collections.ObjectModel;

namespace ProviderSystemManager.WPF.ViewModels.Queries

{

internal class AbonentsByContractsSumAndDateQueryPageViewModel : BindableBase

{

private readonly IAbonentsByContractsSumAndDateQuery \_query;

public AbonentsByContractsSumAndDateQueryPageViewModel(IAbonentsByContractsSumAndDateQuery query)

{

\_query = query;

Data = new();

Date = DateTime.Now;

}

public ObservableCollection<AbonentByContractsSumModel> Data { get; set; }

public int? Count { get; set; }

public DateTime Date { get; set; }

public IAsyncCommand Execute => new AsyncCommand(async () =>

{

var result = await \_query.Execute(Count?.ToString(), Date.ToString("yyyy-MM-dd"));

Data.Clear();

foreach (var item in result)

{

Data.Add(item);

}

});

}

}

using DevExpress.Mvvm;

using ProviderSystemManager.DAL.Queries.Interfaces;

using System.Collections.ObjectModel;

namespace ProviderSystemManager.WPF.ViewModels.Queries

{

internal class AbonentsByContractsSumQueryPageViewModel : BindableBase

{

private readonly IAbonentsByContractsSumQuery \_query;

public AbonentsByContractsSumQueryPageViewModel(IAbonentsByContractsSumQuery query)

{

\_query = query;

Data = new();

}

public ObservableCollection<AbonentByContractsSumModel> Data { get; set; }

public int? Count { get; set; }

public IAsyncCommand Execute => new AsyncCommand(async () =>

{

var result = await \_query.Execute(Count?.ToString());

Data.Clear();

foreach (var item in result)

{

Data.Add(item);

}

});

}

}

using DevExpress.Mvvm;

using ProviderSystemManager.DAL.Queries.Interfaces;

using System;

using System.Collections.ObjectModel;

namespace ProviderSystemManager.WPF.ViewModels.Queries

{

internal class AbonentsByServiceRecievingDateQueryPageViewModel : BindableBase

{

private readonly IAbonentsByServiceRecievingDateQuery \_query;

public AbonentsByServiceRecievingDateQueryPageViewModel(IAbonentsByServiceRecievingDateQuery query)

{

\_query = query;

Date = DateTime.Now;

Data = new();

}

public ObservableCollection<AbonentsByServiceRecievingDateModel> Data { get; set; }

public DateTime Date { get; set; }

public IAsyncCommand Execute => new AsyncCommand(async () =>

{

var result = await \_query.Execute(Date.ToString("yyyy-MM-dd"));

Data.Clear();

foreach (var item in result)

{

Data.Add(item);

}

});

}

}

using DevExpress.Mvvm;

using ProviderSystemManager.WPF.Views.Queries;

using System.Collections.Generic;

using System.Linq;

using System.Windows.Controls;

namespace ProviderSystemManager.WPF.ViewModels.Queries

{

internal class CommonQueryPageViewModel : BindableBase

{

private readonly List<Page> \_queryPages;

public CommonQueryPageViewModel(GetAbonentsByTypeQueryPage getAbonentsByTypeQueryPage,

FirmNamesByOwnTypeQueryPage firmNamesByOwnTypeQueryPage,

FirmServiceSizeByDateQueryPage firmServiceSizeByDateQueryPage,

ContractsInfoQueryPage contractsInfoQueryPage,

ServiceInfoQueryPage serviceInfoQueryPage,

ContractAbonentsEmailNotNullQueryPage contractAbonentsEmailNotNullQueryPage,

FirmHaveServicesQueryPage firmHaveServicesQueryPage,

FirmsByStartDateWithServicesQueryPage firmsByServiceRecievingDateQueryPage,

AbonentsByServiceRecievingDateQueryPage abonentsByServiceRecievingDateQueryPage,

AbonentInfoQueryPage abonentInfoQueryPage,

SumSizeFirmsQueryPage sumSizeFirmsQueryPage,

FirmsCountByOwnTypeQueryPage firmsCountByOwnTypeQueryPage,

AbonentsByContractsSumQueryPage abonentsByContractsSumQueryPage,

AbonentsByContractsSumAndDateQueryPage abonentsByContractsSumAndDateQueryPage,

FirmsSumConnectionCostInflationQueryPage firmsSumConnectionCostInflationQueryPage,

FirmsSumConnectionCostMoreAvgQueryPage firmsSumConnectionCostMoreAvgQueryPage)

{

\_queryPages = new()

{

getAbonentsByTypeQueryPage,

firmNamesByOwnTypeQueryPage,

firmServiceSizeByDateQueryPage,

contractsInfoQueryPage,

serviceInfoQueryPage,

contractAbonentsEmailNotNullQueryPage,

firmHaveServicesQueryPage,

firmsByServiceRecievingDateQueryPage,

abonentsByServiceRecievingDateQueryPage,

abonentInfoQueryPage,

sumSizeFirmsQueryPage,

firmsCountByOwnTypeQueryPage,

abonentsByContractsSumQueryPage,

abonentsByContractsSumAndDateQueryPage,

firmsSumConnectionCostInflationQueryPage,

firmsSumConnectionCostMoreAvgQueryPage

};

CurrentPage = \_queryPages.FirstOrDefault();

QueryTitles = new()

{

"Вывести абонентов с указанным типом",

"Вывести фирмы с заданным типом собственности",

"Вывести все фирмы, которые предоставляли услуги в указанную дату",

"Вывести абонентов и стоимость подключения их контрактов",

"Вывести абонентов и объем сообщения их услуг",

"Вывести все контракты абонентов, у которых указана почта",

"Вывести все фирмы, которые хоть раз отказывали услуги",

"Вывести все фирмы, которые хоть раз отказывали услуги и были открыты в указанный период",

"Вывести всех абонентов и объем сообщения услуг, предоставленных в указанную дату",

"Вывести имена абонентов и их тип",

"Вывести фирмы и их общее число предоставленных МБ",

"Вывести количество фирм с указанным типом собственности",

"Вывести абонентов, которые составили контрактов на подключение в сумме на цену более указанного числа",

"Вывести абонентов, которые составили контрактов на подключение в сумме на цену более указанного числа и которые начинаются в указанную дату",

"Вывести фирмы и их общую прибыль за подключение до и после инфляции (уменьшение на 30%), в период за между двумя датами",

"Вывести фирмы и их общую прибыль за подключения, сумма которой больше средней прибыли за подключения"

};

SelectedTitle = QueryTitles.FirstOrDefault();

}

public Page CurrentPage { get; set; }

public List<string> QueryTitles { get; set; }

public string SelectedTitle {

get => GetValue<string>(nameof(SelectedTitle));

set

{

var selectedIndex = QueryTitles.IndexOf(value);

CurrentPage = \_queryPages[selectedIndex];

SetValue(value, nameof(SelectedTitle));

}

}

}

}

using DevExpress.Mvvm;

using ProviderSystemManager.DAL.Models.Queries;

using ProviderSystemManager.DAL.Queries.Interfaces;

using System.Collections.ObjectModel;

namespace ProviderSystemManager.WPF.ViewModels.Queries

{

internal class ContractAbonentsEmailNotNullQueryPageViewModel : BindableBase

{

private readonly IContractAbonentsEmailNotNullQuery \_query;

public ContractAbonentsEmailNotNullQueryPageViewModel(IContractAbonentsEmailNotNullQuery query)

{ using DevExpress.Mvvm;

using ProviderSystemManager.DAL.Models.Queries;

using ProviderSystemManager.DAL.Queries.Interfaces;

using System.Collections.ObjectModel;

namespace ProviderSystemManager.WPF.ViewModels.Queries

{

internal class FirmHaveServicesQueryPageViewModel : BindableBase

{

private readonly IFirmHaveServicesQuery \_query;

public FirmHaveServicesQueryPageViewModel(IFirmHaveServicesQuery query)

{

\_query = query;

Data = new ObservableCollection<FirmService>();

}

public ObservableCollection<FirmService> Data { get; set; }

public IAsyncCommand Execute => new AsyncCommand(async () =>

{

var result = await \_query.Execute();

Data.Clear();

foreach (var item in result)

{

Data.Add(item);

}

});

}

}

\_query = query;

Data = new ObservableCollection<ContractsInfo>();

}

public ObservableCollection<ContractsInfo> Data { get; set; }

public IAsyncCommand Execute => new AsyncCommand(async () =>

{

var result = await \_query.Execute();

Data.Clear();

foreach (var item in result)

{

Data.Add(item);

}

});

}

}

using DevExpress.Mvvm;

using ProviderSystemManager.BLL.Services.Interfaces;

using ProviderSystemManager.DAL.Queries.Interfaces;

using ProviderSystemManager.Shared.Dtos.Get;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Linq;

namespace ProviderSystemManager.WPF.ViewModels.Queries

{

internal class FirmsByOwnTypeQueryPageViewModel : BindableBase

{

private readonly IFirmsByOwnType \_query;

public FirmsByOwnTypeQueryPageViewModel(IOwnTypeService service, IFirmsByOwnType query)

{

\_query = query;

OwnTypes = new(service.Get().Result);

SelectedOwnType = OwnTypes.FirstOrDefault();

Data = new ObservableCollection<string>();

}

public List<OwnTypeGetDto> OwnTypes { get; set; }

public OwnTypeGetDto SelectedOwnType { get; set; }

public ObservableCollection<string> Data { get; set; }

public IAsyncCommand Execute => new AsyncCommand(async () =>

{

var result = await \_query.Execute(SelectedOwnType.Name);

Data.Clear();

foreach (var item in result)

{

Data.Add(item);

}

});

}

}

using DevExpress.Mvvm;

using ProviderSystemManager.DAL.Models.Queries;

using ProviderSystemManager.DAL.Queries.Interfaces;

using System;

using System.Collections.ObjectModel;

namespace ProviderSystemManager.WPF.ViewModels.Queries

{

internal class FirmsByServiceRecievingDateQueryPageViewModel : BindableBase

{

private readonly IFirmsByServiceRecievingDate \_query;

public FirmsByServiceRecievingDateQueryPageViewModel(IFirmsByServiceRecievingDate query)

{

\_query = query;

Date = DateTime.Now;

Data = new();

}

public ObservableCollection<FirmService> Data { get; set; }

public DateTime Date { get; set; }

public IAsyncCommand Execute => new AsyncCommand(async () =>

{

var result = await \_query.Execute(Date.ToString("yyyy-MM-dd"));

Data.Clear();

foreach (var item in result)

{

Data.Add(item);

}

});

}

}

using DevExpress.Mvvm;

using ProviderSystemManager.DAL.Models.Queries;

using ProviderSystemManager.DAL.Queries.Interfaces;

using System;

using System.Collections.ObjectModel;

namespace ProviderSystemManager.WPF.ViewModels.Queries

{

internal class FirmsByStartDateWithServicesQueryPageViewModel : BindableBase

{

private readonly IFirmsByStartDateWithServicesQuery \_query;

public FirmsByStartDateWithServicesQueryPageViewModel(IFirmsByStartDateWithServicesQuery query)

{

\_query = query;

FirstYear = DateTime.Now.Year;

SecondYear = DateTime.Now.Year;

Data = new();

}

public ObservableCollection<FirmService> Data { get; set; }

public int FirstYear { get; set; }

public int SecondYear { get; set; }

public IAsyncCommand Execute => new AsyncCommand(async () =>

{

var result = await \_query.Execute(FirstYear.ToString(), SecondYear.ToString());

Data.Clear();

foreach (var item in result)

{

Data.Add(item);

}

});

}

}

using DevExpress.Mvvm;

using ProviderSystemManager.BLL.Services.Interfaces;

using ProviderSystemManager.DAL.Queries.Interfaces;

using ProviderSystemManager.Shared.Dtos.Get;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Linq;

namespace ProviderSystemManager.WPF.ViewModels.Queries

{

internal class FirmsCountByOwnTypeQueryPageViewModel : BindableBase

{

private readonly IFirmsCountByOwnTypeQuery \_query;

public FirmsCountByOwnTypeQueryPageViewModel(IOwnTypeService service, IFirmsCountByOwnTypeQuery query)

{

\_query = query;

Data = new();

OwnTypes = new(service.Get().Result);

SelectedOwnType = OwnTypes.FirstOrDefault();

}

public ObservableCollection<int> Data { get; set; }

public List<OwnTypeGetDto> OwnTypes { get; set; }

public OwnTypeGetDto SelectedOwnType { get; set; }

public IAsyncCommand Execute => new AsyncCommand(async () =>

{

Data.Clear();

Data.Add(await \_query.Execute(SelectedOwnType.Name));

});

}

}

using DevExpress.Mvvm;

using ProviderSystemManager.DAL.Queries.Interfaces;

using System;

using System.Collections.ObjectModel;

namespace ProviderSystemManager.WPF.ViewModels.Queries

{

internal class FirmsSumConnectionCostInflationQueryPageViewModel : BindableBase

{

private readonly IFirmsSumConnectionCostInflationQuery \_query;

public FirmsSumConnectionCostInflationQueryPageViewModel(IFirmsSumConnectionCostInflationQuery query)

{

\_query = query;

FirstDate = DateTime.Now;

SecondDate = DateTime.Now;

Data = new();

}

public DateTime FirstDate { get; set; }

public DateTime SecondDate { get; set; }

public ObservableCollection<FirmsSumConnectionCostInflationModel> Data { get; set; }

public IAsyncCommand Execute => new AsyncCommand(async () =>

{

var result = await \_query.Execute(FirstDate.ToString("yyyy-MM-dd"), SecondDate.ToString("yyyy-MM-dd"));

Data.Clear();

foreach (var item in result)

{

Data.Add(item);

}

});

}

}

using DevExpress.Mvvm;

using ProviderSystemManager.DAL.Queries.Interfaces;

using System.Collections.ObjectModel;

namespace ProviderSystemManager.WPF.ViewModels.Queries

{

internal class FirmsSumConnectionCostMoreAvgQueryPageViewModel : BindableBase

{

private readonly IFirmsSumConnectionCostMoreAvgQuery \_query;

public FirmsSumConnectionCostMoreAvgQueryPageViewModel(IFirmsSumConnectionCostMoreAvgQuery query)

{

\_query = query;

Data = new();

}

public ObservableCollection<FirmsSumConnectionCostMoreAvgModel> Data { get; set; }

public IAsyncCommand Execute => new AsyncCommand(async () =>

{

var result = await \_query.Execute();

Data.Clear();

foreach (var item in result)

{

Data.Add(item);

}

});

}

}

using DevExpress.Mvvm;

using ProviderSystemManager.DAL.Models.Queries;

using ProviderSystemManager.DAL.Queries.Interfaces;

using System.Collections.ObjectModel;

namespace ProviderSystemManager.WPF.ViewModels.Queries

{

internal class ServiceInfoQueryPageViewModel

{

public ServiceInfoQueryPageViewModel(IServiceInfoQuery query)

{

\_query = query;

Data = new ObservableCollection<ServiceInfo>();

}

private readonly IServiceInfoQuery \_query;

public ObservableCollection<ServiceInfo> Data { get; set; }

public IAsyncCommand Execute => new AsyncCommand(async () =>

{

var result = await \_query.Execute();

Data.Clear();

foreach (var item in result)

{

Data.Add(item);

}

});

}

}

using DevExpress.Mvvm;

using ProviderSystemManager.BLL.Exporters.Interfaces;

using ProviderSystemManager.DAL.Queries.Interfaces;

using System;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.IO;

namespace ProviderSystemManager.WPF.ViewModels.Queries

{

public class SumSizeFirmsQueryPageViewModel : BindableBase

{

private readonly IExporter<IEnumerable<SumSizeFirmsModel>> \_exporter;

private readonly ISumSizeFirmsQuery \_query;

public SumSizeFirmsQueryPageViewModel(ISumSizeFirmsQuery query, IExporter<IEnumerable<SumSizeFirmsModel>> exporter)

{

\_exporter = exporter;

\_query = query;

Data = new();

}

public ObservableCollection<SumSizeFirmsModel> Data { get; set; }

public IAsyncCommand Execute => new AsyncCommand(async () =>

{

var result = await \_query.Execute();

Data.Clear();

foreach (var item in result)

{

Data.Add(item);

}

});

public IAsyncCommand ToExcel => new AsyncCommand(async () =>

{

var result = await \_query.Execute();

var stream = await \_exporter.Export(result);

using var filestream = new FileStream($"{string.Format("{0:ddMMyyyyHHmm}", DateTime.Now)}.xlsx", FileMode.Create, FileAccess.Write);

await stream.CopyToAsync(filestream);

stream.Close();

});

}

}

using DevExpress.Mvvm;

using ProviderSystemManager.WPF.Views.Queries;

using ProviderSystemManager.WPF.Views.Tables;

using System.Windows.Controls;

using System.Windows.Input;

namespace ProviderSystemManager.WPF.ViewModels.Roles

{

internal class AdminMainPageViewModel : BindableBase

{

private readonly UsersPage \_usersPage;

private readonly CommonQueryPage \_queryPage;

private readonly AbonentTypesPage \_abonentTypesPage;

public AdminMainPageViewModel(UsersPage usersPage, CommonQueryPage queryPage, AbonentTypesPage abonentTypesPage, OwnTypePage ownTypePage)

{

\_usersPage = usersPage;

\_queryPage = queryPage;

\_abonentTypesPage = abonentTypesPage;

\_ownTypePage = ownTypePage;

CurrentTablePage = \_usersPage;

}

private readonly OwnTypePage \_ownTypePage;

public Page CurrentTablePage { get; set; }

public Page OperatorsPage { get; set; }

public ICommand OnTableSelect => new DelegateCommand<string>((tableName) =>

{

switch (tableName)

{

case "users": CurrentTablePage = \_usersPage; break;

case "queries": CurrentTablePage = \_queryPage; break;

case "abonent types": CurrentTablePage = \_abonentTypesPage; break;

case "own types": CurrentTablePage = \_ownTypePage; break;

}

});

}

}

using DevExpress.Mvvm;

using ProviderSystemManager.WPF.Views.Tables;

using System.Windows.Controls;

using System.Windows.Input;

namespace ProviderSystemManager.WPF.ViewModels.Roles

{

internal class OperatorMainPageViewModel : BindableBase

{

public OperatorMainPageViewModel(FirmPage firmPage)

{

CurrentTablePage = firmPage;

}

public Page CurrentTablePage { get; set; }

}

}

using DevExpress.Mvvm;

using ProviderSystemManager.WPF.Views.Tables;

using System.Windows.Controls;

using System.Windows.Input;

namespace ProviderSystemManager.WPF.ViewModels.Roles

{

internal class UserMainPageViewModel : BindableBase

{

private readonly AbonentPage \_abonentPage;

private readonly ContractPage \_contractPage;

private readonly ServicePage \_servicePage;

public UserMainPageViewModel(AbonentPage abonentPage, ContractPage contractPage, ServicePage servicePage)

{

\_abonentPage = abonentPage;

\_contractPage = contractPage;

\_servicePage = servicePage;

CurrentTablePage = contractPage;

}

public Page CurrentTablePage { get; set; }

public ICommand OnTableSelect => new DelegateCommand<string>((tableName) =>

{

switch (tableName.ToString())

{

case "abonent": CurrentTablePage = \_abonentPage; break;

case "contract": CurrentTablePage = \_contractPage; break;

case "service": CurrentTablePage = \_servicePage; break;

}

});

}

}

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

using ProviderSystemManager.DAL.Models;

namespace ProviderSystemManager.DAL.Configuration;

public class AbonentConfiguration : IEntityTypeConfiguration<Abonent>

{

public void Configure(EntityTypeBuilder<Abonent> builder)

{

builder.ToTable("abonents");

builder.HasKey(x => x.Id);

builder.Property(x => x.Id).HasColumnName("id");

builder.Property(x => x.Name)

.HasColumnName("name")

.IsRequired();

builder.HasIndex(x => x.Email).IsUnique();

builder.Property(x => x.Email).HasColumnName("email");

builder.HasIndex(x => x.Address).IsUnique();

builder.Property(x => x.Address)

.HasColumnName("address")

.IsRequired();

builder.Property(x => x.AbonentTypeId).HasColumnName("abonent\_type\_id");

builder.HasMany(x => x.Services)

.WithOne(a => a.Abonent);

builder.HasOne(x => x.AbonentType)

.WithMany(a => a.Abonents)

.HasForeignKey(k => k.AbonentTypeId);

}

} using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

using ProviderSystemManager.DAL.Models;

namespace ProviderSystemManager.DAL.Configuration;

public class AbonentTypeConfiguration: IEntityTypeConfiguration<AbonentType>

{

public void Configure(EntityTypeBuilder<AbonentType> builder)

{

builder.ToTable("abonent\_types");

builder.HasKey(x => x.Id);

builder.Property(x => x.Id)

.HasColumnName("id");

builder.Property(x => x.Name)

.HasColumnName("name")

.HasMaxLength(20);

builder.HasMany(x => x.Abonents)

.WithOne(a => a.AbonentType)

.HasForeignKey(x => x.AbonentTypeId);

}

} using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

using ProviderSystemManager.DAL.Models;

namespace ProviderSystemManager.DAL.Configuration;

public class ContractConfiguration : IEntityTypeConfiguration<Contract>

{

public void Configure(EntityTypeBuilder<Contract> builder)

{

builder.ToTable("contracts");

builder.HasKey(x => x.Id);

builder.Property(x => x.Id).HasColumnName("id");

builder.HasOne(x => x.Firm)

.WithMany(f => f.Contracts)

.HasForeignKey(f => f.FirmId);

builder.HasOne(x => x.Abonent)

.WithMany(f => f.Contracts)

.HasForeignKey(f => f.AbonentId);

builder.Property(x => x.ConnectionCost)

.HasColumnName("connection\_cost")

.IsRequired();

builder.Property(x => x.ConnectionDate)

.HasColumnName("connection\_date")

.IsRequired();

builder.Property(x => x.ForwardingCost)

.HasColumnName("forwarding\_cost")

.IsRequired();

builder.Property(x => x.AbonentId).HasColumnName("abonent\_id");

builder.Property(x => x.FirmId).HasColumnName("firm\_id");

}

} using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

using ProviderSystemManager.DAL.Models;

namespace ProviderSystemManager.DAL.Configuration;

public class FirmConfiguration : IEntityTypeConfiguration<Firm>

{

public void Configure(EntityTypeBuilder<Firm> builder)

{

builder.ToTable("firms");

builder.HasKey(x => x.Id);

builder.Property(x => x.Id).HasColumnName("id");

builder.Property(x => x.Name)

.HasColumnName("name")

.HasMaxLength(40);

builder.Property(x => x.Telephone)

.HasColumnName("telephone")

.HasMaxLength(15);

builder.Property(x => x.Address)

.HasColumnName("address")

.HasMaxLength(30);

builder.Property(x => x.OwnTypeId)

.HasColumnName("own\_type\_id");

builder.Property(x => x.StartWorkingYear)

.HasColumnName("start\_working\_year");

}

} using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

using ProviderSystemManager.DAL.Models;

namespace ProviderSystemManager.DAL.Configuration;

public class OwnTypeConfiguration : IEntityTypeConfiguration<OwnType>

{

public void Configure(EntityTypeBuilder<OwnType> builder)

{

builder.ToTable("own\_types");

builder.HasKey(x => x.Id);

builder.Property(x => x.Id).HasColumnName("id");

builder.Property(x => x.Name)

.HasColumnName("name")

.HasMaxLength(20);

builder.HasMany(x => x.Firms)

.WithOne(m => m.OwnType)

.HasForeignKey(k => k.OwnTypeId);

}

} using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

using ProviderSystemManager.DAL.Models;

namespace ProviderSystemManager.DAL.Configuration;

public class ServiceConfiguration : IEntityTypeConfiguration<Service>

{

public void Configure(EntityTypeBuilder<Service> builder)

{

builder.ToTable("services");

builder.HasKey(x => x.Id);

builder.Property(x => x.Id).HasColumnName("id");

builder.Property(x => x.AbonentId)

.HasColumnName("abonent\_id")

.IsRequired();

builder.Property(x => x.Size)

.HasColumnName("size");

builder.Property(x => x.FirmId)

.HasColumnName("firm\_id");

builder.Property(x => x.RecievingDate)

.HasColumnName("recieving\_date");

builder.HasOne(x => x.Abonent)

.WithMany(a => a.Services)

.HasForeignKey(k => k.AbonentId);

builder.HasOne(x => x.Firm)

.WithMany(m => m.Services)

.HasForeignKey(k => k.FirmId);

}

} using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

using ProviderSystemManager.DAL.Models;

namespace ProviderSystemManager.DAL.Configuration;

public class UserConfiguration : IEntityTypeConfiguration<User>

{

public void Configure(EntityTypeBuilder<User> builder)

{

builder.ToTable("users");

builder.HasKey(x => x.Id);

builder.HasIndex(x => x.Login)

.IsUnique();

builder.Property(x => x.Id).HasColumnName("id");

builder.Property(x => x.Role).HasColumnName("role");

builder.Property(x => x.Login)

.HasColumnName("login")

.IsRequired()

.HasMaxLength(20);

builder.Property(x => x.Password)

.HasColumnName("password")

.IsRequired()

.HasMaxLength(150);

}

} using Microsoft.EntityFrameworkCore;

using ProviderSystemManager.DAL.Database;

using ProviderSystemManager.DAL.QueryCreators;

using ProviderSystemManager.DAL.TableCreators;

namespace ProviderSystemManager.DAL;

public class DbInitializer

{

public static void Initialize(ProviderDbContext dbContext)

{

dbContext.Database.EnsureDeleted();

dbContext.Database.Migrate();

QueryCreator.Init(dbContext);

UserCreator.Init(dbContext);

OwnTypeCreator.Init(dbContext);

AbonentTypeCreator.Init(dbContext);

AbonentCreator.Init(dbContext);

FirmCreator.Init(dbContext);

ContractCreator.Init(dbContext);

ServiceCreator.Init(dbContext);

}

} using Microsoft.EntityFrameworkCore;

using ProviderSystemManager.DAL.Configuration;

using ProviderSystemManager.DAL.Models;

namespace ProviderSystemManager.DAL.Database;

public class ProviderDbContext : DbContext

{

public ProviderDbContext() : base() { }

public ProviderDbContext(DbContextOptions<ProviderDbContext> options) : base(options) { }

public DbSet<OwnType> OwnTypes { get; set; }

public DbSet<AbonentType> AbonentTypes { get; set; }

public DbSet<Abonent> Abonents { get; set; }

public DbSet<Service> Services { get; set; }

public DbSet<Firm> Firms { get; set; }

public DbSet<Contract> Contracts { get; set; }

public DbSet<User> Users { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.ApplyConfiguration(new OwnTypeConfiguration());

modelBuilder.ApplyConfiguration(new AbonentTypeConfiguration());

modelBuilder.ApplyConfiguration(new AbonentConfiguration());

modelBuilder.ApplyConfiguration(new ServiceConfiguration());

modelBuilder.ApplyConfiguration(new FirmConfiguration());

modelBuilder.ApplyConfiguration(new ContractConfiguration());

modelBuilder.ApplyConfiguration(new UserConfiguration());

base.OnModelCreating(modelBuilder);

}

protected override void OnConfiguring(DbContextOptionsBuilder builder)

{

builder.UseNpgsql("User ID=postgres;Password=1956;Host=localhost;Port=5432;Database=ProviderSM;Pooling=true;");

//builder.LogTo(Console.WriteLine);

}

} namespace ProviderSystemManager.DAL.Enums;

public enum UserRole

{

Admin,

Operator,

User

} namespace ProviderSystemManager.DAL.Models.Queries

{

public class ContractsInfo

{

public string AbonentName { get; set; }

public double ConnectionCost { get; set; }

public string Email { get; set; }

}

}

namespace ProviderSystemManager.DAL.Models.Queries

{

public class EmailsByAbonentTypeModel

{

public string Name { get; set; }

public string Email { get; set; }

}

}

namespace ProviderSystemManager.DAL.Models.Queries

{

public class FirmService

{

public string FirmName { get; set; }

public double Size { get; set; }

public DateTime RecievingDate { get; set; }

public int StartWorkingDate { get; set; }

}

}

namespace ProviderSystemManager.DAL.Models.Queries

{

public class ServiceInfo

{

public string AbonentName { get; set; }

public double Size { get; set; }

}

}

namespace ProviderSystemManager.DAL.Models;

public class Abonent : BaseModel

{

public string Name { get; set; }

public string Email { get; set; }

public string Address { get; set; }

public int AbonentTypeId { get; set; }

public AbonentType AbonentType { get; set; }

public virtual ICollection<Service> Services { get; set; }

public virtual ICollection<Contract> Contracts { get; set; }

} namespace ProviderSystemManager.DAL.Models;

public class AbonentType : BaseModel

{

public string Name { get; set; }

public virtual ICollection<Abonent> Abonents { get; set; }

} namespace ProviderSystemManager.DAL.Models;

public class BaseModel

{

public int Id { get; set; }

} namespace ProviderSystemManager.DAL.Models;

public class Contract : BaseModel

{

public int FirmId { get; set; }

public Firm Firm { get; set; }

public int AbonentId { get; set; }

public Abonent Abonent { get; set; }

public DateOnly ConnectionDate { get; set; }

public decimal ConnectionCost { get; set; }

public decimal ForwardingCost { get; set; }

} namespace ProviderSystemManager.DAL.Models;

public class Firm : BaseModel

{

public string Name { get; set; }

public string Telephone { get; set; }

public string Address { get; set; }

public short StartWorkingYear{ get; set; }

public int OwnTypeId { get; set; }

public OwnType OwnType { get; set; }

public virtual ICollection<Service> Services { get; set; }

public virtual ICollection<Contract> Contracts { get; set; }

} namespace ProviderSystemManager.DAL.Models;

public class OwnType : BaseModel

{

public string Name { get; set; }

public virtual ICollection<Firm> Firms { get; set; }

} namespace ProviderSystemManager.DAL.Models;

public class Service : BaseModel

{

public int AbonentId { get; set; }

public Abonent Abonent { get; set; }

public DateOnly RecievingDate { get; set; }

public double Size { get; set; }

public int FirmId { get; set; }

public Firm Firm { get; set; }

} using ProviderSystemManager.DAL.Enums;

namespace ProviderSystemManager.DAL.Models;

public class User : BaseModel

{

public string Login { get; set; }

public string Password { get; set; }

public UserRole Role { get; set; }

} using ProviderSystemManager.DAL.Database;

using ProviderSystemManager.DAL.Queries.Interfaces;

using System.Data.Common;

namespace ProviderSystemManager.DAL.Queries

{

public class AbonentInfoQuery : AbstractQuery<IEnumerable<AbonentInfoModel>>, IAbonentInfoQuery

{

public AbonentInfoQuery(ProviderDbContext dbContext) : base(dbContext)

{

}

public override string CreateQuery => "create or replace view get\_abonent\_info as select a.name as \"абонент\", atp.name as \"тип\" from abonents a inner join abonent\_types atp on atp.id = a.abonent\_type\_id;";

protected override string ExecuteQuery => "SELECT \* FROM get\_abonent\_info";

protected override async Task<IEnumerable<AbonentInfoModel>> ConvertData(DbDataReader reader)

{

var list = new List<AbonentInfoModel>();

while (await reader.ReadAsync())

{

list.Add(new(reader.GetString(0), reader.GetString(1)));

}

return list;

}

}

}

using Microsoft.EntityFrameworkCore;

using ProviderSystemManager.DAL.Database;

using ProviderSystemManager.DAL.Models.Queries;

using ProviderSystemManager.DAL.Queries.Interfaces;

using System.Data.Common;

namespace ProviderSystemManager.DAL.Queries

{

public class AbonentsByAbonentTypeQuery : AbstractQuery<IEnumerable<EmailsByAbonentTypeModel>>, IAbonentsByAbonentTypeQuery

{

public AbonentsByAbonentTypeQuery(ProviderDbContext dbContext) : base(dbContext)

{

}

public override string CreateQuery => "CREATE OR REPLACE FUNCTION get\_abonents\_by\_abonent\_type(abonent\_type TEXT) RETURNS TABLE(\"Имя\" TEXT, \"Почта\" TEXT) AS $$ BEGIN RETURN QUERY SELECT a.name, a.email FROM abonents a INNER JOIN abonent\_types abt ON abt.id = a.abonent\_type\_id WHERE abt.name = abonent\_type ORDER BY a.name; END; $$ LANGUAGE plpgsql;";

protected override string ExecuteQuery => "SELECT \* FROM get\_abonents\_by\_abonent\_type('{0}')";

protected override async Task<IEnumerable<EmailsByAbonentTypeModel>> ConvertData(DbDataReader reader)

{

var list = new List<EmailsByAbonentTypeModel>();

while (await reader.ReadAsync())

{

list.Add(new()

{

Name = reader.GetString(0),

Email = reader.GetString(1)

});

}

return list;

}

}

}

using ProviderSystemManager.DAL.Database;

using ProviderSystemManager.DAL.Queries.Interfaces;

using System.Data.Common;

namespace ProviderSystemManager.DAL.Queries

{

public class AbonentsByContractsSumAndDateQuery : AbstractQuery<IEnumerable<AbonentByContractsSumModel>>, IAbonentsByContractsSumAndDateQuery

{

public AbonentsByContractsSumAndDateQuery(ProviderDbContext dbContext) : base(dbContext)

{

}

public override string CreateQuery => "create or replace function get\_abonents\_by\_contracts\_sum\_and\_date(sum\_value double precision, con\_date date) returns table(\"имя\" text, \"сумма\" numeric) as $$ begin return query select a.name, sum(c.connection\_cost) from contracts c join abonents a on a.id = c.abonent\_id where c.connection\_date=con\_date group by a.name having sum(c.connection\_cost) > sum\_value; end; $$ language plpgsql;";

protected override string ExecuteQuery => "SELECT \* FROM get\_abonents\_by\_contracts\_sum\_and\_date({0}, '{1}')";

protected override async Task<IEnumerable<AbonentByContractsSumModel>> ConvertData(DbDataReader reader)

{

var list = new List<AbonentByContractsSumModel>();

while (await reader.ReadAsync())

{

list.Add(new(reader.GetString(0), reader.GetDouble(1)));

}

return list;

}

}

}

using ProviderSystemManager.DAL.Database;

using ProviderSystemManager.DAL.Queries.Interfaces;

using System.Data.Common;

namespace ProviderSystemManager.DAL.Queries

{

public class AbonentsByContractsSumQuery : AbstractQuery<IEnumerable<AbonentByContractsSumModel>>, IAbonentsByContractsSumQuery

{

public AbonentsByContractsSumQuery(ProviderDbContext dbContext) : base(dbContext)

{

}

public override string CreateQuery => "create or replace function get\_abonents\_by\_contracts\_sum(sum\_value double precision) returns table(\"имя\" text, \"сумма\" numeric) as $$ begin return query select a.name, sum(c.connection\_cost) from contracts c join abonents a on a.id = c.abonent\_id group by a.name having sum(c.connection\_cost) > sum\_value; end; $$ language plpgsql;";

protected override string ExecuteQuery => "SELECT \* FROM get\_abonents\_by\_contracts\_sum({0})";

protected override async Task<IEnumerable<AbonentByContractsSumModel>> ConvertData(DbDataReader reader)

{

var list = new List<AbonentByContractsSumModel>();

while (await reader.ReadAsync())

{

list.Add(new(reader.GetString(0), reader.GetDouble(1)));

}

return list;

}

}

}

using ProviderSystemManager.DAL.Database;

using ProviderSystemManager.DAL.Queries.Interfaces;

using System.Data.Common;

namespace ProviderSystemManager.DAL.Queries

{

public class AbonentsByServiceRecievingDateQuery : AbstractQuery<IEnumerable<AbonentsByServiceRecievingDateModel>>, IAbonentsByServiceRecievingDateQuery

{

public AbonentsByServiceRecievingDateQuery(ProviderDbContext dbContext) : base(dbContext)

{

}

public override string CreateQuery => "create or replace function get\_abonents\_by\_service\_recieving\_date(recieve\_date date) returns table(\"имя\" text, \"объем сообщения\" double precision) as $$ begin return query select a.name, s.size from abonents a inner join services s on s.abonent\_id = a.id where s.recieving\_date = recieve\_date order by a.name; end; $$ language plpgsql;";

protected override string ExecuteQuery => "SELECT\* FROM get\_abonents\_by\_service\_recieving\_date('{0}');";

protected override async Task<IEnumerable<AbonentsByServiceRecievingDateModel>> ConvertData(DbDataReader reader)

{

var list = new List<AbonentsByServiceRecievingDateModel>();

while (await reader.ReadAsync())

{

list.Add(new(reader.GetString(0), reader.GetDouble(1)));

}

return list;

}

}

}

using ProviderSystemManager.DAL.Database;

using ProviderSystemManager.DAL.Models.Queries;

using ProviderSystemManager.DAL.Queries.Interfaces;

using System.Data.Common;

namespace ProviderSystemManager.DAL.Queries

{

public class ContractAbonentsEmailNotNullQuery : AbstractQuery<IEnumerable<ContractsInfo>>, IContractAbonentsEmailNotNullQuery

{

public ContractAbonentsEmailNotNullQuery(ProviderDbContext dbContext) : base(dbContext)

{

}

public override string CreateQuery => "CREATE OR REPLACE VIEW get\_contract\_abonents\_email\_not\_null AS SELECT c.connection\_cost, a.name, a.email FROM contracts c LEFT OUTER JOIN abonents a ON a.id = c.abonent\_id WHERE a.email IS NOT NULL;";

protected override string ExecuteQuery => "SELECT \* FROM get\_contract\_abonents\_email\_not\_null;";

protected override async Task<IEnumerable<ContractsInfo>> ConvertData(DbDataReader reader)

{

var list = new List<ContractsInfo>();

while (await reader.ReadAsync())

{

list.Add(new()

{

ConnectionCost = reader.GetDouble(0),

AbonentName = reader.GetString(1),

Email = reader.GetString(2)

});

}

return list;

}

}

}

using ProviderSystemManager.DAL.Database;

using ProviderSystemManager.DAL.Models.Queries;

using ProviderSystemManager.DAL.Queries.Interfaces;

using System.Data.Common;

namespace ProviderSystemManager.DAL.Queries

{

public class ContractsInfoQuery : AbstractQuery<IEnumerable<ContractsInfo>>, IContractsInfoQuery

{

public ContractsInfoQuery(ProviderDbContext dbContext) : base(dbContext)

{

}

public override string CreateQuery => "CREATE OR REPLACE VIEW get\_contracts\_info AS SELECT c.connection\_cost, a.name FROM contracts c INNER JOIN abonents a ON a.id = c.abonent\_id;";

protected override string ExecuteQuery => "SELECT \* FROM get\_contracts\_info;";

protected override async Task<IEnumerable<ContractsInfo>> ConvertData(DbDataReader reader)

{

var list = new List<ContractsInfo>();

while(await reader.ReadAsync())

{

list.Add(new()

{

ConnectionCost = reader.GetDouble(0),

AbonentName = reader.GetString(1)

});

}

return list;

}

}

}

using ProviderSystemManager.DAL.Database;

using ProviderSystemManager.DAL.Models.Queries;

using ProviderSystemManager.DAL.Queries.Interfaces;

using System.Data.Common;

namespace ProviderSystemManager.DAL.Queries

{

public class FirmHaveServicesQuery : AbstractQuery<IEnumerable<FirmService>>, IFirmHaveServicesQuery

{

public FirmHaveServicesQuery(ProviderDbContext dbContext) : base(dbContext)

{

}

public override string CreateQuery => "CREATE OR REPLACE VIEW get\_firms\_have\_services AS SELECT f.name AS \"Название компании\", s.recieving\_date \"Дата предоставления услуги\" FROM services s RIGHT OUTER JOIN firms f ON f.id = s.firm\_id WHERE s.id IS NOT NULL;";

protected override string ExecuteQuery => "SELECT \* FROM get\_firms\_have\_services";

protected override async Task<IEnumerable<FirmService>> ConvertData(DbDataReader reader)

{

var list = new List<FirmService>();

while (await reader.ReadAsync())

{

list.Add(new()

{

FirmName = reader.GetString(0),

RecievingDate = reader.GetDateTime(1)

});

}

return list;

}

}

}

using ProviderSystemManager.DAL.Database;

using ProviderSystemManager.DAL.Queries.Interfaces;

using System.Data.Common;

namespace ProviderSystemManager.DAL.Queries

{

public class FirmsByOwnType : AbstractQuery<IEnumerable<string>>, IFirmsByOwnType

{

public FirmsByOwnType(ProviderDbContext dbContext) : base(dbContext)

{

}

public override string CreateQuery => "CREATE OR REPLACE FUNCTION get\_firms\_by\_own\_type(own\_type TEXT) RETURNS TABLE(\"Название фирмы\" VARCHAR(40)) AS $$ BEGIN RETURN QUERY SELECT f.name FROM firms f INNER JOIN own\_types ot ON ot.id = f.own\_type\_id WHERE ot.name = own\_type ORDER BY f.name;END;$$ LANGUAGE plpgsql;";

protected override string ExecuteQuery => "SELECT \* FROM get\_firms\_by\_own\_type('{0}')";

protected override async Task<IEnumerable<string>> ConvertData(DbDataReader reader)

{

var list = new List<string>();

while(await reader.ReadAsync())

{

list.Add(reader.GetString(0));

}

return list;

}

}

}

using ProviderSystemManager.DAL.Database;

using ProviderSystemManager.DAL.Models.Queries;

using ProviderSystemManager.DAL.Queries.Interfaces;

using System.Data.Common;

namespace ProviderSystemManager.DAL.Queries

{

public class FirmsByServiceRecievingDate : AbstractQuery<IEnumerable<FirmService>>, IFirmsByServiceRecievingDate

{

public FirmsByServiceRecievingDate(ProviderDbContext dbContext) : base(dbContext)

{

}

public override string CreateQuery => "CREATE OR REPLACE FUNCTION get\_firms\_by\_service\_recieving\_date(recieve\_date date) RETURNS TABLE(\"Название фирмы\" VARCHAR(40), \"Объем сообщения\" DOUBLE PRECISION) AS $$ BEGIN RETURN QUERY SELECT f.name, s.size FROM firms f INNER JOIN services s ON s.firm\_id = f.id WHERE s.recieving\_date = recieve\_date ORDER BY f.name; END; $$ LANGUAGE plpgsql;";

protected override string ExecuteQuery => "SELECT \* FROM get\_firms\_by\_service\_recieving\_date('{0}');";

protected override async Task<IEnumerable<FirmService>> ConvertData(DbDataReader reader)

{

var list = new List<FirmService>();

while(await reader.ReadAsync())

{

list.Add(new()

{

FirmName = reader.GetString(0),

Size = reader.GetDouble(1)

});

}

return list;

}

}

}

using ProviderSystemManager.DAL.Database;

using ProviderSystemManager.DAL.Models.Queries;

using ProviderSystemManager.DAL.Queries.Interfaces;

using System.Data.Common;

namespace ProviderSystemManager.DAL.Queries

{

public class FirmsByStartDateWithServicesQuery : AbstractQuery<IEnumerable<FirmService>>, IFirmsByStartDateWithServicesQuery

{

public FirmsByStartDateWithServicesQuery(ProviderDbContext dbContext) : base(dbContext)

{

}

public override string CreateQuery => "CREATE OR REPLACE FUNCTION get\_firms\_by\_start\_date\_with\_services(start\_date INTEGER, end\_date INTEGER) RETURNS TABLE(\"Название фирмы\" VARCHAR(40), \"Дата предоставления услуги\" DATE, \"Год открытия\" SMALLINT) AS $$ BEGIN RETURN QUERY SELECT f.name, s.recieving\_date, f.start\_working\_year FROM firms f LEFT JOIN services s ON s.firm\_id = f.id WHERE f.start\_working\_year BETWEEN start\_date AND end\_date AND s.id IS NOT NULL; END; $$ LANGUAGE plpgsql;";

protected override string ExecuteQuery => "SELECT \* FROM get\_firms\_by\_start\_date\_with\_services('{0}', '{1}')";

protected override async Task<IEnumerable<FirmService>> ConvertData(DbDataReader reader)

{

var list = new List<FirmService>();

while (await reader.ReadAsync())

{

list.Add(new()

{

FirmName = reader.GetString(0),

RecievingDate = reader.GetDateTime(1),

StartWorkingDate = reader.GetInt32(2)

});

}

return list;

}

}

}

using ProviderSystemManager.DAL.Database;

using ProviderSystemManager.DAL.Queries.Interfaces;

using System.Data.Common;

namespace ProviderSystemManager.DAL.Queries

{

public class FirmsCountByOwnTypeQuery : AbstractQuery<int>, IFirmsCountByOwnTypeQuery

{

public FirmsCountByOwnTypeQuery(ProviderDbContext dbContext) : base(dbContext)

{

}

public override string CreateQuery => "create or replace function get\_firms\_count\_by\_own\_type(own\_type\_name text) returns table(\"количество\" bigint) as $$ begin return query select count(f.name) from firms f join own\_types ot on f.own\_type\_id = ot.id where ot.name = own\_type\_name group by ot.name; end; $$ language plpgsql;";

protected override string ExecuteQuery => "SELECT \* FROM get\_firms\_count\_by\_own\_type('{0}')";

protected override async Task<int> ConvertData(DbDataReader reader)

{

while (await reader.ReadAsync())

{

return reader.GetInt32(0);

}

return 0;

}

}

}

using ProviderSystemManager.DAL.Database;

using ProviderSystemManager.DAL.Queries.Interfaces;

using System.Data.Common;

namespace ProviderSystemManager.DAL.Queries

{

public class FirmsSumConnectionCostInflationQuery : AbstractQuery<IEnumerable<FirmsSumConnectionCostInflationModel>>, IFirmsSumConnectionCostInflationQuery

{

public FirmsSumConnectionCostInflationQuery(ProviderDbContext dbContext) : base(dbContext)

{

}

public override string CreateQuery => "create or replace function get\_firms\_sum\_connection\_cost\_inflation(first\_date date, second\_date date) returns table(\"название\" varchar(40), \"до инфляции\" numeric, \"после инфляции\" numeric) as $$ begin return query select f.name, sum(c.connection\_cost), 0.7 \* sum(c.connection\_cost) :: numeric from (select c.connection\_cost, c.firm\_id from contracts c where c.connection\_date between first\_date and second\_date) c left join firms f on f.id = c.firm\_id group by f.name; end; $$ language plpgsql;";

protected override string ExecuteQuery => "SELECT \* FROM get\_firms\_sum\_connection\_cost\_inflation('{0}', '{1}')";

protected override async Task<IEnumerable<FirmsSumConnectionCostInflationModel>> ConvertData(DbDataReader reader)

{

var list = new List<FirmsSumConnectionCostInflationModel>();

while (await reader.ReadAsync())

{

list.Add(new(reader.GetString(0), reader.GetDouble(1), reader.GetDouble(2)));

}

return list;

}

}

}

using ProviderSystemManager.DAL.Database;

using ProviderSystemManager.DAL.Queries.Interfaces;

using System.Data.Common;

namespace ProviderSystemManager.DAL.Queries

{

public class FirmsSumConnectionCostMoreAvgQuery : AbstractQuery<IEnumerable<FirmsSumConnectionCostMoreAvgModel>>, IFirmsSumConnectionCostMoreAvgQuery

{

public FirmsSumConnectionCostMoreAvgQuery(ProviderDbContext dbContext) : base(dbContext)

{

}

public override string CreateQuery => "create or replace view get\_firms\_sum\_connection\_cost\_more\_avg as select f.name, sum(c.connection\_cost) from firms f join contracts c on c.firm\_id = f.id where c.connection\_cost > (select avg(contracts.connection\_cost) from contracts) group by f.name;";

protected override string ExecuteQuery => "SELECT \* FROM get\_firms\_sum\_connection\_cost\_more\_avg";

protected override async Task<IEnumerable<FirmsSumConnectionCostMoreAvgModel>> ConvertData(DbDataReader reader)

{

var list = new List<FirmsSumConnectionCostMoreAvgModel>();

while (await reader.ReadAsync())

{

list.Add(new(reader.GetString(0), reader.GetDouble(1)));

}

return list;

}

}

}

using ProviderSystemManager.DAL.Database;

using ProviderSystemManager.DAL.Models.Queries;

using ProviderSystemManager.DAL.Queries.Interfaces;

using System.Data.Common;

namespace ProviderSystemManager.DAL.Queries

{

public class ServiceInfoQuery : AbstractQuery<IEnumerable<ServiceInfo>>, IServiceInfoQuery

{

public ServiceInfoQuery(ProviderDbContext dbContext) : base(dbContext)

{

}

public override string CreateQuery => "CREATE OR REPLACE VIEW get\_service\_info AS SELECT s.size, a.name FROM services s INNER JOIN abonents a ON a.id = s.abonent\_id;";

protected override string ExecuteQuery => "SELECT \* FROM get\_service\_info;";

protected override async Task<IEnumerable<ServiceInfo>> ConvertData(DbDataReader reader)

{

var list = new List<ServiceInfo>();

while(await reader.ReadAsync())

{

list.Add(new()

{

Size = reader.GetDouble(0),

AbonentName = reader.GetString(1)

});

}

return list;

}

}

}

using ProviderSystemManager.DAL.Database;

using ProviderSystemManager.DAL.Queries.Interfaces;

using System.Data.Common;

namespace ProviderSystemManager.DAL.Queries

{

public class SumSizeFirmsQuery : AbstractQuery<IEnumerable<SumSizeFirmsModel>>, ISumSizeFirmsQuery

{

public SumSizeFirmsQuery(ProviderDbContext dbContext) : base(dbContext)

{

}

public override string CreateQuery => "create or replace view get\_sum\_size\_firms as select f.name, sum(s.size) from firms f join services s on s.firm\_id = f.id group by f.name;";

protected override string ExecuteQuery => "SELECT \* FROM get\_sum\_size\_firms";

protected override async Task<IEnumerable<SumSizeFirmsModel>> ConvertData(DbDataReader reader)

{

var list = new List<SumSizeFirmsModel>();

while (await reader.ReadAsync())

{

list.Add(new(reader.GetString(0), reader.GetDouble(1)));

}

return list;

}

}

}

using Microsoft.EntityFrameworkCore;

using ProviderSystemManager.DAL.Database;

using ProviderSystemManager.DAL.Queries;

namespace ProviderSystemManager.DAL.QueryCreators

{

internal class QueryCreator

{

public static void Init(ProviderDbContext dbContext)

{

var query = new AbonentsByAbonentTypeQuery(dbContext);

var query2 = new FirmsByOwnType(dbContext);

var query3 = new FirmsByServiceRecievingDate(dbContext);

var query4 = new ContractsInfoQuery(dbContext);

var query5 = new ServiceInfoQuery(dbContext);

var query6 = new ContractAbonentsEmailNotNullQuery(dbContext);

var query7 = new FirmHaveServicesQuery(dbContext);

var query8 = new FirmsByStartDateWithServicesQuery(dbContext);

var query9 = new AbonentsByServiceRecievingDateQuery(dbContext);

var query10 = new AbonentInfoQuery(dbContext);

var query11 = new SumSizeFirmsQuery(dbContext);

var query12 = new FirmsCountByOwnTypeQuery(dbContext);

var query13 = new AbonentsByContractsSumQuery(dbContext);

var query14 = new AbonentsByContractsSumAndDateQuery(dbContext);

var query15 = new FirmsSumConnectionCostInflationQuery(dbContext);

var query16 = new FirmsSumConnectionCostMoreAvgQuery(dbContext);

dbContext.Database.ExecuteSqlRaw(query.CreateQuery);

dbContext.Database.ExecuteSqlRaw(query2.CreateQuery);

dbContext.Database.ExecuteSqlRaw(query3.CreateQuery);

dbContext.Database.ExecuteSqlRaw(query4.CreateQuery);

dbContext.Database.ExecuteSqlRaw(query5.CreateQuery);

dbContext.Database.ExecuteSqlRaw(query6.CreateQuery);

dbContext.Database.ExecuteSqlRaw(query7.CreateQuery);

dbContext.Database.ExecuteSqlRaw(query8.CreateQuery);

dbContext.Database.ExecuteSqlRaw(query9.CreateQuery);

dbContext.Database.ExecuteSqlRaw(query10.CreateQuery);

dbContext.Database.ExecuteSqlRaw(query11.CreateQuery);

dbContext.Database.ExecuteSqlRaw(query12.CreateQuery);

dbContext.Database.ExecuteSqlRaw(query13.CreateQuery);

dbContext.Database.ExecuteSqlRaw(query14.CreateQuery);

dbContext.Database.ExecuteSqlRaw(query15.CreateQuery);

dbContext.Database.ExecuteSqlRaw(query16.CreateQuery);

#region SEQUENCES

dbContext.Database.ExecuteSqlRaw("CREATE SEQUENCE firms\_seq");

dbContext.Database.ExecuteSqlRaw("CREATE SEQUENCE abonents\_seq");

dbContext.Database.ExecuteSqlRaw("CREATE SEQUENCE abonent\_types\_seq");

dbContext.Database.ExecuteSqlRaw("CREATE SEQUENCE contracts\_seq");

dbContext.Database.ExecuteSqlRaw("CREATE SEQUENCE own\_types\_seq");

dbContext.Database.ExecuteSqlRaw("CREATE SEQUENCE services\_seq");

#endregion

#region TRIGGERS

dbContext.Database.ExecuteSqlRaw("CREATE OR REPLACE FUNCTION users\_after\_update() RETURNS trigger AS $$ BEGIN IF OLD.role <> NEW.role THEN IF OLD.role = 1 THEN EXECUTE 'ALTER GROUP operator DROP USER ' || NEW.login; EXECUTE 'GRANT abonent TO ' || NEW.login; END IF; IF OLD.role = 2 THEN EXECUTE 'ALTER GROUP abonent DROP USER ' || NEW.login; EXECUTE 'GRANT operator TO ' || NEW.login; END IF; IF OLD.role = 0 THEN RAISE EXCEPTION 'Администаторов нельзя понижать в должности'; END IF; END IF; RETURN NEW; END; $$ LANGUAGE plpgsql SECURITY DEFINER; CREATE TRIGGER users\_after\_update\_trigger AFTER UPDATE ON users FOR EACH ROW EXECUTE PROCEDURE users\_after\_update();");

dbContext.Database.ExecuteSqlRaw("CREATE OR REPLACE FUNCTION users\_after\_insert() RETURNS trigger AS $$ BEGIN IF NEW.role = 0 THEN EXECUTE 'GRANT admin TO ' || NEW.login; END IF; IF NEW.role = 1 THEN EXECUTE 'GRANT operator TO ' || NEW.login; END IF; IF NEW.role = 2 THEN EXECUTE 'GRANT abonent TO ' || NEW.login; END IF; RETURN NULL; END; $$ LANGUAGE plpgsql SECURITY DEFINER; CREATE TRIGGER users\_after\_insert\_trigger AFTER INSERT ON users FOR EACH ROW EXECUTE PROCEDURE users\_after\_insert();");

dbContext.Database.ExecuteSqlRaw("CREATE OR REPLACE FUNCTION users\_after\_delete() RETURNS trigger AS $$ BEGIN EXECUTE 'DROP USER ' || OLD.login; RETURN NULL; END; $$ LANGUAGE plpgsql SECURITY DEFINER; CREATE TRIGGER users\_after\_delete\_trigger AFTER DELETE ON users FOR EACH ROW EXECUTE PROCEDURE users\_after\_delete();");

dbContext.Database.ExecuteSqlRaw("CREATE OR REPLACE FUNCTION firms\_before\_delete() RETURNS trigger AS $$ BEGIN IF (SELECT COUNT(\*) FROM (SELECT FROM contracts WHERE firm\_id=OLD.id) p) <> 0 THEN RAISE EXCEPTION 'У данной фирмы еще есть контракты с пользователями'; END IF; RETURN NULL; END; $$ LANGUAGE plpgsql SECURITY DEFINER; CREATE TRIGGER firms\_before\_delete\_trigger BEFORE DELETE ON firms FOR EACH ROW EXECUTE PROCEDURE firms\_before\_delete();");

dbContext.Database.ExecuteSqlRaw("CREATE OR REPLACE FUNCTION firms\_before\_insert() RETURNS trigger AS $$ BEGIN IF (SELECT COUNT(\*) FROM (SELECT FROM firms WHERE name=NEW.name AND telephone=NEW.telephone) p) <> 0 THEN RAISE EXCEPTION 'Фирма с таким названием и номером уже существует'; END IF; RETURN NEW; END; $$ LANGUAGE plpgsql SECURITY DEFINER; CREATE TRIGGER firms\_before\_insert\_trigger BEFORE INSERT ON firms FOR EACH ROW EXECUTE PROCEDURE firms\_before\_insert();");

dbContext.Database.ExecuteSqlRaw("CREATE OR REPLACE FUNCTION firms\_before\_update() RETURNS trigger AS $$ BEGIN IF (SELECT COUNT(\*) FROM (SELECT FROM firms WHERE name=NEW.name AND telephone=NEW.telephone) p) <> 0 THEN RAISE EXCEPTION 'Фирма с таким названием и номером уже существует'; END IF; RETURN NEW; END; $$ LANGUAGE plpgsql SECURITY DEFINER; CREATE TRIGGER firms\_before\_update\_trigger BEFORE UPDATE ON firms FOR EACH ROW EXECUTE PROCEDURE firms\_before\_update();");

dbContext.Database.ExecuteSqlRaw("CREATE FUNCTION firms\_before\_insert\_increment() RETURNS trigger AS $$ BEGIN NEW.id := nextval('firms\_seq'); RETURN NEW; END; $$ LANGUAGE plpgsql SECURITY DEFINER; CREATE TRIGGER firms\_before\_insert\_increment\_trigger BEFORE INSERT ON firms FOR EACH ROW EXECUTE PROCEDURE firms\_before\_insert\_increment();");

#endregion

#region SECURITY

dbContext.Database.ExecuteSqlRaw("ALTER TABLE firms ENABLE ROW LEVEL SECURITY;");

dbContext.Database.ExecuteSqlRaw("ALTER TABLE abonents ENABLE ROW LEVEL SECURITY;");

dbContext.Database.ExecuteSqlRaw("ALTER TABLE contracts ENABLE ROW LEVEL SECURITY;");

dbContext.Database.ExecuteSqlRaw("ALTER TABLE services ENABLE ROW LEVEL SECURITY;");

#endregion

#region POLICY

dbContext.Database.ExecuteSqlRaw("CREATE POLICY firms\_operator\_staff ON firms TO operator USING (TRUE);");

dbContext.Database.ExecuteSqlRaw("CREATE POLICY contracts\_staff ON contracts TO operator USING (TRUE);");

dbContext.Database.ExecuteSqlRaw("CREATE POLICY services\_staff ON services TO operator USING (TRUE);");

dbContext.Database.ExecuteSqlRaw("CREATE POLICY abonents\_staff ON abonents TO operator USING (TRUE);");

dbContext.Database.ExecuteSqlRaw("CREATE POLICY abonent\_types\_staff ON abonent\_types TO operator USING (TRUE);");

dbContext.Database.ExecuteSqlRaw("CREATE POLICY own\_types\_staff ON own\_types TO operator USING (TRUE);");

#endregion

}

}

}

using Microsoft.EntityFrameworkCore;

using ProviderSystemManager.DAL.Database;

using ProviderSystemManager.DAL.Models;

using ProviderSystemManager.DAL.Repositories.Interfaces;

namespace ProviderSystemManager.DAL.Repositories;

public class AbonentRepository : AbstractRepository<Abonent>, IAbonentRepository

{

public AbonentRepository(ProviderDbContext dbContext) : base(dbContext) { }

public override async Task<IEnumerable<Abonent>> GetAsync() => await DataSet.Include(x => x.AbonentType).AsNoTracking().ToListAsync();

public override IEnumerable<Abonent> Get() => DataSet.Include(x => x.AbonentType).AsNoTracking();

public override async Task<Abonent> GetByIdAsync(int id) =>

await DataSet.Include(x => x.AbonentType).FirstOrDefaultAsync(x => x.Id == id);

} using ProviderSystemManager.DAL.Database;

using ProviderSystemManager.DAL.Models;

using ProviderSystemManager.DAL.Repositories.Interfaces;

namespace ProviderSystemManager.DAL.Repositories;

public class AbonentTypeRepository : AbstractRepository<AbonentType>, IAbonentTypeRepository

{

public AbonentTypeRepository(ProviderDbContext dbContext) : base(dbContext) { }

} using Microsoft.EntityFrameworkCore;

using ProviderSystemManager.DAL.Database;

using ProviderSystemManager.DAL.Models;

using ProviderSystemManager.DAL.Repositories.Interfaces;

namespace ProviderSystemManager.DAL.Repositories;

public abstract class AbstractRepository<TModel> : IRepository<TModel> where TModel : BaseModel

{

protected DbSet<TModel> DataSet { get; }

protected ProviderDbContext DbContext { get; }

public AbstractRepository(ProviderDbContext dbContext)

{

DbContext = dbContext;

DataSet = dbContext.Set<TModel>();

}

public virtual async Task<IEnumerable<TModel>> GetAsync() => await DataSet.AsNoTracking().ToListAsync();

public virtual IEnumerable<TModel> Get() => DataSet.AsNoTracking();

public virtual async Task<TModel> GetByIdAsync(int id) => await DataSet.AsNoTracking().FirstOrDefaultAsync(x => x.Id == id);

public virtual TModel GetById(int id) => DataSet.AsNoTracking().FirstOrDefault(x => x.Id == id);

public virtual async Task Create(params TModel[] models)

{

await DataSet.AddRangeAsync(models);

await DbContext.SaveChangesAsync();

}

public virtual async Task Update(params TModel[] models)

{

DataSet.UpdateRange(models);

await DbContext.SaveChangesAsync();

}

public virtual async Task Remove(params TModel[] models)

{

var objects = DataSet.Where(x => models.Select(m => m.Id).Contains(x.Id));

foreach (var model in objects)

{

DbContext.Entry(model).State = EntityState.Detached;

}

DataSet.RemoveRange(models);

await DbContext.SaveChangesAsync();

}

public virtual IQueryable<TModel> GetQuery() => DataSet.AsQueryable();

} using Microsoft.EntityFrameworkCore;

using ProviderSystemManager.DAL.Models;

using ProviderSystemManager.DAL.Database;

using ProviderSystemManager.DAL.Repositories.Interfaces;

namespace ProviderSystemManager.DAL.Repositories;

public class ContractRepository : AbstractRepository<Contract>, IContractRepository

{

public ContractRepository(ProviderDbContext dbContext) : base(dbContext)

{

}

public override async Task<IEnumerable<Contract>> GetAsync() =>

await DataSet.Include(x => x.Firm)

.ThenInclude(f => f.OwnType)

.Include(x => x.Abonent)

.ThenInclude(a => a.AbonentType).AsNoTracking().ToListAsync();

public override IEnumerable<Contract> Get() =>

DataSet.Include(x => x.Firm)

.ThenInclude(f => f.OwnType)

.Include(x => x.Abonent)

.ThenInclude(a => a.AbonentType).AsNoTracking();

public override async Task<Contract> GetByIdAsync(int id) => await DataSet.Include(x => x.Firm)

.ThenInclude(f => f.OwnType)

.Include(x => x.Abonent)

.ThenInclude(a => a.AbonentType)

.FirstOrDefaultAsync(x => x.Id == id);

} using Microsoft.EntityFrameworkCore;

using ProviderSystemManager.DAL.Models;

using ProviderSystemManager.DAL.Database;

using ProviderSystemManager.DAL.Repositories.Interfaces;

namespace ProviderSystemManager.DAL.Repositories;

public class FirmRepository : AbstractRepository<Firm>, IFirmRepository

{

public FirmRepository(ProviderDbContext dbContext) : base(dbContext) { }

public override async Task<IEnumerable<Firm>> GetAsync() => await DataSet.Include(x => x.OwnType).AsNoTracking().ToListAsync();

public override IEnumerable<Firm> Get() => DataSet.Include(x => x.OwnType).AsNoTracking();

public override async Task<Firm> GetByIdAsync(int id) => await DataSet.Include(x => x.OwnType).FirstOrDefaultAsync(x => x.Id == id);

} using ProviderSystemManager.DAL.Models;

using ProviderSystemManager.DAL.Database;

using ProviderSystemManager.DAL.Repositories.Interfaces;

namespace ProviderSystemManager.DAL.Repositories;

public class OwnTypeRepository : AbstractRepository<OwnType>, IOwnTypeRepository

{

public OwnTypeRepository(ProviderDbContext dbContext) : base(dbContext) { }

} using Microsoft.EntityFrameworkCore;

using ProviderSystemManager.DAL.Repositories.Interfaces;

using ProviderSystemManager.DAL.Models;

using ProviderSystemManager.DAL.Database;

namespace ProviderSystemManager.DAL.Repositories;

public class ServiceRepository : AbstractRepository<Service>, IServiceRepository

{

public ServiceRepository(ProviderDbContext dbContext) : base(dbContext)

{

}

public override async Task<IEnumerable<Service>> GetAsync() =>

await DataSet

.Include(x => x.Firm)

.Include(x => x.Abonent)

.ThenInclude(x => x.AbonentType)

.AsNoTracking()

.ToListAsync();

public override IEnumerable<Service> Get() =>

DataSet

.Include(x => x.Firm)

.Include(x => x.Abonent)

.ThenInclude(x => x.AbonentType)

.AsNoTracking();

} using Microsoft.EntityFrameworkCore;

using ProviderSystemManager.DAL.Database;

using ProviderSystemManager.DAL.Models;

using ProviderSystemManager.DAL.Repositories.Interfaces;

namespace ProviderSystemManager.DAL.Repositories;

public class UserRepository : AbstractRepository<User>, IUserRepository

{

public UserRepository(ProviderDbContext dbContext) : base(dbContext) { }

public override async Task Create(params User[] models)

{

var createUsersSqlQuery = models.Select(x => $"CREATE USER {x.Login} WITH SUPERUSER PASSWORD '{x.Password}';");

var result = await DbContext.Database.ExecuteSqlRawAsync(string.Join(";", createUsersSqlQuery));

await base.Create(models);

}

public override async Task Update(params User[] models)

{

var updateUsersSqlQuery = models.Select(x => $"UPDATE users SET role = {(int)x.Role}, password = '{x.Password}' where users.id = {x.Id}");

var result = await DbContext.Database.ExecuteSqlRawAsync(string.Join(";", updateUsersSqlQuery));

}

public override async Task Remove(params User[] models)

{

await base.Remove(models);

}

} using ProviderSystemManager.DAL.Database;

using ProviderSystemManager.DAL.Models;

namespace ProviderSystemManager.DAL.TableCreators;

public class AbonentCreator

{

public static int Count => 100;

public static string[] Initials => new string[]

{

"А.",

"Б.",

"С.",

"Д.",

"Е.",

"П.",

"К.",

"Р.",

"В.",

"З.",

"Ф.",

"Л.",

"Н.",

"О.",

"И.",

"Я."

};

public static string[] SecondNames => new string[]

{

"Сергева",

"Сергеев",

"Иванова",

"Иванов",

"Мирова",

"Миров",

"Добролюбов",

"Добролюбова",

"Синеев",

"Синеева",

"Хим",

"Орладно",

"Дорин",

"Дорина",

"Колесников",

"Колестикова",

"Павлолюбов",

"Павлолюбова",

"Коновалов",

"Коновалова"

};

public static void Init(ProviderDbContext dbContext)

{

var random = new Random();

for(int i = 0; i < Count; i++)

{

var firstInitial = Initials[random.Next(Initials.Length - 1)];

var secondInitial = Initials[random.Next(Initials.Length - 1)];

var secondName = SecondNames[random.Next(SecondNames.Length - 1)];

var street = FirmCreator.StreetNames[random.Next(FirmCreator.StreetNames.Length - 1)];

var abonent = new Abonent { Name = $"{firstInitial} {secondInitial} {secondName}", Address = $"ул. {street} д.{i \* 10} кв.{i % 7}", Email = $"test{i}@pic.com", AbonentTypeId = random.Next(1, AbonentTypeCreator.Count) };

dbContext.Abonents?.Add(abonent);

}

dbContext.SaveChanges();

}

} using ProviderSystemManager.DAL.Database;

using ProviderSystemManager.DAL.Models;

namespace ProviderSystemManager.DAL.TableCreators;

public class AbonentTypeCreator

{

public static int Count => 7;

public static void Init(ProviderDbContext context)

{

var abonentType1 = new AbonentType() {Name = "Частное лицо"};

var abonentType2 = new AbonentType() { Name = "ВУЗ" };

var abonentType3 = new AbonentType() { Name = "Школа" };

var abonentType4 = new AbonentType() { Name = "Агентство" };

var abonentType5 = new AbonentType() { Name = "Магазин" };

var abonentType6 = new AbonentType() { Name = "Супермаркет" };

var abonentType7 = new AbonentType() { Name = "Мастерская" };

context.AbonentTypes?.Add(abonentType1);

context.AbonentTypes?.Add(abonentType2);

context.AbonentTypes?.Add(abonentType3);

context.AbonentTypes?.Add(abonentType4);

context.AbonentTypes?.Add(abonentType5);

context.AbonentTypes?.Add(abonentType6);

context.AbonentTypes?.Add(abonentType7);

context.SaveChanges();

}

} using ProviderSystemManager.DAL.Database;

using ProviderSystemManager.DAL.Models;

namespace ProviderSystemManager.DAL.TableCreators;

public class ContractCreator

{

public static int Count => 40000;

public static void Init(ProviderDbContext dbContext)

{

var random = new Random();

for(int i = 0; i < 40000; i++)

{

var contract = new Contract()

{

FirmId = random.Next(1, FirmCreator.Count),

AbonentId = random.Next(1, AbonentCreator.Count),

ConnectionCost = (decimal)(random.Next(1, 10000) + random.NextDouble()),

ConnectionDate = DateOnly.FromDateTime(DateTime.Now.AddDays(random.Next(-1000, 0))),

ForwardingCost = (decimal)(random.Next(1, 10000) + random.NextDouble())

};

dbContext.Contracts?.Add(contract);

}

dbContext.SaveChanges();

}

} using ProviderSystemManager.DAL.Database;

using ProviderSystemManager.DAL.Models;

namespace ProviderSystemManager.DAL.TableCreators;

public class FirmCreator

{

public static int Count => 100;

private static string[] ProviderNames = new string[]

{

"Trinity",

"Matrix",

"Билайн",

"MTC",

"A Связь",

"Пром канал",

"100с",

"Быстрый старт",

"Spike",

"Oww Data",

"Faster",

"No slow Ethernet",

"Sp. Frankov",

"YTL",

"SST",

"QWE"

};

public static string[] StreetNames = new string[]

{

"Пушкина",

"Куйбышева",

"Ватутина",

"Титова",

"Гурова",

"Илонова",

"Битова",

"Сергова",

"Синяя",

"Ленина",

"Кирова",

"Малова",

"Большевиков",

"Иванова",

"Павлова",

"Михеева",

"Карда",

"Кумова",

"Семейная",

"Дружба",

"Дружная",

"Веселова",

"Смирнова",

"Каталова"

};

public static void Init(ProviderDbContext dbContext)

{

var ownTypeId = OwnTypeCreator.Count;

var random = new Random();

for (int i = 0; i < Count; i++)

{

var street = StreetNames[random.Next(StreetNames.Length - 1)];

var name = ProviderNames[random.Next(ProviderNames.Length - 1)];

if(ownTypeId == 0)

ownTypeId = OwnTypeCreator.Count;

var firm = new Firm()

{

Name = name,

Address = $"ул. {street} д. {i \* 13}",

Telephone = $"071{random.Next(1000000, 9999999)}",

OwnTypeId = ownTypeId,

StartWorkingYear = (short)random.Next(1995, 2022)

};

ownTypeId--;

dbContext.Firms?.Add(firm);

dbContext.SaveChanges();

dbContext.Entry(firm).State = Microsoft.EntityFrameworkCore.EntityState.Detached;

}

}

} using ProviderSystemManager.DAL.Database;

using ProviderSystemManager.DAL.Models;

namespace ProviderSystemManager.DAL.TableCreators;

public class OwnTypeCreator

{

public static int Count => 3;

public static void Init(ProviderDbContext context)

{

var ownType1 = new OwnType() {Name = "Частная"};

var ownType2 = new OwnType() { Name = "Собственная" };

var ownType3 = new OwnType() { Name = "Государственная" };

context.OwnTypes?.Add(ownType1);

context.OwnTypes?.Add(ownType2);

context.OwnTypes?.Add(ownType3);

context.SaveChanges();

context.Entry(ownType1).State = Microsoft.EntityFrameworkCore.EntityState.Detached;

context.Entry(ownType2).State = Microsoft.EntityFrameworkCore.EntityState.Detached;

context.Entry(ownType3).State = Microsoft.EntityFrameworkCore.EntityState.Detached;

}

} using ProviderSystemManager.DAL.Database;

using ProviderSystemManager.DAL.Models;

namespace ProviderSystemManager.DAL.TableCreators;

public class ServiceCreator

{

public static int Count => 40000;

public static void Init(ProviderDbContext dbContext)

{

var random = new Random();

for(int i = 0; i < Count; i++)

{

var service = new Service { AbonentId = random.Next(1, AbonentCreator.Count), Size = random.Next(1, 10000) + random.NextDouble(), RecievingDate = DateOnly.FromDateTime(DateTime.Now.AddDays(random.Next(-1000, 0))), FirmId = random.Next(1, FirmCreator.Count) };

dbContext.Services?.Add(service);

}

dbContext.SaveChanges();

}

} using Microsoft.EntityFrameworkCore;

using ProviderSystemManager.DAL.Database;

using ProviderSystemManager.DAL.Enums;

using ProviderSystemManager.DAL.Models;

using System.Security.Cryptography;

namespace ProviderSystemManager.DAL.TableCreators;

public class UserCreator

{

public static void Init(ProviderDbContext context)

{

var user1 = new User() { Login = "operator1", Password = "1954", Role = UserRole.Operator };

var user2 = new User() { Login = "userr", Password = "1955", Role = UserRole.User };

var user3 = new User() { Login = "employee\_1", Password = "1956", Role = UserRole.Admin };

context.Users?.Add(user1);

context.Users?.Add(user2);

context.Users?.Add(user3);

context.SaveChanges();

context.Entry(user1).State = EntityState.Detached;

context.Entry(user2).State = EntityState.Detached;

context.Entry(user3).State = EntityState.Detached;

}

private static string HashPassword(string password)

{

byte[] salt;

new RNGCryptoServiceProvider().GetBytes(salt = new byte[16]);

var pbkdf2 = new Rfc2898DeriveBytes(password, salt, 100000);

var hash = pbkdf2.GetBytes(20);

var hashBytes = new byte[36];

Array.Copy(salt, 0, hashBytes, 0, 16);

Array.Copy(hash, 0, hashBytes, 16, 20);

return Convert.ToBase64String(hashBytes);

}

} using AutoMapper;

using ProviderSystemManager.BLL.Localization;

using ProviderSystemManager.BLL.Services.Interfaces;

using ProviderSystemManager.DAL.Models;

using ProviderSystemManager.DAL.Repositories.Interfaces;

using ProviderSystemManager.Shared;

using ProviderSystemManager.Shared.Dtos.Create;

using ProviderSystemManager.Shared.Dtos.Get;

using ProviderSystemManager.Shared.Dtos.Update;

namespace ProviderSystemManager.BLL.Services;

public class AbonentService : AbstractService<AbonentCreateDto, AbonentUpdateDto, AbonentGetDto, Abonent>, IAbonentService

{

private readonly IAbonentTypeRepository \_abonentTypeRepository;

public AbonentService(IMapper mapper, IAbonentRepository repository, IAbonentTypeRepository abonentTypeRepository) :

base(mapper, repository)

{

\_abonentTypeRepository = abonentTypeRepository;

}

public override async Task<OperationResult<AbonentGetDto>> CreateAsync(params AbonentCreateDto[] dtos)

{

var models = Mapper.Map<Abonent[]>(dtos);

var abonentTypes = \_abonentTypeRepository.GetQuery()

.Where(x => models.Select(x => x.AbonentTypeId).Contains(x.Id));

foreach (var model in models)

{

model.AbonentType = abonentTypes.FirstOrDefault(x => x.Id == model.AbonentTypeId);

if (model.AbonentType == null)

return OperationResponse.Bad<AbonentGetDto>(Errors.WrongAbonentType);

}

await Repository.Create(models);

var getDtos = Mapper.Map<AbonentGetDto[]>(models);

return OperationResponse.Ok<AbonentGetDto>(getDtos.First());

}

public override async Task<OperationResult<AbonentGetDto>> UpdateAsync(params AbonentUpdateDto[] dtos)

{

var models = Mapper.Map<Abonent[]>(dtos);

var abonentTypes = \_abonentTypeRepository.GetQuery()

.Where(x => models.Select(x => x.AbonentTypeId).Contains(x.Id));

foreach (var model in models)

{

model.AbonentType = abonentTypes.FirstOrDefault(x => x.Id == model.AbonentTypeId);

if (model.AbonentType == null)

return OperationResponse.Bad<AbonentGetDto>(Errors.WrongAbonentType);

}

await Repository.Update(models);

var getDtos = Mapper.Map<AbonentGetDto[]>(models);

return OperationResponse.Ok<AbonentGetDto>(getDtos.First());

}

} using AutoMapper;

using ProviderSystemManager.BLL.Services.Interfaces;

using ProviderSystemManager.DAL.Models;

using ProviderSystemManager.DAL.Repositories.Interfaces;

using ProviderSystemManager.Shared.Dtos.Create;

using ProviderSystemManager.Shared.Dtos.Get;

using ProviderSystemManager.Shared.Dtos.Update;

namespace ProviderSystemManager.BLL.Services;

public class AbonentTypeService : AbstractService<AbonentTypeCreateDto, AbonentTypeUpdateDto, AbonentTypeGetDto, AbonentType>, IAbonentTypeService

{

public AbonentTypeService(IMapper mapper, IAbonentTypeRepository repository) : base(mapper, repository) { }

} using AutoMapper;

using Microsoft.EntityFrameworkCore;

using ProviderSystemManager.BLL.Services.Interfaces;

using ProviderSystemManager.DAL.Models;

using ProviderSystemManager.DAL.Repositories.Interfaces;

using ProviderSystemManager.Shared;

using ProviderSystemManager.Shared.Dtos.Create;

using ProviderSystemManager.Shared.Dtos.Get;

using ProviderSystemManager.Shared.Dtos.Update;

namespace ProviderSystemManager.BLL.Services;

public abstract class AbstractService<TCreateDto, TUpdateDto, TGetDto, TModel> : IService<TCreateDto, TUpdateDto, TGetDto>

where TCreateDto : ICreateDto

where TUpdateDto : IUpdateDto

where TGetDto : IGetDto

where TModel: BaseModel

{

public static event Action<IEnumerable<TGetDto>> OnCreate;

public static event Action OnUpdate;

public AbstractService(IMapper mapper, IRepository<TModel> repository)

{

Mapper = mapper;

Repository = repository;

}

protected IRepository<TModel> Repository { get; }

protected IMapper Mapper { get; }

public virtual async Task<OperationResult<IEnumerable<TGetDto>>> GetAsync()

{

var models = await Repository.GetAsync();

var dtos = Mapper.Map<IEnumerable<TGetDto>>(models);

return OperationResponse.Ok(dtos);

}

public virtual OperationResult<IEnumerable<TGetDto>> Get()

{

var models = Repository.Get();

var dtos = Mapper.Map<IEnumerable<TGetDto>>(models);

return OperationResponse.Ok(dtos);

}

public async Task<OperationResult<TGetDto>> GetByIdAsync(int id)

{

var model = await Repository.GetByIdAsync(id);

var dto = Mapper.Map<TGetDto>(model);

return OperationResponse.Ok(dto);

}

public OperationResult<TGetDto> GetById(int id)

{

var model = Repository.GetById(id);

var dto = Mapper.Map<TGetDto>(model);

return OperationResponse.Ok(dto);

}

public virtual async Task<OperationResult<TGetDto>> CreateAsync(params TCreateDto[] dtos)

{

var models = Mapper.Map<TModel[]>(dtos);

await Repository.Create(models);

var getDtos = Mapper.Map<IEnumerable<TGetDto>>(models);

OnCreate?.Invoke(getDtos);

return OperationResponse.Ok(getDtos.First());

}

public virtual async Task<OperationResult<TGetDto>> UpdateAsync(params TUpdateDto[] dtos)

{

var models = Mapper.Map<TModel[]>(dtos);

await Repository.Update(models);

var getDtos = Mapper.Map<IEnumerable<TGetDto>>(models);

OnUpdate?.Invoke();

return OperationResponse.Ok(getDtos.First());

}

public virtual async Task<OperationResult<TGetDto>> RemoveAsync(params int[] ids)

{

var models = await Repository.GetQuery().Where(x => ids.Contains(x.Id)).ToArrayAsync();

await Repository.Remove(models);

return OperationResponse.Ok<TGetDto>();

}

} using AutoMapper;

using Microsoft.EntityFrameworkCore;

using ProviderSystemManager.BLL.Localization;

using ProviderSystemManager.BLL.Services.Interfaces;

using ProviderSystemManager.DAL.Models;

using ProviderSystemManager.DAL.Repositories.Interfaces;

using ProviderSystemManager.Shared;

using ProviderSystemManager.Shared.Dtos.Create;

using ProviderSystemManager.Shared.Dtos.Get;

using ProviderSystemManager.Shared.Dtos.Update;

namespace ProviderSystemManager.BLL.Services;

public class ContractService : AbstractService<ContractCreateDto, ContractUpdateDto, ContractGetDto, Contract>, IContractService

{

private readonly IAbonentRepository \_abonentRepository;

private readonly IFirmRepository \_firmRepository;

public ContractService(IFirmRepository firmRepository, IAbonentRepository abonentRepository, IMapper mapper, IContractRepository repository) : base(mapper, repository)

{

\_firmRepository = firmRepository;

\_abonentRepository = abonentRepository;

}

public override async Task<OperationResult<ContractGetDto>> CreateAsync(params ContractCreateDto[] dtos)

{

var models = Mapper.Map<Contract[]>(dtos);

var firms = \_firmRepository.GetQuery().Where(x => models.Select(m => m.FirmId).Contains(x.Id));

var abonents = \_abonentRepository.GetQuery().Where(x => models.Select(m => m.AbonentId).Contains(x.Id));

foreach (var model in models)

{

var firm = await firms.FirstOrDefaultAsync(x => x.Id == model.FirmId);

if (firm is null)

return OperationResponse.Bad<ContractGetDto>(Errors.WrongFirm);

var abonent = await abonents.FirstOrDefaultAsync(x => x.Id == model.AbonentId);

if (abonent is null)

return OperationResponse.Bad<ContractGetDto>(Errors.WrongAbonent);

model.Firm = firm;

model.Abonent = abonent;

}

await Repository.Create(models);

var getDtos = Mapper.Map<ContractGetDto[]>(models);

return OperationResponse.Ok(getDtos.First());

}

public OperationResult<IEnumerable<ContractGetDto>> GetByAbonentId(int abonentId)

{

var models = Repository

.GetQuery()

.Where(x => x.AbonentId == abonentId);

var getDtos = Mapper.Map<IEnumerable<ContractGetDto>>(models);

return OperationResponse.Ok(getDtos);

}

public override async Task<OperationResult<ContractGetDto>> UpdateAsync(params ContractUpdateDto[] dtos)

{

var models = Mapper.Map<Contract[]>(dtos);

var firms = \_firmRepository.GetQuery().Where(x => models.Select(m => m.FirmId).Contains(x.Id));

var abonents = \_abonentRepository.GetQuery().Where(x => models.Select(m => m.AbonentId).Contains(x.Id));

foreach (var model in models)

{

var firm = await firms.FirstOrDefaultAsync(x => x.Id == model.FirmId);

if (firm is null)

return OperationResponse.Bad<ContractGetDto>(Errors.WrongFirm);

var abonent = await abonents.FirstOrDefaultAsync(x => x.Id == model.AbonentId);

if (abonent is null)

return OperationResponse.Bad<ContractGetDto>(Errors.WrongAbonent);

model.Firm = firm;

model.Abonent = abonent;

}

await Repository.Update(models);

var getDtos = Mapper.Map<ContractGetDto[]>(models);

return OperationResponse.Ok(getDtos.First());

}

} using Microsoft.EntityFrameworkCore;

using ProviderSystemManager.BLL.Localization;

using ProviderSystemManager.DAL.Database;

using ProviderSystemManager.BLL.Services.Interfaces;

using ProviderSystemManager.Shared;

namespace ProviderSystemManager.BLL.Services

{

public class DbContextService : IDbContextService

{

private readonly ProviderDbContext \_dbContext;

public DbContextService(ProviderDbContext dbContext)

{

\_dbContext = dbContext;

}

public async Task<OperationResult<bool>> ChangeConnectionAsync(string connectionString)

{

\_dbContext.Database.SetConnectionString(connectionString);

var canConnect = await \_dbContext.Database.CanConnectAsync();

return canConnect ? OperationResponse.Ok(canConnect) : OperationResponse.Bad<bool>(Errors.ConnectionFail);

}

}

} using AutoMapper;

using Microsoft.EntityFrameworkCore;

using ProviderSystemManager.BLL.Localization;

using ProviderSystemManager.BLL.Services.Interfaces;

using ProviderSystemManager.DAL.Models;

using ProviderSystemManager.DAL.Repositories.Interfaces;

using ProviderSystemManager.Shared;

using ProviderSystemManager.Shared.Dtos.Create;

using ProviderSystemManager.Shared.Dtos.Get;

using ProviderSystemManager.Shared.Dtos.Update;

namespace ProviderSystemManager.BLL.Services;

public class FirmService : AbstractService<FirmCreateDto, FirmUpdateDto, FirmGetDto, Firm>, IFirmService

{

public static event Action<IEnumerable<FirmGetDto>> OnCreate;

public static event Action OnUpdate;

private readonly IOwnTypeRepository \_ownTypesRepository;

public FirmService(IOwnTypeRepository ownTypesRepository, IMapper mapper, IFirmRepository repository)

: base(mapper, repository)

{

\_ownTypesRepository = ownTypesRepository;

}

public override async Task<OperationResult<FirmGetDto>> CreateAsync(params FirmCreateDto[] dtos)

{

var models = Mapper.Map<Firm[]>(dtos);

var ownTypes = \_ownTypesRepository.GetQuery().Where(x => models.Select(m => m.OwnTypeId).Contains(x.Id));

foreach (var model in models)

{

var ownType = await ownTypes.FirstOrDefaultAsync(x => x.Id == model.OwnTypeId);

if (ownType is null)

return OperationResponse.Bad<FirmGetDto>(Errors.WrongOwnType);

model.OwnType = ownType;

}

await Repository.Create(models);

var getDtos = Mapper.Map<IEnumerable<FirmGetDto>>(models);

OnCreate?.Invoke(getDtos);

return OperationResponse.Ok(getDtos.First());

}

public override async Task<OperationResult<FirmGetDto>> UpdateAsync(params FirmUpdateDto[] dtos)

{

var models = Mapper.Map<Firm[]>(dtos);

var ownTypes = \_ownTypesRepository.GetQuery().Where(x => models.Select(m => m.OwnTypeId).Contains(x.Id));

foreach (var model in models)

{

var ownType = await ownTypes.FirstOrDefaultAsync(x => x.Id == model.OwnTypeId);

if (ownType is null)

return OperationResponse.Bad<FirmGetDto>(Errors.WrongOwnType);

model.OwnType = ownType;

}

await Repository.Update(models);

var getDtos = Mapper.Map<FirmGetDto[]>(models);

OnUpdate?.Invoke();

return OperationResponse.Ok(getDtos.First());

}

} using AutoMapper;

using ProviderSystemManager.BLL.Services.Interfaces;

using ProviderSystemManager.DAL.Models;

using ProviderSystemManager.DAL.Repositories.Interfaces;

using ProviderSystemManager.Shared.Dtos.Create;

using ProviderSystemManager.Shared.Dtos.Get;

using ProviderSystemManager.Shared.Dtos.Update;

namespace ProviderSystemManager.BLL.Services;

public class OwnTypeService : AbstractService<OwnTypeCreateDto, OwnTypeUpdateDto, OwnTypeGetDto, OwnType>, IOwnTypeService

{

public OwnTypeService(IMapper mapper, IOwnTypeRepository repository) : base(mapper, repository) { }

} using AutoMapper;

using ProviderSystemManager.BLL.Services.Interfaces;

using ProviderSystemManager.DAL.Models;

using ProviderSystemManager.DAL.Repositories.Interfaces;

using ProviderSystemManager.Shared.Dtos.Create;

using ProviderSystemManager.Shared.Dtos.Get;

using ProviderSystemManager.Shared.Dtos.Update;

namespace ProviderSystemManager.BLL.Services;

public class ServiceService : AbstractService<ServiceCreateDto, ServiceUpdateDto, ServiceGetDto, Service>, IServiceService

{

public ServiceService(IMapper mapper, IServiceRepository repository) : base(mapper, repository)

{

}

} using AutoMapper;

using ProviderSystemManager.BLL.Localization;

using ProviderSystemManager.BLL.Services.Interfaces;

using ProviderSystemManager.DAL.Models;

using ProviderSystemManager.DAL.Repositories.Interfaces;

using ProviderSystemManager.Shared;

using ProviderSystemManager.Shared.Dtos.Create;

using ProviderSystemManager.Shared.Dtos.Get;

using ProviderSystemManager.Shared.Dtos.Update;

using System.Security.Cryptography;

namespace ProviderSystemManager.BLL.Services;

public class UserService : AbstractService<UserCreateDto, UserUpdateDto, UserGetDto, User>, IUserService

{

public UserService(IMapper mapper, IUserRepository repository) : base(mapper, repository)

{

}

public Task<OperationResult<UserGetDto>> GetByLoginAsync(string login)

{

var model = Repository.GetQuery().FirstOrDefault(x => x.Login == login);

if (model is null)

return Task.FromResult(OperationResponse.Bad<UserGetDto>(Errors.WrongLogin));

var dto = Mapper.Map<UserGetDto>(model);

return Task.FromResult(OperationResponse.Ok(dto));

}

}

# ПРИЛОЖЕНИЕ Д

# РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

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

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

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

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

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

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

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

# ПРИЛОЖЕНИЕ Е

# РУКОВОДСТВО АДМИНИСТРАТОРА

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

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

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

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

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

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

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

# ПРИЛОЖЕНИЕ Ж

ОТЧЕТ О ПРОВЕРКЕ НА ЗАИМСТВОВАНИЯ

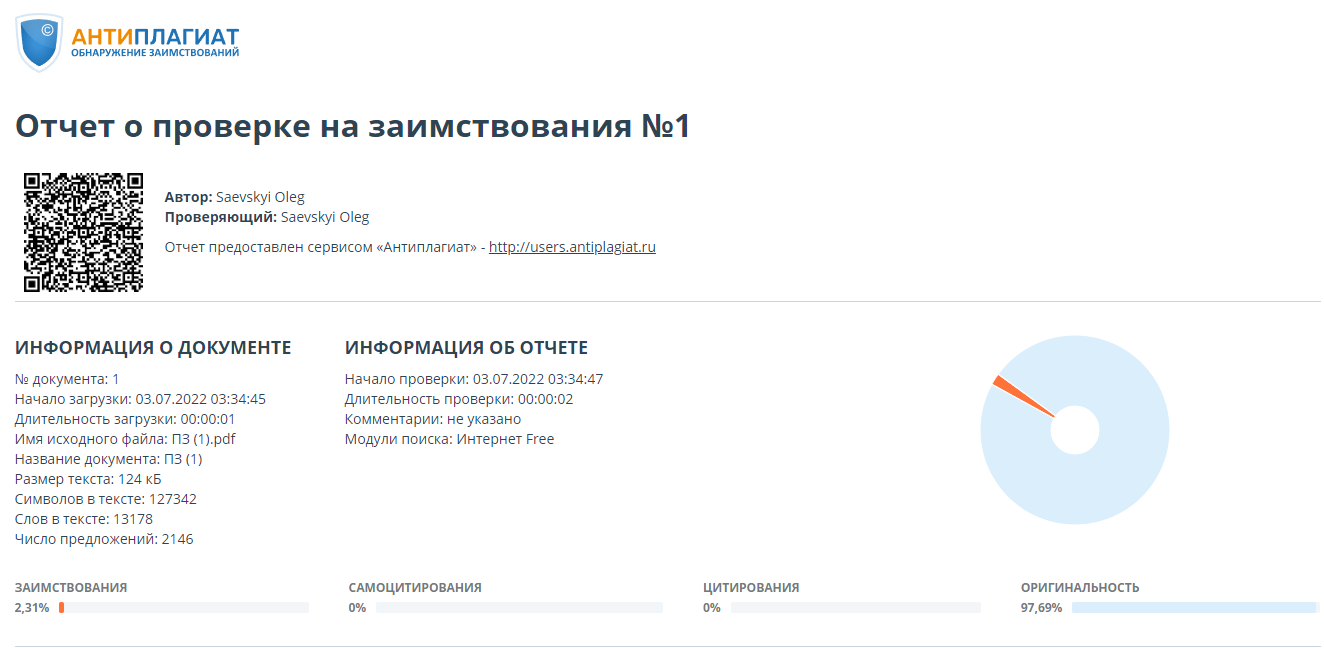


Рисунок В.1 – Заимствования (часть 1)

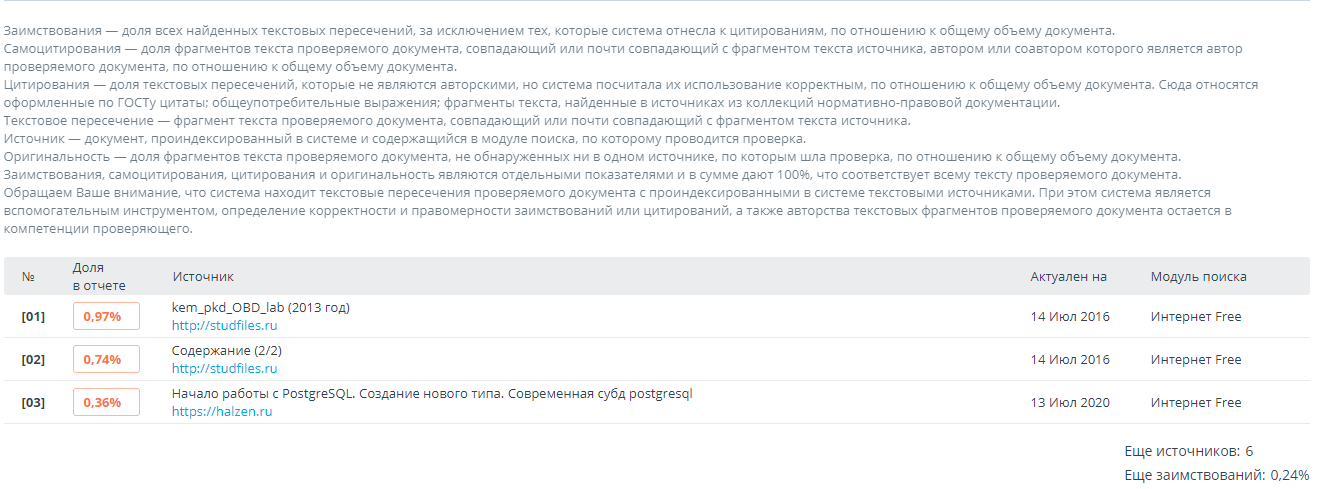


Рисунок В.2 – Заимствования (часть 2)