



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

Разработка Telegram-бота для добавления, поиска
жилья и нахождения соседей

Студент ИУ7-62Б
(Группа)

(Подпись, дата)

И.С.Климов
(И.О.Фамилия)

Руководитель курсовой работы

(Подпись, дата)

О.В.Кузнецова
(И.О.Фамилия)

2022 г.

РЕФЕРАТ

Ключевые слова: база данных, система управления базами данных, PostgreSQL, Telegram-бот, поиск соседей.

Расчетно-пояснительная записка представляет собой 52 страницы, 17 рисунков, 2 таблицы, 3 приложения.

Данная курсовая работа представляет собой реализацию базы данных для поиска и добавления объявлений по аренде квартир, поиску соседей и продаже бытовых товаров, а также Telegram-бот и панель администратора, позволяющие взаимодействовать с базой данных (просматривать, добавлять, удалять и изменять данные). В качестве СУБД используется PostgreSQL, подключаемое к приложению на языке Python через библиотеку psycopg2.

Содержание

ВВЕДЕНИЕ.....	6
1 Аналитическая часть	8
1.1 Постановка задачи	8
1.1.1 Тип сервиса.....	8
1.1.2 Ролевая модель.....	9
1.1.3 Формализация данных.....	12
1.2 Модель данных	13
1.3 Существующие аналоги.....	13
Вывод.....	14
2 Конструкторская часть	16
2.1 Проектирование базы данных	16
2.2 Схемы триггеров	20
Вывод.....	24
3 Технологическая часть	25
3.1 Выбор СУБД.....	25
3.1.1 MySQL.....	25
3.1.2 PostgreSQL	26
3.1.3 Oracle Database	26
Вывод	26
3.2 Выбор инструментов разработки	27
3.3 Детали реализации	27
3.3.1 Разбиение на компоненты.....	27
3.3.2 Создание таблиц, ограничений, ролей и триггеров	28
3.4 Интерфейс приложения	28
Вывод.....	30
4 Исследовательская часть	31
4.1 Примеры работы	31
4.2 Цель эксперимента.....	33
4.3 Технические характеристики.....	34
4.4 Проведение эксперимента.....	34
Вывод.....	34

ЗАКЛЮЧЕНИЕ	36
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	37
ПРИЛОЖЕНИЕ А.....	39
ПРИЛОЖЕНИЕ Б.....	48
ПРИЛОЖЕНИЕ В.....	50

ВВЕДЕНИЕ

Выбор места проживания является одним из самых главных вопросов в жизни человека. Рано или поздно он встанет перед каждым. Особенно это важно для студентов: зачастую ради лучшего образования приходится перебираться в другие города и искать новое жилье, что может быть довольно проблематично. Общежитие является далеко не лучшим вариантом, в своем большинстве студенческие общежития при вузах не отличаются комфортом, нередко здесь можно столкнуться с некачественным ремонтом и плохими условиями. При этом возможное наличие шумных соседей не позволит полноценно отдохнуть и готовиться к учебе. К тому же количество мест здесь ограничено, и существует высокая вероятность остаться без крыши над головой [1]. По состоянию на 2020 год около 20% нуждающихся остались без места [2]. Поэтому оптимальной альтернативой является съем квартиры.

Существует большое количество сервисов с возможностью поиска аренды квартиры. Однако цена может быть довольно высокой, студент попросту не сможет оплатить подходящую квартиру. Решением этой проблемы становится поиск соседа (сожителя). Помимо экономии денег, это отличный повод для нахождения новых людей и товарищей в незнакомом городе.

Целью данной работы является реализация базы данных, используемой в Telegram-боте, который позволит студентам находить жилье и соседей. Для достижения данной цели необходимо решить поставленные **задачи**:

- 1) определить функциональные требования к разрабатываемому программному продукту;
- 2) определить ролевую модель;
- 3) провести анализ моделей данных и выбрать наиболее подходящую;
- 4) спроектировать базу данных, описать ее сущности и связи;
- 5) реализовать спроектированную базу данных;
- 6) реализовать сервис (Telegram-бота), обеспечивающий доступ к базе данных;

- 7) реализовать панель администратора для контроля за ботом;
- 8) провести сравнительный анализ времени выполнения различных запросов к базе данных с использованием индексов и без.

1 Аналитическая часть

В данном разделе ставится задача: определяются тип сервиса, ролевая модель и формализируются данные. Представляются Use-Case диаграммы для четырех типов пользователей (гостя, арендатора, арендодателя и администратора) и ER-диаграмма в нотации Чена. Сравниваются модели данных и выбирается наиболее подходящая. Также приводится сравнение существующих аналогов и обосновывается разработка текущего приложения.

1.1 Постановка задачи

Для поиска соседа по квартире существует не так много возможностей: от оставления комментариев в различных тематических группах до опроса знакомых. Все это не является удобным способом достижения желаемого результата, объявления располагаются хаотично, сложно найти и подобрать подходящую квартиру, найти человека с такими же целями. При этом отсутствует возможность поиска бытовых товаров. Сервис, включающий в себя соответствующий функционал, существенно облегчит поиски.

1.1.1 Тип сервиса

Сервис можно представить в виде десктопного приложения или веб-приложения. В таком случае приходится тратить ресурсы на создание, например, интерфейса. В данной же работе целесообразно взять технологию, позволяющую сфокусироваться на поставленной цели. Таковой является Telegram-бот, который предоставляет удобный API для взаимодействия.

Также бот подразумевает наличие администратора, который будет следить за его состоянием. Поэтому предполагается разработка панели администратора, выполненная в виде десктопного приложения.

1.1.2 Ролевая модель

Сервис предполагает многопользовательское использование, при этом четко выделяются несколько типов пользователей и в соответствии от этого доступность определенного функционала.

1. Гость (неавторизированный пользователь) – просмотр всех типов объявлений без выставления фильтров; регистрация для взаимодействия в системе; авторизация. На рисунке 1.1 представлена Use-Case диаграмма для гостя.

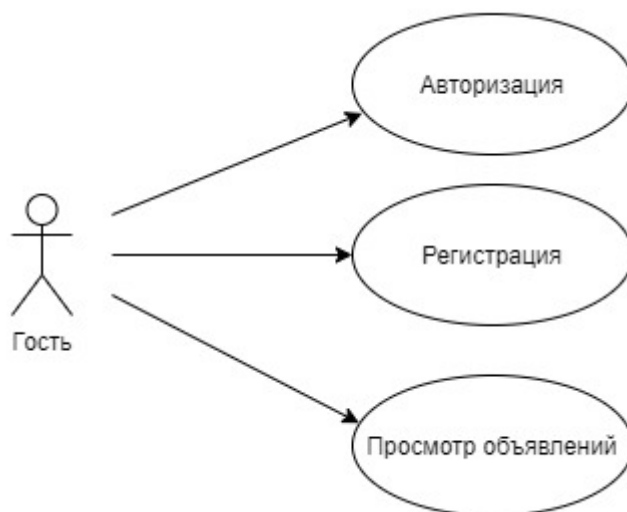


Рисунок 1.1 – Use-Case диаграмма для гостя

2. Арендатор (авторизированный пользователь) – просмотр всех типов объявлений как с фильтрами, так и без; добавление объявлений о поиске соседа и продаже бытовых товаров; подписка на арендодателей и появление квартир; оценка арендодателей; отметка понравившихся квартир; получение уведомлений по подпискам. На рисунке 1.2 представлена Use-Case диаграмма для арендатора.



Рисунок 1.2 – Use-Case диаграмма для разрабатываемой системы

3. Арендодатель (авторизированный пользователь) – просмотр всех типов объявлений как с фильтрами, так и без; добавление объявлений о сдаче квартир в аренду квартир с фотографиями. На рисунке 1.3 представлена Use-Case диаграмма для арендодателя.

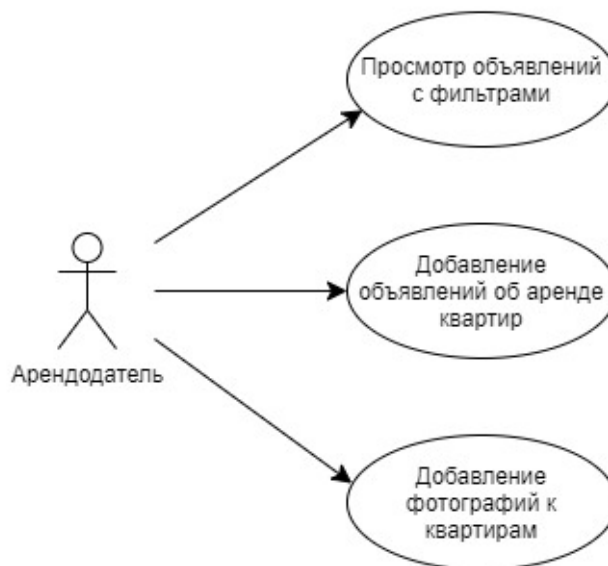


Рисунок 1.3 – Use-Case диаграмма для арендодателя

4. Администратор – добавление удалений пользователей и объявлений всех типов, получение информации о них, ее изменение. На рисунке 1.4 представлена Use-Case диаграмма для администратора.

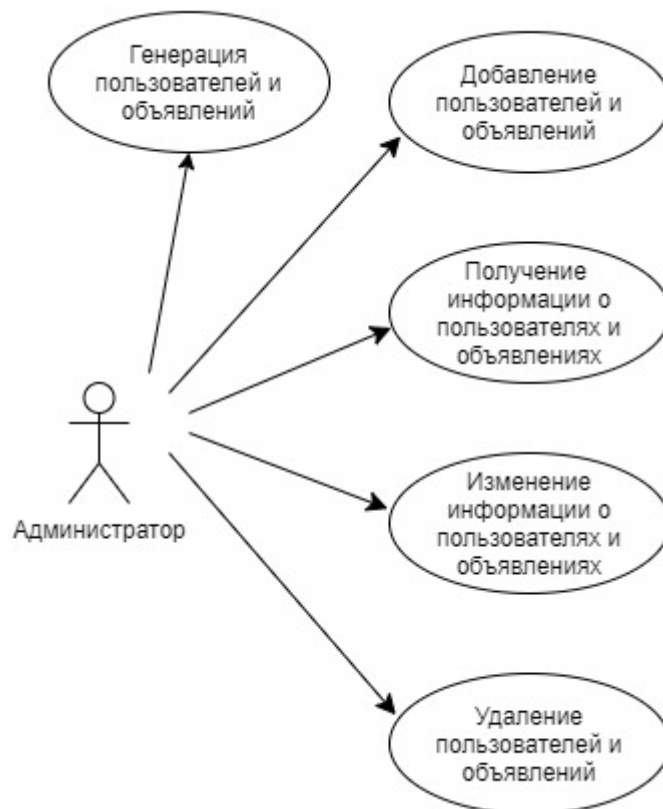


Рисунок 1.4 – Use-Case диаграмма для администратора

1.1.3 Формализация данных

Будущая база данных будет содержать информацию про пользователей (арендаторов и арендодателей), объявления (сдаче квартир в аренду, поиске соседа и продаже бытовых товаров), отношения между ними (подписка на арендодателя, подписка на квартиру по заданным параметрам, оценка квартиры). На рисунке 1.5 представлена соответствующая ER-диаграмма.

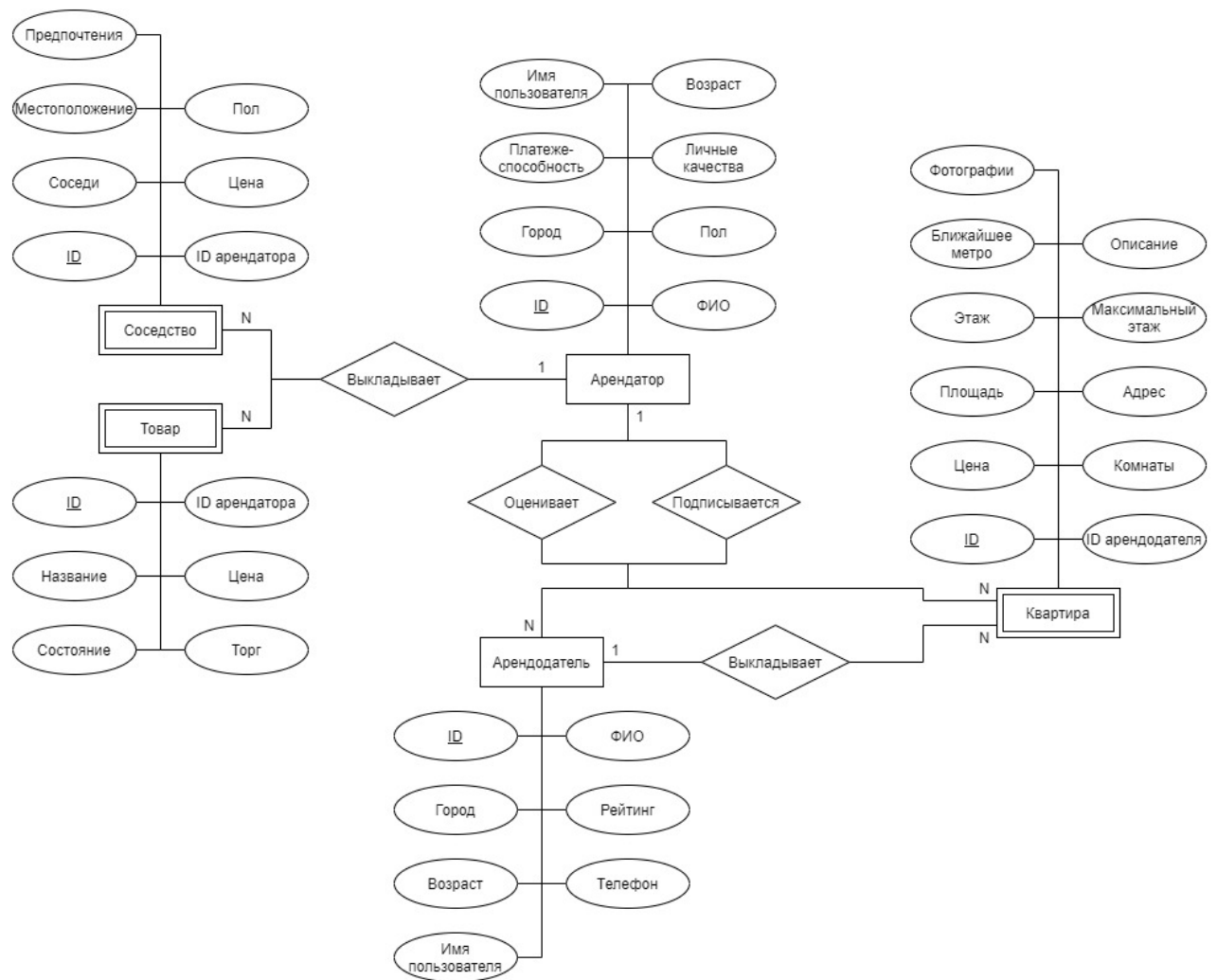


Рисунок 1.5 – ER-диаграмма в нотации Чена

1.2 Модель данных

Модель данных – это совокупность структур данных и операций их обработки [3]. Выделяют три основных типа данных моделей.

1. Дореляционная модель – появилась одной из первых. Она предшествует реляционной и в свою очередь делится на иерархическую и сетевую модели:
 - иерархическая модель – совокупность элементов, расположенных в порядке их подчинения от общего к частному и образующих по структуре дерево;
 - сетевая модель – модель, при которой каждый элемент может быть связан с любым другим элементом.
2. Реляционная модель – модель, в которой объекты и связи представляются в виде таблиц. Эта модель характеризуется простотой структуры данных и удобным табличным представлением. На данный момент является наиболее популярной моделью.
3. Постреляционная модель – представляет собой расширенную реляционную модель, снимающую ограничение неделимости данных, хранящихся в записях таблиц. Она допускает многозначные поля и поддерживает ассоциированные многозначные поля [4].

Реляционная модель является наиболее оптимальным выбором в данной работе. Исходя из составленной модели данных, очевидно, что табличное представление соответствует лучше всего, при этом видна необходимость целостности данных.

1.3 Существующие аналоги

В качестве аналогов рассмотрены два веб-сервиса, похожие по функционалу на разрабатываемый продукт: Quickl (<https://thequickl.ru/>) и

Живем! (<http://sosed.zhivem.ru/>). Был проведен сравнительный анализ данных продуктов с разрабатываемым (ОбщагиНет) по следующим критериям:

- возможность поиска и добавления объявлений о сдаче квартир в аренду;
- возможность поиска и добавления объявлений о поиске соседа;
- возможность поиска и добавления объявлений о продаже бытовых товаров;
- подписка на арендодателей и появление объявлений о сдаче квартир в аренду по выставленным фильтрам;
- оценка арендодателей и квартир;
- получения уведомлений по подпискам.

Для удобства проведем краткий сравнительный анализ при помощи таблицы (таблица 1.1).

Таблица 1.1 – Сравнительная таблица существующих аналогов с разрабатываемым сервисом

Критерий	Quickl	Живем!	ОбщагиНет
Сдача квартир в аренду	+	+	+
Поиск соседа	+	+	+
Продажа бытовых товаров	–	–	+
Подписка на арендодателей и квартиры	–	–	+
Оценка арендодателей и квартир	–	–	+
Получение уведомлений по подпискам	–	–	+

Как видно, из таблицы разрабатываемый продукт имеет ряд преимуществ, что делает обоснованной его разработку.

Вывод

В результате была поставлена задача на данную работу (определены тип сервиса – Telegram-бот; ролевая модель для гостя, арендатора, арендодателя и

администратора; формализованы данные), выбрана реляционная модель данных, представлены Use-Case диаграмма и ER-диаграмма в нотации Чена. Также были проанализированы существующие аналоги и выявлены преимущества разрабатываемого сервиса по сравнению с другими.

2 Конструкторская часть

В данном разделе спроектирована база данных (определены все необходимые таблицы – пользователи, объявления, подписки, описаны поля с указанием типов и ограничений на значения, представлена соответствующая диаграмма). Также приведены схемы работы триггеров, часть которых срабатывают до удаления (удаление данных, связанных с удаляемыми по ключу), часть – до вставки значений (удаление текущей подписки на квартиру).

2.1 Проектирование базы данных

В соответствии с ER-моделью, представленной на рисунке 1.5, база данных должна хранить следующие таблицы:

- таблица арендаторов (tenant);
- таблица арендодателей (landlord);
- таблица квартир (flat);
- таблица фотографий квартир (flat_photo);
- таблица объявлений о соседстве (neighborhood);
- таблица бытовых товаров (goods);
- таблица подписок на арендодателей (subscription_landlord);
- таблица понравившихся квартир (likes_flat);
- таблица подписок на квартиры (subscription_flat);
- таблица станций метро, указанной в подписке на квартиру (subscription_metro).

На рисунке 2.1 представлена диаграмма разрабатываемой базы данных.

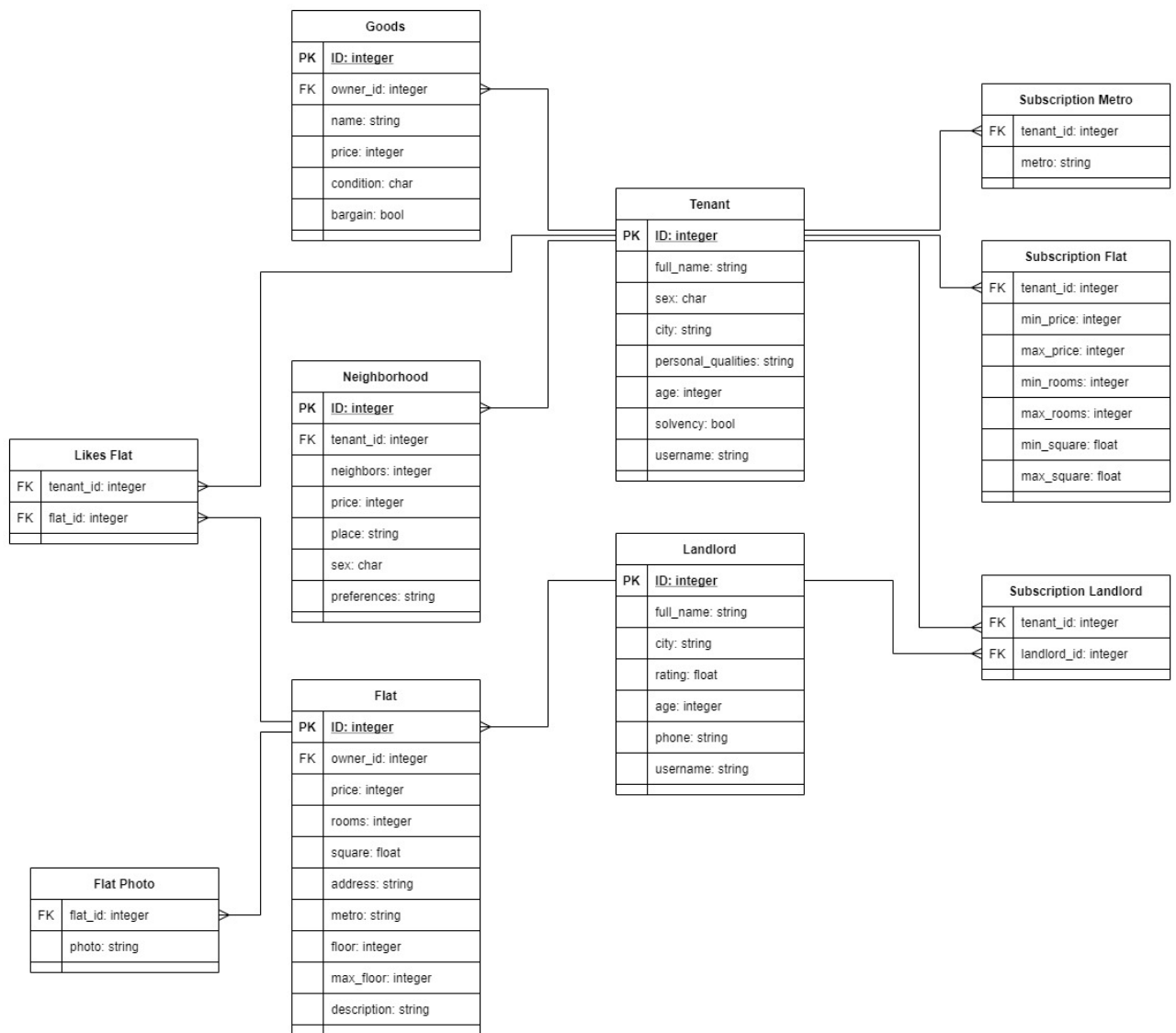


Рисунок 2.1 – Диаграмма базы данных

Таблица *tenant* хранит информацию об арендаторе и содержит следующие поля:

- ID – уникальный идентификатор арендатора, первичный ключ, integer;
- full_name – полное имя, string;
- sex – пол, принимает значения **M** или **F**, char;
- city – город, string;
- personal_qualities – персональные качества, string;
- age – возраст, integer;
- solvency – платежеспособность, bool;
- username – имя пользователя, string.

Таблица *landlord* хранит информацию об арендодателе и содержит следующие поля:

- ID – уникальный идентификатор арендодателя, первичный ключ, integer;
- full_name – полное имя, string;
- city – город, string;
- rating – рейтинг, float;
- age – возраст, integer;
- phone – номер телефона, string;
- username – имя пользователя, string.

Таблица *flat* хранит информацию о квартире и содержит следующие поля:

- ID – уникальный идентификатор квартиры, первичный ключ, integer;
- owner_id – id владельца (арендодателя), внешний ключ, integer;
- price – цена, integer;
- rooms – количество комнат, integer;
- square – площадь, float;
- address – адрес, string;
- metro – ближайшее метро, string;
- floor – этаж, integer;
- max_floor – максимальный этаж, integer;
- description – описание, string.

Таблица *flat_photo* хранит пути к фотографиям квартир и содержит следующие поля:

- flat_id – id квартиры, внешний ключ, integer;
- photo – путь к фотографии, string.

Таблица *neighborhood* хранит информацию об объявлениях о поиске соседа и содержит следующие поля:

- ID – уникальный идентификатор объявления, первичный ключ, integer;
- tenant_id – id арендатора, который ищет соседа, внешний ключ, integer;
- neighbors – количество соседей, integer;
- price – предполагаемая цена, integer;

- place – местоположение, string;
- sex – предпочитаемый пол, char;
- preferences – предпочтения, string.

Таблица *goods* хранит информацию о бытовых товарах и содержит следующие поля:

- ID – уникальный идентификатор товара, первичный ключ, integer;
- owner_id – id владельца (арендатора), внешний ключ, integer;
- name – название, string;
- price – цена, integer;
- condition – состояние, принимает значения **E** (отличное), **G** (хорошее), **S** (удовлетворительное), **U** (неудовлетворительное), **T** (ужасное), char;
- bargain – возможен ли торг, bool.

Таблица *goods* хранит информацию о бытовых товарах и содержит следующие поля:

- ID – уникальный идентификатор товара, первичный ключ, integer;
- owner_id – id владельца (арендатора), внешний ключ, integer;
- name – название, string;
- price – цена, integer;
- condition – состояние, принимает значения **E** (отличное), **G** (хорошее), **S** (удовлетворительное), **U** (неудовлетворительное), **T** (ужасное), char;
- bargain – возможен ли торг, bool.

Таблица *tenant* и *landlord* связаны отношением многие-ко-многим (подписка арендатором на арендодателей), таблица *subscription_landlord* хранит данную связь и содержит следующие поля:

- tenant_id – id арендатора, внешний ключ, integer;
- landlord_id – id арендодателя, внешний ключ, integer.

Таблица *tenant* и *flat* связаны отношением многие-ко-многим (арендаторы отмечают понравившуюся квартиру), таблица *likes_flat* хранит данную связь и содержит следующие поля:

- tenant_id – id арендатора, внешний ключ, integer;

- flat_id – id квартиры, внешний ключ, integer.

Таблица *subscription_flat* хранит информацию о параметрах квартиры, которые были указаны при подписке арендатором, и содержит следующие поля:

- tenant_id – id арендатора, внешний ключ, integer;
- min_price – минимальная цена, integer;
- max_price – максимальная цена, integer;
- min_rooms – минимальное количество комнат, integer;
- max_rooms – максимальное количество комнат, integer;
- min_square – минимальная площадь, float;
- max_square – максимальная площадь, float.

Таблица *subscription_metro* хранит информацию о станциях метро, указанных в подписке на квартиру, и содержит следующие поля:

- tenant_id – id арендатора, внешний ключ, integer;
- metro – станция метро, string.

2.2 Схемы триггеров

При удалении некоторых данных необходимо также удалять связанные с ними строки в других таблицах (внешние ключи), также выставлять соответствующее значение. Для этого необходимы триггеры.

На рисунке 2.2 представлена схема триггера, срабатывающего после удаления арендатора.



Рисунок 2.2 – Схема триггера `delete_tenant`

На рисунке 2.3 представлена схема триггера, срабатывающего после удаления арендодателя.

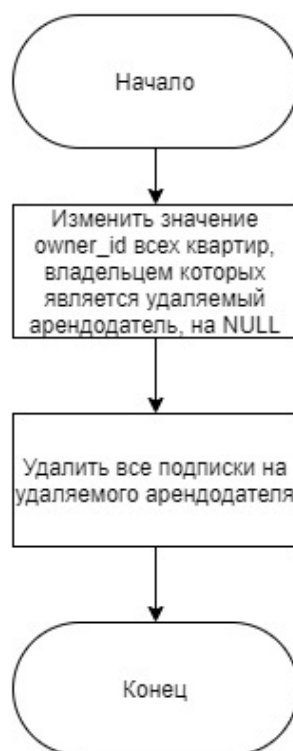


Рисунок 2.3 – Схема триггера `delete_landlord`

На рисунке 2.4 представлена схема триггера, срабатывающего после удаления квартиры.

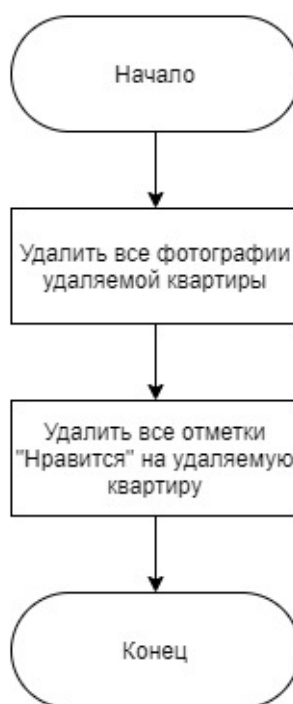


Рисунок 2.4 – Схема триггера `delete_flat`

На рисунке 2.5 представлена схема триггера, срабатывающего после удаления подписки на квартиру.



Рисунок 2.5 – Схема триггера `delete_subscription_flat`

Также при добавлении новой подписки на квартиру необходимо, что старая удалялась. На рисунке 2.6 представлена схема соответствующего триггера.



Рисунок 2.6 – Схема триггера `insert_subscription_flat`

Вывод

В данном разделе была спроектирована база данных, определены таблицы (арендаторов, арендодателей, квартир, фотографий квартир, объявлений о поиске соседа, бытовых товаров, подписок на арендодателей, понравившихся квартир, подписок на квартиры и станций метро, указанных в данной подписке), построена соответствующая диаграмма, указаны поля, их типы и ограничения к ним. Также приведены схемы работы триггеров, срабатывающих при удалении и добавлении данных.

3 Технологическая часть

В данном разделе приведен выбор инструментов разработки (языка программирования, библиотек и фреймворков). Также проводится анализ наиболее популярных реляционных СУБД и выбирается наиболее подходящая. Проведено разбиение приложения на компоненты (с выделением трех частей). Также представлены детали реализации (создание таблиц, ограничений, ролей, триггеров) и интерфейс приложения.

3.1 Выбор СУБД

Система управления базами данных (СУБД) — совокупность программных средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных [5]. С помощью нее можно проводить различные операции над базой данных: выбирать, обновлять, удалять, редактировать и т.д. При этом СУБД гарантирует сохранность, целостность и безопасность хранения данных. На основе выбранной реляционной модели рассмотрим наиболее популярные соответствующие СУБД.

3.1.1 MySQL

MySQL — это реляционная СУБД с открытым исходным кодом. В настоящее время это одна из наиболее популярных в веб-приложениях, почти все веб-фреймворки поддерживают MySQL уже на уровне базовой конфигурации (без дополнительных модулей) [6]. К преимуществам относятся простота в использовании, масштабируемость, скорость. Однако СУБД имеет определенные ограничения в функционале и недостаточную надежность [7].

3.1.2 PostgreSQL

PostgreSQL — это свободная объектно-реляционная СУБД, которая базируется на языке SQL и поддерживает многочисленные возможности. Она отличается высокой надёжностью и хорошей производительностью. PostgreSQL поддерживает транзакции, обладающие свойствами ACID, репликация реализована встроенными механизмами. Можно создавать свои типы данных и индексов, а также расширять поведение при помощи языков программирования [8].

3.1.3 Oracle Database

Oracle Database — это объектно-реляционная СУБД, созданная компанией Oracle. Является наиболее популярной в мире. Она поддерживает множество функций, является очень надёжной, может применяться практически для любых задач. Особенностью является быстрая работа с большими объемами данных. Однако стоимость пользования данной СУБД является довольно высокой, и работа с ней может требовать достаточного количества ресурсов [9].

Вывод

На основе анализа рассматриваемых СУБД был сделан выбор в пользу PostgreSQL. Благодаря большому количеству преимуществ, простоте и удобству она является наиболее оптимальным вариантом для разрабатываемого сервиса. Также имеется опыт работы с данной СУБД. При этом для PostgreSQL есть загружаемый процедурный язык PL/pgSQL, который позволяет довольно эффективно и просто писать функции и триггерные процедуры [10].

3.2 Выбор инструментов разработки

В качестве языка программирования был выбран Python [11]. Данный язык позволяет достаточно быстро разрабатывать, при этом поддерживает объектно-ориентированную парадигму программирования. Также Python имеет обширное количество библиотек, что открывает большой выбор возможностей. Идеальным решением для разработки Telegram-бота стала библиотека aiogram [12], которая постоянно обновляется и имеет ряд преимуществ по сравнению с другими аналогичными (асинхронность, наличие удобных инструментов для разработки).

Для разработки desktop-приложения для панели администратора выбран фреймворк PyQt5 [13] из-за простоты использования, высокой производительности, кроссплатформенности. При этом для создания графического интерфейса можно использовать QtDesigner [14].

3.3 Детали реализации

3.3.1 Разбиение на компоненты

Перед реализацией программного продукта необходимо произвести верхнеуровневое разбиение на компоненты.

1. Пользовательский интерфейс:

- компонент, отвечающий за получение запросов от бота и отправку ответов;
- компонент, отвечающий за взаимодействие пользователя с панелью администратора.

2. Бизнес-логика:

- модели, соответствующие выделенным сущностям;
- контроллеры для четырех видов пользователей;
- компоненты, отвечающие за генерацию данных.

3. Доступ к данным:

- репозитории, служащие промежуточным звеном между приложением и базой данных;
- компонент, отвечающий за обработку запросов к базе данных.

3.3.2 Создание таблиц, ограничений, ролей и триггеров

Необходимо реализовать создание таблиц и ограничений к ним. В приложении А приведены соответствующие листинги. Были написаны ограничения по внешним и первичным ключам, проверка на NULL, проверка на неотрицательность чисел. В случае пола – это проверка на принадлежность значения установленным.

Роль – это разрешение, которое предоставляется пользователям для доступа к определенным данным. В приложении Б представлена реализация ролевой модели: создаются роли гости, арендатора, арендодателя и администратора и выдаются им права.

Триггер – это хранимая процедура специального типа, которая автоматически выполняется при наступлении определенного события. В приложении В приведено создание триггеров.

3.4 Интерфейс приложения

Пользователь начинает работу с ботом вызовом команды /start, после чего выводится приветственное сообщение. Команда /help выводит список всех возможностей. На рисунке 3.1 представлен результат вызова данных команд.

Пользователь может:

- зарегистрироваться как арендатор или арендодатель;
- добавить объявление о сдаче квартиры в аренду, поиске соседа и продаже бытовых товаров;
- посмотреть аналогичные объявления как с фильтрами, так и без;

- посмотреть информацию об арендодателе, поставить ему оценку, подписаться;
- оценить квартиру;
- подписаться на квартиры с определенными параметрами.

Каждому типу пользователя (гостю, арендатору и арендодателю) доступна часть функционала.

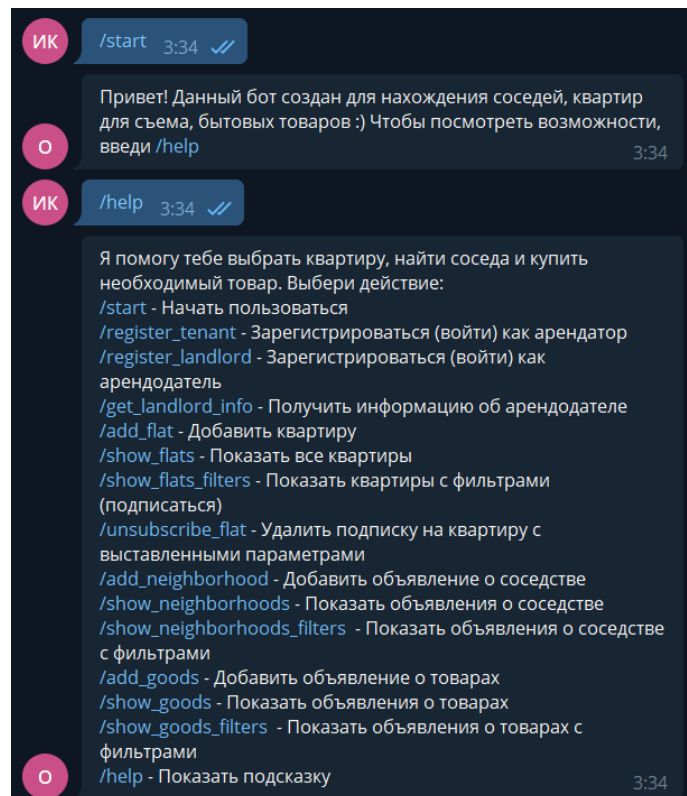


Рисунок 3.1 – Вызов приветственных в боте

Для администратора разработана отдельная панель, благодаря которой он может отслеживать текущее состояние данных, генерировать новые, удалять, редактировать и добавлять. На рисунке 3.2 представлен вид окна панели администратора.

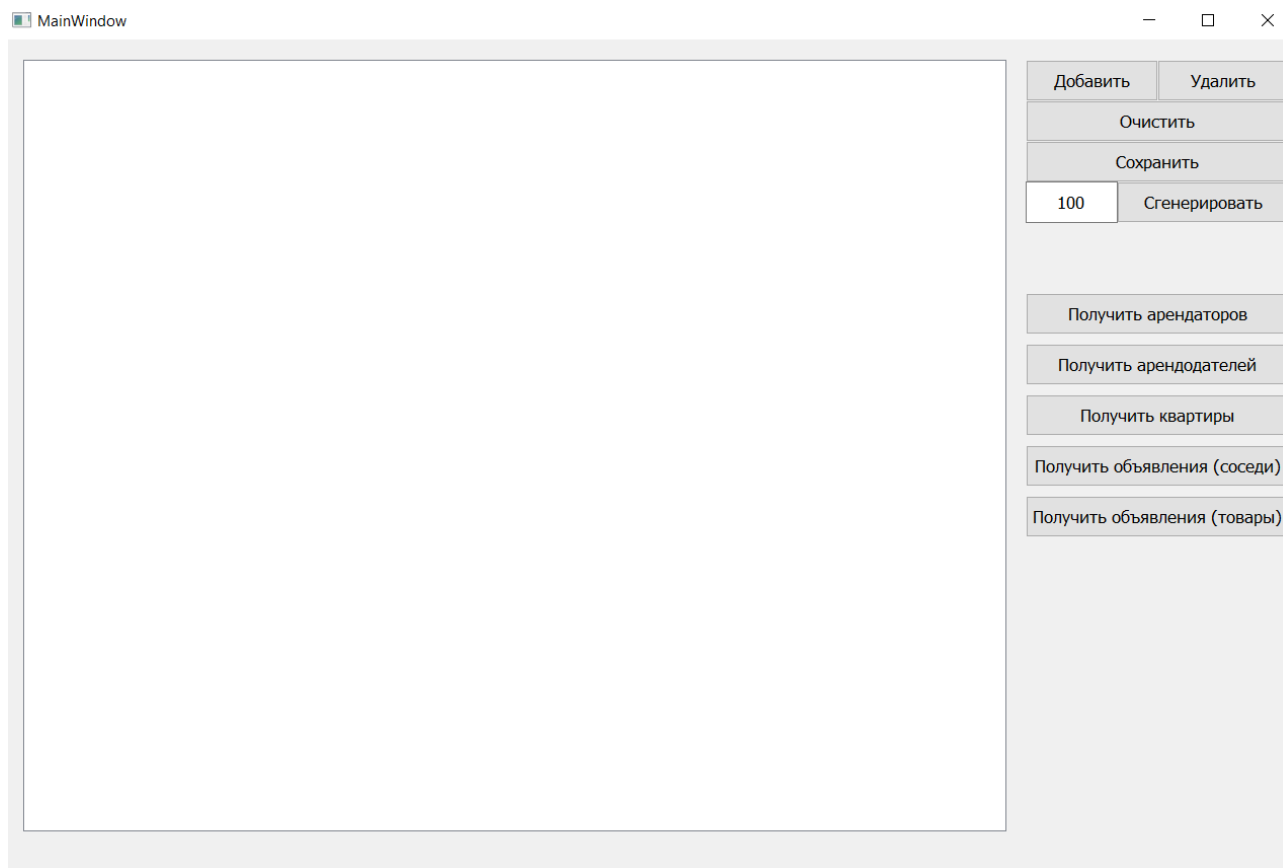


Рисунок 3.2 – Панель администратора

Вывод

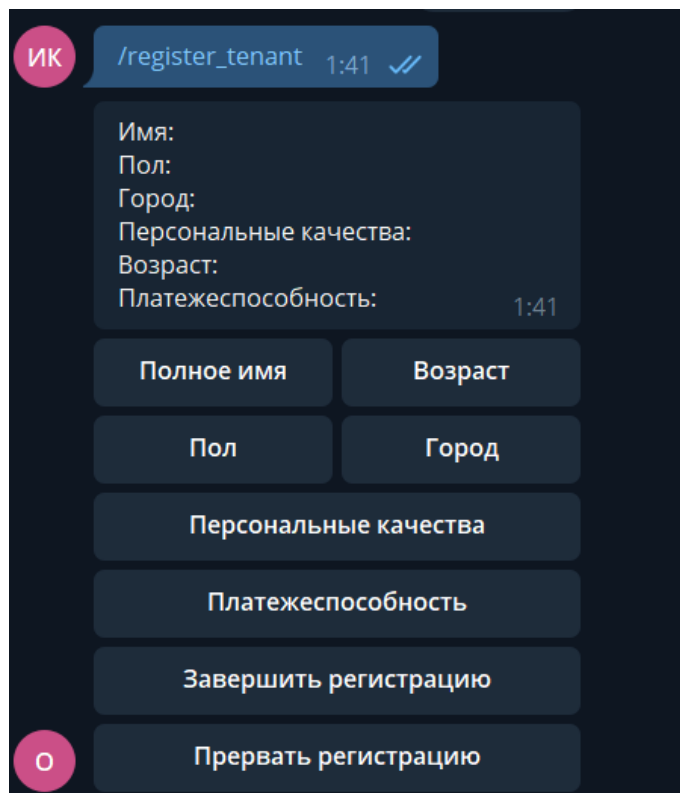
В результате анализа реляционных СУБД был сделан выбор в пользу PostgreSQL. Были выбрана инструменты разработки (язык программирования – Python, библиотека для разработки бота – aiogram, фреймворк для разработки панели администратора – PyQt5), а также проведено верхнеуровневое разбиение (выделены три «слоя» с соответствующими компонентами), приведены детали реализации (создание таблиц, ограничений, ролей, триггеров) и интерфейс приложения.

4 Исследовательская часть

В данном разделе представлены примеры работы разработанного приложения (как бота, так и панели администратора), ставится цель эксперимента, приводятся технические характеристики устройства, на котором он проводится. В результате выполнения эксперимента делается сравнительный анализ времени выполнения запросов к базе данных с использованием индексов и без.

4.1 Примеры работы

После запуска бота пользователь может воспользоваться предложенными функциями. На рисунках 4.1 - 4.2 показаны формы регистрации в качестве арендатора и арендодателя соответственно.



The image shows a Telegram chat interface with a bot. At the top, a blue message bubble contains the command `/register_tenant`, the time `1:41`, and a checkmark. Below it, a list of registration fields is displayed: `Имя:`, `Пол:`, `Город:`, `Персональные качества:`, `Возраст:`, and `Платежеспособность:`. To the right of the last field is the time `1:41`. Below the fields are several buttons: `Полное имя` and `Возраст` (top row), `Пол` and `Город` (second row), `Персональные качества` (third row), `Платежеспособность` (fourth row), `Завершить регистрацию` (fifth row), and `Прервать регистрацию` (bottom row, preceded by a pink circle with a white 'O' icon).

Рисунок 4.1 – Форма регистрации для арендатора в боте

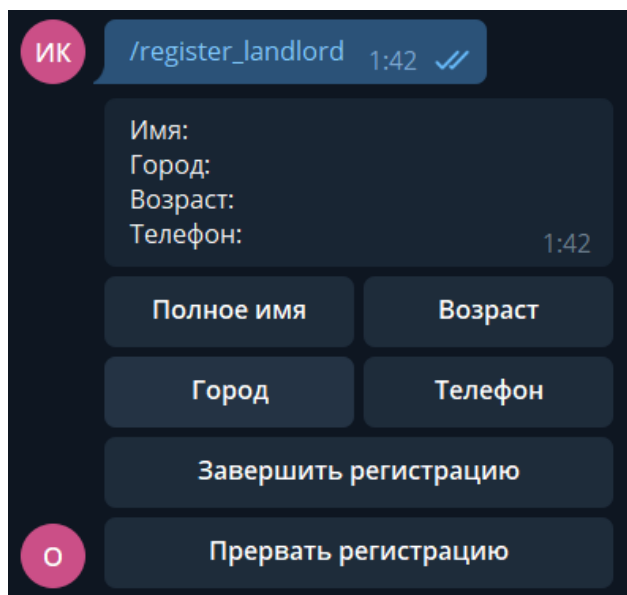


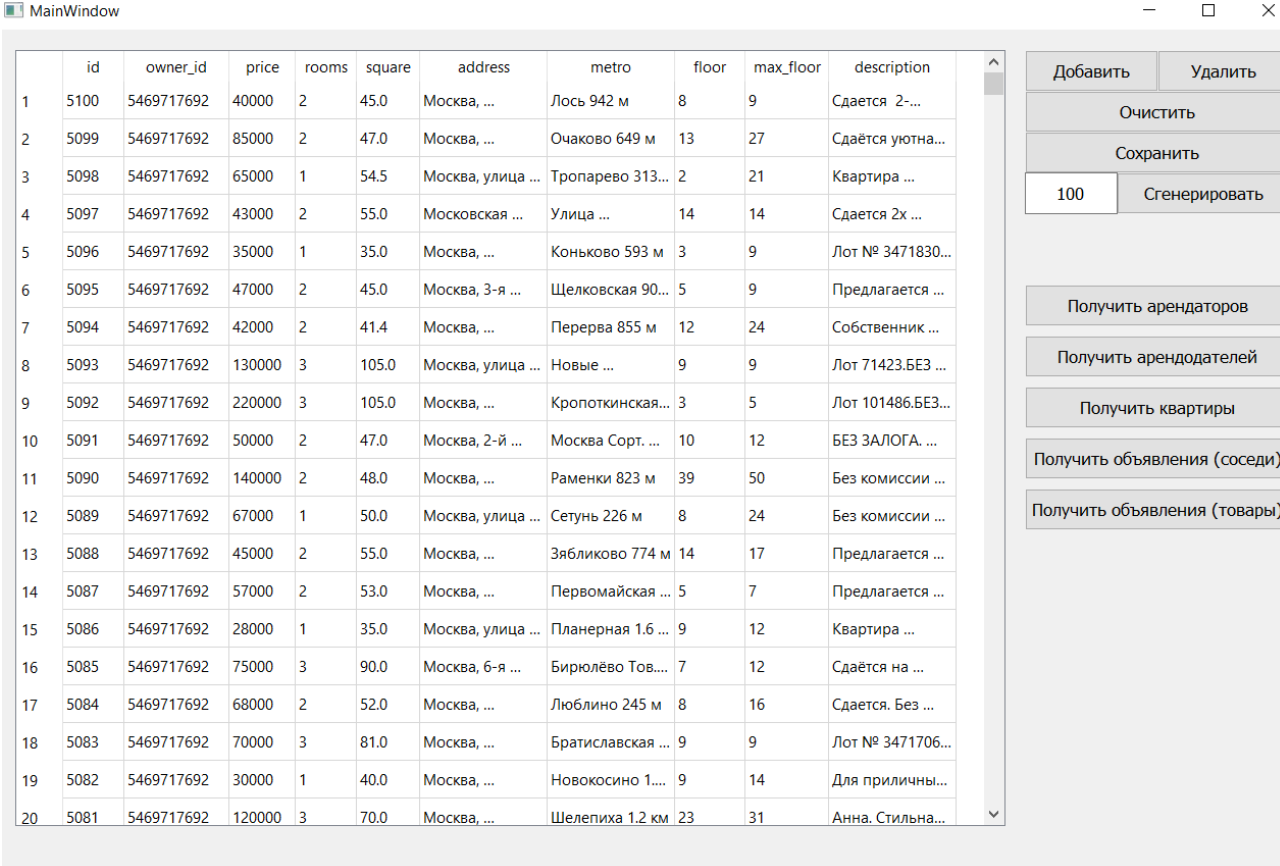
Рисунок 4.2 – Форма регистрации для арендодателя в боте

На рисунке 4.3 представлен результат вызова команды показа всех квартир (присутствует пагинация).



Рисунок 4.3 – Просмотр квартир в боте

В качестве примера для демонстрации работы в панели администратора на рисунке 4.4 приведена таблица, содержащая информацию о выложенных квартирах.



	id	owner_id	price	rooms	square	address	metro	floor	max_floor	description
1	5100	5469717692	40000	2	45.0	Москва, ...	Лось 942 м	8	9	Сдается 2-...
2	5099	5469717692	85000	2	47.0	Москва, ...	Очаково 649 м	13	27	Сдаётся уютна...
3	5098	5469717692	65000	1	54.5	Москва, улица ...	Тропарево 313...	2	21	Квартира ...
4	5097	5469717692	43000	2	55.0	Московская ...	Улица ...	14	14	Сдается 2х ...
5	5096	5469717692	35000	1	35.0	Москва, ...	Коньково 593 м	3	9	Лот № 3471830...
6	5095	5469717692	47000	2	45.0	Москва, 3-я ...	Щелковская 90...	5	9	Предлагается ...
7	5094	5469717692	42000	2	41.4	Москва, ...	Перерва 855 м	12	24	Собственник ...
8	5093	5469717692	130000	3	105.0	Москва, улица ...	Новые ...	9	9	Лот 71423.БЕЗ ...
9	5092	5469717692	220000	3	105.0	Москва, ...	Кропоткинская...	3	5	Лот 101486.БЕЗ...
10	5091	5469717692	50000	2	47.0	Москва, 2-й ...	Москва Сорт. ...	10	12	БЕЗ ЗАЛОГА. ...
11	5090	5469717692	140000	2	48.0	Москва, ...	Раменки 823 м	39	50	Без комиссии ...
12	5089	5469717692	67000	1	50.0	Москва, улица ...	Сетунь 226 м	8	24	Без комиссии ...
13	5088	5469717692	45000	2	55.0	Москва, ...	Зябликово 774 м	14	17	Предлагается ...
14	5087	5469717692	57000	2	53.0	Москва, ...	Первомайская ...	5	7	Предлагается ...
15	5086	5469717692	28000	1	35.0	Москва, улица ...	Планерная 1.6 ...	9	12	Квартира ...
16	5085	5469717692	75000	3	90.0	Москва, 6-я ...	Бирюлёво Тов....	7	12	Сдаётся на ...
17	5084	5469717692	68000	2	52.0	Москва, ...	Люблино 245 м	8	16	Сдается. Без ...
18	5083	5469717692	70000	3	81.0	Москва, ...	Братиславская ...	9	9	Лот № 3471706...
19	5082	5469717692	30000	1	40.0	Москва, ...	Новокосино 1....	9	14	Для приличны...
20	5081	5469717692	120000	3	70.0	Москва, ...	Шелепиха 1.2 км	23	31	Анна. Стильна...

Добавить Удалить
Очистить
Сохранить
100 Сгенерировать
Получить арендаторов
Получить арендодателей
Получить квартиры
Получить объявления (соседи)
Получить объявления (товары)

Рисунок 4.4 – Таблица в панели администратора, содержащая информацию о выложенных квартирах

4.2 Цель эксперимента

Индексы — это средство увеличения производительности БД. Используя индекс, сервер баз данных может находить и извлекать нужные строки гораздо быстрее, чем без него [15]. Целью эксперимента является сравнение времени выполнения запросов с использованием индексов и без. Будут выполнены три типа запросов — к таблице квартир по значению поля `owner_id`, к таблице объявлений о поиске соседа по значению поля `tenant_id` и к таблице товаров по значению поля `owner_id`.

4.3 Технические характеристики

Эксперимент ставился на ноутбуке Huawei MateBook D 14. Важно отметить, что ноутбук был включен в сеть питания и нагружен только встроенными приложениями окружения и непосредственно самой тестируемой системой. Технические характеристики данного устройства:

- операционная система: Windows 10 (64-разрядная);
- оперативная память: 8 Гб;
- процессор: AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx 2.10 GHz;
- количество ядер: 4;
- количество логических процессоров: 8.

4.4 Проведение эксперимента

Во время проведения эксперимента замерялось время выполнения каждого из запросов 1000 раз с индексами и без и находилось среднее арифметическое. В таблице 4.1 показан результат в миллисекундах.

Таблица 4.1 – Время выполнения запросов с индексами и без

Запрос	Время выполнения без индексов, мс	Время выполнения с индексами, мс
Поиск квартир по владельцу	0.203159	0.087331
Поиск объявления по поиску соседа по арендатору	0.148975	0.047203
Поиск товара по владельцу	0.117043	0.044737

Вывод

В данном разделе были представлены примеры работы разработанного приложения, а также проведен эксперимент, в результате которого проведен

сравнительный анализ времени выполнения запросов с использованием индексов и без. В результате исследования было установлено, что использование индексов позволило ускорить работу в среднем на 60%.

ЗАКЛЮЧЕНИЕ

В результате выполнения работы была достигнута цель: реализована база данных, используемая в Telegram-боте, который позволяет пользователям находить жилье и соседей. Были достигнуты все поставленные задачи:

- 1) определены функциональные требования к разрабатываемому программному продукту;
- 2) определена ролевую модель;
- 3) проведен анализ моделей данных и выбран наиболее подходящую;
- 4) спроектирована база данных, описаны ее сущности и связи;
- 5) реализована спроектированная база данных;
- 6) реализован сервис, обеспечивающий доступ к базе данных;
- 7) реализована панель администратора для контроля за ботом;
- 8) проведен сравнительный анализ времени выполнения различных запросов к базе данных с использованием индексов и без.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Жилье для студентов в Москва // Сеть московских общежитий [Электронный ресурс]. Режим доступа: <https://msopro.ru/zhilye-dlja-studentov-v-moskve.html> (дата обращения: 13.05.2022).
2. Почему некоторые студенты остаются без общежития // Горизонтальная Россия [Электронный ресурс]. Режим доступа: <https://semnasem.org/articles/2021/09/08/nehvatka-mest-v-studencheskih-obshezhitiyah> (дата обращения: 13.05.2022).
3. Модели данных / Каптерев А.И. // Электронный учебник по информатике [Электронный ресурс]. Режим доступа: http://www.mediagnosis.ru/Autorun/Page6/10_3_.htm (дата обращения: 14.05.2022).
4. Постреляционная модель: понятие, достоинства и недостатки [Электронный ресурс]. Режим доступа: <https://studfile.net/preview/5407090/page:4/> (дата обращения 14.05.2022).
5. Обзор систем управления базами данных (СУБД) для систем контроля и управления доступом (СКУД) [Электронный ресурс]. Режим доступа: <https://www.parsec.ru/articles/obzor-sistem-upravleniya-bazami-dannykh-subd-dlya-sistem-kontrolya-i-upravleniya-dostupom-skud/> (дата обращения: 15.05.2022).
6. MySQL — система управления базами данных [Электронный ресурс]. Режим доступа: <https://web-creator.ru/articles/mysql> (дата обращения: 10.06.2022).
7. Система управления базами данных MySQL [Электронный ресурс]. Режим доступа: https://depix.ru/articles/sistema_upravleniya_bazami_dannyh_mysql (дата обращения: 10.06.2022).

8. PostgreSQL — объектно-реляционная система управления базами данных [Электронный ресурс]. Режим доступа: <https://web-creator.ru/articles/postgresql> (дата обращения: 11.06.2022).
9. СУБД Oracle: обзор характеристик и возможностей базы данных [Электронный ресурс]. Режим доступа: <https://oracle-patches.com/oracle/prof/субд-oracle-обзор-характеристик-и-возможностей-базы-данных> (дата обращения: 11.06.2022).
10. Chapter 41. PL/pgSQL - SQL Procedural Language [Электронный ресурс]. Режим доступа: <https://www.postgresql.org/docs/9.6/plpgsql.html> (дата обращения: 25.08.2022).
11. Python 3.10.7 documentation [Электронный ресурс]. Режим доступа: <https://docs.python.org/3/> (дата обращения 25.08.2022).
12. Aiogram's documentation [Электронный ресурс]. Режим доступа: <https://docs.aiogram.dev/en/latest/> (дата обращения 25.08.2022).
13. Qt for Python [Электронный ресурс]. Режим доступа: <https://doc.qt.io/qtforpython/> (дата обращения 26.08.2022).
14. Qt Designer Manual [Электронный ресурс]. Режим доступа: <https://doc.qt.io/qt-5/qtdesigner-manual.html> (дата обращения: 26.08.2022).
15. Глава 11. Индексы [Электронный ресурс]. Режим доступа: <https://postgrespro.ru/docs/postgresql/9.6/indexes> (дата обращения: 03.09.2022).

ПРИЛОЖЕНИЕ А

Создание таблиц и ограничений

В листингах А.1 - А.2 представлен скрипт создания таблиц, в листингах А.3 - А.9 – создания ограничений.

Листинг А.1 – Создание таблиц. Часть 1

```
CREATE TABLE IF NOT EXISTS public.tenant (  
    id BIGINT,  
    full_name VARCHAR(100),  
    sex CHAR,  
    city VARCHAR(30),  
    personal_qualities TEXT,  
    age INTEGER,  
    solvency BOOLEAN,  
    username VARCHAR(35)  
);  
  
CREATE TABLE IF NOT EXISTS public.landlord (  
    id BIGINT,  
    full_name VARCHAR(100),  
    city VARCHAR(30),  
    rating REAL,  
    age INTEGER,  
    phone VARCHAR(15),  
    username VARCHAR(35)  
);  
  
CREATE TABLE IF NOT EXISTS public.flat (  
    id SERIAL,  
    owner_id BIGINT,  
    price INTEGER,  
    rooms INTEGER,  
    square REAL,  
    address VARCHAR(200),  
    metro VARCHAR(30),  
    floor INTEGER,  
    max_floor INTEGER,  
    description TEXT  
);  
  
CREATE TABLE IF NOT EXISTS public.flat_photo (  
    flat_id INTEGER,  
    photo VARCHAR(200)  
);
```

Листинг А.2 – Создание таблиц. Часть 2

```
CREATE TABLE IF NOT EXISTS public.neighborhood (
    id SERIAL,
    tenant_id BIGINT,
    neighbors INTEGER,
    price INTEGER,
    place TEXT,
    sex CHAR,
    preferences TEXT
);

CREATE TABLE IF NOT EXISTS public.goods (
    id SERIAL,
    owner_id BIGINT,
    name VARCHAR(50),
    price INTEGER,
    condition CHAR,
    bargain BOOLEAN
);

CREATE TABLE IF NOT EXISTS public.subscription_landlord (
    tenant_id BIGINT,
    landlord_id BIGINT
);

CREATE TABLE IF NOT EXISTS public.likes_flat (
    tenant_id BIGINT,
    flat_id INTEGER
);

CREATE TABLE IF NOT EXISTS public.subscription_flat (
    tenant_id BIGINT,
    min_price INTEGER,
    max_price INTEGER,
    min_rooms INTEGER,
    max_rooms INTEGER,
    min_square REAL,
    max_square REAL
);

CREATE TABLE IF NOT EXISTS public.subscription_metro (
    tenant_id BIGINT,
    metro VARCHAR(30)
);
```

Листинг А.3 – Создание ограничений. Часть 1

```
-- function that adds constraint if it doesn't exist
CREATE OR REPLACE FUNCTION public.create_constraint (t_name text, c_name text,
c_sql text)
RETURNS VOID AS
$$
BEGIN
    IF NOT EXISTS (SELECT constraint_name
                    FROM information_schema.constraint_column_usage
                    WHERE table_name = t_name AND constraint_name = c_name) THEN
        EXECUTE c_sql;
    END IF;
END;
$$
LANGUAGE 'plpgsql';

-- create constraints to public.tenant
SELECT public.create_constraint (
    'tenant',
    'tenant_pkey',
    'ALTER TABLE public.tenant ADD CONSTRAINT tenant_pkey PRIMARY KEY (id);'
);

SELECT public.create_constraint (
    'tenant',
    'tenant_id_check',
    'ALTER TABLE public.tenant ADD CONSTRAINT tenant_id_check CHECK (id > 0 AND
id IS NOT NULL);'
);

SELECT public.create_constraint(
    'tenant',
    'tenant_full_name_check',
    'ALTER TABLE public.tenant ADD CONSTRAINT tenant_full_name_check CHECK
(full_name IS NOT NULL);'
);

SELECT public.create_constraint(
    'tenant',
    'tenant_sex_check',
    'ALTER TABLE public.tenant ADD CONSTRAINT tenant_sex_check CHECK (sex IN
(''M'', ''F'') AND sex IS NOT NULL);'
);

SELECT public.create_constraint(
    'tenant',
    'tenant_city_check',
    'ALTER TABLE public.tenant ADD CONSTRAINT tenant_city_check CHECK (city IS
NOT NULL);'
);

SELECT public.create_constraint(
    'tenant',
    'tenant_age_check',
    'ALTER TABLE public.tenant ADD CONSTRAINT tenant_age_check CHECK (age >= 14
AND age <= 100 AND age IS NOT NULL);'
);
```


Листинг А.4 – Создание ограничений. Часть 2

```
-- create constraints to public.landlord
SELECT public.create_constraint(
    'landlord',
    'landlord_pkey',
    'ALTER TABLE public.landlord ADD CONSTRAINT landlord_pkey PRIMARY KEY (id);'
);

SELECT public.create_constraint (
    'landlord',
    'landlord_id_check',
    'ALTER TABLE public.landlord ADD CONSTRAINT landlord_id_check CHECK (id > 0
AND id IS NOT NULL);'
);

SELECT public.create_constraint(
    'landlord',
    'landlord_full_name_check',
    'ALTER TABLE public.landlord ADD CONSTRAINT landlord_full_name_check CHECK
(full_name IS NOT NULL);'
);

SELECT public.create_constraint(
    'landlord',
    'landlord_city_check',
    'ALTER TABLE public.landlord ADD CONSTRAINT landlord_city_check CHECK (city
IS NOT NULL);'
);

SELECT public.create_constraint(
    'landlord',
    'landlord_rating_check',
    'ALTER TABLE public.landlord ADD CONSTRAINT landlord_rating_check CHECK
(rating >= 0.0 AND rating <= 10.0 AND rating IS NOT NULL);'
);

SELECT public.create_constraint(
    'landlord',
    'landlord_age_check',
    'ALTER TABLE public.landlord ADD CONSTRAINT landlord_age_check CHECK (age >=
14 AND age <= 100 AND age IS NOT NULL);'
);

SELECT public.create_constraint(
    'landlord',
    'landlord_phone_check',
    'ALTER TABLE public.landlord ADD CONSTRAINT landlord_phone_check CHECK
(char_length(phone) > 10 AND phone IS NOT NULL);'
);

-- create constraints to public.flat
SELECT public.create_constraint(
    'flat',
    'flat_pkey',
    'ALTER TABLE public.flat ADD CONSTRAINT flat_pkey PRIMARY KEY (id);'
);
```

Листинг А.5 – Создание ограничений. Часть 3

```
SELECT public.create_constraint (  
    'landlord',  
    'flat_owner_id_fkey',  
    'ALTER TABLE public.flat ADD CONSTRAINT flat_owner_id_fkey FOREIGN KEY  
(owner_id) REFERENCES public.landlord (id);'  
);  
  
SELECT public.create_constraint(  
    'flat',  
    'flat_price_check',  
    'ALTER TABLE public.flat ADD CONSTRAINT flat_price_check CHECK (price > 0  
AND price IS NOT NULL);'  
);  
  
SELECT public.create_constraint(  
    'flat',  
    'flat_rooms_check',  
    'ALTER TABLE public.flat ADD CONSTRAINT flat_rooms_check CHECK (rooms > 0  
AND rooms IS NOT NULL);'  
);  
  
SELECT public.create_constraint(  
    'flat',  
    'flat_square_check',  
    'ALTER TABLE public.flat ADD CONSTRAINT flat_square_check CHECK (square > 0  
AND square IS NOT NULL);'  
);  
  
SELECT public.create_constraint(  
    'flat',  
    'flat_address_check',  
    'ALTER TABLE public.flat ADD CONSTRAINT flat_address_check CHECK (address IS  
NOT NULL);'  
);  
  
SELECT public.create_constraint(  
    'flat',  
    'flat_floor_check',  
    'ALTER TABLE public.flat ADD CONSTRAINT flat_floor_check CHECK (floor >=  
0);'  
);  
  
SELECT public.create_constraint(  
    'flat',  
    'flat_max_floor_check',  
    'ALTER TABLE public.flat ADD CONSTRAINT flat_max_floor_check CHECK  
(max_floor >= 0 AND max_floor >= floor);'  
);  
  
-- create constraints to public.flat_photo  
SELECT public.create_constraint (  
    'flat',  
    'flat_photo_flat_id_fkey',  
    'ALTER TABLE public.flat_photo ADD CONSTRAINT flat_photo_flat_id_fkey  
FOREIGN KEY (flat_id) REFERENCES public.flat (id);'  
);
```

Листинг А.6 – Создание ограничений. Часть 4

```
SELECT public.create_constraint (
    'flat_photo',
    'flat_photo_flat_id_check',
    'ALTER TABLE public.flat_photo ADD CONSTRAINT flat_photo_flat_id_check CHECK
(flat_id IS NOT NULL);'
);

SELECT public.create_constraint(
    'flat_photo',
    'flat_photo_check',
    'ALTER TABLE public.flat_photo ADD CONSTRAINT flat_photo_check CHECK (photo
IS NOT NULL);'
);

-- create constrains to public.neighborhood
SELECT public.create_constraint (
    'neighborhood',
    'neighborhood_pkey',
    'ALTER TABLE public.neighborhood ADD CONSTRAINT neighborhood_pkey PRIMARY
KEY (id);'
);

SELECT public.create_constraint (
    'tenant',
    'neighborhood_tenant_id_fkey',
    'ALTER TABLE public.neighborhood ADD CONSTRAINT neighborhood_tenant_id_fkey
FOREIGN KEY (tenant_id) REFERENCES public.tenant (id);'
);

SELECT public.create_constraint(
    'neighborhood',
    'neighborhood_neighbors_check',
    'ALTER TABLE public.neighborhood ADD CONSTRAINT neighborhood_neighbors_check
CHECK (neighbors > 0 AND neighbors IS NOT NULL);'
);

SELECT public.create_constraint(
    'neighborhood',
    'neighborhood_price_check',
    'ALTER TABLE public.neighborhood ADD CONSTRAINT neighborhood_price_check
CHECK (price > 0 AND price IS NOT NULL);'
);

SELECT public.create_constraint(
    'neighborhood',
    'neighborhood_sex_check',
    'ALTER TABLE public.neighborhood ADD CONSTRAINT neighborhood_sex_check CHECK
(sex IN ('M', 'F', 'N') AND sex IS NOT NULL);'
);

-- create constraints to public.goods
SELECT public.create_constraint (
    'goods',
    'goods_pkey',
    'ALTER TABLE public.goods ADD CONSTRAINT goods_pkey PRIMARY KEY (id);'
```

Листинг А.7 – Создание ограничений. Часть 5

```
SELECT public.create_constraint (
    'tenant',
    'goods_owner_id_fkey',
    'ALTER TABLE public.goods ADD CONSTRAINT goods_owner_id_fkey FOREIGN KEY
(owner_id) REFERENCES public.tenant (id);'
);

SELECT public.create_constraint(
    'goods',
    'goods_name_check',
    'ALTER TABLE public.goods ADD CONSTRAINT goods_name_check CHECK (name IS NOT
NULL);'
);

SELECT public.create_constraint(
    'goods',
    'goods_price_check',
    'ALTER TABLE public.goods ADD CONSTRAINT goods_price_check CHECK (price > 0
AND price IS NOT NULL);'
);

-- E - excellent
-- G - good
-- S - satisfactory
-- U - unsatisfactory
-- T - terrible
SELECT public.create_constraint(
    'goods',
    'goods_condition_check',
    'ALTER TABLE public.goods ADD CONSTRAINT goods_condition_check CHECK ' ||
    '(condition IN (''E'', ''G'', ''S'', ''U'', ''T'')) AND condition IS NOT
NULL);'
);

-- create constraints to public.subscription_landlord
SELECT public.create_constraint (
    'tenant',
    'subscription_landlord_tenant_id_fkey',
    'ALTER TABLE public.subscription_landlord ADD CONSTRAINT
subscription_landlord_tenant_id_fkey' ||
    ' FOREIGN KEY (tenant_id) REFERENCES public.tenant (id);'
);

SELECT public.create_constraint (
    'subscription_landlord',
    'subscription_landlord_tenant_id_check',
    'ALTER TABLE public.subscription_landlord ADD CONSTRAINT
subscription_landlord_tenant_id_check' ||
    ' CHECK (tenant_id IS NOT NULL);'
);

SELECT public.create_constraint (
    'landlord',
    'subscription_landlord_id_fkey',
    'ALTER TABLE public.subscription_landlord ADD CONSTRAINT
subscription_landlord_id_fkey' ||
    ' FOREIGN KEY (landlord_id) REFERENCES public.landlord (id);');
```

Листинг А8 – Создание ограничений. Часть 6

```
SELECT public.create_constraint (  
    'subscription_landlord',  
    'subscription_landlord_id_check',  
    'ALTER TABLE public.subscription_landlord ADD CONSTRAINT  
subscription_landlord_id_check' ||  
    ' CHECK (landlord_id IS NOT NULL);'  
);  
  
-- create constraints to public.likes_flat  
SELECT public.create_constraint (  
    'tenant',  
    'likes_flat_tenant_id_fkey',  
    'ALTER TABLE public.likes_flat ADD CONSTRAINT likes_flat_tenant_id_fkey' ||  
        ' FOREIGN KEY (tenant_id) REFERENCES public.tenant (id);'  
);  
  
SELECT public.create_constraint (  
    'likes_flat',  
    'likes_flat_tenant_id_check',  
    'ALTER TABLE public.likes_flat ADD CONSTRAINT likes_flat_tenant_id_check  
CHECK (tenant_id IS NOT NULL);'  
);  
  
SELECT public.create_constraint (  
    'flat',  
    'likes_flat_id_fkey',  
    'ALTER TABLE public.likes_flat ADD CONSTRAINT likes_flat_id_fkey' ||  
        ' FOREIGN KEY (flat_id) REFERENCES public.flat (id);'  
);  
  
SELECT public.create_constraint (  
    'likes_flat',  
    'likes_flat_id_check',  
    'ALTER TABLE public.likes_flat ADD CONSTRAINT likes_flat_id_check CHECK  
(flat_id IS NOT NULL);'  
);  
  
-- create constraints to public.subscription_flat  
SELECT public.create_constraint (  
    'tenant',  
    'subscription_flat_tenant_id_fkey',  
    'ALTER TABLE public.subscription_flat ADD CONSTRAINT  
subscription_flat_tenant_id_fkey' ||  
        ' FOREIGN KEY (tenant_id) REFERENCES public.tenant (id);'  
);  
  
SELECT public.create_constraint (  
    'subscription_flat',  
    'subscription_flat_tenant_id_check',  
    'ALTER TABLE public.subscription_flat ADD CONSTRAINT  
subscription_flat_tenant_id_check' ||  
        ' CHECK (tenant_id IS NOT NULL);'  
);
```

Листинг А.9 – Создание ограничений. Часть 7

```
SELECT public.create_constraint(  
    'subscription_flat',  
    'subscription_flat_price_check',  
    'ALTER TABLE public.subscription_flat ADD CONSTRAINT  
subscription_flat_price_check' ||  
    ' CHECK (min_price <= max_price AND min_price >= 0);'  
);  
  
SELECT public.create_constraint(  
    'subscription_flat',  
    'subscription_flat_rooms_check',  
    'ALTER TABLE public.subscription_flat ADD CONSTRAINT  
subscription_flat_rooms_check' ||  
    ' CHECK (min_rooms <= max_rooms AND min_rooms >= 0);'  
);  
  
SELECT public.create_constraint(  
    'subscription_flat',  
    'subscription_flat_square_check',  
    'ALTER TABLE public.subscription_flat ADD CONSTRAINT  
subscription_flat_square_check' ||  
    ' CHECK (min_square <= max_square AND min_square >= 0);'  
);  
  
-- create constraints to public.subscription_metro  
SELECT public.create_constraint (  
    'tenant',  
    'subscription_metro_tenant_id_fkey',  
    'ALTER TABLE public.subscription_metro ADD CONSTRAINT  
subscription_metro_tenant_id_fkey ' ||  
    'FOREIGN KEY (tenant_id) REFERENCES public.tenant (id);');  
  
SELECT public.create_constraint (  
    'subscription_metro',  
    'subscription_metro_tenant_id_check',  
    'ALTER TABLE public.subscription_metro ADD CONSTRAINT  
subscription_metro_tenant_id_check CHECK (tenant_id IS NOT NULL);'  
);
```

ПРИЛОЖЕНИЕ Б

Создание ролей

В листингах Б.1 - Б.2 приведено создание ролей (гость, арендатор, арендодатель и администратор).

Листинг Б.1 – Создание ролей. Часть 1

```
-- create role if it doesn't exist
CREATE OR REPLACE FUNCTION public.create_role (r_name text, r_sql text)
RETURNS VOID AS
$$
BEGIN
    IF NOT EXISTS (SELECT FROM pg_catalog.pg_roles
                    WHERE rolname = r_name) THEN
        EXECUTE r_sql;
    END IF;
END;
$$
LANGUAGE 'plpgsql';

-- delete role if it exists
CREATE OR REPLACE FUNCTION public.delete_role (r_name text)
RETURNS VOID AS
$$
BEGIN
    IF EXISTS (SELECT FROM pg_catalog.pg_roles
                WHERE rolname = r_name) THEN
        EXECUTE FORMAT('REASSIGN OWNED BY %s TO postgres;' ||
                        'DROP OWNED BY %s;' ||
                        'DROP ROLE %s;', r_name, r_name, r_name);
    END IF;
END;
$$
LANGUAGE 'plpgsql';

-- create guest role
SELECT public.delete_role('guest');
SELECT public.create_role('guest', 'CREATE ROLE guest LOGIN PASSWORD
'guest''');
GRANT SELECT, INSERT ON public.tenant TO guest;
GRANT SELECT, INSERT ON public.landlord TO guest;
GRANT SELECT ON public.flat TO guest;
GRANT SELECT ON public.flat_photo TO guest;
GRANT SELECT ON public.neighborhood TO guest;
GRANT SELECT ON public.goods TO guest;
```

Листинг Б.2 – Создание ролей. Часть 2

```
-- create tenant role
SELECT public.delete_role('tenant');
SELECT public.create_role('tenant','CREATE ROLE tenant LOGIN PASSWORD
'tenant';');
GRANT SELECT ON public.tenant TO tenant;
GRANT SELECT, INSERT, UPDATE ON public.landlord TO tenant;
GRANT SELECT ON public.flat TO tenant;
GRANT SELECT ON public.flat_photo TO tenant;
GRANT INSERT, SELECT, DELETE ON public.subscription_landlord TO tenant;
GRANT INSERT, SELECT, DELETE ON public.likes_flat TO tenant;
GRANT INSERT, SELECT, DELETE, UPDATE ON public.subscription_flat TO tenant;
GRANT INSERT, SELECT, DELETE ON public.subscription_metro TO tenant;
GRANT INSERT, SELECT ON public.neighborhood TO tenant;
GRANT USAGE, SELECT ON SEQUENCE neighborhood_id_seq TO tenant;
GRANT INSERT, SELECT ON public.goods TO tenant;
GRANT USAGE, SELECT ON SEQUENCE goods_id_seq TO tenant;

-- create landlord role
SELECT public.delete_role('landlord');
SELECT public.create_role('landlord','CREATE ROLE landlord LOGIN PASSWORD
'landlord';');
GRANT SELECT, INSERT ON public.tenant TO landlord;
GRANT SELECT ON public.landlord TO landlord;
GRANT INSERT, SELECT ON public.flat TO landlord;
GRANT USAGE, SELECT ON SEQUENCE flat_id_seq TO landlord;
GRANT INSERT, SELECT ON public.flat_photo TO landlord;
GRANT SELECT ON public.subscription_landlord TO landlord;
GRANT SELECT ON public.subscription_flat TO landlord;
GRANT SELECT ON public.subscription_metro TO landlord;
GRANT SELECT ON public.neighborhood TO landlord;
GRANT SELECT ON public.goods TO landlord;

-- create admin role
SELECT public.delete_role('admin');
SELECT public.create_role('admin','CREATE ROLE admin LOGIN PASSWORD
'admin';');
GRANT postgres TO admin;
```


ПРИЛОЖЕНИЕ В

Создание триггеров

В листингах В.1 - В.2 приведено создание триггеров, срабатывающих после удаления арендатора, арендодателя, квартиры, подписок на квартиру и добавления очередной подписки.

Листинг В.1 – Создание триггеров. Часть 1

```
-- trigger to delete flats and subscriptions before deleting landlord
CREATE OR REPLACE FUNCTION public.delete_landlord_dependencies ()
RETURNS TRIGGER AS
$$
BEGIN
    UPDATE public.flat SET owner_id = NULL WHERE owner_id = old.id;
    DELETE FROM public.subscription_landlord WHERE landlord_id = old.id;
    RETURN old;
END;
$$
LANGUAGE 'plpgsql';

DROP TRIGGER IF EXISTS delete_landlord on public.landlord;

CREATE TRIGGER delete_landlord
BEFORE DELETE on public.landlord
FOR EACH ROW
EXECUTE PROCEDURE public.delete_landlord_dependencies();

-- trigger to delete flats photos before deleting flat
CREATE OR REPLACE FUNCTION public.delete_flat_dependencies ()
RETURNS TRIGGER AS
$$
BEGIN
    DELETE FROM public.flat_photo WHERE flat_id = old.id;
    DELETE FROM public.likes_flat WHERE flat_id = old.id;
    RETURN old;
END;
$$
LANGUAGE 'plpgsql';

DROP TRIGGER IF EXISTS delete_flat on public.flat;

CREATE TRIGGER delete_flat
BEFORE DELETE on public.flat
FOR EACH ROW
EXECUTE PROCEDURE public.delete_flat_dependencies();
```

Листинг В.2 – Создание триггеров. Часть 2

```
-- trigger to delete neighborhoods, goods, subscriptions and likes before tenant
CREATE OR REPLACE FUNCTION public.delete_tenant_dependencies ()
RETURNS TRIGGER AS
$$
BEGIN
    UPDATE public.neighborhood SET tenant_id = NULL WHERE tenant_id = old.id;
    UPDATE public.goods SET owner_id = NULL WHERE owner_id = old.id;
    DELETE FROM public.subscription_landlord WHERE tenant_id = old.id;
    DELETE FROM public.likes_flat WHERE tenant_id = old.id;
    DELETE FROM public.subscription_flat WHERE tenant_id = old.id;
    RETURN old;
END;
$$
LANGUAGE 'plpgsql';

DROP TRIGGER IF EXISTS delete_tenant on public.tenant;

CREATE TRIGGER delete_tenant
BEFORE DELETE on public.tenant
FOR EACH ROW
EXECUTE PROCEDURE public.delete_tenant_dependencies();

-- trigger to delete metro before deleting subscription_flat
CREATE OR REPLACE FUNCTION public.delete_subscription_metro ()
RETURNS TRIGGER AS
$$
BEGIN
    DELETE FROM public.subscription_metro WHERE tenant_id = old.tenant_id;
    RETURN old;
END;
$$
LANGUAGE 'plpgsql';

DROP TRIGGER IF EXISTS delete_subscription_flat on public.subscription_flat;

CREATE TRIGGER delete_subscription_flat
BEFORE DELETE on public.subscription_flat
FOR EACH ROW
EXECUTE PROCEDURE public.delete_subscription_metro();

-- trigger to delete subscription_flat before inserting subscription_flat
CREATE OR REPLACE FUNCTION public.delete_flat_subscription ()
RETURNS TRIGGER AS
$$
BEGIN
    DELETE FROM public.subscription_flat WHERE tenant_id = new.tenant_id;
    RETURN new;
END;
$$
LANGUAGE 'plpgsql';

DROP TRIGGER IF EXISTS insert_subscription_flat on public.subscription_flat;

CREATE TRIGGER insert_subscription_flat
BEFORE INSERT on public.subscription_flat
FOR EACH ROW
EXECUTE PROCEDURE public.delete_flat_subscription();
```