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 _____ И.С.Климов
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсовой работы _____ О.В.Кузнецова
(Подпись, дата) (И.О.Фамилия)

2022 г.

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)

УТВЕРЖДАЮ
Заведующий кафедрой ИУ7
(Индекс)
_____ И.В. Рудаков
(И.О.Фамилия)
« _____ » _____ 2022 г.

З А Д А Н И Е
на выполнение курсовой работы

по дисциплине Базы данных

Студент группы ИУ7-62Б

Климов Илья Сергеевич
(Фамилия, имя, отчество)

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

Направленность КР (учебная, исследовательская, практическая, производственная, др.)

учебная

Источник тематики (кафедра, предприятие, НИР) кафедра

График выполнения работы: 25% к 4 нед., 50% к 7 нед., 75% к 11 нед., 100% к 14 нед.

Задание разработать Telegram-бота, позволяющего регистрироваться пользователям, добавлять и искать объявления об аренде квартир, поиске соседей и продаже бытовых товаров. Предоставить возможность подписки на определенных арендодателей и на появление объявлений с выставленными фильтрами. Разработать функционал, позволяющий пользователю выставлять оценки арендодателям, отмечать понравившиеся квартиры и получать уведомления о новых пользователях, поставивших аналогичную отметку.

Оформление курсовой работы:

2.1. Расчетно-пояснительная записка на 25-30 листах формата А4.
Расчетно-пояснительная записка должна содержать постановку введение, аналитическую часть, конструкторскую часть, технологическую часть, экспериментально-исследовательский раздел, заключение, список литературы, приложения.
2.2. Перечень графического материала (плакаты, схемы, чертежи и т.п.) На защиту проекта должна быть представлена презентация, состоящая из 15-20 слайдов. На слайдах должны быть отражены: постановка задачи, использованные методы и алгоритмы, расчетные соотношения, структура комплекса программ, диаграмма классов, интерфейс, характеристики разработанного ПО, результаты проведенных исследований.

Дата выдачи задания « ___ » _____ 2022 г.

Руководитель курсовой работы _____ О.В. Кузнецова
(Подпись, дата) (И.О.Фамилия)

Студент _____ И.С. Климов
(Подпись, дата) (И.О.Фамилия)

Содержание

Введение **4**
1 Аналитическая часть **6**
1.1 Постановка задачи 6
1.1.1 Тип приложения 6
1.1.2 Ролевая модель 7
1.1.3 Формализация данных 10
1.2 Модель данных 10
1.3 Существующие аналоги 11
Вывод 12
Список использованных источников **29**

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 может стать 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 позволяет 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) заполнить базу данных данными
6) реализовать сервис, позволяющий решить поставленную задачу и обеспечивающий доступ к базе данных;

7) провести сравнительный анализ …

1 Аналитическая часть

В данном разделе ставится постановка задачи, определяются тип сервиса, ролевая модель, модель данных и представляются Use-Case диаграммы и ER-модель.

1.1 Постановка задачи

Для поиска соседа по квартире существует не так много способов: от оставления комментариев в различных тематических группах до опроса знакомых. Все это не является удобным способом достижения желаемого результата. Сервис, включающий в себя соответствующий функционал, существенно облегчит поиски.

1.1.1 Тип сервиса

Сервис можно представить в виде десктопного приложения и 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, которая будет выполнена 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.

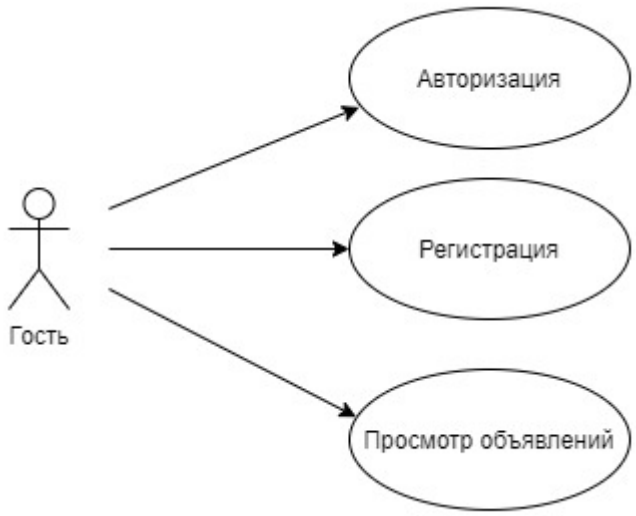

Рисунок 1.1 – Use-Case диаграмма для гостя

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.

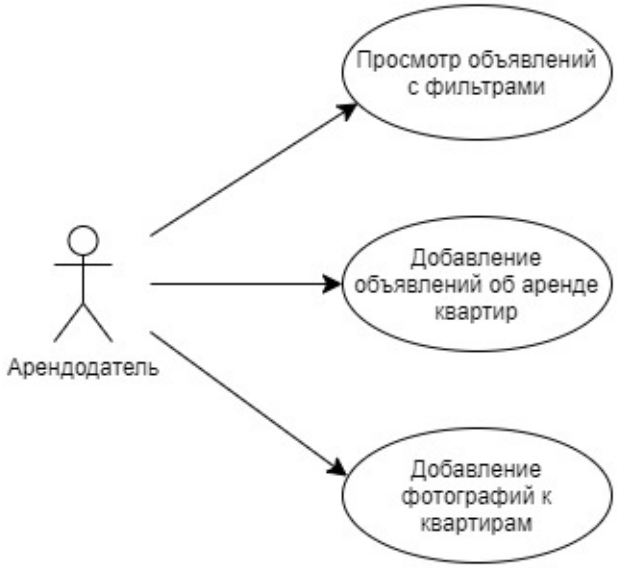

Рисунок 1.3 – Use-Case диаграмма для арендодателя

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.

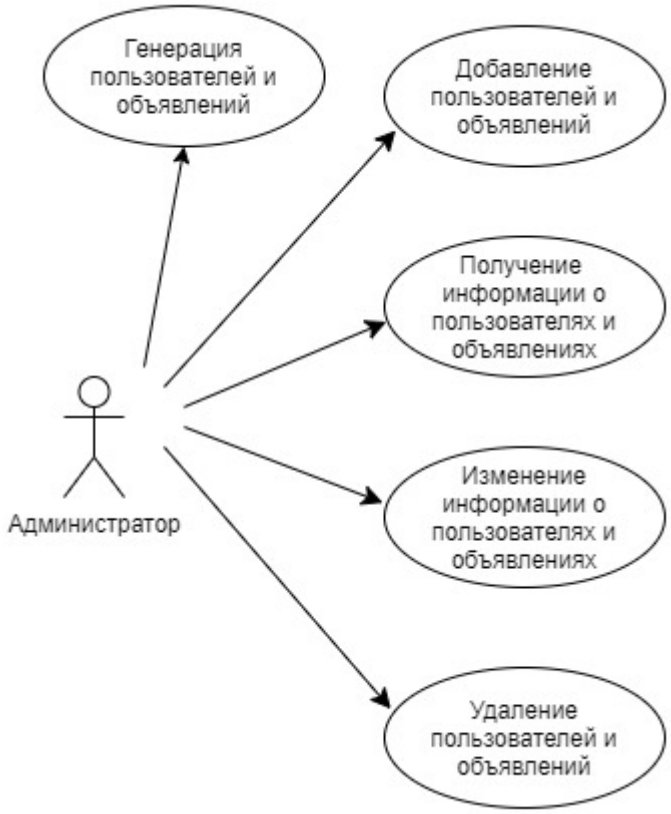

Рисунок 1.4 – Use-Case диаграмма для администратора

1.1.3 Формализация данных

База данных будет содержать информацию про пользователей (арендаторов и арендодателей), объявления (сдаче квартир в аренду, поиске соседа и продаже бытовых товаров), отношения между ними (подписки. На рисунке 1.5 представлена соответствующая ER-модель.

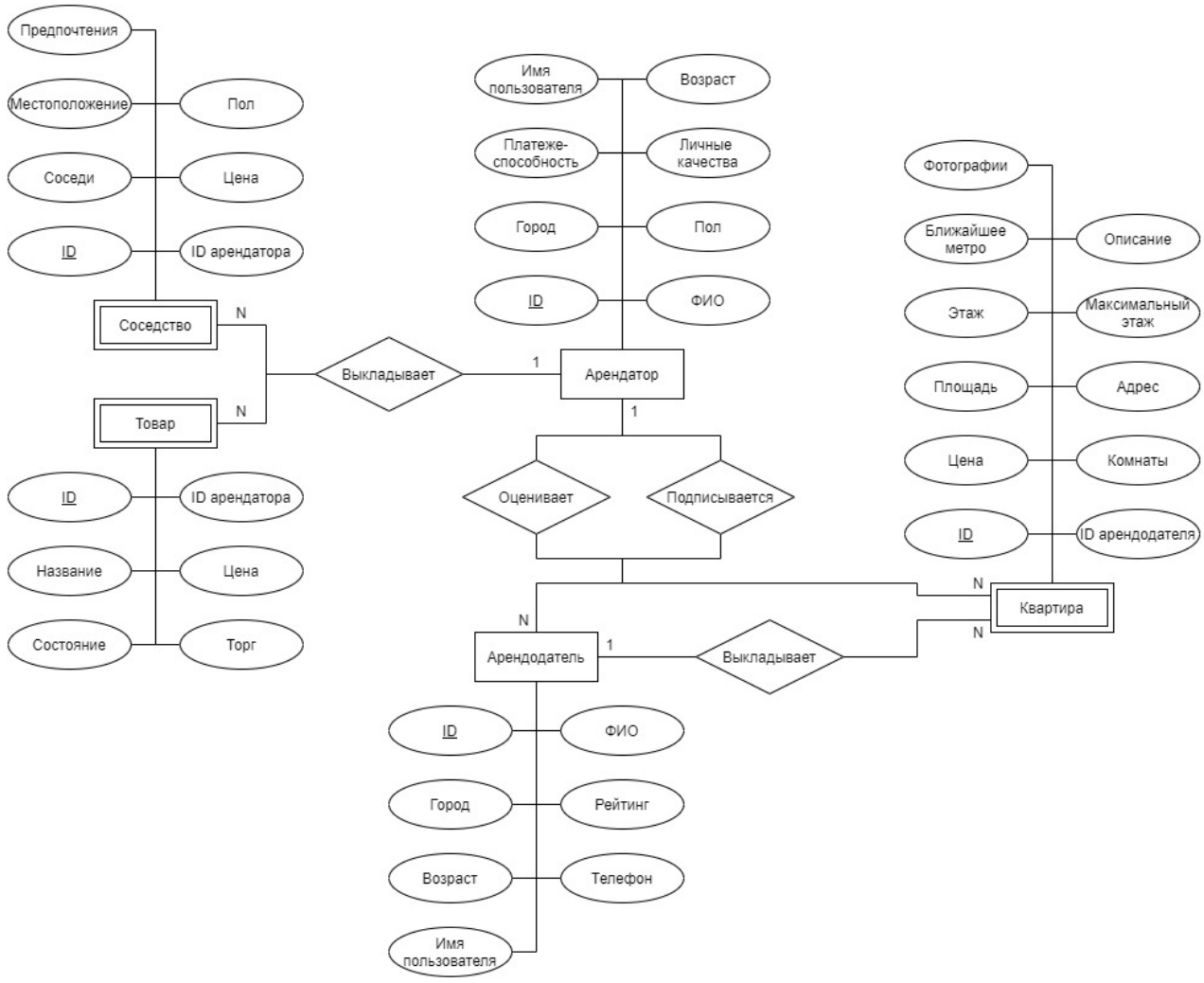

Рисунок 1.5 – ER-модель разрабатываемой БД в нотации Чена

1.2 Модель данных

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

1. Дореляционная модель – появилась одной из первых. Она предшествуют 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;
– получений уведомлений по подпискам.

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

Таблиц 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, ролевая модель), выбрана реляционная модель данных, представлены Use-Case диаграмма и ER-модель. Также были проанализированы существующие аналоги и выявлены преимущества разрабатываемого сервиса по сравнению с другими.

2 Конструкторская часть

В данном разделе представлена диаграмма базы данных, описаны таблицы, приведены схемы триггеров.

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 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.

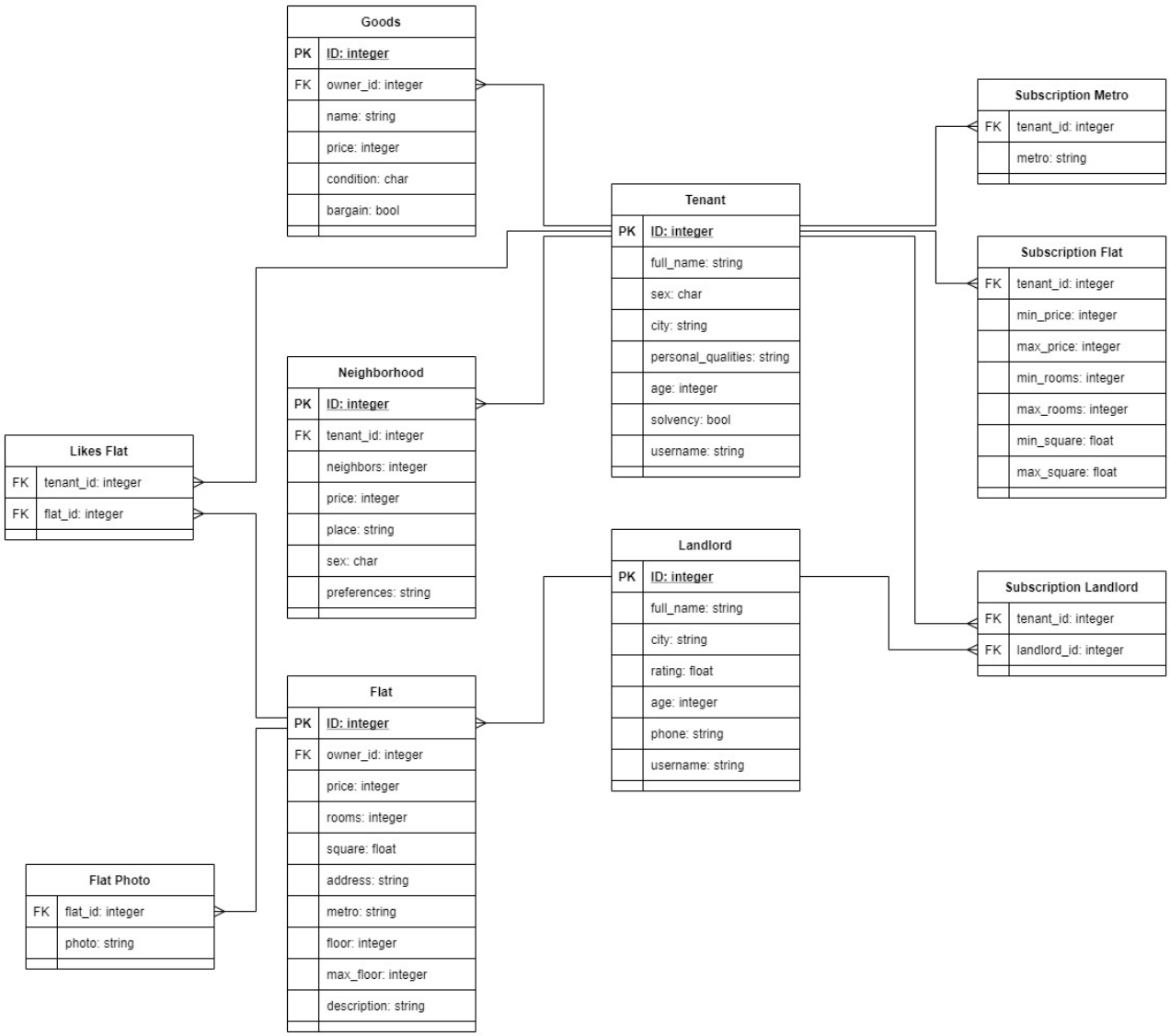

Рисунок 2.1 – Диаграмма базы данных

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). 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.

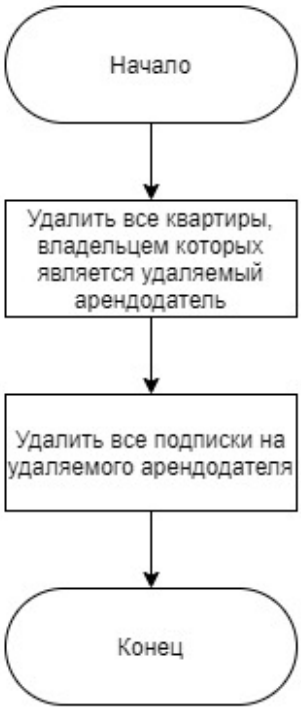

Рисунок 2.3 – Схема триггера delete_landlord

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.

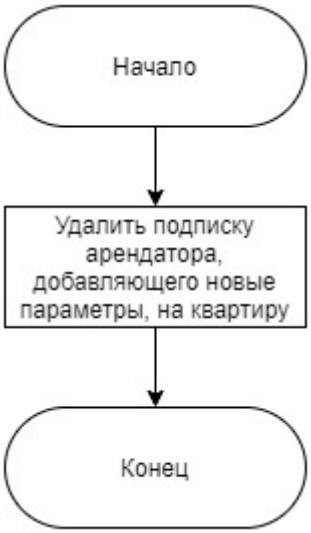

Рисунок 2.6 – Схема триггера insert_subscription_flat

Вывод

В данном разделена была спроектирована база данных и приведены схемы триггеров, срабатывающих при удалении и добавлении данных.

3 Технологическая часть

В данном разделе приведен анализ и выбор инструментов разработки и СУБД. Проведено разбиение приложения на компоненты. Также представлены детали реализации и интерфейс приложения.

3.1 Выбор СУБД

Система управления базами данных (СУБД) — совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных [5]. На основе выбранной реляционной модели данных 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 надежной, 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.

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 с данной СУБД имеется опыт работы. 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 объектную-ориентированную парадигму программирования. Также Python имеет обширное количество библиотек, что открывает большой выбор.

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 выбрана библиотека 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 внутренним 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 квартиру 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.

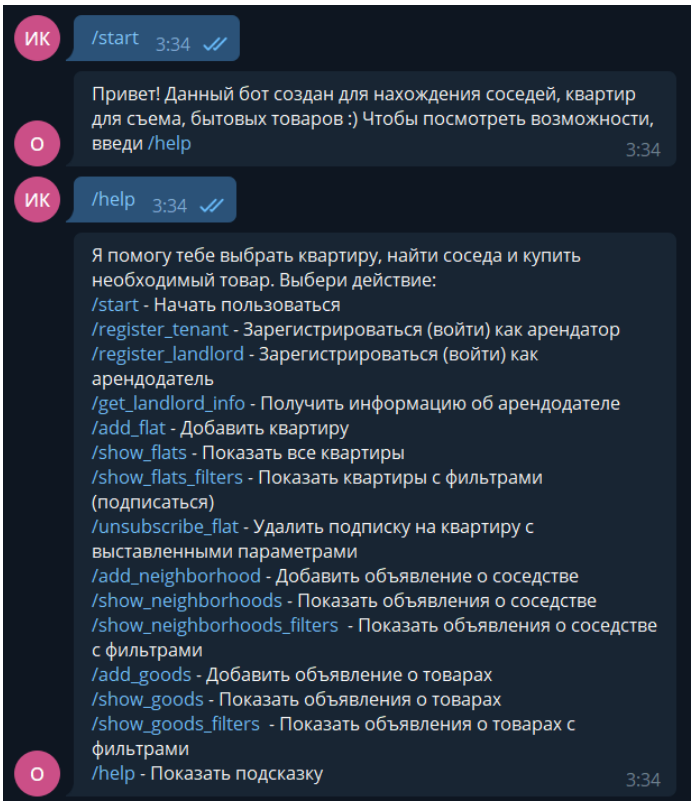

Рисунок 3.1 – Вызов приветственных в боте

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.

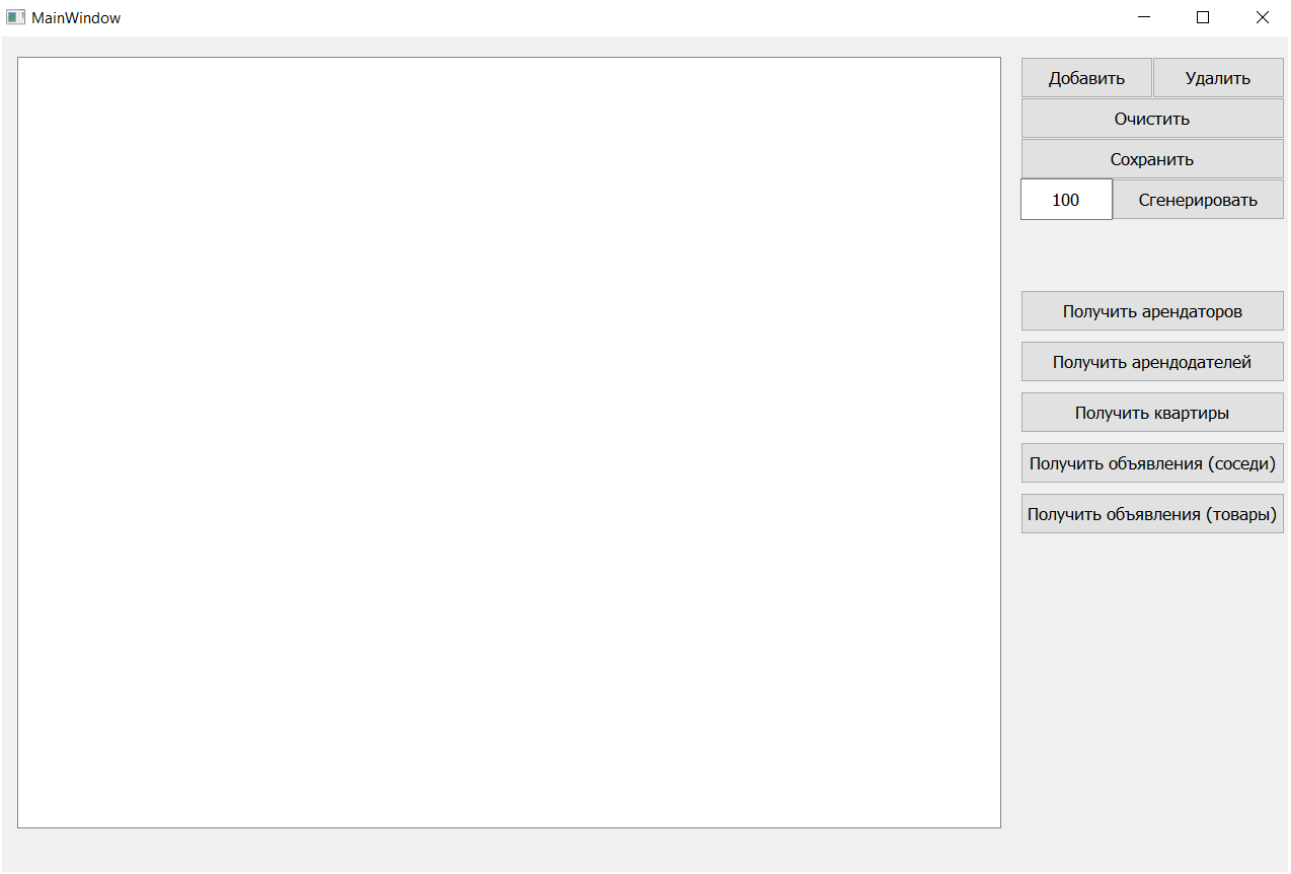

Рисунок 3.2 – Панель администратора

Вывод

В результате анализа наиболее популярных реляционных СУБД был сделан выбор в пользу PostgreSQL. Были выбрана инструменты разработки, а также проведено верхнеуровневое разбиение на компоненты, приведены детали реализации и интерфейс приложения.

4 Исследовательская часть

Список использованных источников

1) Источник 1

ПРИЛОЖЕНИЕ А

Создание таблиц и ограничений

В листингах А.1 - А.2 представлен скрипт создания таблиц, в листингах А.3 - А.10 – 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_owner_id_check',
    'ALTER TABLE public.flat ADD CONSTRAINT flat_owner_id_check CHECK (owner_id
IS NOT NULL);'
);

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);'
);
```

Листинг А.6 – Создание ограничений. Часть 4

```
-- 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);'
);

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_tenant_id_check',
    'ALTER TABLE public.neighborhood ADD CONSTRAINT neighborhood_tenant_id_check
CHECK (tenant_id IS NOT NULL);'
);

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);'
);
```

Листинг А.7 – Создание ограничений. Часть 5

```
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);'
);

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_owner_id_check',
    'ALTER TABLE public.goods ADD CONSTRAINT goods_owner_id_check CHECK
(owner_id IS NOT NULL);'
);
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);'
);
```

Листинг А8 – Создание ограничений. Часть 6

```
-- 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);'
);

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);'
);
```

Листинг А.9 – Создание ограничений. Часть 7

```
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);'
);

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);'
);
```

Листинг А.10 – Создание ограничений. Часть 8

```
-- 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
$$
BEGIN
    DELETE FROM public.flat WHERE owner_id = old.id;
    DELETE FROM public.subscription_landlord WHERE landlord_id = old.id;
    RETURN old;
END;
$$
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 B.2 – Создание триггеров. Часть 2

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