|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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.

**ЗАДАНИЕ**

**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 5](#_Toc113411165)

[1 Аналитическая часть 7](#_Toc113411166)

[**1.1** **Постановка задачи** 7](#_Toc113411167)

[**1.1.1 Тип сервиса** 7](#_Toc113411168)

[**1.1.2 Ролевая модель** 8](#_Toc113411169)

[**1.1.3 Формализация данных** 11](#_Toc113411170)

[**1.2** **Модель данных** 11](#_Toc113411171)

[**1.3** **Существующие аналоги** 12](#_Toc113411172)

[**Вывод** 13](#_Toc113411173)

[2 Конструкторская часть 14](#_Toc113411174)

[**2.1 Проектирование базы данных** 14](#_Toc113411175)

[**2.2 Схемы триггеров** 18](#_Toc113411176)

[**Вывод** 22](#_Toc113411177)

[3 Технологическая часть 23](#_Toc113411178)

[**3.1 Выбор СУБД** 23](#_Toc113411179)

[**3.1.1 MySQL** 23](#_Toc113411180)

[**3.1.2 PostgreSQL** 23](#_Toc113411181)

[**3.1.3 Oracle Database** 24](#_Toc113411182)

[**Вывод** 24](#_Toc113411183)

[**3.2 Выбор инструментов разработки** 24](#_Toc113411184)

[**3.3 Детали реализации** 25](#_Toc113411185)

[**3.3.1 Разбиение на компоненты** 25](#_Toc113411186)

[**3.3.2 Создание таблиц, ограничений, ролей и триггеров** 26](#_Toc113411187)

[**3.4 Интерфейс приложения** 26](#_Toc113411188)

[**Вывод** 28](#_Toc113411189)

[4 Исследовательская часть 29](#_Toc113411190)

[**4.1 Примеры работы** 29](#_Toc113411191)

[**4.2 Цель эксперимента** 31](#_Toc113411192)

[**4.3 Технические характеристики** 32](#_Toc113411193)

[**4.4 Проведение эксперимента** 32](#_Toc113411194)

[**Вывод** 32](#_Toc113411195)

[ЗАКЛЮЧЕНИЕ 33](#_Toc113411196)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 34](#_Toc113411197)

[ПРИЛОЖЕНИЕ А 35](#_Toc113411198)

[ПРИЛОЖЕНИЕ Б 45](#_Toc113411199)

[ПРИЛОЖЕНИЕ В 47](#_Toc113411200)

# **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 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 сервис, позволяющий решить поставленную задачу и обеспечивающий доступ к базе данных;
6. провести сравнительный анализ времени выполнения различных запросов к базе данных с использованием индексов и без.

# **Аналитическая часть**

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

## **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.

### **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.

### **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.

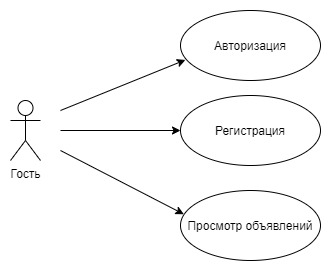


Рисунок 1.1 – Use-Case диаграмма для гостя

1. 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.

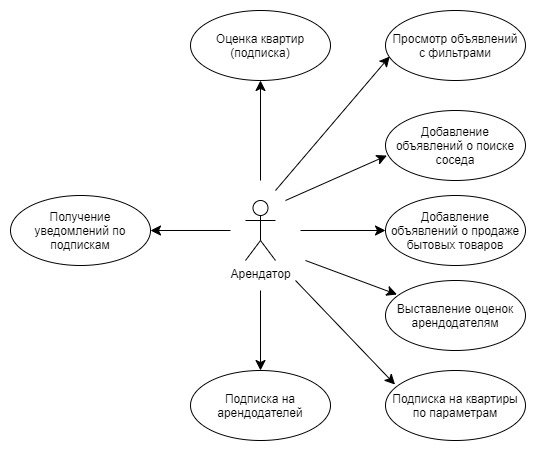


Рисунок 1.2 – Use-Case диаграмма для разрабатываемой системы

1. 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.

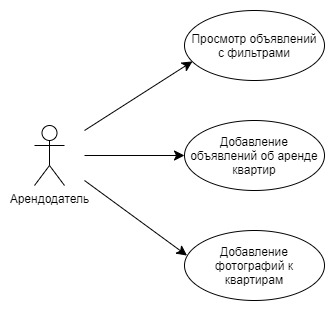


Рисунок 1.3 – Use-Case диаграмма для арендодателя

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

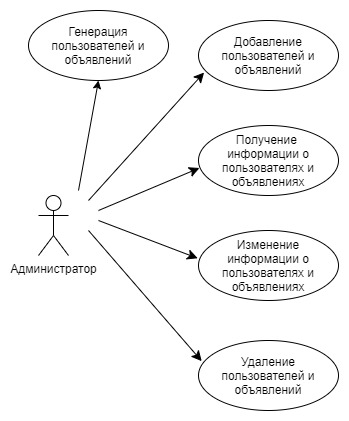


Рисунок 1.4 – Use-Case диаграмма для администратора

### **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.

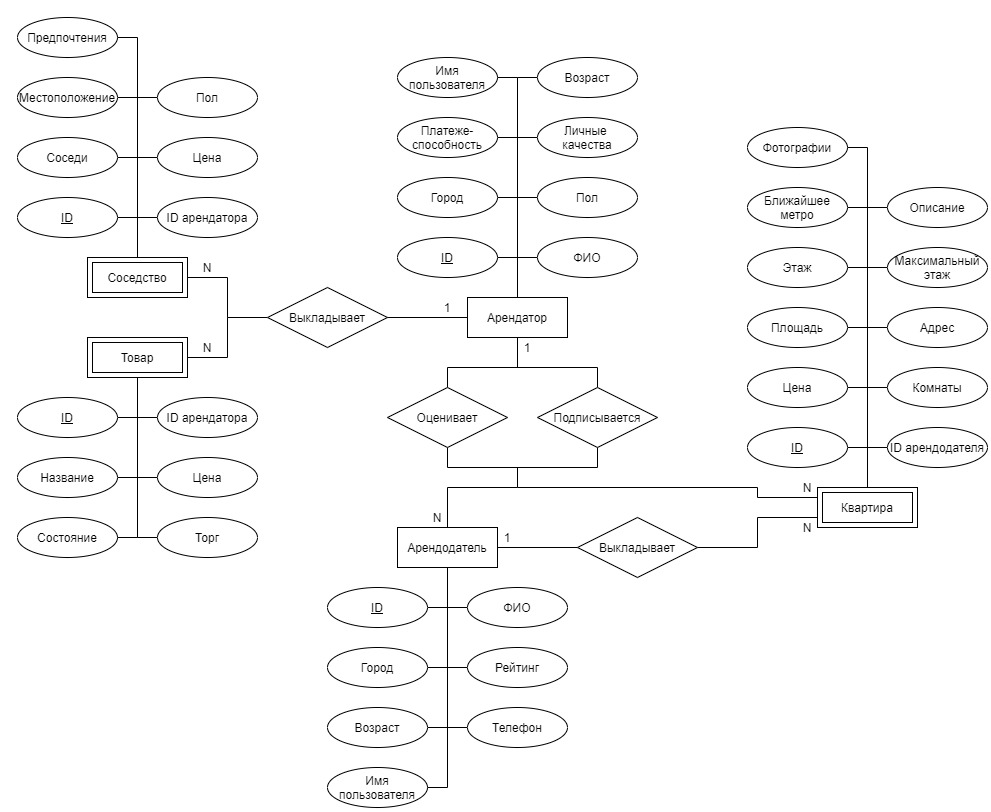


Рисунок 1.5 – ER-модель разрабатываемой БД в нотации Чена

## **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.

## **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.

# **Конструкторская часть**

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

## **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.

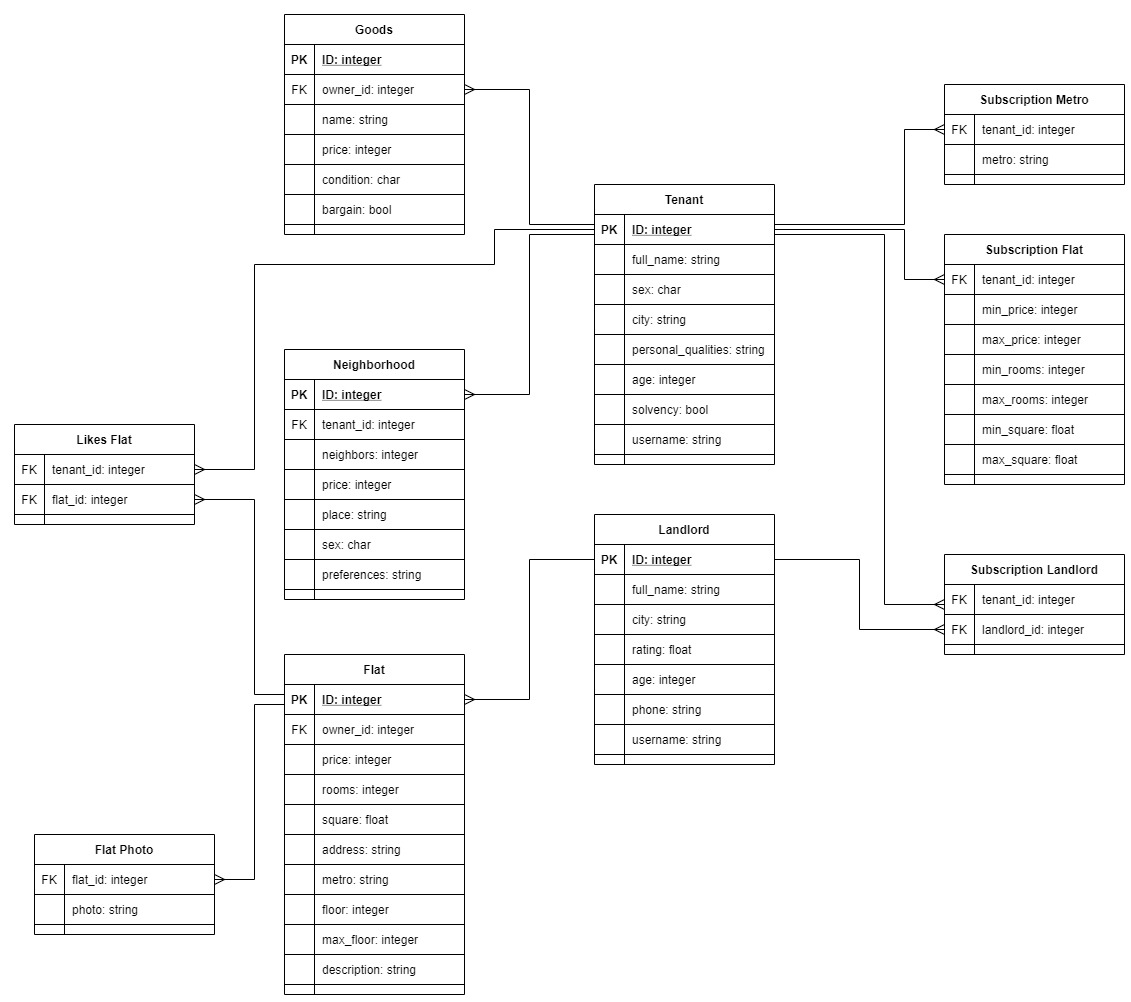


Рисунок 2.1 – Диаграмма базы данных

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.

## **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.

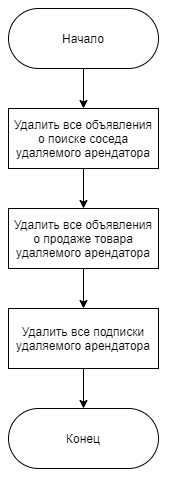


Рисунок 2.2 – Схема триггера delete\_tenant

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

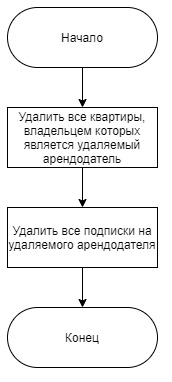


Рисунок 2.3 – Схема триггера delete\_landlord

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.

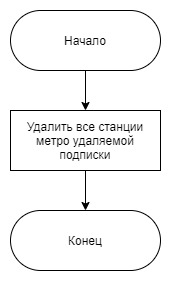


Рисунок 2.5 – Схема триггера delete\_subscription\_flat

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

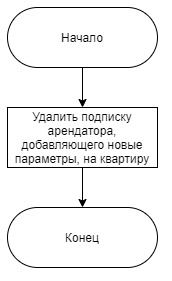


Рисунок 2.6 – Схема триггера insert\_subscription\_flat

## **Вывод**

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

# **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.

## **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.

### **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].

### **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].

### **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].

## **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].

## **Детали реализации**

### **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.

1. Доступ к данным:

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

### **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.

## **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.

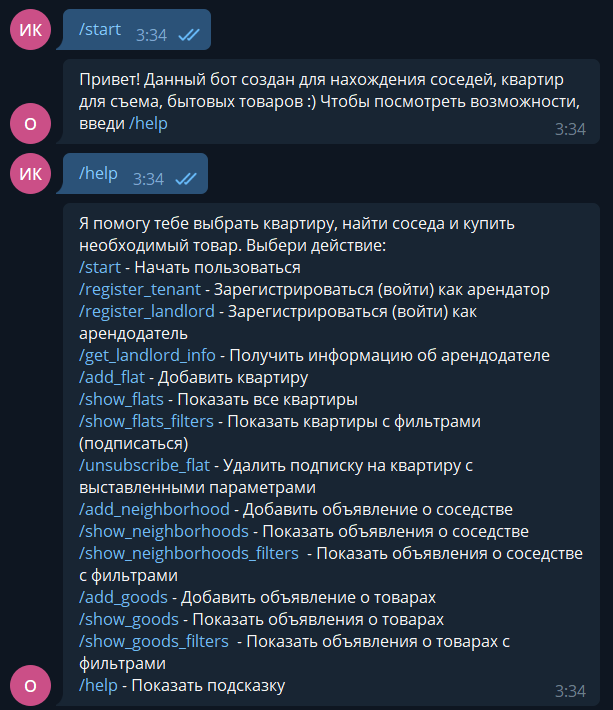


Рисунок 3.1 – Вызов приветственных в боте

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.

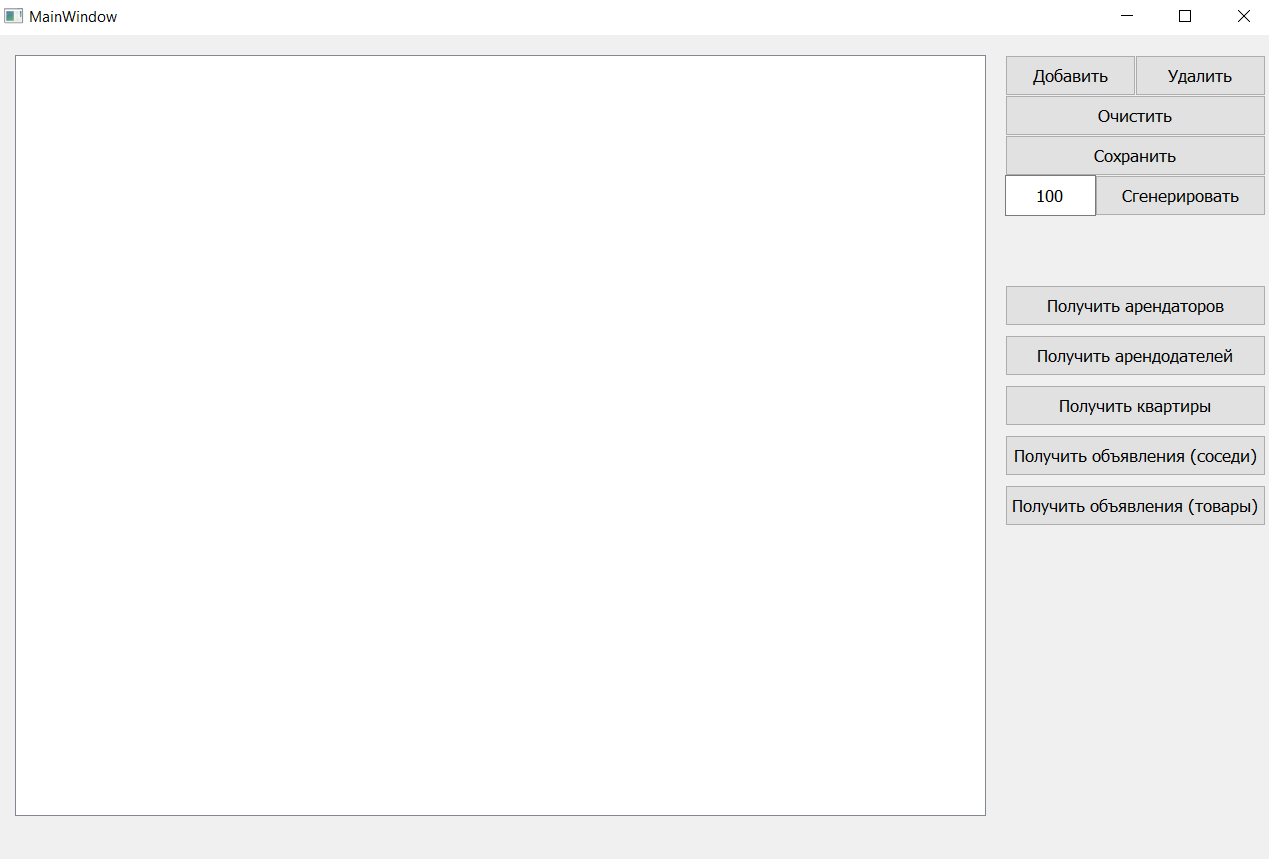


Рисунок 3.2 – Панель администратора

## **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.

# **Исследовательская часть**

В данном разделе представлены примеры работы разработанного приложения, проведен сравнительный анализ времени выполнения запросов к базе данных с использованием индексов и без.

## **Примеры работы**

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

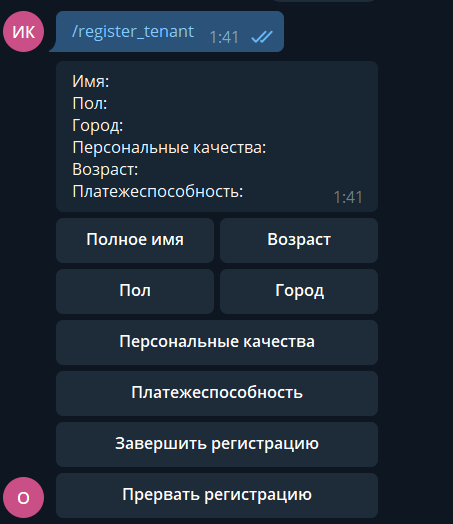


Рисунок 4.1 – Форма регистрации для арендатора в боте

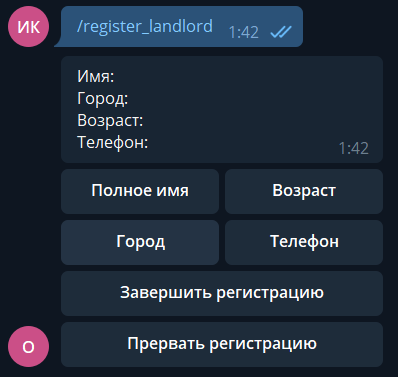


Рисунок 4.2 – Форма регистрации для арендодателя в боте

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



Рисунок 4.3 – Просмотр квартир в боте

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

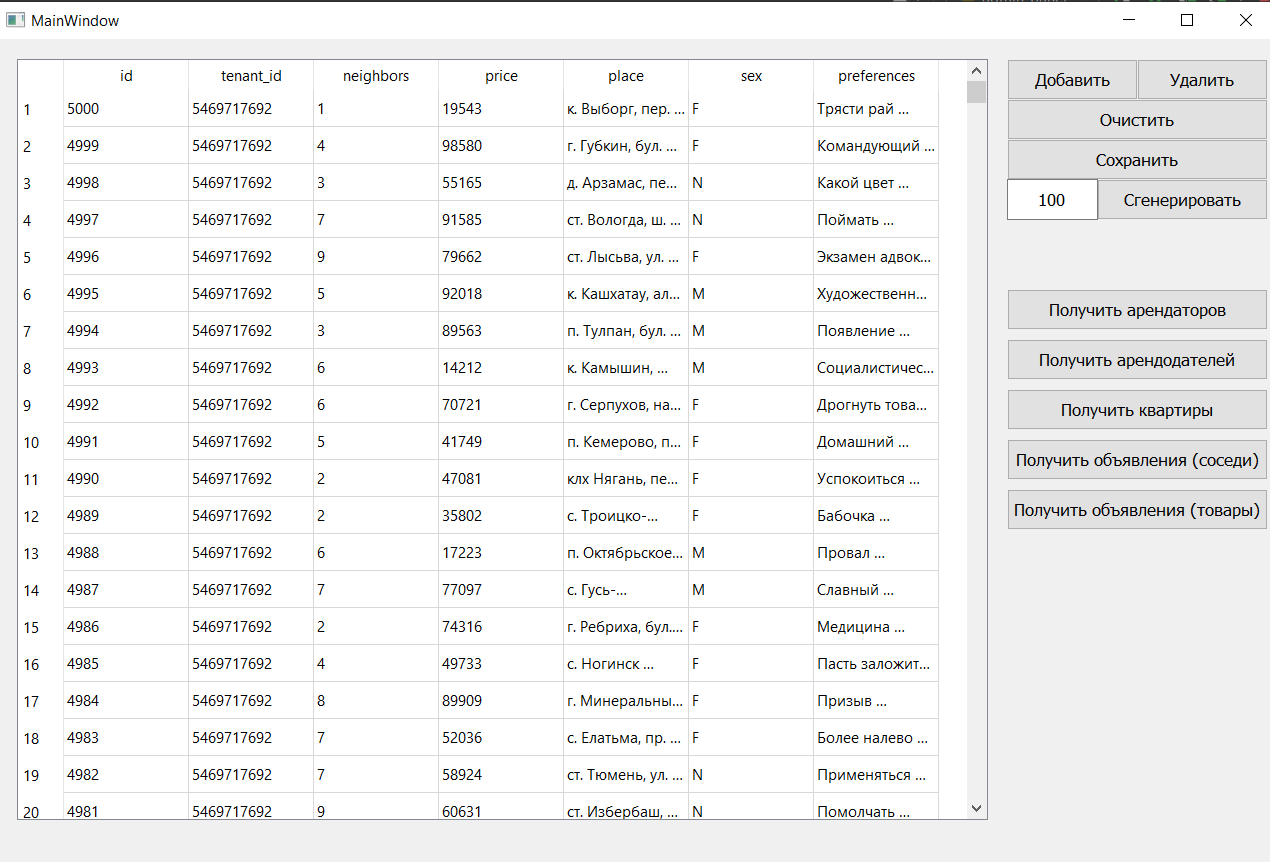


Рисунок 4.4 – Таблица в панели администратора, содержащая информацию о выложенных квартирах

## **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 вида 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.

## **Технические характеристики**

Технические характеристики устройства, на котором проводится эксперимент:

* операционная система: Windows 10;
* оперативная память: 8 Гб;
* процессор: AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx 2.10 GHz.

## **Проведение эксперимента**

Во время проведения эксперимента замерялось время выполнения каждого из запросов 100 раз и находилось среднее арифметическое. В таблице 4.1 показан результат в миллисекундах.

|  |  |  |
| --- | --- | --- |
| Запрос | Время выполнения без индексов, мс | Время выполнения с индексами, мс |
| Поиск квартир по владельцу | 2.37795 | 1.84497 |
| Поиск объявления по поиску соседа по арендатору | 1.71215 | 0.89556 |
| Поиск товара по владельцу | 1.31333 | 1.00274 |

## **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 можно сделать вывод, что использование индексов позволяет ускорить работу в среднем на 30%.

# **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. реализован сервис, позволяющий решить поставленную задачу и обеспечивающий доступ к базе данных;
6. проведен сравнительный анализ времени выполнения различных запросов к базе данных с использованием индексов и без.

# **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 В.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();