МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ**

**УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

Институт информационных технологий

Кафедра/департамент «Информационные системы»

**Пояснительная записка**

к расчетно-графической работе

по дисциплине «Управление данными»

на тему “База данных медицинской страховой компании”

Выполнил: обучающийся 3 курса

группы: ИС/б-20-2-о

Направления подготовки

09.03.02

Информационные системы и технологии

Воронухин А.А.

« \_\_« \_\_\_\_\_\_\_20 22 г.

Научный руководитель:

«\_\_ « \_\_\_\_\_\_\_\_20 22 г.

Оценка\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

« \_\_«\_\_\_\_\_\_\_\_\_ 20 22 г.

Севастополь 2022

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc122068238)

[1. АНАЛИТИЧЕСКАЯ ЧАСТЬ 4](#_Toc122068239)

[1.1. Анализ предметной области (концептуальное моделирование) 4](#_Toc122068240)

[1.2. Постановка задачи 5](#_Toc122068241)

[2. РАЗРАБОТКА ЛОГИЧЕСКОЙ МОДЕЛИ БАЗЫ ДАННЫХ 6](#_Toc122068242)

[2.1. Построение простой и сложной сетевых моделей 6](#_Toc122068243)

[2.2. Построение диаграммы «сущность-связь» в нотации П.Чена 7](#_Toc122068244)

[2.3. Построение модели основанной на ключах (Key Based model, KB) 8](#_Toc122068245)

[2.4. Построение полной атрибутивной модели в нотации IDEF1X 9](#_Toc122068246)

[3. РАЗРАБОТКА ФИЗИЧЕСКОЙ МОДЕЛИ БАЗЫ ДАННЫХ 10](#_Toc122068247)

[3.1. Выбор аппаратной и программной платформы для реализации БД 10](#_Toc122068248)

[3.2. Реализация базы данных 10](#_Toc122068249)

[3.3. Тестирование базы данных 10](#_Toc122068250)

[4. РАЗРАБОТКА КЛИЕНТСКОГО ПРИЛОЖЕНИЯ 19](#_Toc122068251)

[4.1. Обоснование выбора языка программирования 19](#_Toc122068252)

[4.2. Разработка интерфейса пользователя 19](#_Toc122068253)

[4.3. Алгоритм работы модулей 21](#_Toc122068254)

[4.4. Тестирование работы приложения 21](#_Toc122068255)

[ЗАКЛЮЧЕНИЕ 24](#_Toc122068256)

# **ВВЕДЕНИЕ**

В рамках настоящей работы ведется разработка мобильного приложения для работы с базой данных на тему «База данных медицинской страховой компании» на основании документа – техническое задание и в рамках организации – Севастопольский государственный университет. Дата выдачи задания: 17.09.2022.

Целью расчетно-графической работы является систематизация, закрепление и углубление знаний в области управления данными на SQL подобных языках и их совершенствование путем применения при разработке простого мобильного приложения.

Для достижения цели на разных этапах курсового проектирования должны быть решены следующие задачи:

* анализ предметной области;
* разработка логической модели
* реализация базы данных;
* разработка клиентского приложения.

В качестве задания был выбран вариант 21 – база данных туристической базы

1. **АНАЛИТИЧЕСКАЯ ЧАСТЬ**
   1. **Анализ предметной области (концептуальное моделирование)**

В ходе анализа предметной области с помощью документации, расположенной в открытом доступе, реальных прототипов приложений и информации специалистов были выявлены основные связи и объекты рассматриваемой базы данных. Таким образом, были выделены следующие основные объекты предметной области: туристическая база, коттедж или же другие жилые помещения, выделяемые постояльцам, клиент и бронь, которую может осуществить клиент для отдельного коттеджа на определенный промежуток времени. Сущность «Коттедж» должна содержать в себе поле с ID турбазы, к которой данный коттедж принадлежит и данные о классе обслуживания. Сущность «Бронь» должна содержать информацию о пользователе, осуществляющим бронирование, и о коттедже, который бронируется.

Коттедж обладает следующими атрибутами: цена за единицу времени(сутки или ночь), свой номер внутри турбазы, количество спальных мест, используемых для размещения постояльцев, классом обслуживания и оснащения. Бронь обладает следующими атрибутами: итоговая цена, формируемая из цены за единицу времени для коттеджа, дополнительной стоимости обслуживания, задаваемой классом обслуживания и временем пребывания; дата заселения, дата выселения. Атрибуты клиента в свою очередь, как правило, могут быть ограничены паспортными данными (серия и номер паспорта, ФИО) и контактными данными (телефон, эл. почта и другие). Основной задачей, ставящийся перед подобными системами, обычно является упрощение подсчетов, контроля заселения и выселения, бронирования клиентами коттеджей, хранение данных о бронировании для упрощения составления учетов.

* 1. **Постановка задачи**

Основные объекты предметной области, информация о которых должна храниться в базе данных:

* Клиент
* Бронь
* Коттедж
* Турбаза

База данных доступна для рядовых пользователей, администраторов и управляющих турбазами.

В ходе исследования связей между вышеуказанными объектами, были выявлены следующие связи: связь между клиентом и бронью, связь между клиентом и комнатой, связь между бронью и коттеджами, связь между отдельными коттеджами и туристической базой. Между клиентом и коттеджем существует связь многие ко многим, которая заключается в том, что множество клиент может снять множество коттеджей и один коттедж может быть снят множеством клиентов. Связь между клиентом и бронью заключается в том, что один клиент может иметь множество броней, что выражает связь один ко многим. Бронь, в свою очередь, также имеет связь с коттеджем в виде, что на один коттедж может быть множество броней. Связь между турбазой и коттеджем является связью один ко многим в виду того, что турбаза включает в себя множество коттеджей. Помимо связей, были также выделены основные атрибуты каждого объекта. Таким образом, турбаза обладает следующим рядом атрибутов: местоположение(адрес), количество гостевых помещений, название.

Представители группы пользователей должны обладать функционалом просмотра данных о турбазах и коттеджах, принадлежащих данным турбазам.

Представители группы управляющих турбазами обладают функционалом регистрации пользователей, добавления, редактирования и удаления информации о коттеджах, классах обслуживания, бронях и пользователях.

Представители группы администраторов обладают полным контролем над данными в базе.

# **РАЗРАБОТКА ЛОГИЧЕСКОЙ МОДЕЛИ БАЗЫ ДАННЫХ**

## **Построение простой и сложной сетевых моделей**

В рамках схематичного отображения связей между объектами базы данных была построена диаграмма простой сетевой модели. Данная диаграмма приведена на рисунке 1.

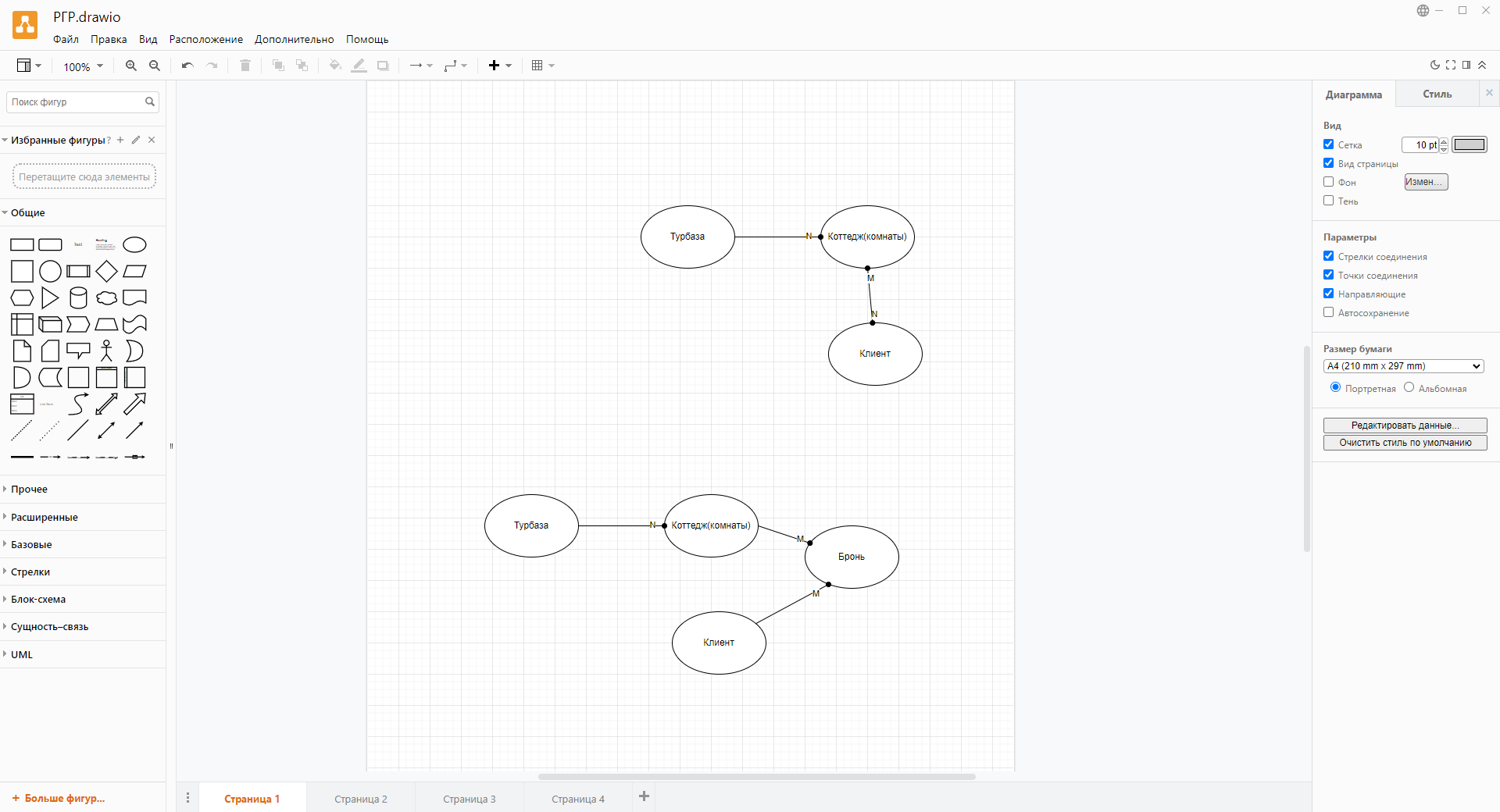


Рисунок 1 – Диаграмма простой сетевой модели

Как видно по диаграмме простой сетевой модели, между сущностями «Коттедж» и «Клиент» существует связь многие ко многим. Данная проблема была устранена внедрением дополнительного объекта «Бронь». В результате была получена сложная сетевая модель, представленная на рисунке 2.

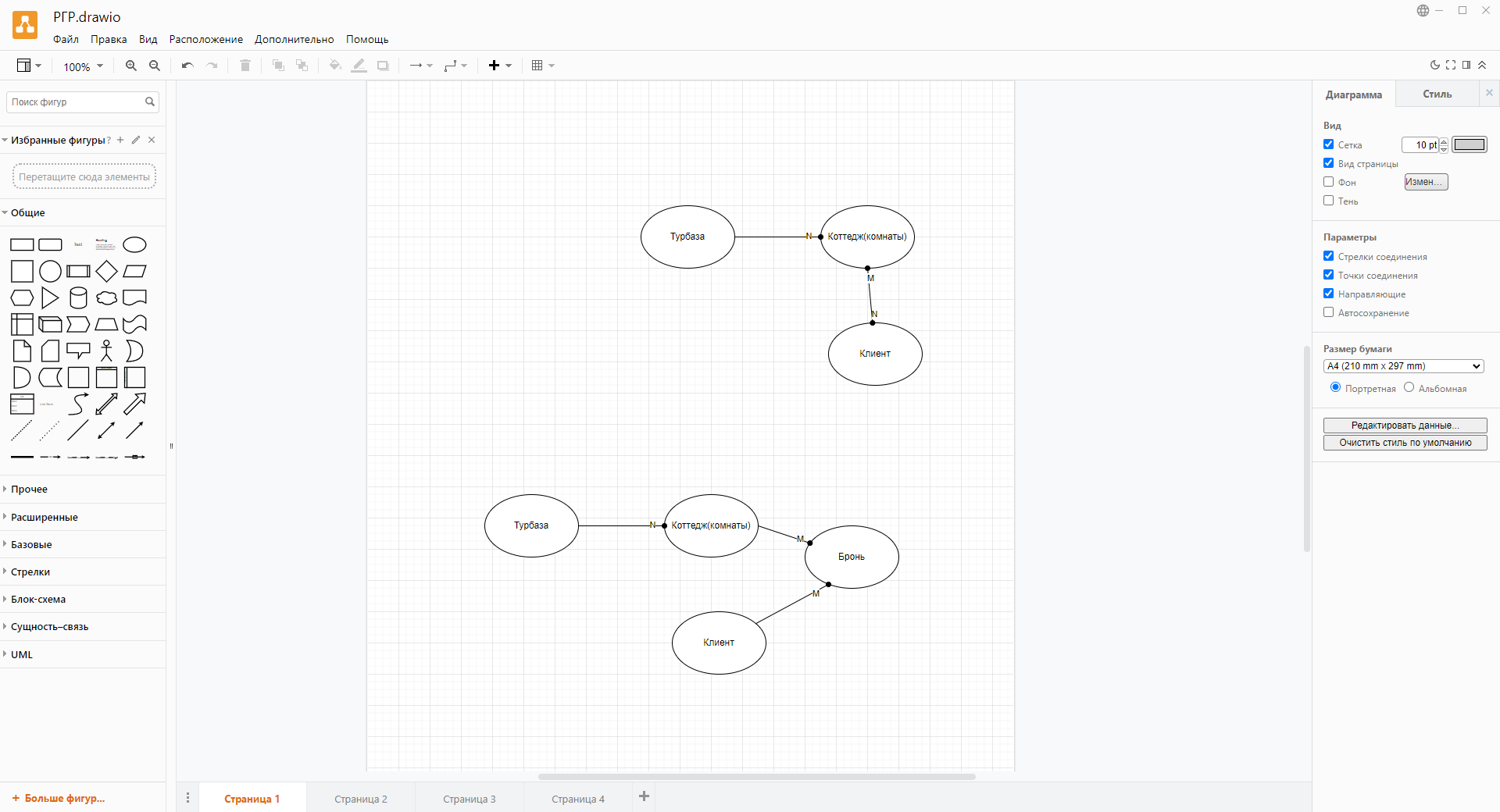


Рисунок 2 – Диаграмма сложной сетевой модели

Диаграмма сложной сетевой модели в дальнейшем будет использована для построения диаграммы сущность-связь

* 1. **Построение диаграммы «сущность-связь» в нотации П.Чена**

По полученной в предыдущем пункте сложной сетевой модели и данным, полученным при анализе предметной области, а также в процессе установления связи между сущностями была составлена диаграмма сущность-связь, представленная на рисунке 3.

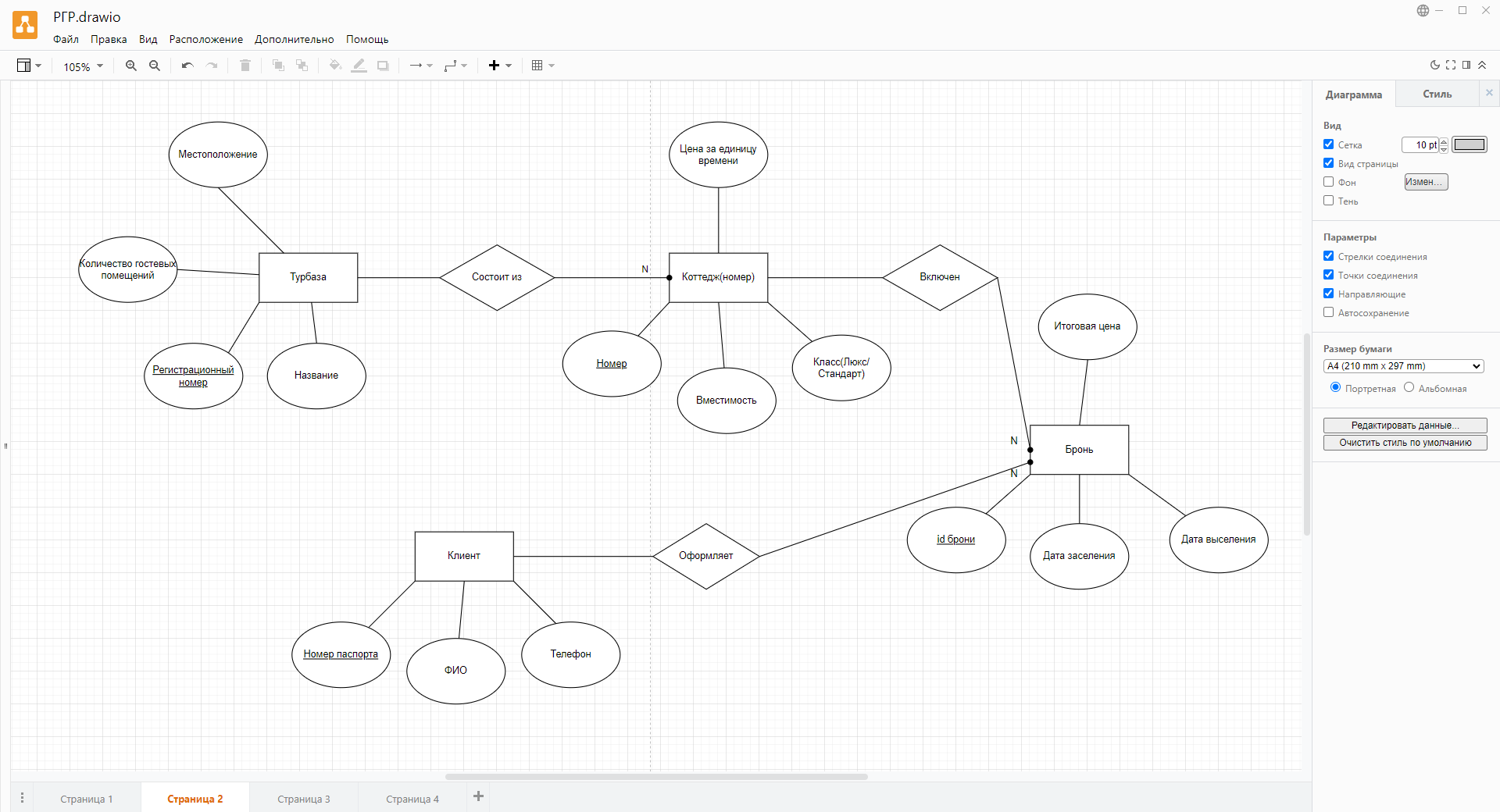


Рисунок 3 – Диаграмма сущность-связь для заданной предметной области

Как видно на рисунке 3, для сущности «Турбаза» были добавлены атрибуты «местоположение», «название», «регистрационный номер», «количество жилых помещений». В результате анализа предметной области, в роли первичного ключа был выбран атрибут «регистрационный номер».

Для сущности «Коттедж» были выделены атрибуты «цена за единицу времени», «класс обслуживания», «вместимость» и «номер», который был выбран в качестве первичного ключа в виду того, что данный атрибут может однозначно идентифицировать коттедж.

Для сущности «Бронь» были выделены атрибуты «дата заселения», «дата выселения», «id брони». Поле «id брони» было выбрано как первичный ключ.

Для сущности «Клиент» были выделены атрибуты «ФИО», «телефон» и «номер паспорта». Атрибут «номер паспорта» был выбран как первичный ключ, способный уникально определить клиента.

Полученная диаграмма в дальнейшем будет использована для построения модели, основанной на ключах. Все связи между сущностями наглядно отражены на рисунке 3.

* 1. **Построение модели основанной на ключах (Key Based model, KB)**

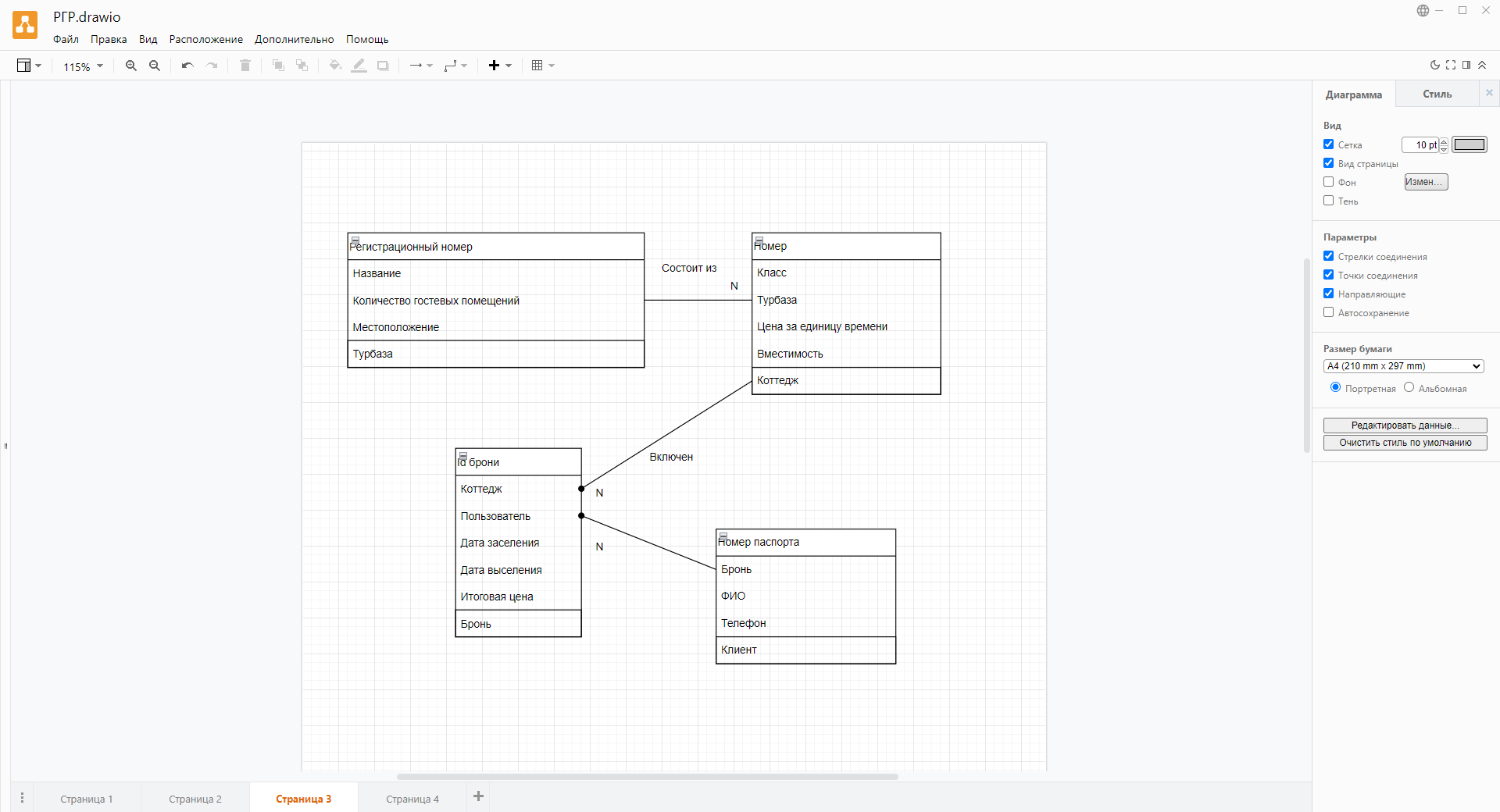
На основе полученной в предыдущем пункте диаграммы, была построена модель, основанная на ключах. Данная модель приведена на рисунке 4.

Рисунок 4 – Модель, основанная на ключах

Данная диаграмма может в дальнейшем быть использована для построения полной атрибутивной модели в ходе нормализации.

* 1. **Построение полной атрибутивной модели в нотации IDEF1X**

Модель, полученная в пункте 2.3, была подвергнута нормализации с целью доведения до нотации IDEF1X. В результате была получена полная атрибутивная модель, приведенная на рисунке 5. Данная модель находится в первой нормальной форме так как все атрибуты всех сущностей удовлетворяют требованию атомарности. Поля местоположение (адрес), ФИО и номер паспорта было решено не разбивать на дополнительные атрибуты в связи с тем, что предметная область не предполагает активного использования данных атрибутов. Отношение находится во второй нормальной форме, так как каждый неключевой атрибут полностью зависит от первичного ключа. Для приведения отношения к третьей нормальной форме была упразднена транзитивная зависимость дополнительной стоимости обслуживания коттеджа от его класса. Для разрешения данной транзитивной зависимости была введена дополнительная сущность «Класс обслуживания». В результате было получено отношение, находящееся в третьей нормальной форме, так как отношение не имеет транзитивных зависимостей и, более того, отношения имеют лишь один возможный первичный ключ, который и является их первичным ключом, что свидетельствует о состоятельности Изображение выглядит как текст, карта, внутренний

Автоматически созданное описаниеопределения третьей нормальной формы.

Рисунок 5 – Полная атрибутивная модель

Полученная полная атрибутивная модель в дальнейшем будет использована для физического проектирования базы данных

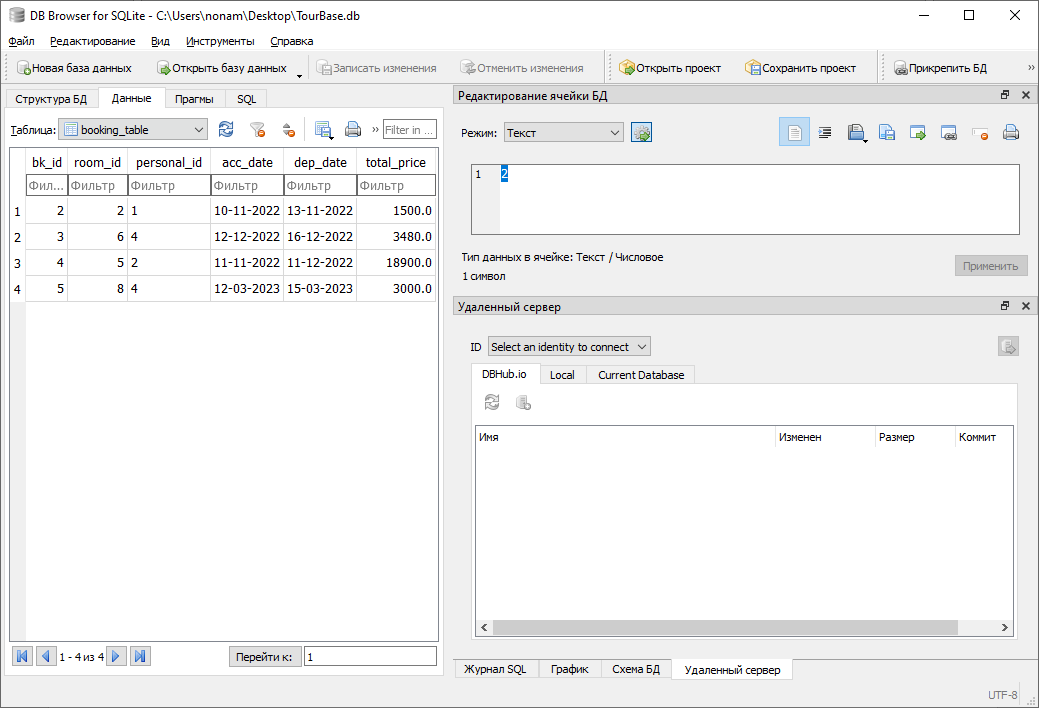
1. **РАЗРАБОТКА ФИЗИЧЕСКОЙ МОДЕЛИ БАЗЫ ДАННЫХ**
   1. **Выбор аппаратной и программной платформы для реализации БД**

Для реализации физической модели базы данных была выбрана платформа SQLite, поскольку она позволяет взаимодействовать с базой данных, оперируя командами SQL, и обладает удобным и интуитивно понятным интерфейсом. Также SQLite можно использовать при написании пользовательской программы, поскольку Android Studio позволяет взаимодействовать с выбранной СУБД.

* 1. **Реализация базы данных**

Для полей «Название» и «Местоположение» сущности «Турбаза» был выбран символьный тип данных, поскольку они хранят строки. Аналогично с полями «ФИО», «Телефон» и «Название класса» в сущностях «Клиент» и «Класс» соответственно. Поля «Дата заселения» и «Дата выселения» имеют символьный тип данных в виду того, что в СУБД SQLite не существует отдельного типа Date и вместо него разработчики предлагают использовать тип TEXT. Полям «Цена за единицу времени», «Доп. стоимость» и «Итоговая стоимость» сущностей «Коттедж», «Класс» и «Бронь» был задан тип Real, так как над данными полями возможны сложные арифметические операции и данные поля отражают меру стоимости. Остальным полям БД был дан тип Integer в связи с тем, что они либо являются идентификаторами, либо отражают целочисленные меры. Полям «Регистрационный номер», «Номер», «ИД Класса», «ИД Брони» был приписан модификатор Primary key и autoincrement для автоматической генерации ключей.

* 1. **Тестирование базы данных**

Для проведения тестирования базы данных, база данных была заполнена осмысленными значениями (рисунки 6–10).

Изображение выглядит как текст, снимок экрана, внутренний, ноутбук

Автоматически созданное описаниеРисунок 6 – Значения внутри таблицы booking\_table

Рисунок 7 – Значения внутри таблицы classes\_table

Изображение выглядит как текст, снимок экрана, внутренний, ноутбук

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, внутренний

Автоматически созданное описаниеРисунок 8 – Значения внутри таблицы client\_table

Рисунок 9 – Значения внутри таблицы cottage

Изображение выглядит как текст

Автоматически созданное описаниеРисунок 10 – Значения в таблице tour\_hostels

Для проведения тестирования корректности работы составленной БД, были написаны 15 разноплановых тестовых запроса. Их коды и результаты выполнения приведены ниже.

Был составлен запрос для поиска в таблице cottage записи с reg\_id=4 (рисунок 11).

Листинг 1 – Код запроса

Изображение выглядит как текст, снимок экрана, компьютер, внутренний

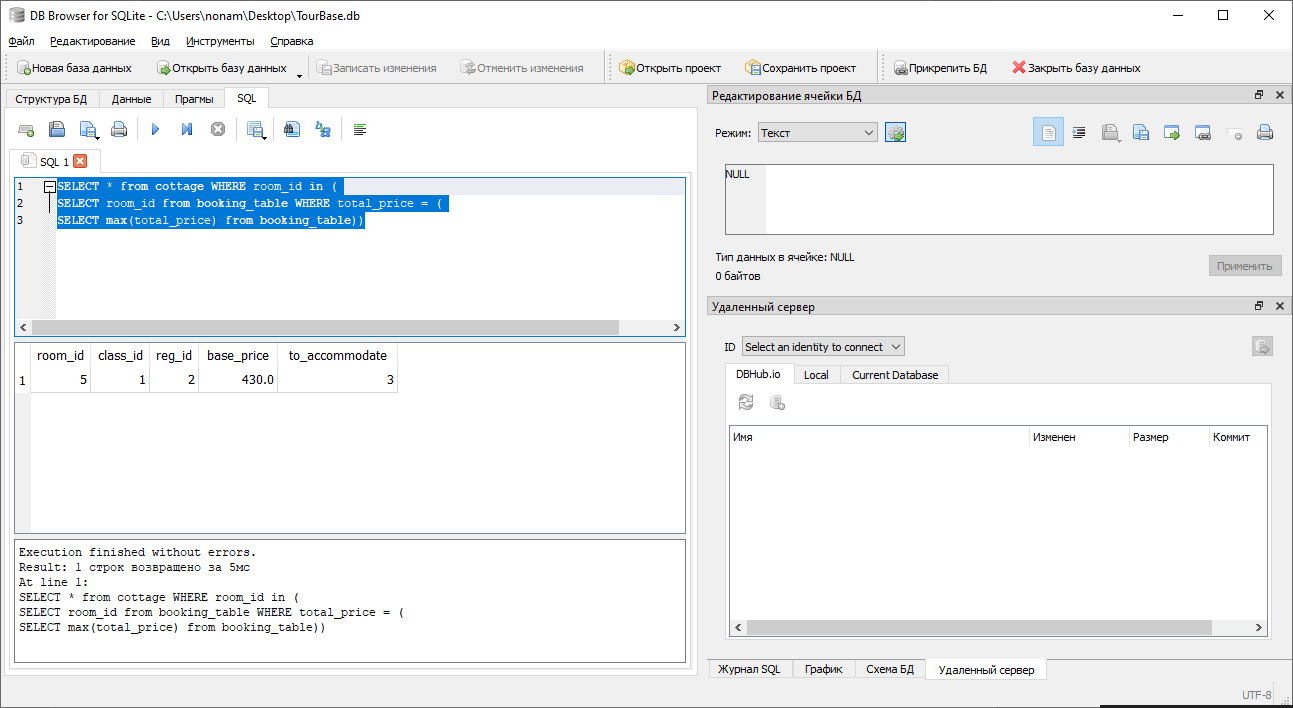
Автоматически созданное описаниеSELECT \* FROM cottage WHERE reg\_id = 4;

Рисунок 11 – Результат выполнения запроса по поиску записи с reg\_id = 4

Был соcтавлен запрос для поиска клиента с именем alex (рисунок 12)

Листинг 2 – Код запроса

SELECT \* from client\_table WHERE client\_name = 'alex'

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 12 – Результат выполнения запроса с поиском клиента по имени

Был составлен запрос для поиска броней для клиента с именем «Савелий» (рисунок 13).

Листинг 3 – Код запроса

SELECT \* FROM booking\_table WHERE personal\_id in (

Изображение выглядит как текст

Автоматически созданное описаниеSELECT personal\_id from client\_table WHERE client\_name = 'Савелий')

Рисунок 13 – Результат выполнения запроса по поиску броней

Был написан запрос с поиском коттеджа который забронирован с максимальной суммой среди всех бронирований (рисунок 14)

Листинг 4 – Код запроса

SELECT \* from cottage WHERE room\_id in (

SELECT room\_id from booking\_table WHERE total\_price = (

SELECT max(total\_price) from booking\_table))

Рисунок 14 – Результат выполнения запроса по поиску коттеджа с максимальной величиной total\_price

Был написан запрос для поиска среднего значения total\_price (рисунок 15)

Листинг 5 – Код запроса

SELECT avg(total\_price) FROM booking\_table

Изображение выглядит как текст, снимок экрана, внутренний

Автоматически созданное описание

Рисунок 15 – Результат поиска среднего значения поля total\_price

Был написан запрос с поиском среднего значения total\_prce среди коттеджей с премиумным классом (рисунок 16).

Листинг 6 – Кол запроса

SELECT avg(total\_price) FROM booking\_table WHERE room\_id in (

SELECT room\_id from cottage where class\_id

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 16 – Результат поиска среднего total\_price для коттеджей с премиальным классом

Был написан запрос с объединением таблиц booking\_table и client\_table для поиска клиента, заплатившего максимальную сумму (рисунок 17).

Листинг 7 – Код запроса

SELECT \* FROM client\_table join booking\_table

on client\_table.personal\_id=booking\_table.personal\_id WHERE total\_price = (

SELECT max(total\_price) FROM booking\_table)

Изображение выглядит как текст

Автоматически созданное описаниеРисунок 17 – Результат выполнения запроса с поиском клиента, забронировавшего на максимальную сумму

Был составлен запрос с поиском клиентов, не имеющих бронь (рисунок 18)

Листинг 8 – Код запроса

SELECT \* FROM client\_table WHERE personal\_id not in (

SELECT personal\_id FROM booking\_table)

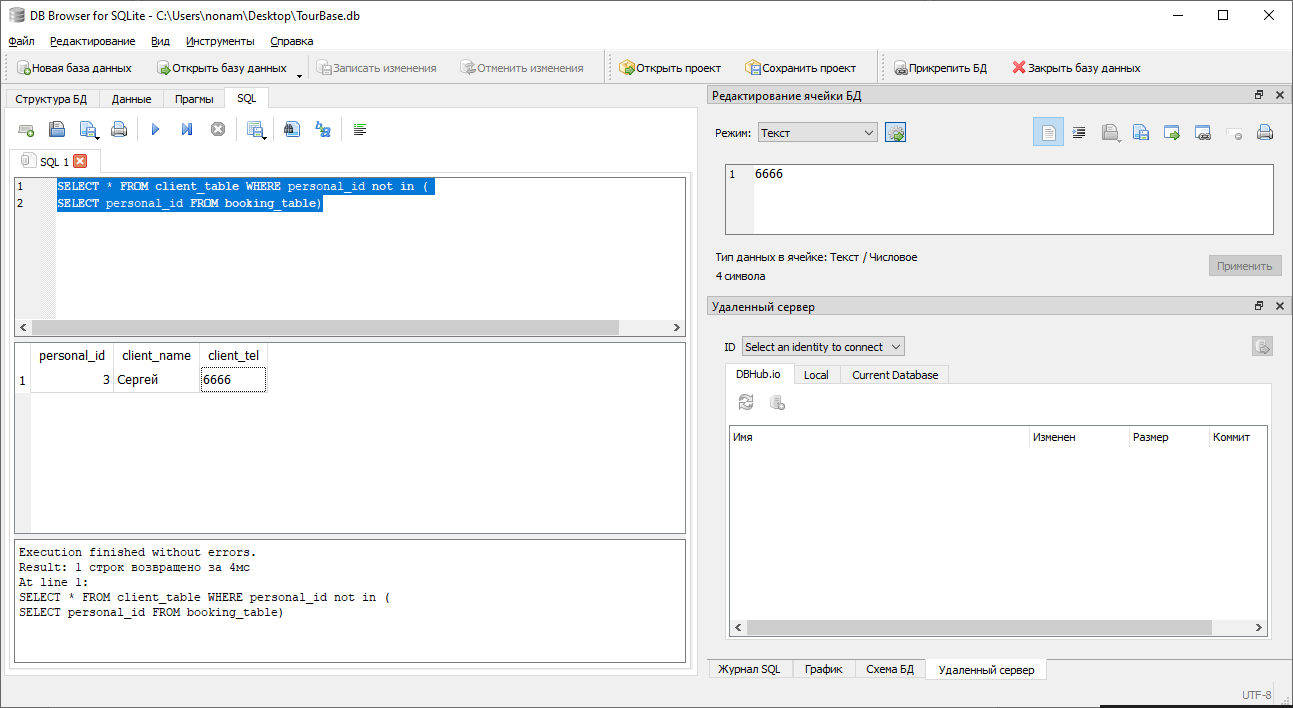


Рисунок 18 – Результат поиска клиента без забронированных коттеджей

Был составлен запрос по изменению personal\_id клиента, не имеющего брони (рисунок 19)

Листинг 9 – Код запроса

UPDATE client\_table SET personal\_id=0 WHERE personal\_id not in (

Изображение выглядит как текст

Автоматически созданное описаниеSELECT personal\_id FROM booking\_table)

Рисунок 19 – Результат изменения personal\_id человека, не имеющего брони

Был составлен запрос для поиска reg\_id турбаз, в которых будет или же отдых «Савелий» (рисунок 20)

Листинг 10 – Код запроса

SELECT reg\_id from cottage WHERE room\_id in (

SELECT room\_id FROM booking\_table WHERE personal\_id in (

Изображение выглядит как текст

Автоматически созданное описаниеSELECT personal\_id FROM client\_table WHERE client\_name = 'Савелий'))

Рисунок 20 – Результат выполнения запроса с поиском reg\_id турбаз, в которых отдыхал Савелий

Был составлен запрос для тестирования целостности данных при удалении клиента. На рисунке 21 представлены данные таблицы booking\_table до удаления пользователя с personal\_id=4. На рисунке 22 продемонстрирован результат каскадирования.

Листинг 11 – Код запроса

Изображение выглядит как текст

Автоматически созданное описаниеDELETE FROM client\_table WHERE personal\_id = 4

Рисунок 21 – Данные в таблице booking\_table до удаления

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 22 – Результат каскадирования данных

Было протестирована целостность данных при удалении записи с class\_id=1 из таблицы classes\_table. Данные в таблицах cottages и booking\_talbe до удаления продемострированы на рисунках 23 и 24. Данные после каскадирования продемонстрированы на рисунках 25 и 26

Листинг 12 – Код запроса

DELETE FROM classes\_table WHERE class\_id = 1

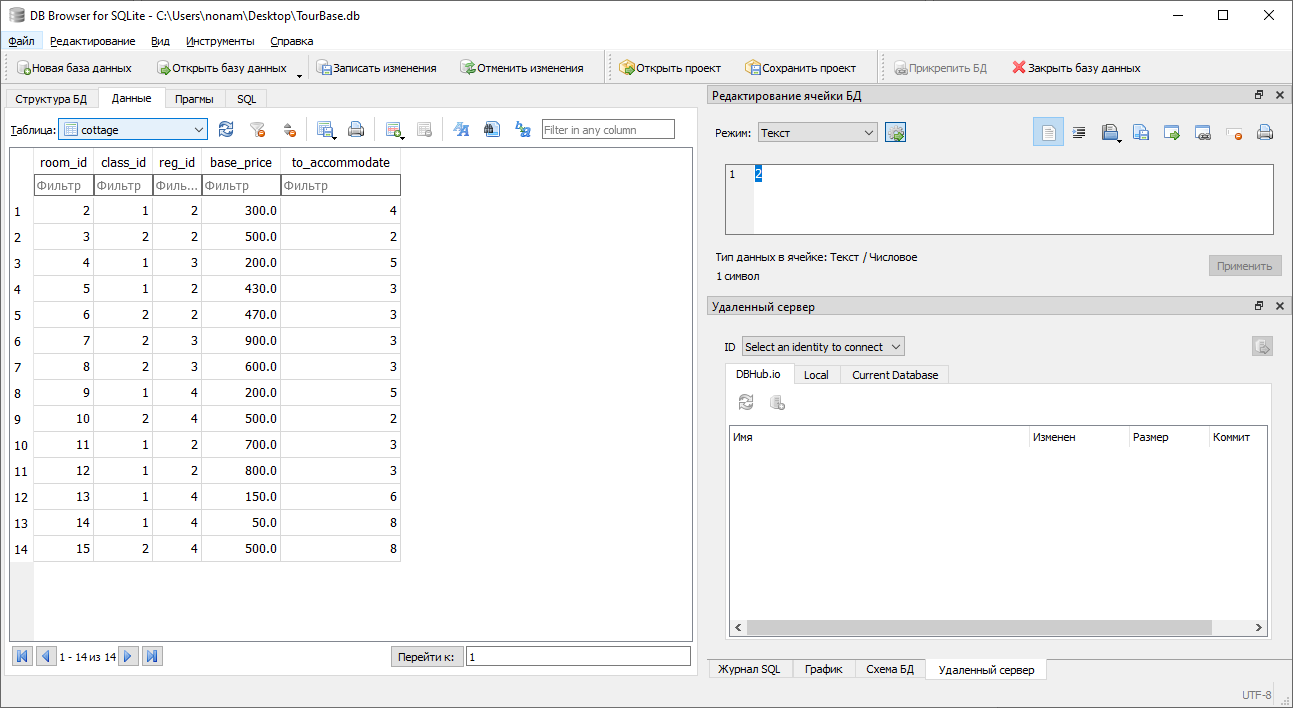


Рисунок 23 – Данные в таблице cottage до каскадирования

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 24 – Данные в таблице booking\_table до каскадирования

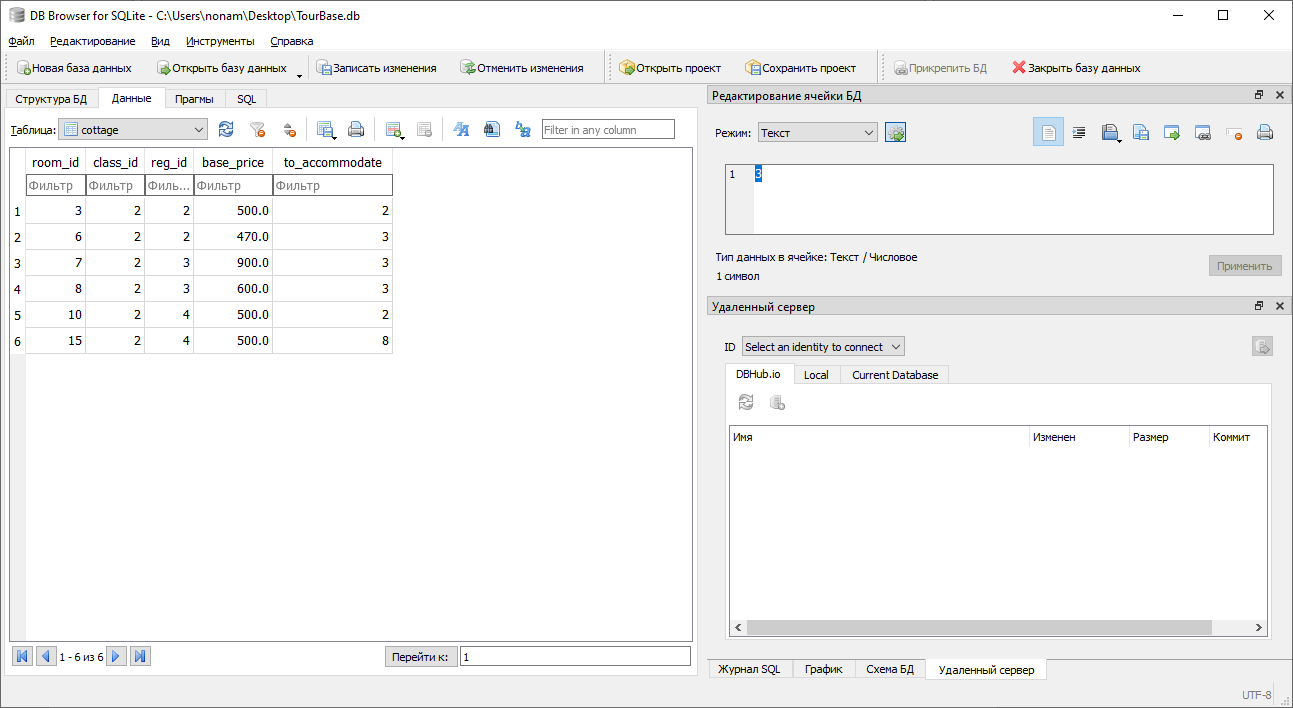


Рисунок 25 – Результат каскадирования в таблице cottage

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 26 – Результат каскадирования в табилце booking\_table

Было протестирована целостность данных при удалении турбазы с reg\_id=3. Данные до каскадирования показанны на рисунке 25. Данные после каскадирования продемонстрированы на рисунке 27.

Листинг 13 – Код запроса

DELETE FROM tour\_hostels WHERE reg\_id = 3

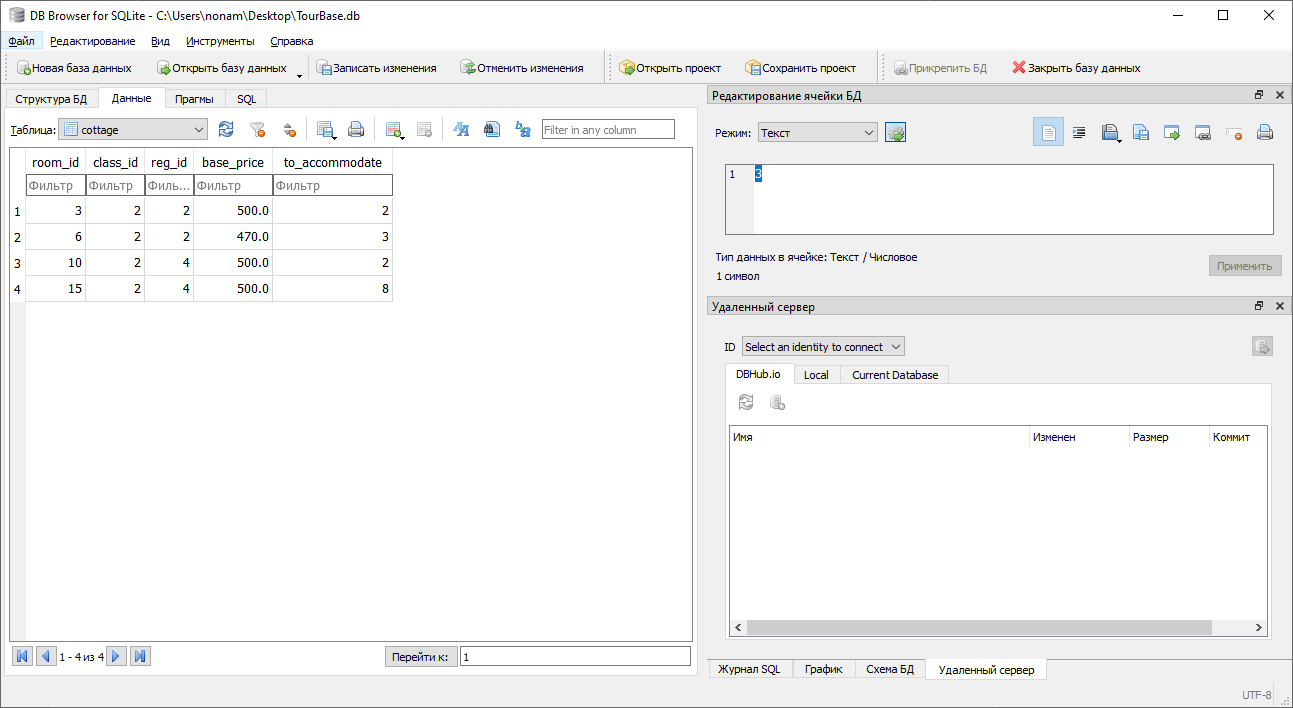


Рисунок 27 – Результаты каскадирования данных при удалении турбазы

По итогам проведенного тестирования, можно сделать вывод, что спроектированная база данных работает без ошибок на всех составленых запросах и сохраняет целостность данных при удалении данных. Спроектированная база данных в дальнейшем будет использована в клиентской программе.

1. **РАЗРАБОТКА КЛИЕНТСКОГО ПРИЛОЖЕНИЯ**
   1. **Обоснование выбора языка программирования**

В качества языка для разработки клиентской программы был выбран язык программирования Kotlin. Данный выбор может быть обоснован тем, что Kotlin сейчас является предпочтительным языком для разработки Android-приложений. Также Kotlin обладает большим спектром возможностей для быстрого написания программного обеспечения с помощью среды Android Studio.

* 1. **Разработка интерфейса пользователя**

Так как в качестве платформы для разработки клиентского приложения был выбран Android Kotlin, разработка интерфейса пользователя была осуществленна с помощью технологии Jetpack Navigation Component.

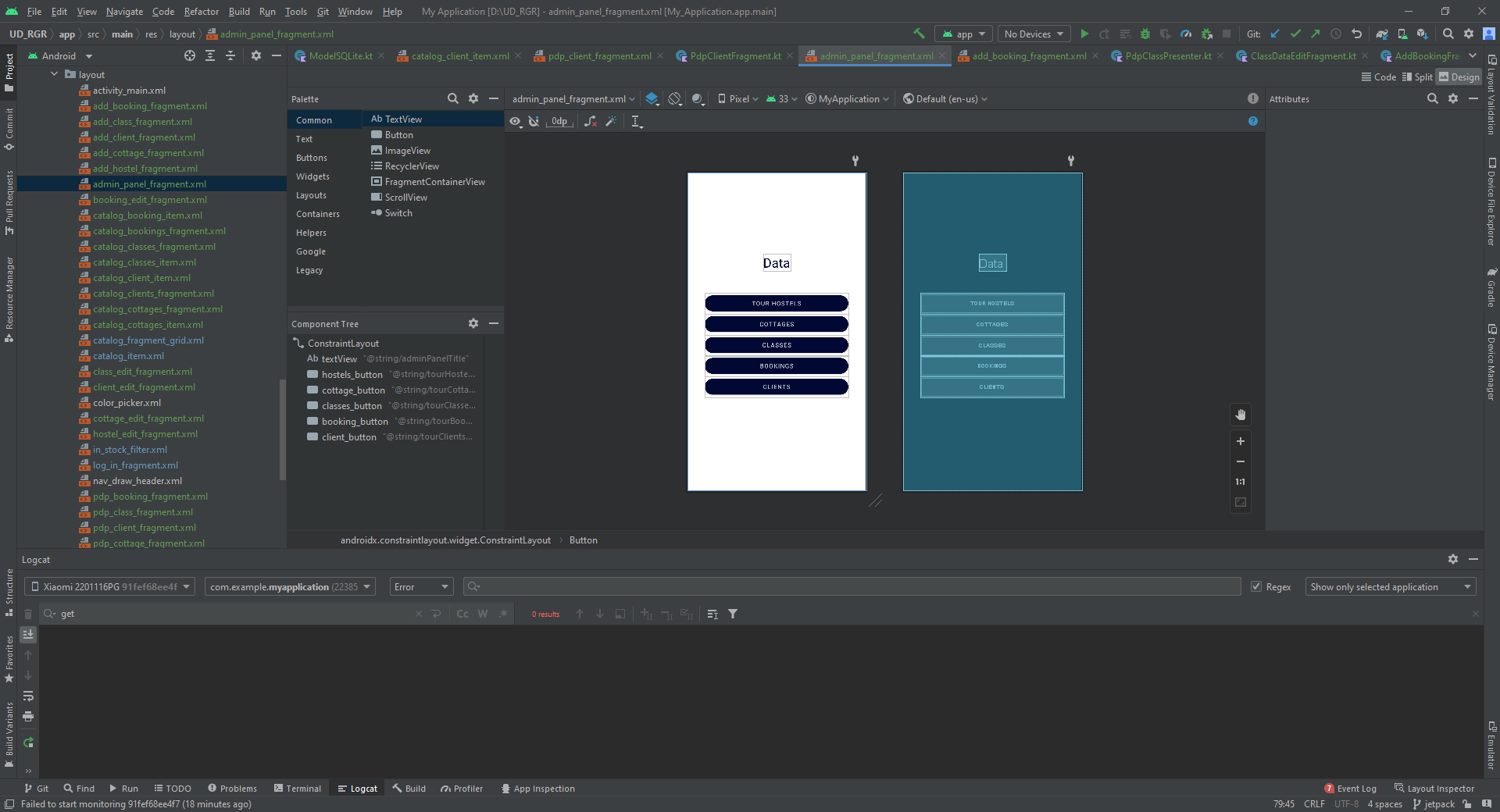
В качестве реализации разного уровня доступа к базе данных, было решено разделить интерфейс на подсистему пользователя и подсистему администратора. В подсистеме администратора, пользователю приложения предлагается выбрать данные какой из таблиц просмотреть в данный момент времени. Данный фрагмент представлен на рисунке 28.

Рисунок 28 – Страница администратора для выбора таблицы для просмотра

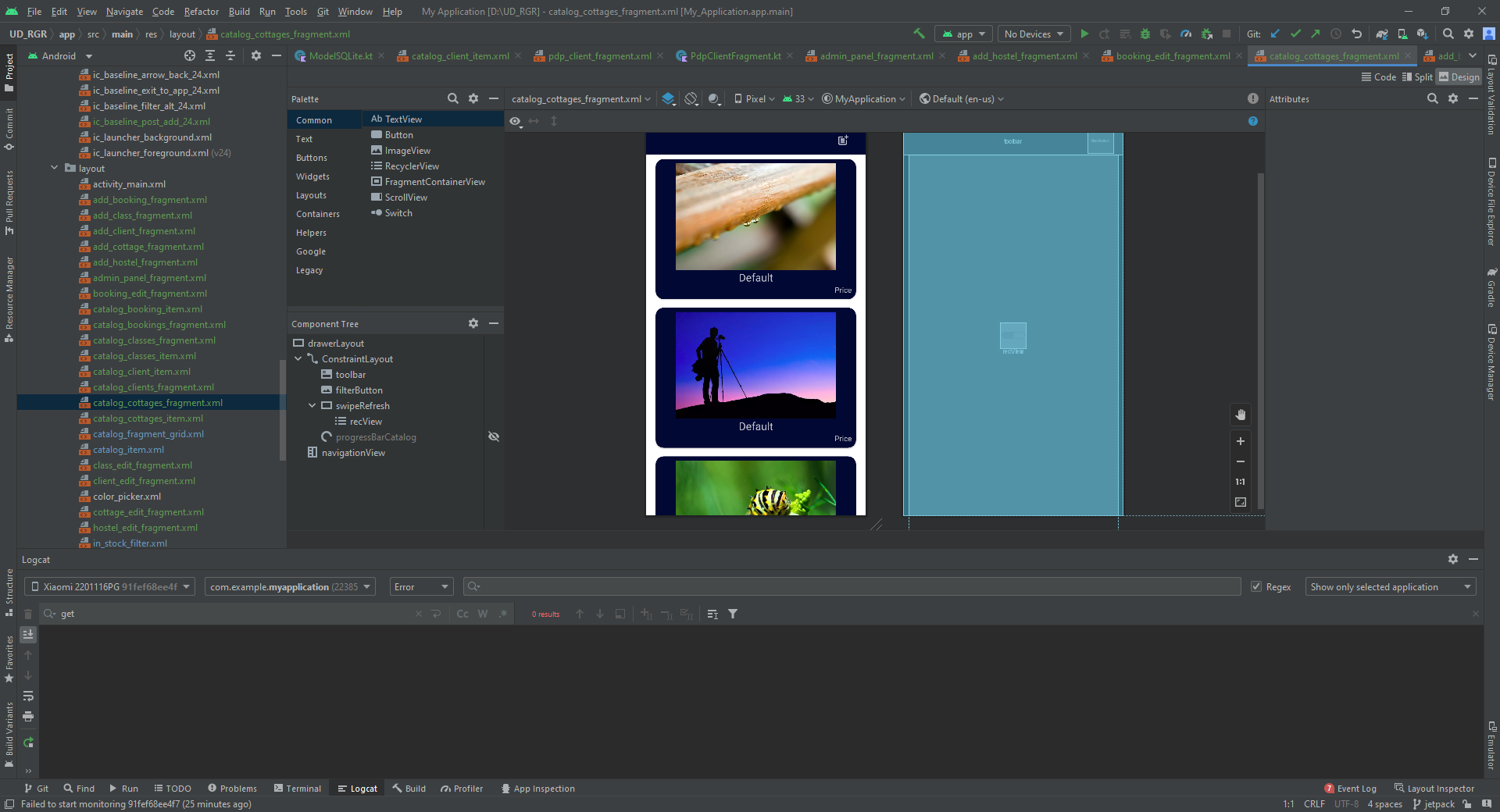
После выбора какого-либо пункта меню, администратора направляет на просмотр списка элементов. В данном окне, администратор может открыть подробные сведения об отдельной записи либо же добавить новые (рисунки 29 –30).

Рисунок 29 – Страница просмотра списка элементов

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 30 – Страница добавления элементов

Сценарий работы с приложением пользователя прав администратора отличается тем, что у обычного пользователя при входе в приложение сразу же произойдет переход на каталог турбаз, из которого он может либо перейти в каталог коттеджей для отдельной турбазы, либо же посмотреть более подробные сведения о турбазе. Элементы редактирования и добавления у обычного пользователя закрыты. На рисунке 31 приведен сценарий работы пользователя с Изображение выглядит как текст, снимок экрана, внутренний, черный

Автоматически созданное описаниеприложением.

Рисунок 31 – Nav\_graph возможных переходов пользователя по приложению

* 1. **Алгоритм работы модулей**

Алгоритм работы всех модулей программы строится на архитектуре MVP. При создании пользователем какого-либо события, обрабатываемого программой, управление действиями переходит с слушателя событий на классы Presenter. Данные классы осуществляют обработку события в зависимости от текущего контекста и обеспечивают взаимодействие графической части приложения с Model, которая управляет СУБД SQLite.

* 1. **Тестирование работы приложения**

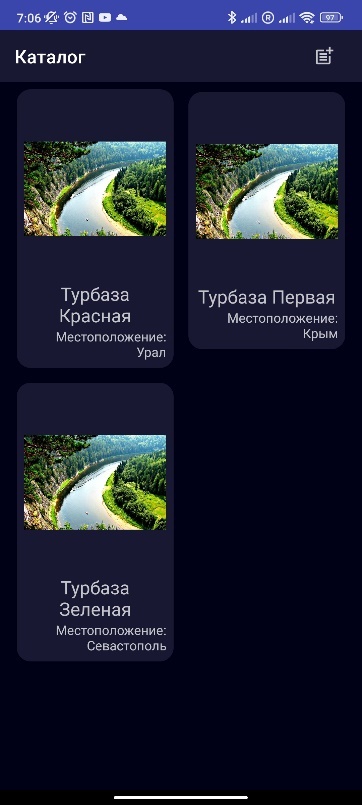
С помощью средств Android Studio была проведена отладка приложения. В ходе этой отладки протестированы основные функции приложения а именно: добавление новых данных в БД, редактирование старых данных в БД, просмотр данных в БД в зависимости от текущего запроса. Результаты тестирования представлены на рисунках 32-35.

Рисунок 32 – Результат добавления турбаз

Изображение выглядит как текст, электроника, снимок экрана, дисплей

Автоматически созданное описание

Рисунок 33 – Результат редактирования записи

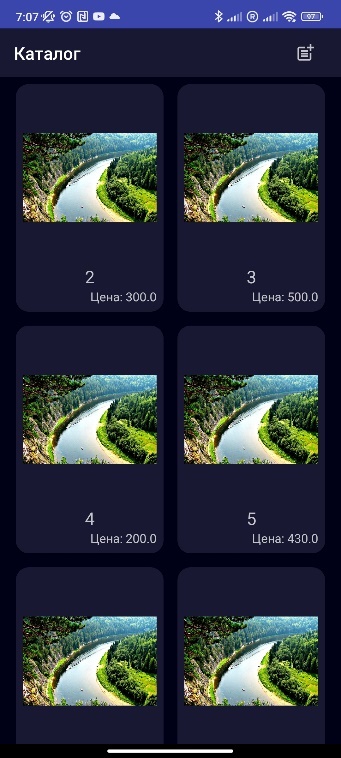
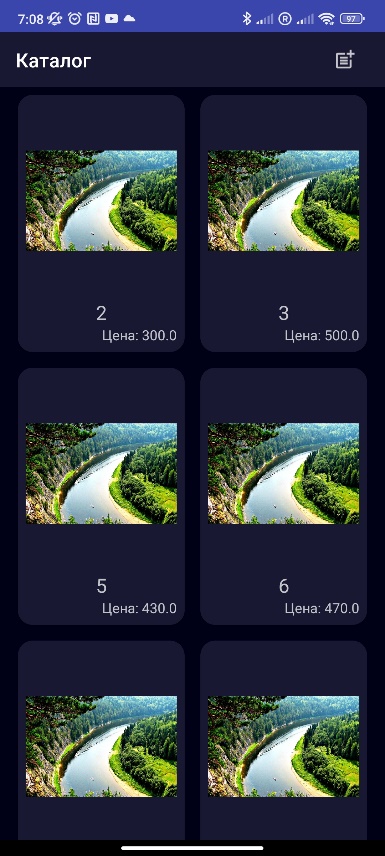


Рисунок 34 – Результат добавления коттеджей

Рисунок 35 – Результат удаления турбазы с каскадированием данных

По результатам проведенного тестирования, можно сделать вывод, что программа работает корректно во всех предоставляемых режимах работы

# **ЗАКЛЮЧЕНИЕ**

В соответсвии с вариантом задания была разработана база данных, позволяющая добавлять, редактировать, просматривать и удалять данные. При построении базы данных были пройдены такие этапы проектирования как: концептуальное проектирование, логическое проектирование, физическое проектирование и использования полученной физической модели в разрабатываемом приложении. На этапе концептуального проектирования были рассмотрены все необходимые для данной предметной области сущности, их связи между собой. На этапе логического проектирования была спроектирована полная атрибутивная модель базы данных, удовлетвояющая нотации IDEF1X. Для этого при логическом проектировании были пройдены этапы создания моделей сущность-связь, создания модели, основанной на ключах с последующей нормализацией данной модели. В результате по полученной логической модели была создана физическая модель, используемая в программе.

Таким образом, была достигнута цель расчетно-графической работы – были получены и закреплены практические и теоритические навыки концептуального, логического и физического проектирования баз данных а также использования внутри разрабатываемых приложений СУБД, использующих принципы SQL.

**ПРИЛОЖЕНИЕ А  
Листинг программы**

**Код класса ModelSQLite**

package com.example.myapplication.models\_and\_DB

import android.content.ContentValues

import android.content.Context

import android.database.sqlite.SQLiteDatabase

import android.database.sqlite.SQLiteOpenHelper

import android.util.Log

import com.example.myapplication.interfaces.IModelSQLite

import com.example.myapplication.objects.Consts

import io.realm.kotlin.Realm

import io.realm.kotlin.ext.query

import io.realm.kotlin.internal.platform.runBlocking

import io.realm.kotlin.mongodb.App

import io.realm.kotlin.mongodb.Credentials

import io.realm.kotlin.mongodb.User

import io.realm.kotlin.mongodb.exceptions.AuthException

import io.realm.kotlin.mongodb.exceptions.InvalidCredentialsException

import io.realm.kotlin.mongodb.exceptions.ServiceException

import io.realm.kotlin.mongodb.exceptions.UserAlreadyExistsException

import io.realm.kotlin.mongodb.sync.SyncConfiguration

import java.text.SimpleDateFormat

class ModelSQLite(context: Context?) : SQLiteOpenHelper(

context, DATABASE\_NAME,

null, DATABASE\_VERSION

), IModelSQLite {

companion object {

const val DATABASE\_NAME = "TourBase.db"

const val DATABASE\_VERSION = 2

const val TOUR\_HOSTELS\_TABLE\_NAME = "tour\_hostels"

const val COTTAGE\_TABLE\_NAME = "cottage"

const val HOSTEL\_NAME = "hostel\_name"

const val REG\_ID = "reg\_id"

const val LIVING\_APARTMENTS\_AMOUNT = "living\_apartments"

const val LOCATION = "location"

const val ROOM\_ID = "room\_id"

const val BASE\_PRICE = "base\_price"

const val AMOUNT\_TO\_ACCOMMODATE = "to\_accommodate"

const val BOOKING\_ID = "bk\_id"

const val ACCOMMODATION\_DATE = "acc\_date"

const val DEPARTURE\_DATE = "dep\_date"

const val TOTAL\_PRICE = "total\_price"

const val BOOKING\_TABLE\_NAME = "booking\_table"

const val PERSONAL\_ID = "personal\_id"

const val FULL\_CLIENT\_NAME = "client\_name"

const val CLIENT\_TELEPHONE = "client\_tel"

const val CLIENT\_TABLE\_NAME = "client\_table"

const val CLASS\_ID = "class\_id"

const val CLASS\_NAME = "class\_name"

const val ADDITIONAL\_PRICE = "add\_price"

const val CLASSES\_TABLE\_NAME = "classes\_table"

}

private var realmUserCfg: SyncConfiguration? = RealmUserDB.getSyncCfg()

private var realmUserDB: Realm? = realmUserCfg?.let {

Realm.open(it)

//RealmUserDB.getInstance(it)

}

private var user: User? = null

private val app = App.create(Consts.app\_id)

fun logIn(\_login: String, \_password: String): Boolean {

try {

runBlocking {

user = app.login(Credentials.emailPassword(\_login, \_password))

user?.let {

realmUserCfg = RealmUserDB.getSyncCfg(it, \_login)

}

realmUserCfg?.let {

realmUserDB = Realm.open(it)

}

}

} catch (exc: AuthException) {

throw exc

} catch (exc: InvalidCredentialsException) {

throw exc

} catch (exc: IllegalArgumentException) {

throw exc

} catch (exc: IllegalStateException) {

throw exc

} catch (exc: ServiceException) {

Log.d("debug", "3"); throw exc

}

return true

}

fun signUp(user: RealmUserData): Boolean {

return try {

runBlocking {

app.emailPasswordAuth.registerUser(user.\_id, user.password)

Log.d("debug", "user registered")

}

logIn(user.\_id, user.password)

} catch (exc: UserAlreadyExistsException) {

Log.e("debug", "User already exists")

false

} catch (exc: ServiceException) {

Log.e("debug", "Some error occurred")

throw exc

}

}

fun logOut() {

try {

runBlocking {

app.currentUser?.logOut()

realmUserDB?.close()

}

} catch (exc: ServiceException) {

throw exc

} finally {

RealmUserDB.reset()

}

}

suspend fun updateData(user: RealmUserData) {

Log.d("debug", "Saving user")

realmUserDB?.write {

copyToRealm(user)

}

}

fun getUser(): RealmUserData? {

return realmUserDB?.query<RealmUserData>("\_id = $0", RealmUserDB.getLogin())?.find()

?.first()

}

override fun onConfigure(db: SQLiteDatabase?) {

db?.setForeignKeyConstraintsEnabled(true)

super.onConfigure(db)

}

override fun onCreate(db: SQLiteDatabase?) {

db?.execSQL(

"CREATE TABLE $TOUR\_HOSTELS\_TABLE\_NAME ( " +

"$REG\_ID INTEGER PRIMARY KEY AUTOINCREMENT, " +

"$HOSTEL\_NAME TEXT, " +

"$LIVING\_APARTMENTS\_AMOUNT INTEGER, " +

"$LOCATION TEXT )"

)

db?.execSQL(

"CREATE TABLE $CLASSES\_TABLE\_NAME (" +

"$CLASS\_ID INTEGER PRIMARY KEY AUTOINCREMENT, " +

"$CLASS\_NAME TEXT, " +

"$ADDITIONAL\_PRICE REAL)"

)

db?.execSQL(

"CREATE TABLE $COTTAGE\_TABLE\_NAME ( " +

"$ROOM\_ID INTEGER PRIMARY KEY AUTOINCREMENT, " +

"$CLASS\_ID INTEGER, " +

"$REG\_ID INTEGER, " +

"$BASE\_PRICE REAL, " +

"$AMOUNT\_TO\_ACCOMMODATE INTEGER, " +

"FOREIGN KEY ($CLASS\_ID) " +

"REFERENCES $CLASSES\_TABLE\_NAME($CLASS\_ID)" +

"ON DELETE CASCADE, " +

"FOREIGN KEY ($REG\_ID) " +

"REFERENCES $TOUR\_HOSTELS\_TABLE\_NAME ($REG\_ID) " +

"ON DELETE CASCADE)"

)

db?.execSQL(

"CREATE TABLE $CLIENT\_TABLE\_NAME (" +

"$PERSONAL\_ID INTEGER PRIMARY KEY AUTOINCREMENT, " +

"$FULL\_CLIENT\_NAME TEXT, " +

"$CLIENT\_TELEPHONE TEXT)"

)

db?.execSQL(

"CREATE TABLE $BOOKING\_TABLE\_NAME (" +

"$BOOKING\_ID INTEGER PRIMARY KEY AUTOINCREMENT," +

"$ROOM\_ID INTEGER, " +

"$PERSONAL\_ID TEXT, " +

"$ACCOMMODATION\_DATE TEXT, " +

"$DEPARTURE\_DATE TEXT, " +

"$TOTAL\_PRICE REAL, " +

"FOREIGN KEY($ROOM\_ID) " +

"REFERENCES $COTTAGE\_TABLE\_NAME ($ROOM\_ID) " +

"ON DELETE CASCADE," +

"FOREIGN KEY($PERSONAL\_ID)" +

"REFERENCES $CLIENT\_TABLE\_NAME($PERSONAL\_ID)" +

"ON DELETE CASCADE )"

)

}

override fun onUpgrade(db: SQLiteDatabase?, oldVersion: Int, newVersion: Int) {

//db?.execSQL("DROP TABLE IF EXISTS $TOUR\_HOSTELS\_TABLE\_NAME")

//db?.execSQL("DROP TABLE IF EXISTS $CLASSES\_TABLE\_NAME")

//db?.execSQL("DROP TABLE IF EXISTS $CLIENT\_TABLE\_NAME")

//db?.execSQL("DROP TABLE IF EXISTS $COTTAGE\_TABLE\_NAME")

//

//db?.execSQL("DROP TABLE IF EXISTS $BOOKING\_TABLE\_NAME")

onCreate(db)

}

override fun insertTourHostel(hostelName: String, livingApartments: Int, location: String) {

val values = ContentValues()

values.put(HOSTEL\_NAME, hostelName)

values.put(LIVING\_APARTMENTS\_AMOUNT, livingApartments)

values.put(LOCATION, location)

if (this.writableDatabase.insert(TOUR\_HOSTELS\_TABLE\_NAME, null, values) == -1L) {

throw Exception("unable to insert hostel")

}

}

override fun insertClasses(className: String, addPrice: Float) {

val values = ContentValues()

values.put(CLASS\_NAME, className)

values.put(ADDITIONAL\_PRICE, addPrice)

if (this.writableDatabase.insert(CLASSES\_TABLE\_NAME, null, values) == -1L) {

throw Exception("unable to insert class")

}

}

override fun insertCottage(classId: Int, regId: Int, basePrice: Float, toAccommodate: Int) {

val values = ContentValues()

values.put(CLASS\_ID, classId)

values.put(REG\_ID, regId)

values.put(BASE\_PRICE, basePrice)

values.put(AMOUNT\_TO\_ACCOMMODATE, toAccommodate)

try {

if (this.writableDatabase.insert(COTTAGE\_TABLE\_NAME, null, values) == -1L) {

Log.d("debug", "some shit happened")

throw Exception("unable to insert cottage")

} else

Log.d("debug", "it's fine")

} catch (exc: Exception) {

Log.d("debug", "no constr")

}

}

override fun insertClient(clientFullName: String, tel: String) {

val values = ContentValues()

//values.put(PERSONAL\_ID, personId)

values.put(FULL\_CLIENT\_NAME, clientFullName)

values.put(CLIENT\_TELEPHONE, tel)

try {

if (this.writableDatabase.insert(CLIENT\_TABLE\_NAME, null, values) == -1L) {

throw Exception("unable to insert client")

}

} catch (exc: Exception) {

Log.d("debug", "no constr")

}

}

override fun insertBooking(roomId: Int, personId: String, accDate: String, depDate: String) {

val values = ContentValues()

values.put(ROOM\_ID, roomId)

values.put(PERSONAL\_ID, personId)

values.put(ACCOMMODATION\_DATE, accDate)

values.put(DEPARTURE\_DATE, depDate)

values.put(TOTAL\_PRICE, getTotalPrice(roomId, accDate, depDate))

try {

if (this.writableDatabase.insert(BOOKING\_TABLE\_NAME, null, values) == -1L) {

throw Exception("unable to insert booking")

}

} catch (exc: Exception) {

Log.d("debug", "no constr")

}

}

override fun getTourHostels(filter: String): List<TourHostelsData> {

val cursor =

this.readableDatabase.rawQuery("SELECT \* FROM $TOUR\_HOSTELS\_TABLE\_NAME $filter", null)

val list: ArrayList<TourHostelsData> = ArrayList()

while (cursor.moveToNext()) {

list.add(

TourHostelsData(

cursor.getInt(0),

cursor.getString(1),

cursor.getString(2),

cursor.getString(3)

)

)

}

cursor.close()

return list

}

private fun getTotalPrice(roomId: Int, accDate: String, depDate: String): Float {

val cursor = this.readableDatabase.rawQuery(

"SELECT $BASE\_PRICE, $ADDITIONAL\_PRICE " +

"FROM $COTTAGE\_TABLE\_NAME " +

"JOIN $CLASSES\_TABLE\_NAME ON " +

"$COTTAGE\_TABLE\_NAME.$CLASS\_ID = $CLASSES\_TABLE\_NAME.$CLASS\_ID " +

"WHERE $ROOM\_ID = $roomId ",

null

)

cursor.moveToFirst()

val totalPrice = (cursor.getFloat(0)

+ cursor.getFloat(1)) \*

(SimpleDateFormat("dd-MM-yyyy")

.parse(depDate).time - SimpleDateFormat("dd-MM-yyyy")

.parse(accDate).time) / 86400000

cursor.close()

return totalPrice

}

override fun getCottages(filter: String): List<CottagesData> {

val cursor =

this.readableDatabase.rawQuery("SELECT \* FROM $COTTAGE\_TABLE\_NAME $filter", null)

val list: ArrayList<CottagesData> = ArrayList()

while (cursor.moveToNext()) {

list.add(

CottagesData(

cursor.getInt(0),

cursor.getInt(1),

cursor.getInt(2),

cursor.getFloat(3),

cursor.getInt(4)

)

)

}

cursor.close()

return list

}

override fun getBooking(filter: String): List<BookingData> {

val cursor =

this.readableDatabase.rawQuery("SELECT \* FROM $BOOKING\_TABLE\_NAME $filter", null)

val list: ArrayList<BookingData> = ArrayList()

while (cursor.moveToNext()) {

list.add(

BookingData(

cursor.getInt(0),

cursor.getInt(1),

cursor.getInt(2),

cursor.getString(3),

cursor.getString(4),

cursor.getFloat(5)

)

)

}

cursor.close()

return list

}

override fun getClasses(filter: String): List<ClassesData> {

val cursor =

this.readableDatabase.rawQuery("SELECT \* FROM $CLASSES\_TABLE\_NAME $filter", null)

val list: ArrayList<ClassesData> = ArrayList()

while (cursor.moveToNext()) {

list.add(

ClassesData(

cursor.getInt(0),

cursor.getString(1),

cursor.getFloat(2)

)

)

}

cursor.close()

return list

}

override fun getClients(filter: String): List<ClientData> {

val cursor =

this.readableDatabase.rawQuery("SELECT \* FROM $CLIENT\_TABLE\_NAME $filter", null)

val list: ArrayList<ClientData> = ArrayList()

while (cursor.moveToNext()) {

list.add(

ClientData(

cursor.getInt(0),

cursor.getString(1),

cursor.getString(2)

)

)

}

cursor.close()

return list

}

override fun updateTourHostel(data: TourHostelsData) {

val values = ContentValues()

values.put(HOSTEL\_NAME, data.hostelName)

values.put(LIVING\_APARTMENTS\_AMOUNT, data.livingApartments)

values.put(LOCATION, data.location)

this.writableDatabase.update(

TOUR\_HOSTELS\_TABLE\_NAME,

values,

"$REG\_ID=?",

arrayOf(data.regId.toString())

)

}

override fun updateCottages(data: CottagesData) {

val values = ContentValues()

values.put(CLASS\_ID, data.classId)

values.put(REG\_ID, data.regId)

values.put(BASE\_PRICE, data.basePrice)

values.put(AMOUNT\_TO\_ACCOMMODATE, data.amountToAccommodate)

this.writableDatabase.update(

COTTAGE\_TABLE\_NAME,

values,

"$ROOM\_ID = ?",

arrayOf(data.roomId.toString())

)

}

override fun updateBooking(data: BookingData) {

val values = ContentValues()

values.put(PERSONAL\_ID, data.personalId)

values.put(ROOM\_ID, data.roomId)

values.put(ACCOMMODATION\_DATE, data.accommodationDate)

values.put(DEPARTURE\_DATE, data.departureDate)

values.put(

TOTAL\_PRICE,

getTotalPrice(data.roomId, data.accommodationDate, data.departureDate)

)

this.writableDatabase.update(

BOOKING\_TABLE\_NAME,

values,

"$BOOKING\_ID = ?",

arrayOf(data.bookId.toString())

)

}

override fun updateClass(data: ClassesData) {

val values = ContentValues()

values.put(CLASS\_NAME, data.className)

values.put(ADDITIONAL\_PRICE, data.additionalPrice)

this.writableDatabase.update(

CLASSES\_TABLE\_NAME,

values,

"$CLASS\_ID = ?",

arrayOf(data.classId.toString())

)

}

override fun updateClient(data: ClientData) {

val values = ContentValues()

values.put(FULL\_CLIENT\_NAME, data.clientFullName)

values.put(CLIENT\_TELEPHONE, data.clientTelephone)

this.writableDatabase.update(

CLIENT\_TABLE\_NAME,

values,

"$PERSONAL\_ID = ?",

arrayOf(data.personalId.toString())

)

}

override fun deleteTourHostel(regId: Int) {

this.writableDatabase.delete(

TOUR\_HOSTELS\_TABLE\_NAME,

"$REG\_ID = ?",

arrayOf(regId.toString())

)

}

override fun deleteCottage(roomId: Int) {

this.writableDatabase.delete(

COTTAGE\_TABLE\_NAME,

"$ROOM\_ID = ?",

arrayOf(roomId.toString())

)

}

override fun deleteBooking(bookId: Int) {

this.writableDatabase.delete(

BOOKING\_TABLE\_NAME,

"$BOOKING\_ID = ?",

arrayOf(bookId.toString())

)

}

override fun deleteClass(classId: Int) {

this.writableDatabase.delete(

CLASSES\_TABLE\_NAME,

"$CLASS\_ID = ?",

arrayOf(classId.toString())

)

}

override fun deleteClient(personId: String) {

this.writableDatabase.delete(

CLIENT\_TABLE\_NAME,

"$PERSONAL\_ID = ?",

arrayOf(personId)

)

}

}

**Код класса CatalogCottagesFragment**

package com.example.myapplication.main.cottages

import android.annotation.SuppressLint

import android.content.Context

import android.graphics.Color

import android.os.Bundle

import android.util.Log

import android.view.View

import android.widget.\*

import androidx.appcompat.widget.SwitchCompat

import androidx.fragment.app.Fragment

import androidx.navigation.fragment.findNavController

import androidx.recyclerview.widget.GridLayoutManager

import com.example.myapplication.R

import com.example.myapplication.databinding.CatalogCottagesFragmentBinding

import com.example.myapplication.interfaces.ICatalogCottages

import com.example.myapplication.interfaces.IToastRealmSessionErrors

import com.example.myapplication.main.FilterValues

import com.example.myapplication.models\_and\_DB.CottagesData

import com.example.myapplication.models\_and\_DB.RealmUserDB

import com.example.myapplication.models\_and\_DB.RealmUserData

import com.example.myapplication.objects.SharedPrefsIDs

import com.google.android.material.slider.RangeSlider

import com.jaredrummler.android.colorpicker.ColorPanelView

import com.jaredrummler.android.colorpicker.ColorPickerDialog

import com.jaredrummler.android.colorpicker.ColorPickerDialogListener

import com.jaredrummler.android.colorpicker.ColorShape

class CatalogCottagesFragment : Fragment(R.layout.catalog\_cottages\_fragment), IToastRealmSessionErrors,

ICatalogCottages, CottageRecyclerAdapter.IItemNodeListener {

private var binding: CatalogCottagesFragmentBinding? = null

private val presenter by lazy {

PresenterCatalogCottages(

this, activity?.getSharedPreferences(

SharedPrefsIDs.sharedPrefLogInData,

Context.MODE\_PRIVATE

)

)

}

private val recyclerAdapter = CottageRecyclerAdapter(this)

private val filter = FilterValues(null, null, 80F, 20F , null, null)

private var priceFilter: RangeSlider?=null

private var menuColorFilter: ColorPanelView?=null

private var inStockFilter:SwitchCompat?=null

private var applyButton:Button?=null

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {

binding = CatalogCottagesFragmentBinding.bind(view)

initBinding()

Log.d("debug", "item\_idCottages: " + arguments?.getString("itemId"))

presenter.init( arguments?.getString("itemId"))

preparePriceFilter()

setupListeners()

binding?.toolbar?.title=getString(R.string.catalogTitle)

if(!RealmUserDB.isAdmin)

binding?.filterButton?.visibility=View.GONE

super.onViewCreated(view, savedInstanceState)

Log.d("debug","Color: ${Color.parseColor("#ffee8383")}")

Log.d("debug", "onCreateCatalog completed")

}

override fun onViewStateRestored(savedInstanceState: Bundle?) {

presenter.updateData(filter)//pass current hostel id

super.onViewStateRestored(savedInstanceState)

}

override fun onDestroy() {

presenter.onFragmentClose()

super.onDestroy()

}

private fun initBinding(){

priceFilter=binding?.navigationView

?.menu?.findItem(R.id.menuPriceFilter)?.actionView as RangeSlider?

menuColorFilter=binding?.navigationView

?.menu?.findItem(R.id.menuColorFilter)?.actionView

?.findViewById(R.id.colorPick) as ColorPanelView?

inStockFilter=binding?.navigationView

?.menu?.findItem(R.id.menuInStockFilter)

?.actionView?.findViewById(R.id.inStockFilter) as SwitchCompat?

applyButton= binding?.navigationView

?.menu?.findItem(R.id.menuSubmitFilters)

?.actionView?.findViewById(R.id.apply\_filters) as Button?

}

private fun setupListeners(){

menuColorFilter?.setOnClickListener {

createColorPickerDialog(1)

}

inStockFilter?.setOnCheckedChangeListener { \_, isChecked ->

filter.inStock = isChecked

Log.d("debug", "inStock=" + filter.inStock)

}

//applying filters

applyButton?.setOnClickListener {

filter.minPrice = priceFilter?.values?.get(0)

filter.maxPrice = priceFilter?.values?.get(1)

filter.color = getColorFilterValue()

filter.inStock = getInStockFilterValue()

Log.d("debug", "Apply button")

presenter.updateData(filter)

}

binding?.swipeRefresh?.setOnRefreshListener {

presenter.updateData(filter)

binding?.swipeRefresh?.isRefreshing = false

}

binding?.recView?.adapter = recyclerAdapter

binding?.filterButton?.setOnClickListener {

findNavController().navigate(R.id.action\_catalogCottagesFragment\_to\_addCottageFragment)

}

binding?.navigationView?.getHeaderView(0)

?.findViewById<ImageButton>(R.id.exitAccountButton)

?.setOnClickListener {

presenter.logOut()

}

}

private fun getColorFilterValue(): Int? {

if ((binding?.navigationView

?.menu?.findItem(R.id.menuColorFilter)

?.actionView

?.findViewById(R.id.checkBoxColorFilter) as CheckBox).isChecked

)

return menuColorFilter?.color

return null

}

private fun getInStockFilterValue(): Boolean? {

return inStockFilter?.isChecked

}

override fun setPriceFilterValueTo(value: Float) {

priceFilter?.valueTo = value

priceFilter?.values?.set(1, value\*0.8F)

}

@SuppressLint("ClickableViewAccessibility")

private fun preparePriceFilter() {

val menuPriceFilter = binding?.navigationView

?.menu?.findItem(R.id.menuPriceFilter)?.actionView as RangeSlider

menuPriceFilter.addOnChangeListener { slider, \_, \_ ->

Log.d("debug", "Slider THIS values: " + slider.values[0] + " max " + slider.values[1])

filter.minPrice = menuPriceFilter.values[0]

filter.maxPrice = menuPriceFilter.values[1]

}

}

private fun createColorPickerDialog(id: Int) {

val dialog = ColorPickerDialog.newBuilder()

.setColor(Color.RED)

.setDialogType(ColorPickerDialog.TYPE\_PRESETS)

.setAllowCustom(true)

.setAllowPresets(true)

.setColorShape(ColorShape.CIRCLE)

.setDialogId(id)

.create()

dialog.setColorPickerDialogListener(object : ColorPickerDialogListener {

override fun onColorSelected(dialogId: Int, color: Int) {

val colorIcon = binding?.navigationView

?.menu?.findItem(R.id.menuColorFilter)

?.actionView

?.findViewById(R.id.colorPick) as ColorPanelView

colorIcon.color = color

filter.color = color

Log.d("debug", "OnColorSelected inside fragment")

}

override fun onDialogDismissed(dialogId: Int) {

}

})

dialog.show(childFragmentManager, "Color Picker")

}

override fun onItemClickListener(position: Int) {

Log.d("debug", "item click: ${recyclerAdapter.getItems()?.get(position)?.roomId}")

val bundle=Bundle()

bundle.putString("itemId", recyclerAdapter.getItems()?.get(position)?.roomId?.toString())

findNavController().navigate(R.id.action\_catalogCottagesFragment\_to\_pdpCottageFragment, bundle)

// navigate(R.id.action\_catalogFragment\_to\_pdpFragment, Bundle().putString("itemId","value"))

}

override fun showProgressBar() {

binding?.progressBarCatalog?.visibility=ProgressBar.VISIBLE

}

override fun hideProgressBar() {

binding?.progressBarCatalog?.visibility=ProgressBar.INVISIBLE

}

override fun setRecyclerData(itemList: List<CottagesData>) {

recyclerAdapter.updateList(itemList)

}

override fun setRecyclerLayoutManagerToGrid() {

binding?.recView?.layoutManager = GridLayoutManager(context, 2)

}

override fun setupNavHeaderWithUserData(userData: RealmUserData) {

binding?.navigationView

?.getHeaderView(0)?.findViewById<TextView>(R.id.tvUserFullName)

?.text="${userData.userFirstName} ${userData.userLastName}"

binding?.navigationView

?.getHeaderView(0)?.findViewById<TextView>(R.id.tvUserLogin)

?.text=userData.\_id

}

override fun returnToLoginScreen() {

findNavController().popBackStack(R.id.logInFragment, inclusive = false)

}

override fun showToastUnableToLogIN() {

Toast.makeText(

requireContext(),

getString(R.string.logInErrorConnectionIssue),

Toast.LENGTH\_LONG

).show()

}

override fun showToastInternalRealmError() {

Toast.makeText(requireContext(), getString(R.string.unknownRealmError), Toast.LENGTH\_LONG)

.show()

}

}

**Код класса PresenterCatalogCottages**

package com.example.myapplication.main.cottages

import android.content.SharedPreferences

import android.util.Log

import androidx.core.content.edit

import com.example.myapplication.interfaces.ICatalogCottages

import com.example.myapplication.main.FilterValues

import com.example.myapplication.models\_and\_DB.ModelSQLite

import com.example.myapplication.models\_and\_DB.SQLiteQueryBuilder

import com.example.myapplication.objects.SharedPrefsIDs.isLogged

import io.realm.kotlin.mongodb.exceptions.ServiceException

import kotlinx.coroutines.CoroutineScope

import kotlinx.coroutines.Dispatchers

import kotlinx.coroutines.async

class PresenterCatalogCottages(

private var view: ICatalogCottages,

private val sharedPref: SharedPreferences?

) {

private var model = ModelSQLite(view.requireContext())

fun init(itemId: String?) {

/\*runBlocking {

model.insertTestItemData(

RealmItemData(

"https://cdn.danielonline.ru/upload/resize\_cache/iblock/ec4/363\_477\_1/ha45abt99cygpooyvp9j9or56ou9a5sh.JPG",

"ItemName5", "descr5", true, 1600,-432555, "default"

)

)

}\*/

view.setRecyclerLayoutManagerToGrid()

model.getUser()?.let {

view.setupNavHeaderWithUserData(it)

}

Log.d("debug", "Query"+SQLiteQueryBuilder().addObligatoryFilter(

ModelSQLite.REG\_ID,

"=",

itemId

).getQuery())

view.setRecyclerData(

model.getCottages(

SQLiteQueryBuilder().addObligatoryFilter(

ModelSQLite.REG\_ID,

"=",

itemId

).getQuery()

)

)

}

private fun clearPrefs() {

sharedPref?.edit(commit = true) {

putBoolean(isLogged, false)

}

}

fun onFragmentClose() {

model.close()

}

fun updateData(filterValues: FilterValues) {

}

fun logOut() {

Log.d("debug", "Logout")

view.showProgressBar()

clearPrefs()

try {

CoroutineScope(Dispatchers.IO).async {

model.logOut()

}

} catch (exc: ServiceException) {

Log.d("debug", "Service error while logging out")

} finally {

view.hideProgressBar()

}

view.returnToLoginScreen()

}

}

**Код класса CottageRecyclerAdapter**

package com.example.myapplication.main.cottages

import android.annotation.SuppressLint

import android.content.Context

import android.view.LayoutInflater

import android.view.View

import android.view.ViewGroup

import androidx.recyclerview.widget.RecyclerView

import com.example.myapplication.R

import com.example.myapplication.databinding.CatalogCottagesItemBinding

import com.example.myapplication.models\_and\_DB.CottagesData

import com.squareup.picasso.Picasso

class CottageRecyclerAdapter(private val onNodeListener: IItemNodeListener) :

RecyclerView.Adapter<CottageRecyclerAdapter.ItemHolder>() {

private var itemList: List<CottagesData>? = null

class ItemHolder(

item: View,

val context: Context?,

private val onNodeListener: IItemNodeListener

) : RecyclerView.ViewHolder(item), View.OnClickListener {

private val binding = CatalogCottagesItemBinding.bind(item)

fun bindElement(item: CottagesData)=with(binding) {

tvTitle.text = item.roomId.toString()

//добавление изображения

//Picasso.with(context).load(item.\_id).into(itemImg)

Picasso.get().load("https://www.mal-kar.ru/tours/bazi.otdiha/3.jpg").into(itemImg)

tvBasePrice.text = "${context?.getString(R.string.tvPrice)}: ${item.basePrice}"

}

override fun onClick(v: View?) {

onNodeListener.onItemClickListener(adapterPosition)

}

init {

item.setOnClickListener(this)

}

}

interface IItemNodeListener {

fun onItemClickListener(position: Int)

}

fun getItems(): List<CottagesData>? {

return itemList

}

override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ItemHolder {

return ItemHolder(

LayoutInflater.from(parent.context).inflate(R.layout.catalog\_cottages\_item, parent, false),

parent.context,

onNodeListener

)

}

override fun onBindViewHolder(holder: ItemHolder, postition: Int) {

itemList?.get(postition)?.let { holder.bindElement(it) }

}

override fun getItemCount(): Int {

return itemList?.size ?: return 0

}

@SuppressLint("NotifyDataSetChanged")

fun updateList(updatedList: List<CottagesData>) {

itemList = null

itemList = updatedList

notifyDataSetChanged()

}

}

**Код класса CatalogDataEdit**

package com.example.myapplication.main.cottages

import android.content.Context

import android.os.Bundle

import android.util.Log

import android.view.View

import androidx.fragment.app.Fragment

import com.example.myapplication.R

import com.example.myapplication.databinding.CottageEditFragmentBinding

import com.example.myapplication.models\_and\_DB.CottagesData

import com.example.myapplication.objects.SharedPrefsIDs

class CottageDataEditFragment: Fragment(R.layout.cottage\_edit\_fragment){

private var binding: CottageEditFragmentBinding? = null

private val presenter by lazy {

PresenterCottageDataEdit(

this, activity?.getSharedPreferences(

SharedPrefsIDs.sharedPrefLogInData,

Context.MODE\_PRIVATE

)

)

}

private var item: CottagesData?=null

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {

binding = CottageEditFragmentBinding.bind(view)

super.onViewCreated(view, savedInstanceState)

Log.d("debug", "item\_id: " + arguments?.getString("itemId"))

arguments?.getString("itemId")?.let { presenter.init(it) }

binding?.changeButton?.setOnClickListener {

item?.basePrice=binding?.edBasePrice?.editText?.text.toString().toFloat()

item?.amountToAccommodate= binding?.edToAccommodate?.editText?.text.toString().toInt()

item?.let { presenter.changeData(it) }

}

binding?.buttonDelete?.setOnClickListener {

presenter.deleteData()

}

}

fun setupWithData(data: CottagesData){

item=data

binding?.edBasePrice?.editText?.setText(data.basePrice.toString())

binding?.edToAccommodate?.editText?.setText(data.amountToAccommodate.toString())

}

}

**Код класса PresenterCatalogDataEdit**

package com.example.myapplication.main.cottages

import android.content.SharedPreferences

import com.example.myapplication.models\_and\_DB.CottagesData

import com.example.myapplication.models\_and\_DB.ModelSQLite

class PresenterCottageDataEdit(private var view: CottageDataEditFragment, private val sharedPref: SharedPreferences?) {

private var model: ModelSQLite? = ModelSQLite(view.requireContext())

private var item: CottagesData?=null

fun init(itemId: String){

model?.getCottages("WHERE ${ModelSQLite.ROOM\_ID} = $itemId")?.let {

view.setupWithData(it.first())

item=it.first()

}

}

fun changeData(data: CottagesData){

model?.updateCottages(data)

}

fun deleteData(){

item?.let { model?.deleteTourHostel(it.roomId) }

}

}