|  |  |  |
| --- | --- | --- |
|  |  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |

Институт Информационных технологий

Кафедра Математического обеспечения и стандартизации информационных технологий

**Отчет по практической работе №10**

по дисциплине «Проектирование и разработка мобильных приложений»

|  |  |
| --- | --- |
| **Выполнил:**  Студент группыИКБО-21-23 | Лисовский И.В. |
| **Проверил:**  Старший преподаватель кафедры МОСИТ | Шешуков Л.С. |

Москва 2025 г.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc197548642)

[1 ТЕОРЕТИЧЕСКОЕ ВВЕДЕНИЕ 4](#_Toc197548643)

[1.1 Класс SharedPreference 4](#_Toc197548644)

[1.2 Основные понятия баз данных 9](#_Toc197548645)

[1.3 Основные команды языка SQL 11](#_Toc197548646)

[1.4 Работа с СУБД в Android 18](#_Toc197548647)

[2 ПРАКТИЧЕСКАЯ ЧАСТЬ 35](#_Toc197548648)

[2.1 Реализация записи, получения, изменения и удаления имени пользователя приложения через SharedPreference 35](#_Toc197548649)

[2.2 Создание тематической базы данных. Реализация методов записи, поиска, изменения и удаления данных 37](#_Toc197548650)

[ЗАКЛЮЧЕНИЕ 45](#_Toc197548651)

# ВВЕДЕНИЕ

В данной практической работе были реализованы две функциональные задачи, направленные на закрепление навыков работы с локальным хранилищем данных в Android-приложениях на языке Java.

Первая задача заключалась в использовании механизма SharedPreferences для сохранения, получения, изменения и удаления имени пользователя. Этот подход является простым и эффективным решением для хранения небольших объёмов данных, таких как настройки или персонализированная информация.

Вторая задача была посвящена созданию и управлению локальной базой данных SQLite, где на примере автосалона реализована таблица с информацией об автомобилях. Для работы с базой данных были созданы методы для добавления, отображения, обновления и удаления записей. Все взаимодействия с базой данных происходят через класс, унаследованный от SQLiteOpenHelper.

# 1 ТЕОРЕТИЧЕСКОЕ ВВЕДЕНИЕ

## Класс SharedPreference

В предыдущей практике мы использовали файловую систему для сохранения данных, что удобно при передаче информации между активностями и даже приложениями. Однако для хранения простых данных — таких как строки, числа или логические значения — использование отдельных файлов не всегда оправдано. В таких случаях гораздо эффективнее применять механизм SharedPreferences.

SharedPreferences — это встроенный инструмент Android для хранения простых данных в формате «ключ-значение». Он отлично подходит для сохранения настроек приложения, авторизационной информации и других небольших данных, которые должны быть доступны между сессиями работы приложения.

Где используется SharedPreferences:

- настройки пользователя: например, выбранная тема, язык интерфейса или другие предпочтения,

- состояние приложения: можно сохранить, какую вкладку пользователь открыл последней или что он ввёл в форму, чтобы при следующем запуске восстановить то же состояние,

- небольшие объёмы данных: SharedPreferences избавляет от необходимости подключать базу данных, если нужно хранить лишь немного информации.

Пример сохранения данных показан на рисунке 1.



Рисунок 1 – Сохранение данных

Метод getSharedPreferences() принимает два параметра: имя файла настроек (в данном случае "myPreferences") и режим доступа (MODE\_PRIVATE, при котором настройки доступны только вашему приложению). Если файл настроек ещё не существует, он будет создан автоматически.

Метод edit() возвращает объект SharedPreferences.Editor, с помощью которого можно добавлять или изменять данные.

Чтобы получить ранее сохранённые значения, сначала нужно получить доступ к объекту SharedPreferences. Это показано на рисунке 2.

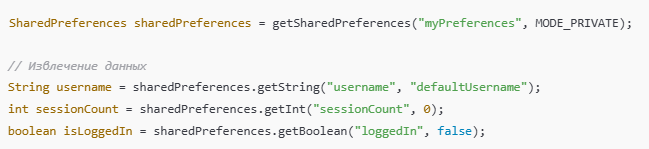


Рисунок 2 – Чтение данных

Метод getString(key, defaultValue) возвращает строковое значение, связанное с указанным ключом. Если такого ключа нет в настройках, будет возвращено значение по умолчанию, переданное вторым параметром. Аналогичным образом работают методы getInt() и getBoolean() для получения чисел и логических значений соответственно.

Этот подход позволяет безопасно извлекать данные без риска возникновения ошибок в случае отсутствия нужного ключа.

На рисунке 3 показан пример работы программы.



Рисунок 3 – Работоспособность программы

Чтобы удалить определённые значения из настроек или полностью очистить все сохранённые данные, используется редактор SharedPreferences.Editor. Пример показан на рисунке 4.

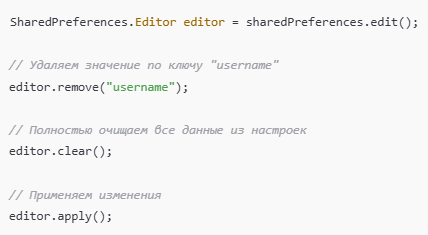


Рисунок 4 – Удаление данных

- метод remove("ключ") удаляет конкретное значение, связанное с заданным ключом,

- метод clear() удаляет все данные, хранящиеся в SharedPreferences,

- метод apply() сохраняет изменения асинхронно.

Это удобно, например, при выходе пользователя из приложения или при сбросе настроек к значениям по умолчанию.

Удаление данных продемонстрировано на рисунке 5.



Рисунок 5 – Работоспособность программы

Если попробовать получить данные после того, как они были удалены, то получим данные, прописанные по умолчанию. Это показано на рисунке 6.



Рисунок 6 – Получение дефолтных значений

## Основные понятия баз данных

Главный недостаток метода хранения данных, описанного выше — это неструктурированность, что значительно затруднит дальнейшую работу с данными.

Одним из самых популярных способов хранения данных является их структуризация в виде таблиц. Такая концепция основана на структурировании информации в виде таблиц, состоящих из строк и столбцов.

Это выглядит примерно как Excel таблица. Есть колонки с заголовками, и информация внутри. На рисунке 7 показан пример интерфейса таблицы.

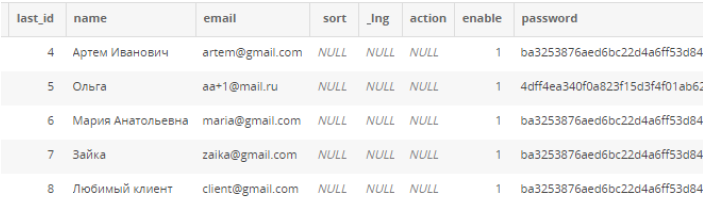


Рисунок 7 – Пример интерфейса таблицы

Это одна из наиболее распространённых моделей организации данных, которая обеспечивает удобное управление и быстрый доступ к информации. Ниже приведены ключевые элементы данной концепции:

1. таблицы - данные размещаются в таблицах, каждая из которых соответствует определённой сущности (например, клиенты, товары, заказы),
2. строки и столбцы - каждая строка представляет собой отдельный объект (экземпляр сущности), а столбцы содержат его характеристики. К примеру, в таблице с клиентами строки будут отражать конкретных клиентов, а столбцы — такие данные, как имя, адрес и номер телефона,
3. первичные ключи (Primary Keys) - для уникальной идентификации каждой строки используется первичный ключ — уникальное значение, которое однозначно определяет запись. Обычно такие ключи генерируются автоматически, чтобы избежать дублирования,
4. внешние ключи (Foreign Keys) - обеспечивают связь между таблицами. Например, в таблице заказов внешний ключ может ссылаться на таблицу клиентов, указывая, кто именно сделал заказ,
5. нормализация - чтобы минимизировать избыточность и предотвратить ошибки, данные проходят нормализацию — процесс разделения информации по нескольким связанным таблицам. Это упрощает сопровождение и обновление данных,
6. индексы - создаются для ускорения поиска и фильтрации данных. Они могут быть заданы для одного или нескольких столбцов, позволяя системе быстро находить нужные записи.

Эта структура лежит в основе реляционных баз данных, в которых для работы с данными применяется язык SQL (Structured Query Language — язык структурированных запросов).

## Основные команды языка SQL

SQL (Structured Query Language) — это язык, с помощью которого осуществляется взаимодействие с базой данных. Прежде чем выполнять запросы, необходимо создать саму базу и таблицы в ней.

Для создания таблицы используется команда CREATE TABLE. Её синтаксис показан на рисунке 8.

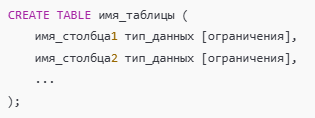


Рисунок 8 – Создание таблицы

Имя таблицы должно быть уникальным в рамках базы данных и не должно начинаться с sqlite\_, так как такие имена зарезервированы системой SQLite для служебных нужд.

Внутри скобок указываются имена столбцов, их типы данных и, при необходимости, ограничения или дополнительные атрибуты.

Создадим таблицу student для хранения информации о студентах — их ФИО, возрасте и поле и покажем это на рисунке 9.

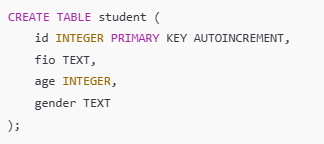


Рисунок 9 – Создание таблицы student

id INTEGER PRIMARY KEY AUTOINCREMENT — создаёт уникальный идентификатор, который автоматически увеличивается с добавлением каждой новой записи.

Если попытаться повторно выполнить такую команду, возникнет ошибка, потому что таблица с таким именем уже существует. Чтобы этого избежать, можно использовать конструкцию IF NOT EXISTS, которая позволяет создать таблицу только если она ещё не была создана, что показано на рисунке 10.

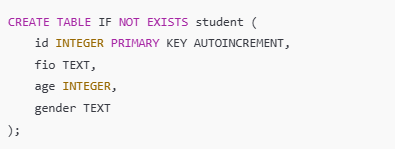


Рисунок 10 – Использование конструкции IF NOT EXISTS

Для вставки записей в таблицу используется команда INSERT. Общий синтаксис показан на рисунке 11.



Рисунок 11 – Добавление данных

После имени таблицы указываются нужные столбцы, а затем в VALUES — соответствующие значения. Значения передаются строго в том же порядке, что и указанные столбцы.

На рисунке 12 покажем вставку данных на примере.



Рисунок 12 – Пример вставки данных

В данном случае:

- fio получит значение 'Муравьёва Екатерина Андреевна',

- age — 25,

- gender — 'женский'.

После выполнения этой команды новая запись появится в таблице student.

Важно понимать, что при добавлении записей в таблицу необязательно указывать значения для всех столбцов. Например, в приведённом ранее примере мы не задавали значение для поля id, поскольку оно формируется автоматически благодаря атрибуту AUTOINCREMENT.

Можно также выполнять вставку без явного перечисления столбцов, как на рисунке 13.



Рисунок 13 – Вставка без явного перечисления столбцов

Однако в таком случае необходимо обязательно указать значения для всех столбцов, в том числе и для id, и при этом соблюдать точный порядок следования столбцов, как он определён при создании таблицы.

Теперь в таблице student появилась первая запись. Но для полноценной работы с данными этого недостаточно. Добавим ещё несколько студентов, чтобы впоследствии можно было выполнять выборку, фильтрацию и другие SQL-операции. На рисунке 14 показана новая таблица.

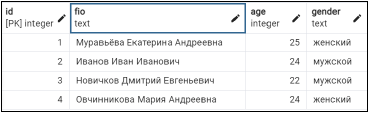


Рисунок 14 – Обновленная база данных

Для получения данных из базы данных SQLite используется команда SELECT. В простейшем виде её синтаксис показан на рисунке 15.



Рисунок 15 – Получение данных в общем виде

Часто нам нужно извлекать из базы только те записи, которые соответствуют определённому условию. Для этого в команде SELECT используется оператор WHERE, за которым следует само условие. Это показано на рисунке 16.



Рисунок 16 – Использование оператора WHERE

- SELECT — указывает, какие столбцы мы хотим получить,

- FROM — из какой таблицы берутся данные,

- WHERE — какое условие должны удовлетворять строки.

Например, предположим, что мы хотим получить информацию обо всех студентах, которым 24 года. Формируем запрос по смыслу: «получить все записи о студентах, у которых возраст равен 24». В SQL это будет выглядеть как показано на рисунке 17.



Рисунок 17 – Получение конкретных данных

Символ \* означает, что мы выбираем все столбцы. Конечно, при необходимости мы можем указать конкретные столбцы вместо звёздочки, если нужны не все данные.

Если бы у нас не было базы данных, а были обычные Excel-файлы, мы бы выполнили то же самое следующим образом:

1. Открыли бы файл с нужными данными (например, student).
2. Установили бы фильтр на колонку «Возраст» со значением 24.

То есть, независимо от инструмента — будь то база данных или электронная таблица — мы должны знать, где хранятся данные (название таблицы или файла) и по какому столбцу мы хотим их отфильтровать. Это не какая-то уникальная особенность баз данных — аналогичные действия можно выполнять и в обычном Excel.

Для изменения уже существующих записей в SQLite применяется команда UPDATE. Чтобы обновить только нужные строки, мы можем уточнить условие с помощью оператора WHERE. Синтаксис показан на рисунке 18.

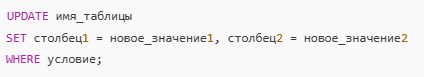


Рисунок 18 – Обновление данных

Предположим, что мы хотим изменить данные в таблице student: всем студенткам (то есть тем, у кого пол — "женский") мы устанавливаем возраст 21. Покажем это на рисунке 19.

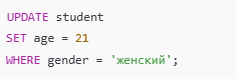


Рисунок 19 – Обновление данных на примере

Таким образом, мы обновляем только те записи, которые соответствуют заданному условию, не затрагивая остальные.

Обновленную базу данных покажем на рисунке 20.

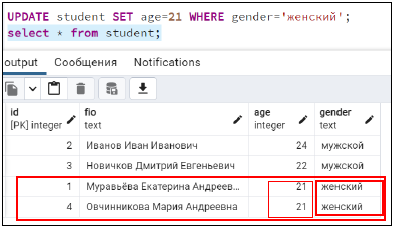


Рисунок 20 – Обновленная база данных

Команда DELETE используется для удаления данных из базы. Её основной синтаксис показан на рисунке 21.



Рисунок 21 – Синтаксис команды DELETE

Допустим, мы хотим удалить из таблицы student всех студентов, которым больше 23 лет. Тогда наш запрос будет выглядеть как показано на рисунке 22.



Рисунок 22 – Удаление конкретных данных

Обновленную базу данных покажем на рисунке 23.

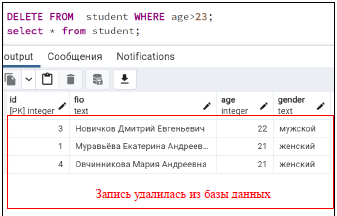


Рисунок 23 – Обновленная база данных

Если мы хотим удалить абсолютно все записи из таблицы, не задавая условий, можно опустить WHERE. Покажем синтаксис на рисунке 24.



Рисунок 24 – Удаление всех записей

Результат такого запроса покажем на рисунке 25.

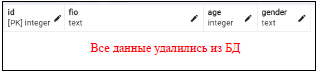


Рисунок 25 – Обновленная база данных

Чтобы удалить саму таблицу вместе со всеми её данными, используется команда DROP TABLE. Синтаксис такого запроса показан на рисунке 26.



Рисунок 26 – Удаление таблицы

Результат работы такого запроса покажем на рисунке 27.

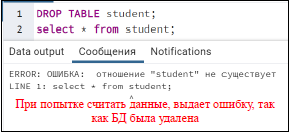


Рисунок 27 – Результат удаления базы данных

По аналогии с созданием таблицы, если мы попытаемся удалить таблицу, которая не существует, то мы столкнемся с ошибкой. В этом случае опять же с помощью операторов IF EXISTSпроверять наличие таблицы перед удалением. Покажем это на рисунке 28.

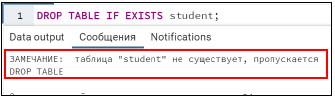


Рисунок 28 – Использование IF EXISTS при удалении

## 1.4 Работа с СУБД в Android

В Android-разработке для управления базами данных мы часто используем SQLite — встроенную, лёгкую систему управления базами данных (СУБД), которая поддерживает большую часть возможностей SQL.

Для взаимодействия с базой через SQLite мы можем использовать интерфейс Cursor, который обеспечивает произвольный доступ к результатам SQL-запросов.

Основные средства работы с базами данных предоставляет пакет android.database, а работа с SQLite организована через android.database.sqlite.

* класс SQLiteDatabase представляет саму базу данных и позволяет выполнять с ней различные операции: чтение, запись, обновление и удаление данных,
* класс SQLiteCursor используется для выполнения запросов и получения набора строк, соответствующих этим запросам,
* SQLiteQueryBuilder помогает нам формировать SQL-запросы программно,
* SQL-выражения представлены с помощью SQLiteStatement, что позволяет вставлять в запросы значения динамически, используя плейсхолдеры,
* класс SQLiteOpenHelper используется для создания и обновления базы данных. С его помощью мы можем создавать базу и все необходимые таблицы, если они ещё не существуют.

Если мы попытаемся прочитать данные из базы, которая была ранее удалена, — произойдёт ошибка, так как файл базы данных больше не существует.

В качестве примера давайте создадим приложение для хранения контактов. У нас будет одна сущность — контакт, включающая имя, номер телефона и уникальный идентификатор (первичный ключ).

Сначала создадим Java-класс для представления контактов и покажем его на рисунке 29.

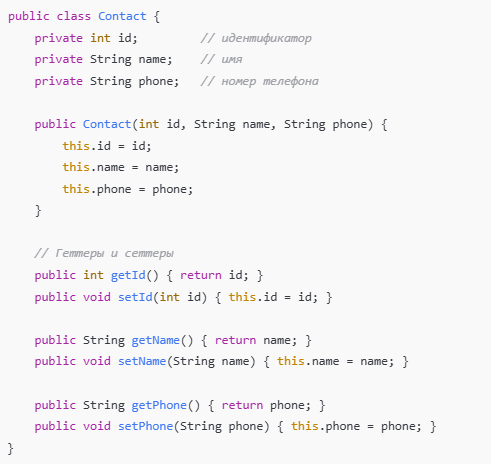


Рисунок 29 – Созданный Java класс

Далее мы создаём отдельный класс-помощник DatabaseHelper для работы с базой данных. Этот класс должен наследовать SQLiteOpenHelper и переопределять как минимум два метода: onCreate() и onUpgrade(). В этом классе будет реализована логика создания и обновления базы.

Метод onCreate(). Этот метод вызывается, когда мы впервые пытаемся получить доступ к базе данных, но она ещё не создана. Внутри него мы формируем SQL-запрос для создания таблицы и выполняем его с помощью execSQL(). Покажем код этого метода на рисунке 30.

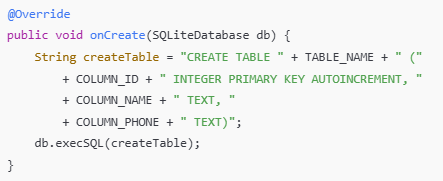


Рисунок 30 – Метод onCreate()

Метод onUpgrade() вызывается, если требуется обновить структуру базы данных, например, при изменении её версии. В простейшем случае мы можем удалить существующую таблицу и создать новую. Покажем код этого метода на рисунке 31.

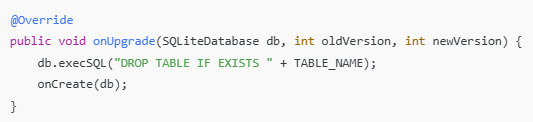


Рисунок 31 – Метод onUpgrade()

Здесь используется команда DROP TABLE, чтобы удалить существующую таблицу перед её пересозданием.

Стоит отметить, что такой способ обновления подходит только для примеров или тестов. В реальных проектах, когда важно сохранить существующие данные, метод onUpgrade() должен содержать более сложную логику: добавление новых столбцов, удаление устаревших, миграция данных и т. д.

Теперь, когда мы создали базу данных и таблицу, следующим шагом будет её наполнение данными.

Кстати, созданную базу можно найти и открыть в Device File Explorer по пути /data/data/имя\_вашего\_пакета/databases.

Для выполнения операций добавления, изменения и удаления данных в SQLite мы используем методы insert(), update() и delete() класса SQLiteDatabase.

Чтобы получить доступ к базе данных, мы можем воспользоваться методами getReadableDatabase() (для чтения) или getWritableDatabase() (для записи). Так как в этом случае мы планируем добавлять данные, используем второй вариант. Покажем на рисунке 32 код для добавления нового контакта.

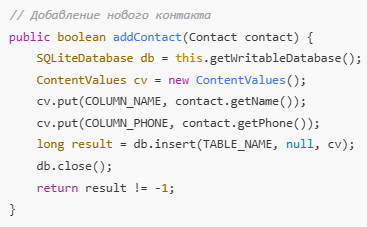


Рисунок 32 – Добавление нового контакта

Для добавления или обновления данных мы создаём объект ContentValues, который представляет собой словарь с парами «ключ–значение». Метод put() позволяет нам добавить данные в этот словарь: первый параметр — это название столбца (ключ), второй — соответствующее значение.

Метод insert() принимает три параметра:

* имя таблицы,
* nullColumnHack — имя столбца, в который можно вставить NULL, если остальные значения отсутствуют (часто передаётся как null),
* объект ContentValues с данными для добавления.

После завершения работы с базой данных мы закрываем соединение с помощью close().

Теперь, на рисунке 33 покажем удаление записи.

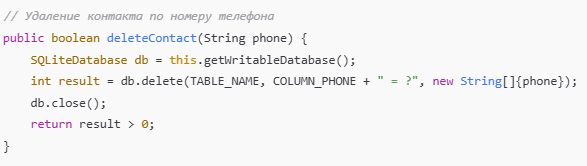


Рисунок 33 – Удаление записи

Метод delete() принимает:

* имя таблицы;
* условие WHERE, в котором ? — это плейсхолдер;
* массив значений, которые будут подставлены вместо ?.

Такой подход позволяет безопасно передавать параметры и защищает от SQL-инъекций.

На рисунке 34 покажем метод поиска записи.

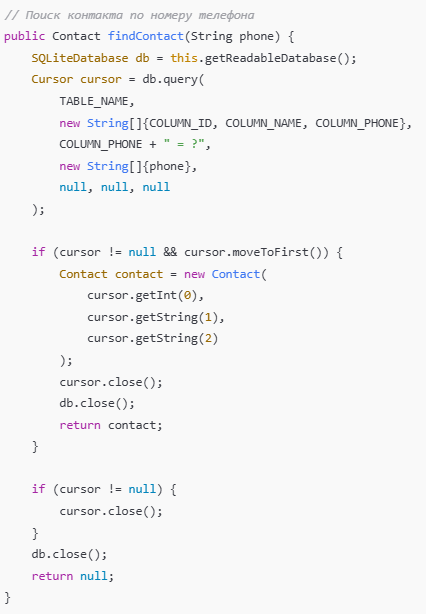


Рисунок 34 – Метод удаления записи

Метод query() позволяет нам выполнять выборку по заданному условию. Если запись найдена, мы создаём объект Contact на основе данных из курсора и возвращаем его. В конце не забываем закрыть Cursor и базу данных.

Когда мы ищем контакт по номеру телефона, нам не нужно вносить изменения в базу данных — достаточно использовать метод getReadableDatabase(), который предоставляет доступ к данным только для чтения.

Для управления результатами выборки мы используем класс Cursor, который предоставляет набор методов:

* moveToFirst() — перемещает курсор к первой строке выборки;
* moveToNext() — переходит к следующей строке;
* методы get\*() (например, getString(), getInt(), getLong()) позволяют получить данные из определённого столбца по его индексу в текущей строке.

Когда работа с курсором завершена, мы обязательно закрываем его с помощью close(), чтобы освободить ресурсы.

На рисунке 35 покажем метод для получения всех записей из базы.



Рисунок 35 – Получение всех записей

Метод rawQuery() выполняет произвольный SQL-запрос и возвращает Cursor с результатами. Сначала мы проверяем, удалось ли получить хотя бы одну строку — если moveToFirst() вернёт false, значит в базе данных нет данных. Далее, используя moveToNext(), мы проходим по всем строкам результата, создавая объекты Contact и добавляя их в список.

На рисунке 36 покажем метод обновления данных.

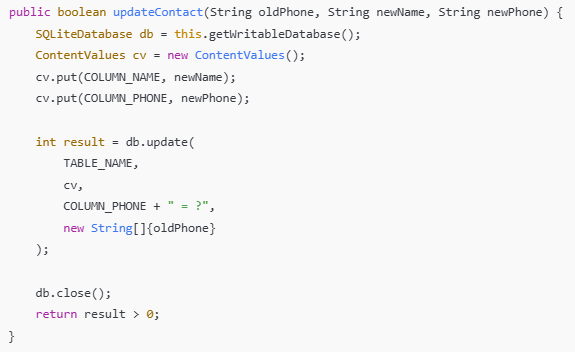


Рисунок 36 – Метод обновления данных

Здесь мы обновляем строку, в которой номер телефона совпадает со старым значением (oldPhone), заменяя имя и номер на новые значения. Возвращаем true, если обновление прошло успешно.

В результате всей работы мы создали полноценный класс DatabaseHelper, который позволяет управлять базой данных SQLite в Android-приложении. Этот класс расширяет SQLiteOpenHelper и инкапсулирует в себе все основные операции: создание базы, добавление, удаление, обновление и поиск контактов.

На рисунке 37 покажем первую часть итогового класса.



Рисунок 37 – Класс DatabaseHelper ч.1

На рисунке 38 покажем вторую часть этого же класса.



Рисунок 38 – Класс DatabaseHelper ч.2

На рисунке 39 покажем финальную третью часть кода этого же файла.



Рисунок 39 – Класс DatabaseHelper ч.3

Затем мы создаём главную активность MainActivity, в которой отображается динамический список всех контактов. При добавлении, обновлении или удалении записи список будет автоматически изменяться на экране.

На рисунке 40 покажем первую часть MainActivity.



Рисунок 40 – Класс MainActivity ч.1

На рисунке 41 покажем вторую часть кода MainActivity.



Рисунок 41 – Класс MainActivity ч.2

На рисунке 42 покажем третью и заключительную часть класса MainActivity.



Рисунок 42 – Класс MainActivity ч.3

Так как мы используем RecyclerView для отображения списка контактов, нам необходимо реализовать собственный адаптер, который будет отвечать за отображение каждого элемента и обновление данных на экране при изменениях в базе.

На рисунке 43 покажем код класса ContactAdapter.



Рисунок 43 – Класс ContactAdapter

На рисунке 44 покажем работоспособность нашей программы.



Рисунок 44 – Работоспособность программы

Даже если перезапустить приложение, то при открытии будет выведен список сохраненных ранее контактов.

Таким образом, получилось готовое приложение для хранения телефонных контактов.

# 2 ПРАКТИЧЕСКАЯ ЧАСТЬ

## 2.1 Реализация записи, получения, изменения и удаления имени пользователя приложения через SharedPreference

Для решения данной задачи на рисунке 1 покажем разметку и интерфейс для activity\_main.xml.

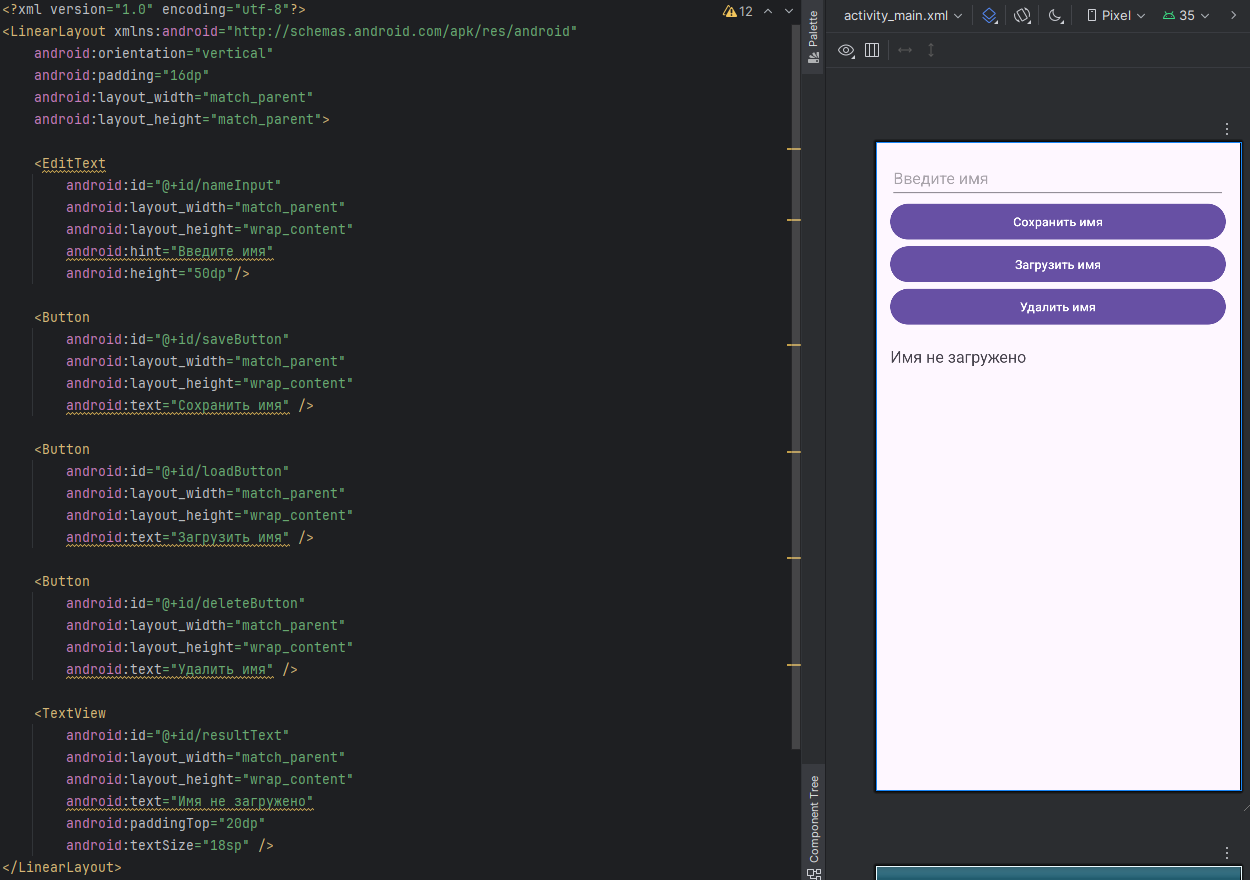


Рисунок 45 – Разметка и интерфейс activity\_main.xml

На рисунке 46 покажем код MainActiivty.



Рисунок 46 – Код MainActivity

На рисунке 47 покажем работоспособность программы.

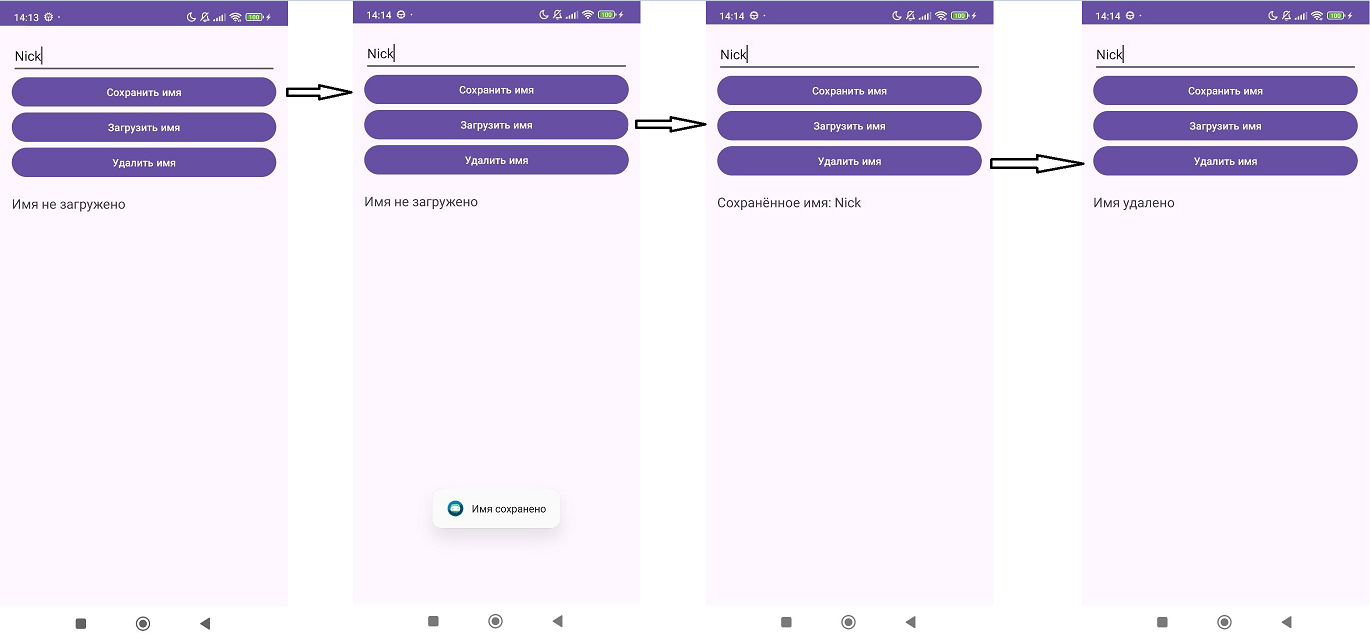


Рисунок 47 – Работоспособность программы

## 2.2 Создание тематической базы данных. Реализация методов записи, поиска, изменения и удаления данных

Для решения данной задачи напишем класс Car, который будет иметь 5 полей, как и требует того задание. Первую часть кода класса Car покажем на рисунке 48.

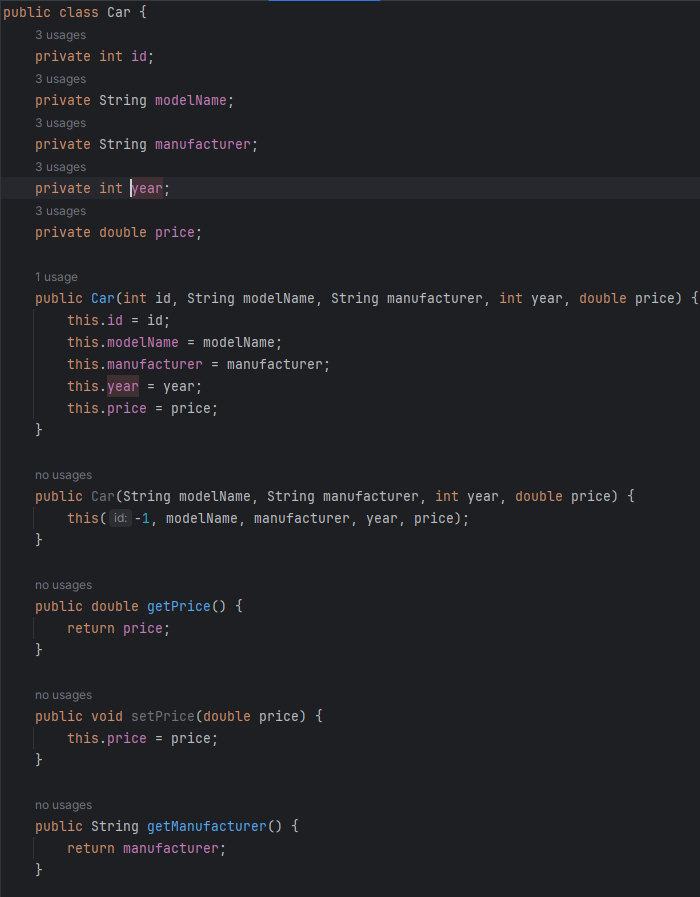


Рисунок 48 – Класс Car ч.1

На рисунке 49 покажем вторую часть кода класса Car.

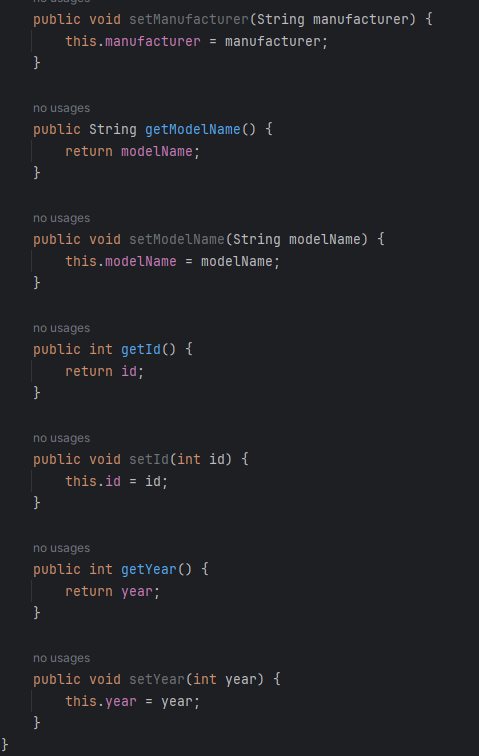


Рисунок 49 – Класс Car ч.2

Далее напишем класс CarDatabaseHelper. На рисунке 50 покажем первую часть кода этого класса.

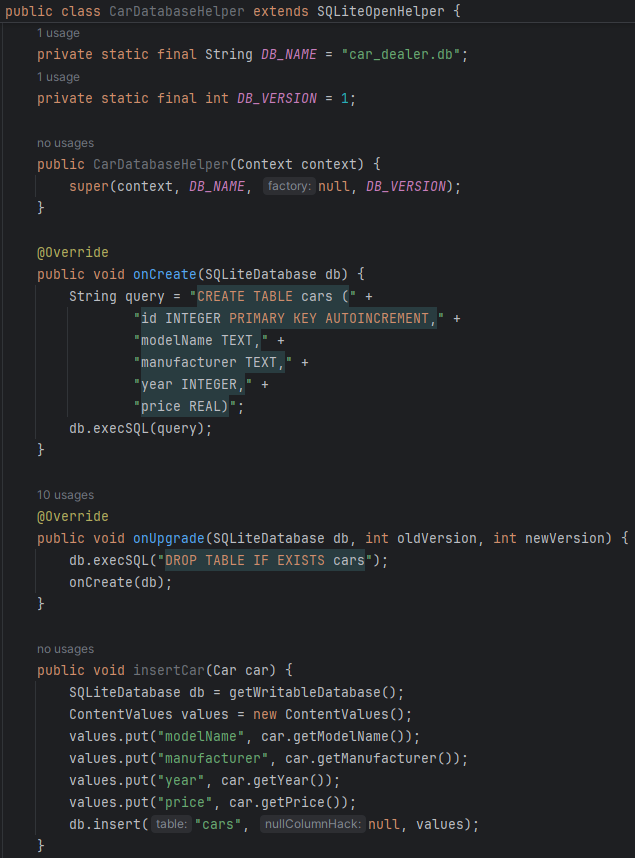


Рисунок 50 – Класс CarDatabaseHelper ч.1

На рисунке 51 покажем вторую часть кода этого же класса.



Рисунок 51 – Класс CarDatabaseHelper ч.2

Теперь перепишем разметку для activity\_main.xml под новые условия и покажем результат на рисунке 52.

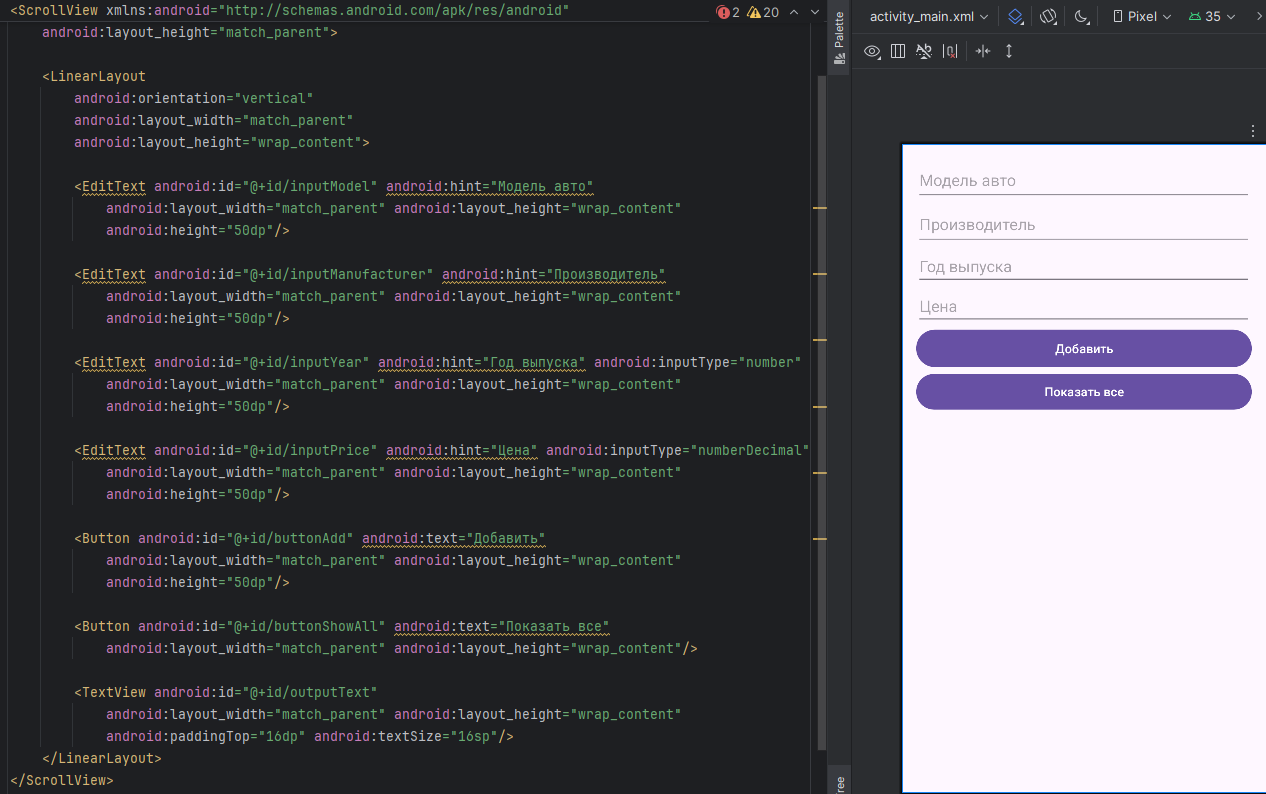


Рисунок 52 – Разметка и интерфейс activity\_main.xml

Далее на рисунке 53 покажем первую часть кода класса MainActivity.

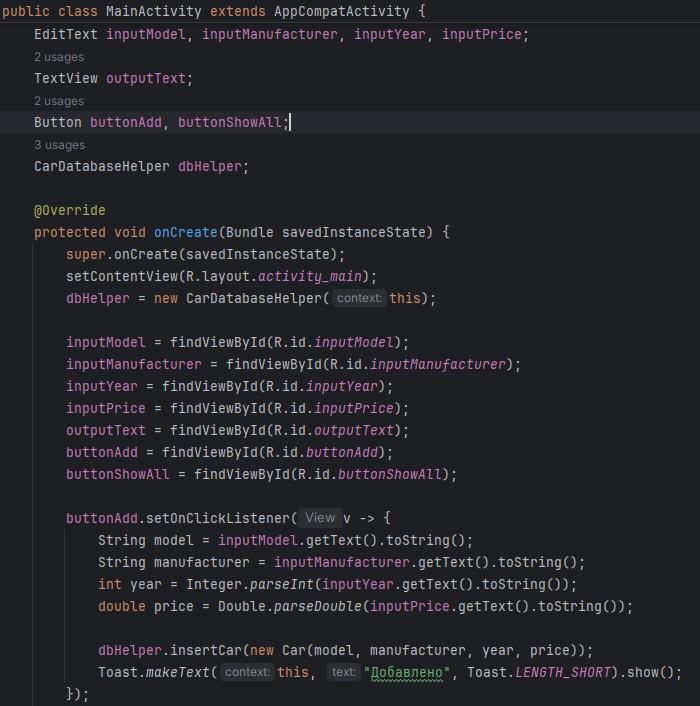


Рисунок 53 – Класс MainActivity ч.1

На рисунке 54 покажем вторую часть кода класса MainActivity.

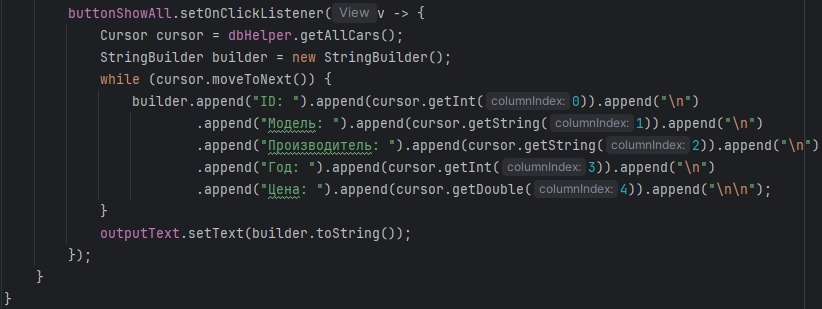


Рисунок 54 – Класс MainActivity ч.2

На рисунке 55 покажем работоспособность программы.

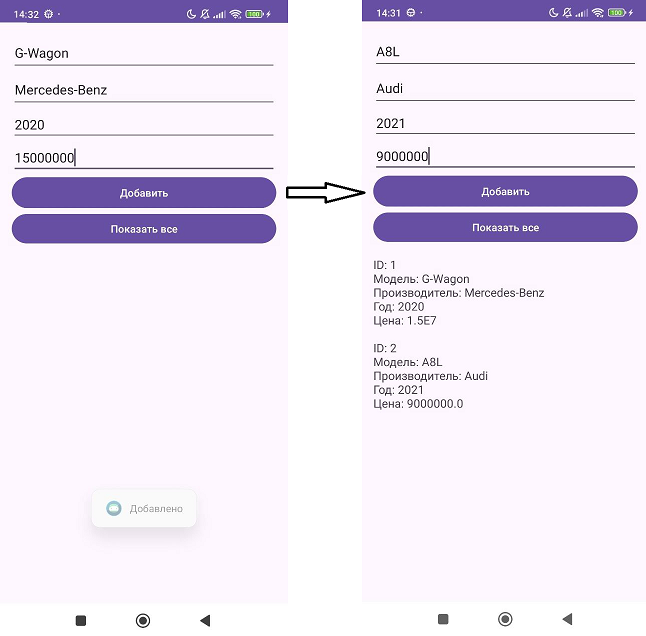


Рисунок 55 – Работоспособность программы

# ЗАКЛЮЧЕНИЕ

В ходе выполнения практической работы были успешно реализованы два способа локального хранения данных в Android: SharedPreferences и база данных SQLite.

Была создана форма для взаимодействия с пользователем, позволяющая сохранять и редактировать имя пользователя с помощью SharedPreferences, что продемонстрировало удобство работы с настройками приложения.

Кроме того, была спроектирована структура базы данных на тему автосалона, где каждая запись содержит информацию о модели автомобиля, производителе, годе выпуска и цене. Пользователь получил возможность выполнять основные операции: добавление, просмотр, изменение и удаление записей.

Практика показала, как комбинировать разные подходы к хранению информации в Android, в зависимости от объёма и характера данных, что является важным шагом к созданию полноценных и устойчивых приложений.