УДК 004

Использование Room для сохранения данных в Android

Андриенко Иван Сергеевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В данной статье описан процесс использования библиотеки Room для сохранения данных в Android-приложении. Рассмотрены ключевые аспекты работы с базой данных Room, включая создание сущностей, определение DAO (Data Access Object) и настройку базы данных. В качестве примера приведена реализация системы хранения данных о тренировках, включающая операции вставки, удаления и выборки данных. Показано, как с помощью Room можно эффективно управлять локальными данными, обеспечивая их сохранность и доступность в приложении.
Ключевые слова: Android, Room, база данных, локальное хранение данных, сущности, DAO, RoomDatabase, сохранение данных, управление данными.

Using Room to save data in Android

Andrienko Ivan Sergeevich
Sholom-Aleichem Priamursky State University
Student

Abstract
This article describes the process of using the Room library to save data in an Android application. The key aspects of working with the Room database are considered, including creating entities, defining a DAO (Data Access Object) and configuring the database. As an example, the implementation of a training data storage system is shown, including operations for inserting, deleting and fetching data. It shows how Room can be used to effectively manage local data, ensuring their safety and accessibility in the application.
Keywords: Android, Rom, Database, Local Data storage, Entities, DAO, Room Database, Data Storage, Data management.

1 Введение

1.1 Актуальность

В условиях современного мобильного приложения важное значение имеет надежное и эффективное управление локальными данными. Room, являясь официальной библиотекой для работы с базами данных в Android, предоставляет удобный и безопасный способ для сохранения и обработки данных. Эта библиотека абстрагирует низкоуровневые детали работы с SQLite, обеспечивая разработчикам более простое и удобное API для

взаимодействия с базой данных. Использование Room способствует улучшению производительности и безопасности приложений, а также упрощает процессы миграции и управления схемой базы данных. В условиях растущих требований к качеству и функциональности мобильных приложений интеграция Room становится все более актуальной и востребованной среди разработчиков.

1.2 Обзор исследований

А. Куанушбек в своей работе исследует разработку приложения для создания базы данных на языке программирования Kotlin для платформы Android. В статье подробно рассматриваются особенности использования Kotlin и преимущества его применения в контексте работы с базами данных [1]. В работе С.А. Проскурина и Л.В. Гаев изучен вопрос эффективности использования памяти при разработке локальной базы данных в Android-приложениях. Авторы анализируют различные методы оптимизации и предлагают подходы для улучшения производительности и снижения потребления памяти [2]. В работе П.А. Сергеева рассматривается использование библиотеки Room для работы с базой данных в Android-приложениях. В статье описываются ключевые преимущества Room, такие как простота интеграции, безопасность данных и удобство миграции схемы базы данных [3]. А.И. Власов, Д.Н. Крамарь и А.А. Цветков анализируют методы хранения и визуальные модели данных при разработке приложений для ОС Android. В статье рассматриваются различные подходы к организации и визуализации данных, что позволяет повысить эффективность работы с ними [4]. О.А. Смелов проводит сравнительный анализ решений для организации локальных баз данных на ОС Android. В статье сравниваются популярные библиотеки и подходы, такие как SQLite, Room и другие, с целью определения их преимуществ и недостатков в различных сценариях использования [5].

1.3 Цель исследования

Цель исследования – разработать эффективный подход к использованию библиотеки Room для сохранения данных в Android-приложении. Это включает в себя создание сущностей, определение DAO (Data Access Object) и настройку базы данных, что позволяет обеспечить надежное и безопасное хранение данных.

2 Материалы и методы

Для исследования использовались Android Studio, язык программирования Java и библиотека Room. Для работы с библиотекой были созданы сущности, DAO (Data Access Object) и база данных с поддержкой миграций.

3 Результаты и обсуждения

Room – это библиотека, являющаяся частью Android Jetpack, которая предоставляет абстракцию поверх SQLite, упрощая работу с базами данных в Android-приложениях. Она позволяет разработчикам работать с объектами Java, используя аннотации для определения сущностей, запросов и миграций, что значительно упрощает управление данными. Room состоит из трех основных компонентов:

1. Entity (Сущность) – класс, представляющий таблицу в базе данных. Каждое поле этого класса становится столбцом таблицы. Например, класс Workout представляет сущность для хранения информации о тренировках, включая идентификатор, дату, продолжительность и вес.
2. DAO (Data Access Object) – интерфейс, содержащий методы для доступа к данным. DAO определяет методы для выполнения операций вставки, обновления, удаления и выборки данных с использованием аннотаций, таких как @Insert, @Delete и @Query. В интерфейсе WorkoutDao определены методы для вставки новой тренировки, удаления по идентификатору и выборки всех тренировок или последних четырех тренировок.
3. Database (База данных) – абстрактный класс, расширяющий RoomDatabase. Этот класс связывает сущности и DAO и управляет базой данных. Метод getDatabase класса WorkoutDatabase реализует паттерн Singleton для обеспечения единственного экземпляра базы данных в приложении. (рис. 1).

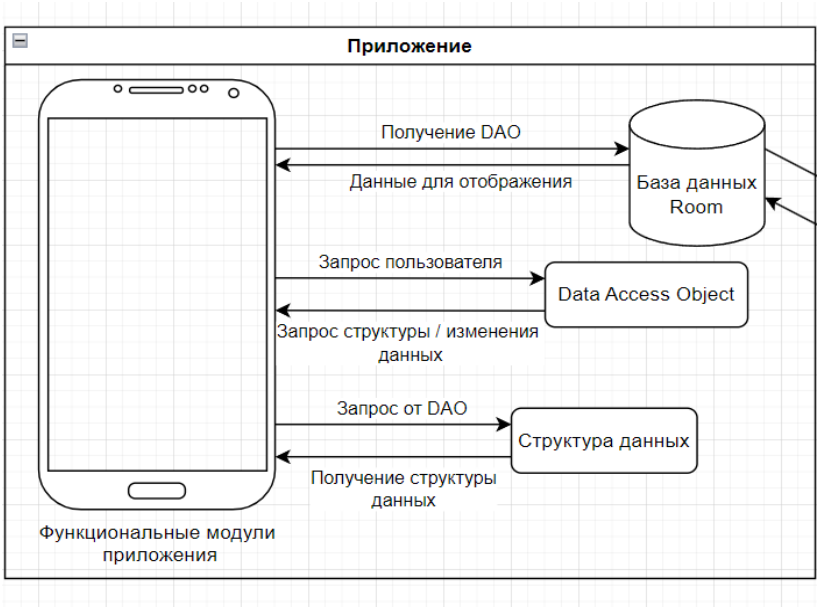


Рис. 1. Схема работы базы данных

Класс Workout представляет собой сущность, которая будет храниться в базе данных. Этот класс аннотирован @Entity, что указывает на то, что он является таблицей в базе данных. В данном примере таблица называется workouts. В классе Workout определены следующие поля: id, уникальный идентификатор записи, который автоматически генерируется; date, дата

тренировки в формате long; duration, продолжительность тренировки в формате String; и weight, вес, связанный с тренировкой в формате double (рис. 2).

```java
package com.modeg.workoutapp.data;

import ...

@Entity(tableName = "workouts")
public class Workout {
    @PrimaryKey(autoGenerate = true)  5 usages
    private int id;
    private long date;  5 usages
    private String duration;  3 usages
    private double weight;  3 usages

    public Workout(long date, String duration, double weight) {
        this.date = date;
        this.duration = duration;
        this.weight = weight;
    }

    public int getId() { return id; }

    public void setId(int id) { this.id = id; }

    public long getDate() { return date; }

    public void setDate(long date) { this.date = date; }

    public String getDuration() { return duration; }

    public void setDuration(String duration) { this.duration = duration; }

    public double getWeight() { return weight; }

    public void setWeight(double weight) { this.weight = weight; }
}
```

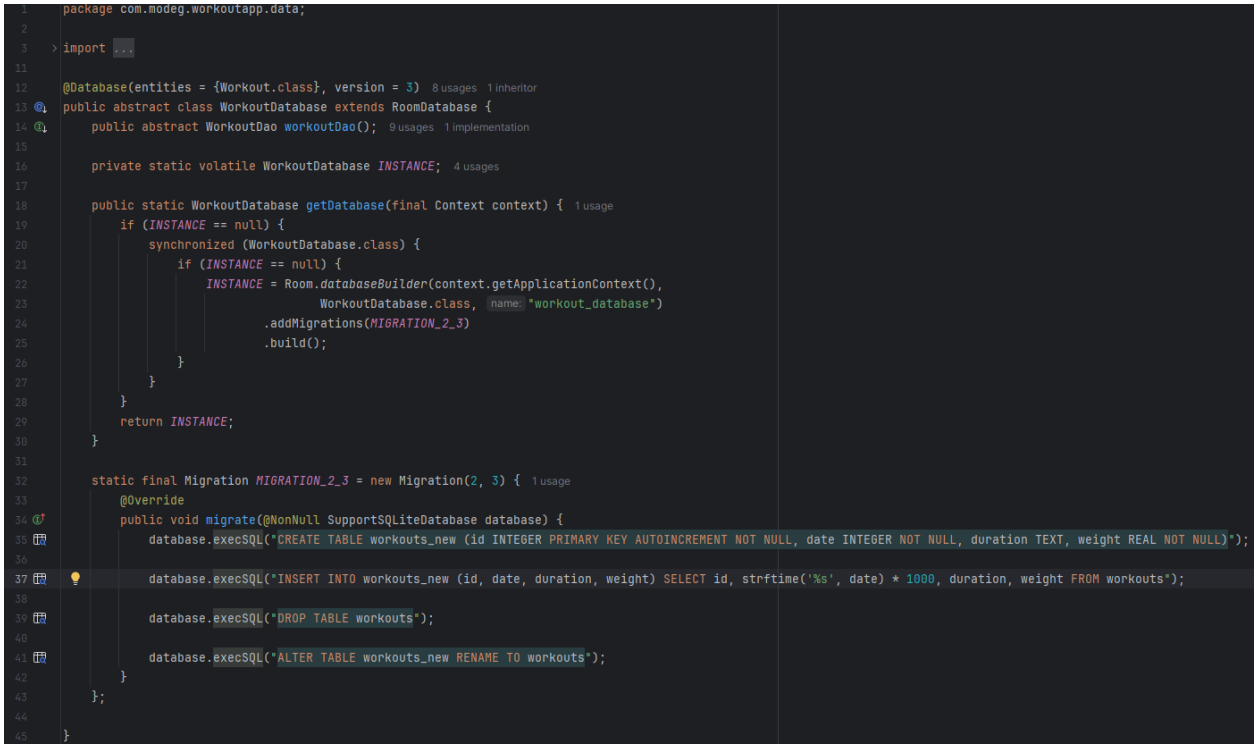
Рис. 2. Создание entity для базы данных

Интерфейс WorkoutDao содержит методы для взаимодействия с таблицей workouts в базе данных. Он аннотирован @Dao и включает методы для вставки новой записи, удаления записи по идентификатору, получения всех записей, получения последних четырех записей и получения записей, дата которых больше или равна указанной. (рис. 3).

```java
package com.modeg.workoutapp.data;

import ...

@Dao  5 usages  1 implementation
public interface WorkoutDao {
    @Insert  1 usage  1 implementation
    void insert(Workout workout);

    @Query("DELETE FROM workouts WHERE id = :id")  1 usage  1 implementation
    void deleteById(int id);

    @Query("SELECT * FROM workouts ORDER BY id DESC")  2 usages  1 implementation
    List<Workout> getAllWorkouts();

    @Query("SELECT * FROM workouts ORDER BY id DESC LIMIT 4")  2 usages  1 implementation
    List<Workout> getLastFourWorkouts();

    @Query("SELECT * FROM workouts WHERE date >= :date ORDER BY date ASC")  3 usages  1 implementation
    List<Workout> getWorkoutsFromDate(long date);
}
```

Рис. 3. Создание Data Access Object для БД

Абстрактный класс WorkoutDatabase расширяет RoomDatabase и представляет собой базу данных для приложения. Он аннотирован @Database с указанием сущностей и версии базы данных. Класс включает метод workoutDao(), который возвращает объект WorkoutDao для взаимодействия с таблицей workouts, и статический метод getDatabase(final Context context), который обеспечивает создание и получение экземпляра базы данных, используя паттерн Singleton. Кроме того, включена поддержка миграций, таких как переход с версии 2 на версию 3, для управления изменениями схемы базы данных (рис. 4).



```
package com.modeg.workoutapp.data;

import ...

@Database(entities = {Workout.class}, version = 3)  8 usages  1 inheritor
public abstract class WorkoutDatabase extends RoomDatabase {
    public abstract WorkoutDao workoutDao();  9 usages  1 implementation

    private static volatile WorkoutDatabase INSTANCE;  4 usages

    public static WorkoutDatabase getDatabase(final Context context) {  1 usage
        if (INSTANCE == null) {
            synchronized (WorkoutDatabase.class) {
                if (INSTANCE == null) {
                    INSTANCE = Room.databaseBuilder(context.getApplicationContext(),
                                    WorkoutDatabase.class,  name: "workout_database")
                            .addMigrations(MIGRATION_2_3)
                            .build();
                }
            }
        }
        return INSTANCE;
    }

    static final Migration MIGRATION_2_3 = new Migration(2, 3) {  1 usage
        @Override
        public void migrate(@NonNull SupportSQLiteDatabase database) {
            database.execSQL("CREATE TABLE workouts_new (id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, date INTEGER NOT NULL, duration TEXT, weight REAL NOT NULL)");

            database.execSQL("INSERT INTO workouts_new (id, date, duration, weight) SELECT id, strftime('%s', date) * 1000, duration, weight FROM workouts");

            database.execSQL("DROP TABLE workouts");

            database.execSQL("ALTER TABLE workouts_new RENAME TO workouts");
        }
    };

}
```

Рис. 4. Создание класса WorkoutDatabase.java

После настройки базы данных Room следующим шагом является использование и сохранение данных в приложении. Для создания и сохранения новой записи в таблице workouts используется метод insert интерфейса WorkoutDao. В классе DashboardViewModel создается метод addWorkout, который принимает параметры даты, продолжительности и веса, создает новый объект Workout и сохраняет его в базе данных (рис. 5).

```
    public void addWorkout(String date, String duration, double weight) {  1 usage
        executorService.execute(() -> {
            try {
                SimpleDateFormat sdf = new SimpleDateFormat( pattern: "yyyy-MM-dd", Locale.getDefault());
                Date parsedDate = sdf.parse(date);
                Workout workout = new Workout(parsedDate.getTime(), duration, weight);
                workoutDatabase.workoutDao().insert(workout);
                // Загрузка всех данных после добавления новой тренировки
                List<Workout> allWorkoutsList = workoutDatabase.workoutDao().getAllWorkouts();
                List<Workout> lastFourWorkoutsList = workoutDatabase.workoutDao().getLastFourWorkouts();
                if (!allWorkoutsList.isEmpty()) {
                    lastWorkout.postValue(allWorkoutsList.get(0)); // Обновление последней тренировки
                }
                updateWorkoutSummary(allWorkoutsList);
                lastWorkouts.postValue(convertToStringList(lastFourWorkoutsList));
                allWorkouts.postValue(allWorkoutsList);
                // Обновление данных для графика
                Calendar cal = Calendar.getInstance();
                cal.add(Calendar.MONTH,  amount: -1);
                long oneMonthAgo = cal.getTimeInMillis();
                List<Workout> monthlyWorkoutsList = workoutDatabase.workoutDao().getWorkoutsFromDate(oneMonthAgo);
                monthlyWeightData.postValue(monthlyWorkoutsList);
            } catch (ParseException e) {
                e.printStackTrace();
            }
        });
    }
```

Рис. 5. Создание метода для добавления записи в БД

Для получения данных из таблицы workouts используются методы getAllWorkouts, getLastFourWorkouts и getWorkoutsFromDate интерфейса WorkoutDao. В DashboardViewModel создается метод loadWorkouts, который загружает все тренировки и обновляет соответствующие LiveData объекты (рис. 6).

```
    private void loadWorkouts() {  2 usages
        executorService.execute(() -> {
            List<Workout> allWorkoutsList = workoutDatabase.workoutDao().getAllWorkouts();
            List<Workout> lastFourWorkoutsList = workoutDatabase.workoutDao().getLastFourWorkouts();
            if (!allWorkoutsList.isEmpty()) {
                lastWorkout.postValue(allWorkoutsList.get(0)); // Обновление последней тренировки
            }
            updateWorkoutSummary(allWorkoutsList);
            lastWorkouts.postValue(convertToStringList(lastFourWorkoutsList));
            allWorkouts.postValue(allWorkoutsList);

            // Обновление данных для графика
            Calendar cal = Calendar.getInstance();
            cal.add(Calendar.MONTH,  amount: -1);
            long oneMonthAgo = cal.getTimeInMillis();
            List<Workout> monthlyWorkoutsList = workoutDatabase.workoutDao().getWorkoutsFromDate(oneMonthAgo);
            monthlyWeightData.postValue(monthlyWorkoutsList);
        });
    }
```

Рис. 6. Создание метода для загрузки записей из БД

Для удаления записи из таблицы используется метод deleteById интерфейса WorkoutDao. В DashboardViewModel создается метод deleteWorkoutById, который удаляет запись по идентификатору и обновляет данные (рис. 7).

```
60      public void deleteWorkoutById(int id) {  2 usages
61          executorService.execute(() -> {
62              workoutDatabase.workoutDao().deleteById(id);
63              loadWorkouts(); // Обновление данных после удаления
64          });
65      }
```

Рис. 7. Создание метода для удаления записей

После реализации классов записи успешно создаются, выводятся и удаляются (рис. 8).

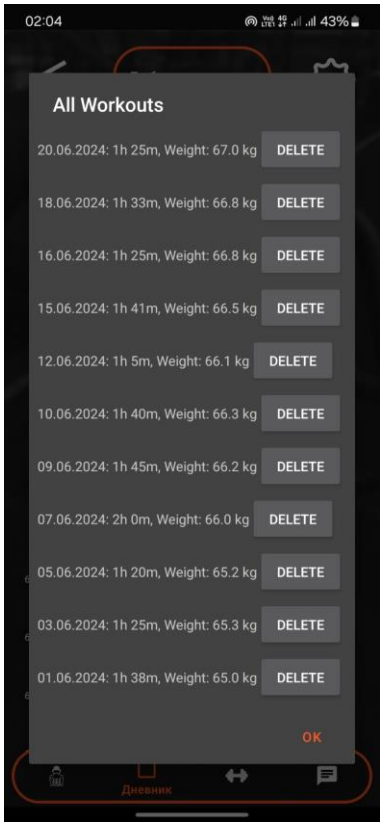


Рис. 8. Вывод из базы данных

Выводы

В данной статье представлен процесс использования библиотеки Room для сохранения данных в Android-приложении. Рассмотрены ключевые аспекты, включая создание сущностей, определение DAO и настройку базы данных. Приведены примеры реализации, которые демонстрируют, как библиотека Room упрощает работу с локальными базами данных. Использование Room обеспечивает надежное и эффективное управление данными, позволяя разработчикам легко сохранять, обновлять и получать данные. Библиотека предоставляет мощный инструмент для разработки надежных и производительных Android-приложений, способных эффективно управлять локальными данными.

Библиографический список

1. Kuanyshbek A. Research and develop application for creating a database in the Kotlin programming language based on Android // Научному прогрессу – творчество молодых. 2022. № 1. С. 336-338.
2. Проскурина С.А., Гаев Л.В. Исследование вопроса эффективности использования памяти при разработке локальной базы данных в Android-приложениях // В сборнике: Генезис и онтология инновационно ориентированной политики в условиях цифровизации. сборник статей Международной научно-практической конференции. Уфа, 2022. С. 18-20.
3. Сергеев П.А. Использование библиотеки Room для работы с базой данных Android-приложения // В сборнике: Научные исследования и современное образование. Сборник материалов IX Международной научно-практической конференции. Редколлегия: О.Н. Широков [и др.]. 2019. С. 127-128.
4. Власов А.И., Крамарь Д.Н., Цветков А.А. Анализ методов хранения и визуальных моделей данных при разработке приложений для ОС Android // Автоматизация. Современные технологии. 2023. Т. 77. № 4. С. 181-187.
5. Смелов О.А. Сравнительный анализ решений для организации локальных баз данных на ОС Android // Интернаука. 2021. № 31-1 (207). С. 14-16.
6. URL: https://developer.android.com/training/data-storage/room (дата обращения 25.06.2024)