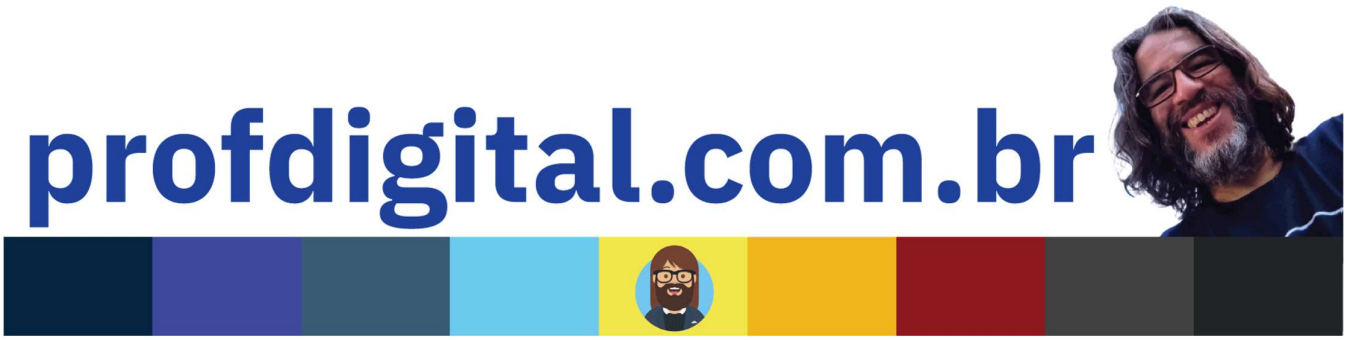
Desenvolvimento de aplicativo para o Sistema Operacional Android – Biblioteca de Persistência Room

Marcos Vinícius da Silva Santos e Marcos Antonio dos Santos



ISENÇÃO DE RESPONSABILIDADE E OUTRAS INFORMAÇÕES LEGAIS

Todas as opiniões quando expressas neste documento ou no site em que é apresentado este documento não representam as opiniões de nenhuma entidade com a qual os autores tenham sido associados, são agora associados ou serão associados futuramente.

Qualquer ação envolvendo programação que você execute com as informações deste documento é feita estritamente sob a sua responsabilidade para o desenvolvimento de habilidades técnicas e profissionais, portanto, os autores não se responsabilizam por quaisquer erros ou omissões, ou pelos resultados obtidos com o uso das informações contidas nesse documento.

Todas as informações neste documento são fornecidas "no estado em que se encontram", sem a garantia dos resultados obtidos com o uso destas informações. Parte das informações aqui apresentadas foram obtidas em diversas fontes e, mesmo com todo o cuidado em sua coleta e manuseio, os autores não se responsabilizam pela publicação acidental de dados incorretos que possam levar a qualquer tipo de dano ou prejuízo.

Os autores recomendam que em caso de dúvidas o utilizador deste documento busque saná-las recorrendo à documentação oficial do fabricante que é disponibilizada nos respectivos sites.

Room

Faremos uso da biblioteca de persistência de dados Room que oferece uma camada de abstração sobre o SQlite para permitir acesso fluente e robusto ao banco de dados.

Room é uma das bibliotecas existentes dentro do conjunto “Android JetPack” apresentado durante o Google I/O de 2018, ela auxilia os desenvolvedores criando uma abstração da camada de banco de dados. Essa camada irá em nosso app utilizar o banco SQLite.

Aplicativos que processam quantidades não triviais de dados estruturados podem se beneficiar muito da persistência desses dados localmente, ou seja, com o uso de um banco de dados criado em nível de dispositivo físico.

O caso de uso mais comum é armazenar as partes de dados relevantes em cache (na memória física do dispositivo). Dessa forma, quando o dispositivo não conseguir acessar a rede, o usuário ainda poderá navegar pelo conteúdo enquanto estiver off-line. Todas as alterações de conteúdo feitas pelo usuário serão sincronizadas com o servidor quando o dispositivo ficar on-line novamente. Portanto, é altamente recomendável usar o Room em vez do SQLite “puro”, porque o Room cuida desses problemas para você.

É necessário importar a dependência Room para o projeto do aplicativo em Build.Gradle(:app). O link https://bit.ly/3liLXXz apresenta como fazer isso.

Para o correto desenvolvimento do seu aplicativo e fazendo uso da linguagem Groove temos abaixo a declaração de dependência necessária:

```
dependencies {
...
  //ROOM banco de dados SQLite
    def room_version = "2.3.0"
    implementation "androidx.room:room-runtime:$room_version"
    annotationProcessor "androidx.room:room-compiler:$room_version"
    testImplementation "androidx.room:room-testing:$room_version"
...
}
```

Anotação @Entity

Por fazermos uso da Room teremos que possuir um conjunto de dados relacionados que são definidos como entidades, ou seja, para cada entidade, uma tabela é criada no banco de dados (objeto Database) associado para armazenar os itens.

Tais entidades são classes em linguagem Java que possuem os respectivos métodos assessores e construtor da classe e além da anotação inicial @Entity.

Veja o exemplo da codificação em linguagem Java que representa a tabela ‘tasks’:

```
package com.example.gerenciamentodetarefas2g;

import androidx.annotation.NonNull;
import androidx.room.ColumnInfo;
import androidx.room.Entity;
import androidx.room.PrimaryKey;

@Entity(tableName = "tasks")
public class Task {

    @PrimaryKey(autoGenerate = true)
    @NonNull
    @ColumnInfo(name="taskId")
    private int mId;

    @ColumnInfo(name = "descriptionTask")
    private String mDescription;

    public int getId() {
        return mId;
    }

    public void setId(int id) {
        mId = id;
    }

    public String getDescription() {
        return mDescription;
    }

    public void setDescription(String description) {
        mDescription = description;
    }

    public String getNote() {
```

```
    return mNote;
}

public void setNote(String note) {
    mNote = note;
}

public float getPriorityLevel() {
    return mPriorityLevel;
}

public void setPriorityLevel(float priorityLevel) {
    mPriorityLevel = priorityLevel;
}

public long getEstimatedDate() {
    return mEstimatedDate;
}

public void setEstimatedDate(long estimatedDate) {
    mEstimatedDate = estimatedDate;
}

public long getInsertionDate() {
    return mInsertionDate;
}

public void setInsertionDate(long insertionDate) {
    mInsertionDate = insertionDate;
}

public long getUpdateDate() {
    return mUpdateDate;
}

public void setUpdateDate(long updateDate) {
```

```
        mUpdateDate = updateDate;
    }

    public int getIsDeleted() {
        return mIsDeleted;
    }

    public void setIsDeleted(int isDeleted) {
        mIsDeleted = isDeleted;
    }

    public int getIsFinished() {
        return mIsFinished;
    }

    public void setIsFinished(int isFinished) {
        mIsFinished = isFinished;
    }

    private String mNote;
    private float mPriorityLevel;
    private long mEstimatedDate;
    private long mInsertionDate;
    private long mUpdateDate;
    private int mIsDeleted;
    private int mIsFinished;

    public Task(String description, String note, float priorityLevel, long estimatedDate,
long insertionDate, long updateDate, int isDeleted, int isFinished) {
        mDescription = description;
        mNote = note;
        mPriorityLevel = priorityLevel;
        mEstimatedDate = estimatedDate;
        mInsertionDate = insertionDate;
        mUpdateDate = updateDate;
        mIsDeleted = isDeleted;
        mIsFinished = isFinished;
```

```
    }
}
```

Num próximo momento o nome da tabela 'tasks' deverá ser informado no objeto Database. Se após a primeira execução com sucesso do aplicativo haver a necessidade de alguma mudança na estrutura de dados que foi definida AQUI nesta classe 'Task' uma nova versão do banco de dados deverá ser informada na classe 'DbRoomDatabase', afinal de contas é necessário avisar na classe 'DbRoomDatabase' que uma mudança ocorreu aqui. IMPORTANTE: A classe 'DbRoomDatabase' que irá representar o objeto Database não foi codificada até este momento.

Em resumo a anotação **@Entity cria uma tabela para cada classe** com a qual está anotada e, portanto, a própria classe é essencialmente uma **classe de dados que não contém lógica**. Essa anotação indica que a classe de dados deve ser mapeada para um banco de dados.

A classe Task utilizada como exemplo

A estrutura de dados definida AQUI para esta classe/entidade faz parte do seu estudo sobre modelagem de dados. Essa modelagem é originária de muitos estudos e análises realizada junto ao seu cliente com o levantamento de todas as necessidades, dados e requisitos necessários, inclusive com pesquisas externas, para o desenvolvimento do aplicativo buscando atender a máxima defendida pelo seu professor que é "Nós temos a solução para o seu problema".

O cliente deste aplicativo é o Sr. Santos que tem uma agenda em papel com vários espaços para anotações onde ele registra os livros, museus, filmes, tarefas (tasks) e etc. Santos faz avalições destes e outros temas. Há um espaço para avaliação diária de questões ligadas ao seu dia-a-dia. Relatórios com estatísticas e alertas serão disponibilizados para o aplicativo. Desta breve descrição veio o nome do aplicativo - 'GERENCIAMENTO DE TAREFAS'.

O que é uma interface em Java?

Como você já aprendeu, os objetos definem suas interações com o mundo exterior por meio dos métodos que expõem o que é possível de fazer. Os métodos formam a interface do objeto com o mundo exterior; os botões na frente de seu aparelho de televisão ou do seu controle remoto, por exemplo, são a interface entre você e a fiação elétrica do outro lado de sua caixa 'mágica' de plástico. Você pressiona o botão "power" para ligar e desligar a televisão e nem precisa saber como isso acontece.

Em sua forma mais comum, uma interface é um grupo de métodos relacionados com blocos de códigos vazios. Para nossos estudos e considerando a Room faremos uso de algumas anotações como @Dao, @Query, @Insert e etc.

Fazer uso de uma interface, ou seja, **implementar uma interface permite que uma classe se torne mais formal sobre o comportamento que promete fornecer**, o que de certo modo estabelece um contrato de comunicação entre a classe e o mundo exterior.

Veja a implementação da interface 'TaskDao':

```
package com.example.gerenciamentodetarefas2g;

import androidx.lifecycle.LiveData;
import androidx.room.Dao;
import androidx.room.Delete;
import androidx.room.Insert;
import androidx.room.Query;
import androidx.room.Update;

import java.util.List;

@Dao
public interface TaskDao {

    @Insert
    void insertTask(Task task);

    @Update
    void updateTask(Task task);

    @Delete
    void deleteTask(Task task);

    @Query("DELETE FROM tasks")
    void deleteAllTasks();

    @Query("SELECT * FROM tasks ORDER BY UPPER(descriptionTask) ASC")
    LiveData<List<Task>> loadAllTasks();
```

```
    @Query("SELECT * FROM tasks ORDER BY UPPER(descriptionTask) DESC")
    LiveData<List<Task>> loadAllTasksDescending();

    @Query("SELECT * FROM tasks WHERE mIsDeleted = 0 ORDER BY UPPER(descriptionTask) ASC")
    LiveData<List<Task>> loadActiveTasks();

    @Query("SELECT * FROM tasks WHERE mIsDeleted = 1 ORDER BY UPPER(descriptionTask) ASC")
    LiveData<List<Task>> loadInactiveTasks();

    @Query("SELECT * FROM tasks WHERE descriptionTask LIKE :search")
    LiveData<List<Task>> findTasksLikePattern(String search);

    @Query("SELECT * FROM tasks WHERE descriptionTask MATCH :search")
    LiveData<List<Task>> findTasksMatch(String search);

    @Query("SELECT COUNT(*) FROM tasks WHERE mIsFinished = 0")
    int totalTasksNotFinished();

    @Query("SELECT COUNT(*) FROM tasks")
    int totalTasks();

}
```

A interface TaskDao utilizada como exemplo

Uma aplicação interage com uma data source (fonte de dados) por meio de operações CRUD (create, retrieve, update e delete). Essas operações permitem criar (C), recuperar (R), atualizar (U) e deletar (D) objetos.

As fontes de dados podem ser as mais variadas com destaque para:

- banco de dados relacional;
- banco orientado a objetos;
- banco de dados NoSQL;
- sistema de arquivos;
- web services baseados nos protocolos SOAP ou REST;
- repositório baseado no protocolo LDAP (Lightweight Directory Access Protocol - exemplo de LDAP uma lista telefônica = diretório de telefones).

Desta necessidade de interação entre a classe Dao e a fonte de dados, iremos trabalhar com a nossa Dao funcionando como uma Interface.

Para compreender o conceito de Interface pense na classe Dao como sendo o 'controle remoto' para ligar a TV (TV será o banco de dados) e a sua escolha no controle remoto (escolha serão os dados).

O padrão Data Access Object (DAO, Objetos de acesso a dados) tem como objetivo encapsular o acesso ao data source fornecendo uma interface para que as diversas outras camadas da aplicação possam se comunicar com o data source.

Objetos de acesso a dados são as principais classes nas quais você define suas interações com o banco de dados. Eles podem incluir uma variedade de métodos de consulta, inserção, atualização e exclusão física das linhas de uma tabela.

É recomendável ter várias classes Dao na sua base de código, uma para cada entidade.

Anotação @Dao

A classe marcada com @Dao deve ser uma interface ou uma classe abstrata para funcionar com a Room. Em tempo de compilação, o Room irá gerar uma implementação dessa classe quando for referenciada para utilizar a fonte de dados com o insert, update, delete e select.

Dito de forma mais simples a anotação @Dao **marca a classe como sendo um objeto de acesso a dados**.

Sugestão de leitura

- https://github.com/1268marcos/youtube_android/blob/master/Museum.java
- Saiba mais sobre como aplicar e usar os recursos da Room no link https://bit.ly/3xh7Gl9
- https://medium.com/@kinnerapriyap/entity-embedded-and-composite-primary-keys-with-room-db-8cb6ca6256e8
- https://github.com/1268marcos/youtube_android/blob/master/MuseumDao.java
- https://developer.android.com/reference/androidx/room/Dao