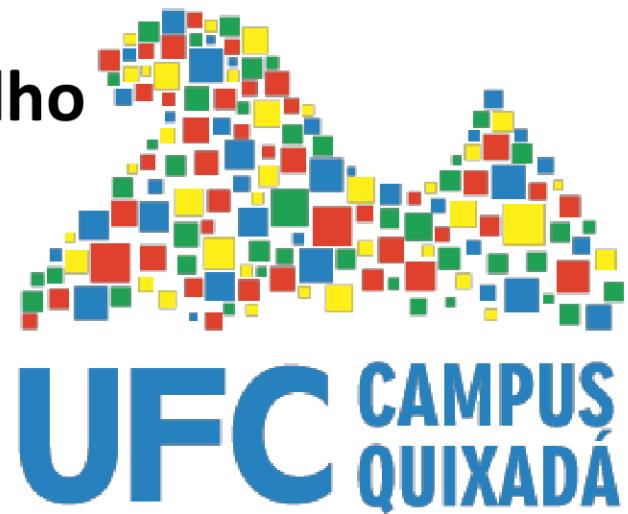


Desenvolvimento de software para Dispositivos Móveis

**CRUD (Create, Read, Update, Delete) com ROOM
Database**



Prof. Sidartha Carvalho



ROOM Database

ROOM DB

- Room é uma biblioteca de persistencia de dados que provê uma camada de abstração sob o banco de dados relacional SQLite. Com essa abstração é possível operar o banco de dados relacional de forma simplificada usando classes.



ROOM DB: Principais componentes

- Entity
 - Classe que serve de modelo para representar uma tabela no banco de dados. A classe que for anotada com `@Entity` será convertida em uma tabela no banco de dados SQLite nativo do Android.
- Database
 - Classe abstrata que guarda os acessos ao banco de dados e onde podemos chamar as `@Entity` criadas.
- DAO (Data Access Object)
 - Interface que permite acessar os dados usando classes, permitindo realizar operações (queries SQL) de forma facilitada.

ROOM DB: Macro passos

1. Adicionar ROOM db no build.gradle.kts
(module: app)
2. Criar o layout da MainActivity
3. Criar a classe de Modelo (@Entity)
4. Criar a classe de acesso aos dados – DAO
5. Criar database
6. Usar o banco de dados na MainActivity

1. Dependencias

No arquivo dentro do seu projeto:

Gradle Scripts > build.gradle.kts (module: app)

adicionar:

```
// dependencia para usar o room db.  
val room_version = "2.6.1"  
implementation("androidx.room:room-runtime:$room_version")  
annotationProcessor("androidx.room:room-compiler:$room_version")
```

2. MainActivity XML

- Adicionar 2 componentes:
 - RecyclerView para listar os itens
 - Floating Action Button para ações.



2. MainActivity XML

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/rv_itens"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_marginStart="1dp"
        android:layout_marginTop="1dp"
        android:layout_marginEnd="1dp"
        android:layout_marginBottom="1dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <com.google.android.material.floatingactionbutton.FloatingActionButton
        android:id="@+id/fab_add"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginEnd="14dp"
        android:layout_marginBottom="25dp"
        android:clickable="true"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:srcCompat="@android:drawable/ic_input_add" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

3. Classe de Modelo (@Entity)

- Criar a classe Pessoa e adicionar o @Entity acima dela, indicando que é uma classe representativa do banco de dados.
 - Atributos: Id, Nome, Curso e Idade
- Criar o construtor sem incluir o id, pois ele será gerado automaticamente.
- Usar @Entity(tableName = "pessoa") para especificar o nome da tabela no banco de dados.
- Usar o @PrimaryKey(autoGenerate = true) para criar um Id automático no banco de dados.

3. Classe de Modelo (@Entity)

```
@Entity(tableName = "person")
public class Person {
    @PrimaryKey
    public int uid;

    @ColumnInfo(name = "first_name")
    public String firstName;

    @ColumnInfo(name = "last_name")
    public String lastName;
}
```

4. Interface DAO

- Esta interface irá incluir as operações que poderão ser realizadas no banco de dados
 - Adicionar, ler, atualizar e deletar os dados
 - Também é possível criar queries SQL customizadas para outras necessidades
- Criar uma interface PessoaDao
- Usar as anotações @Insert, @Update, @Delete e @Query("SQL AQUI")
- O @Insert e similares criam a query SQL automaticamente

4. DAO

```
@Dao
public interface PersonDao {
    @Query("SELECT * FROM person")
    List<Person> getAll();

    @Query("SELECT * FROM person WHERE uid IN (:userIds)")
    List<Person> loadAllByIds(int[] userIds);

    @Query("SELECT * FROM person WHERE first_name LIKE :first AND " +
        "last_name LIKE :last LIMIT 1")
    Person findByName(String first, String last);

    @Insert
    void insertAll(Person... users);

    @Delete
    void delete(Person user);
}
```

5. Database

- Criar classe abstrata (extends RoomDatabase)
- Definir @Database
- Implementar método getInstance()
 - Opcional – Singleton design pattern

5. Database

```
@Database(entities = {Person.class}, version = 1)
public abstract class AppDatabase extends RoomDatabase {
    public abstract PersonDao personDao();
}
```

6. Usar database

- Na MainActivity vamos criar uma conexão ao banco de dados, criar uma Pessoa, adicionar ao banco de dados e exibir todas as Pessoas cadastradas.

6. Usar database

```
Person p1 = new Person("Sidartha", "Carvalho");

AppDatabase db = Room.databaseBuilder(getApplicationContext(),
    AppDatabase.class, "database-name")
    .enableMultiInstanceInvalidation()
    .allowMainThreadQueries()
    .fallbackToDestructiveMigration().build();

PersonDao personDao = db.personDao();

personDao.insertAll(p1);

List<Person> persons = personDao.getAll();

for (Person p: persons) {
    Log.d("sid-tag", p.toString());
}

}
```

6. Usar database

- Perceba que chamamos alguns métodos no momento de criação da base de dados
- `enableMultiInstanceInvalidation()`
 - Permite somente uma instância da base de dados
- `allowMainThreadQueries()`
 - Permite executar queries sem usar Threads, podendo travar a interface do usuário
- `.fallbackToDestructiveMigration()`
 - Permite recriar o banco de dados se o Model for alterado

FINALIZAMOS

- Agora você já possui acesso a uma instância do banco SQLite usando Room database.
- Agora você pode adicionar, remover, atualizar e realizar queries SQL da forma que preferir.

ROOM Database + RecyclerView

ROOM DB + RecyclerView

- Agora você já viu como usar o RecyclerView para **listar** itens no seu aplicativo Android, que tal revisitar os projetos anteriores e relembrar como faz?

ROOM DB + RecyclerView



ROOM DB + RecyclerView

1. Alterar a MainActivity XML para incluir o RecyclerView
2. Criar o item.xml e o item.java
3. Criar o adapter
4. Iniciar o RecyclerView e setar o adapter na MainActivity
5. Resgatar os dados do Banco de Dados e popular a RecyclerView a partir deles

ROOM DB + RecyclerView

- NOVAS FUNCIONALIDADES
 1. Incluir novo item na lista ao clicar no botão “ + ”
 2. Adicionar Swipe para remover item da lista
 3. Exibir Toast ao tentar inserir dados inválidos

Agora você vai ter que pensar um pouco ao achar um problema no código 😊

- Usar o projeto fornecido como modelo

ROOM DB + RecyclerView

```
private void swipeToDelete() {
    //Swipe to delete feature
    new ItemTouchHelper(new ItemTouchHelper.SimpleCallback(0, ItemTouchHelper.LEFT) {
        @Override
        public boolean onMove(@NonNull RecyclerView recyclerView, @NonNull
        RecyclerView.ViewHolder viewHolder, @NonNull RecyclerView.ViewHolder target) {
            return false;
        }

        @Override
        public void onSwiped(@NonNull RecyclerView.ViewHolder viewHolder, int direction) {
        }
    }).attachToRecyclerView(rv_itens);
}
```

Referências

- <https://developer.android.com/>

Dúvidas?



www.shutterstock.com · 744867163