

Programação para Dispositivos Móveis I

Prof. Marcos Vasconcelos de Oliveira

Marcos.vasconcelos4@fatec.sp.gov.br



Aula 14 – App 06 – Lista de Compras com Banco de Dados - Persistência de Dados

Prof. Marcos Vasconcelos de Oliveira

Marcos.vasconcelos4@fatec.sp.gov.br



Tela do Aplicativo

- Criar o Banco de Dados
- Criar a tabela: Produto
- Gravar na Tabela



Recursos Utilizados ???

- SQLite
- Anko

SQLite



- Quando criamos um banco de dados para o nosso aplicativo, ele estará acessível em todas as nossas classes, mas não em outro aplicativo.
- Quando o aplicativo é desinstalado, seus dados também são removidos.

Biblioteca ANKO SQLite



- O objetivo dessa biblioteca é deixar o desenvolvimento de aplicativos com Kotlin mais rápido e mais fácil.
- Desenvolvida pela equipe da JetBrains
- Quatro partes

Biblioteca ANKO SQLite



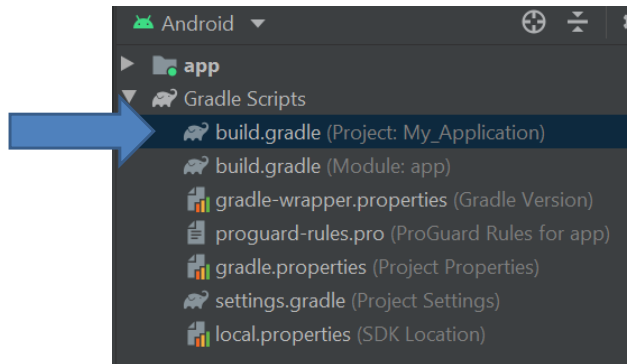
- Quatro partes
- 1. Anko Commons: uma caixa de ferramentas para diversas funções para as tarefas mais comuns do Android.
- 2. Anko Layouts: criação de layouts dinâmicos de forma rápida e eficiente.
- 3. Anko SQLite: ferramentas que facilitam a criação e manipulação de banco de dados SQLite.
- 4. Anko Coroutines: utilitários baseados na biblioteca `kotlinx.coroutines` .

Biblioteca ANKO SQLite



- Mais informações:
<https://github.com/Kotlin/anko>
- Classe SQLiteOpenHelper
- <https://developer.android.com/training/data-storage/sqlite.html>

Adicionando Anko SQLite



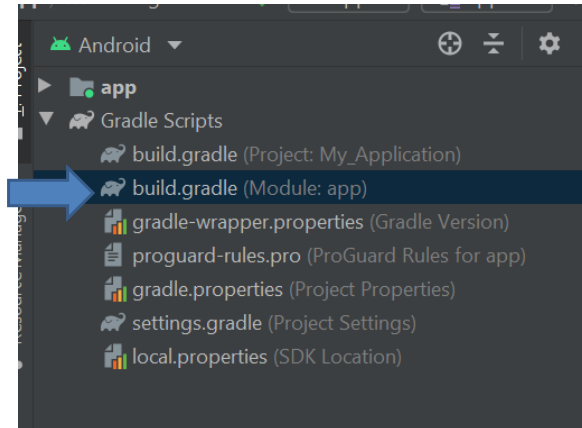
Adicionando Anko SQLite



Gradle files have changed since last project sync. A project sync may be necessary for the IDE to work properly. [Sync Now](#)

```
1 // Top-level build file where you can add configuration options common to all sub-projects/modules.
2 buildscript {
3     ext.kotlin_version = "1.3.72"
4     ext.anko_version = '0.10.4'
5     repositories {
6         google()
7         jcenter()
8     }
}
```

Adicionando Anko SQLite



Adicionando Anko SQLite



Gradle files have changed since last project sync. A project sync may be necessary for the IDE to work properly. [Sync Now](#)

```
18
19 buildTypes {
20     release {
21         minifyEnabled false
22         proguardFiles getDefaultProguardFile('proguard-android-optimize.txt')
23     }
24 }
25
26
27 dependencies {
28     implementation fileTree(dir: "libs", include: ["*.jar"])
29     implementation "org.jetbrains.kotlin:kotlin-stdlib:$kotlin_version"
30     implementation 'androidx.core:core-ktx:1.3.1'
31     implementation 'androidx.appcompat:appcompat:1.2.0'
32     implementation 'androidx.constraintlayout:constraintlayout:2.0.1'
33     testImplementation 'junit:junit:4.12'
34     androidTestImplementation 'androidx.test.ext:junit:1.1.2'
35     androidTestImplementation 'androidx.test.espresso:espresso-core:3.3.0'
36     implementation "org.jetbrains.anko:anko-sqlite:$anko_version"
37 }
```





Adicionando Anko SQLite



- implementation "org.jetbrains.anko:anko-sqlite:\$anko_version"

MainActivity.kt



```
1 package com.example.myapplication
2
3 import android.content.Intent
4 import androidx.appcompat.app.AppCompatActivity
5 import android.os.Bundle
6 import android.widget.*
7 import kotlinx.android.synthetic.main.activity_main.*
8 import org.w3c.dom.Text
9
10 import org.jetbrains.anko.startActivity
```

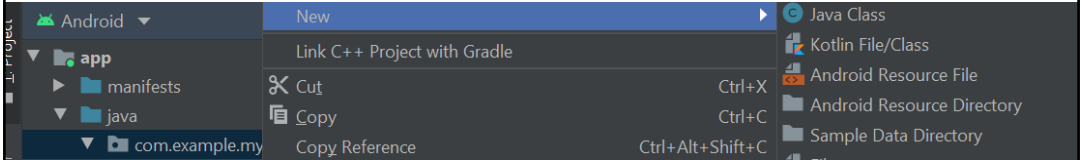
MainActivity.kt



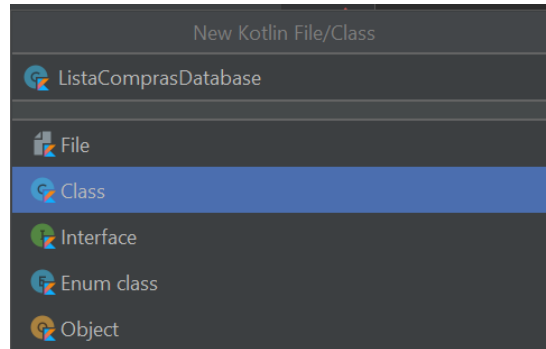
- Utilizando a biblioteca anko podemos fazer a trocar do startActivity

```
btnAdicionar.setOnClickListener { it: View!  
    //val intent = Intent(this, CadastroActivity::class.java)  
    //startActivity(intent)  
  
    startActivity<CadastroActivity>()  
}
```

Nova Classe



Classe ListaComprasDatabase



Classe ListaComprasDatabase

- Classe ManagedSQLiteOpenHelper → faz o gerenciamento do banco de dados, desde a criação do banco de dados.
- 3 parâmetros obrigatórios

```
package com.example.myapplication

import android.content.Context
import org.jetbrains.anko.db.ManagedSQLiteOpenHelper

class ListaComprasDatabase (context: Context) :ManagedSQLiteOpenHelper<ctx = context,
    name = "listaCompras.db", version = 1){

    Fat
```

Classe ListaComprasDatabase

```
import android.content.Context
import android.database.sqlite.SQLiteDatabase

import org.jetbrains.anko.db.ManagedSQLiteOpenHelper

class ListaComprasDatabase(context: Context) : ManagedSQLiteOpenHelper(
    ctx = context,
    name = "listaCompras.db", version = 1
) {
    override fun onCreate(db: SQLiteDatabase) {
        //criação de tabelas
    }

    override fun onUpgrade(db: SQLiteDatabase, oldVersion: Int, newVersion: Int) {
        //atualizado do banco de dados
    }
}
```

Classe ListaComprasDatabase

- * import org.jetbrains.anko.db.ManagedSQLiteOpenHelper
import org.jetbrains.anko.db.createTable
import org.jetbrains.anko.db.INTEGER
import org.jetbrains.anko.db.PRIMARY_KEY
import org.jetbrains.anko.db.UNIQUE
import org.jetbrains.anko.db.TEXT
import org.jetbrains.anko.db.REAL

import org.jetbrains.anko.db.BLOB

```
import org.jetbrains.anko.db.*
```

```
override fun onCreate(db: SQLiteDatabase) {  
    //criacao de tabelas  
  
    db.createTable( tableName: "Produtos", ifNotExists: true,  
        ...columns: "id" to INTEGER + PRIMARY_KEY + UNIQUE,  
        "nome" to TEXT,  
        "quantidade" to INTEGER,  
        "valor" to REAL,  
        "foto" to BLOB)  
}
```

Tipos de Dados - SQLite



- INTEGER
- REAL
- TEXT
- BLOB: valor de modo binário

Singleton



- Uma instância única para toda a aplicação
- Em Kotlin, por definição, não temos variáveis estáticas, mas existem os blocos companion object que, no final das contas, são a mesma coisa.
- <https://www.devmedia.com.br/padrao-de-projeto-singletonem-java/26392>
- Criar uma variável estática do mesmo tipo da classe, geralmente chamada de instance, e implementar um método estático que retorna essa variável, geralmente chamado getInstance

Singleton



- Em Kotlin, por definição não temos variáveis estáticas, mas existem os blocos companion object que, no final das contas, são a mesma coisa.
- Quando têm variáveis e métodos dentro de blocos companion object, é possível acessá-los sem a necessidade de criar uma instância de classe

Classe ListaComprasDatabase

```
class ListaComprasDatabase(context: Context) : ManagedSQLiteOpenHelper(  
    ctx = context,  
    name = "listaCompras.db", version = 1  
) {  
  
    //singleton da classe  
    companion object {  
        private var instance: ListaComprasDatabase? = null  
  
        @Synchronized  
        fun getInstance(ctx: Context): ListaComprasDatabase {  
            if (instance == null) {  
                instance = ListaComprasDatabase(ctx.getApplicationContext())  
            }  
            return instance!!  
        }  
    }  
  
    override fun onCreate(db: SQLiteDatabase) {
```


Singleton



- A anotação `@Synchronized` protege o método contra acesso concorrente de múltiplas threads.

Classe ListaComprasDatabase

```
// Acesso a propriedade pelo contexto
val Context.database: ListaComprasDatabase
    get() = ListaComprasDatabase.getInstance(getApplicationContext())
```

```
import org.jetbrains.annotations.Nullable

// Acesso a propriedade pelo contexto
val Context.database: ListaComprasDatabase
    get() = ListaComprasDatabase.getInstance(getApplicationContext())

class ListaComprasDatabase(context: Context) : ManagedSQLiteOpenHelper(
    ctx = context,
```

Data class Produto adicionar id

```
package com.example.myapplication

import android.graphics.Bitmap

data class Produto (
    val id: Int,
    val nome: String,
    val quantidade: Int,
    val valor: Double,
    val foto: Bitmap? = null)
```

CadastroActivity.kt

```
if(produto.isNotEmpty() && qtde.isNotEmpty() && valor.isNotEmpty() ){  
    //produtosAdapter.add(produto)  
    //apagar essas linhas CadastroActivity.kt  
    //val prod = Produto(produto, qtde.toInt(), valor.toDouble())  
    //produtosGlobal.add(prod)  
    txtProduto.text.clear()  
    txtQtde.text.clear()  
    txtValor.text.clear()  
  
} else{  
    txtProduto.error = if (txtProduto.text.isEmpty()) "Preencha o nome do Produto" else null  
    txtQtde.error = if(txtQtde.text.isEmpty()) "Preencha a quantidade" else null  
    txtValor.error = if(txtValor.text.isEmpty()) "Preencha o valor" else null  
}  
}
```

CadastroActivity.kt



- Insert na tabela Produtos

```
if(produto.isNotEmpty() && qtde.isNotEmpty() && valor.isNotEmpty() ){  
  
    //produtosAdapter.add(produto)  
    //apagar essas linhas CadastroActivity.kt  
    //val prod = Produto(produto, qtde.toInt(), valor.toDouble())  
    //produtosGlobal.add(prod)  
  
    database.use{ this: SQLiteDatabase  
        val idProduto = insert( table: "Produtos",  
            nullColumnHack: "nome" to produto,  
            values: "quantidade" to qtde,  
                    "valor" to valor.toDouble(),  
                    "foto" to imageBitMap)  
    }  
  
    txtProduto.text.clear()  
    txtQtde.text.clear()  
    txtValor.text.clear()  
}
```

CadastroActivity.kt



```
import org.jetbrains.anko.toast
import org.jetbrains.anko.db.insert
```

```
database.use { this: SQLiteDatabase
    val idProduto = insert(
        tableName: "Produtos",
        ...values: "nome" to produto,
        "quantidade" to qtde,
        "valor" to valor.toDouble(),
        "foto" to imageBitMap
    )

    if (idProduto != -1L) {
        toast("Item inserido com sucesso")
        txtProduto.text.clear()
        txtQtde.text.clear()
        txtValor.text.clear() ^use
    } else {
        toast("Erro ao inserir no banco de dados") ^use
    }
}
```

Utils.kt

```
import android.graphics.Bitmap
import android.graphics.BitmapFactory
import java.io.ByteArrayOutputStream

val produtosGlobal = mutableListOf<Produto>()

fun Bitmap.toByteArray(): ByteArray {
    val stream = ByteArrayOutputStream()
    //comprimindo a imagem
    this.compress(android.graphics.Bitmap.CompressFormat.PNG, quality: 0, stream)
    //transformando em um array de caracteres
    return stream.toByteArray()
}

fun ByteArray.toBitmap(): Bitmap {
    return BitmapFactory.decodeByteArray(data: this, offset: 0, this.size)
}
```

Gravar a foto no banco de dados

- Não podemos enviar o objeto Bitmap, pois o tipo BLOB não suporta um BitMap diretamente.
- Devemos enviar um vetor de bytes em Kotlin um objeto do tipo ByteArray
- Transformar BitMap em ByteArray
- Transformar ByteArray em Bitmap

Função Bitmap.toByteArray



- 3 partes
- 1ª declarar um objeto do tipo `ByteArrayOutputStream`, que será responsável por fazer a transformação do `Bitmap` em `ByteArray`
- 2ª fazer a compressão da imagem dentro do objeto stream. (3 parâmetros, o formato da imagem, a qualidade de compressão, sendo 0 como qualidade máxima e 100 como máximo de compressão e objeto stream)

Função Bitmap.toByteArray



- 3ª retornar o resultado da função

CadastroActivity.kt



```
database.use { this: SQLiteDatabase
    val idProduto = insert(
        tableName: "Produtos",
        ...values: "nome" to produto,
        "quantidade" to qtde,
        "valor" to valor.toDouble(),
        "foto" to imageBitmap?.toByteArray()
    )

    if (idProduto != -1L) {
        toast("Item inserido com sucesso")
        txtProduto.text.clear()
        txtQtde.text.clear()
        txtValor.text.clear() ^use
    } else{
        toast("Erro ao inserir no banco de dados") ^use
    }
}
```

Comando Select



- Vamos supor que você queira trazer todos os produtos com quantidade maior que 5, o comando ficaria:
- `select("produtos").whereArgs("quantidade > {qtd}", "qtd" to 5)`


Comando Select



- `parseSingle`: converte exatamente 1 linha; usamos essa função quando sabemos que nossa consulta retornará exatamente 1 registro.
- `parseOpt`: converte 1 ou nenhuma linha; diferente do `parseSingle`, o `parseOpt` pode ter um retorno nulo caso não tenha nenhum registro.
- `parseList`: converte vários registros em uma lista, para consultas que retornaram mais de 1 resultado

MainActivity.kt – mudar código

```
override fun onResume() {  
    super.onResume()  
  
    val adapter = listViewProdutos.adapter as ProdutoAdapter  
  
    adapter.clear()  
    adapter.addAll(produtosGlobal)  
  
    var soma = 0.0  
    for (item in produtosGlobal){  
        soma += item.valor * item.quantidade  
    }  
    txtTotal.text = soma.toString()  
}
```




```
import org.jetbrains.anko.db.parseList
import org.jetbrains.anko.db.rowParser
import org.jetbrains.anko.db.select
```

```
override fun onResume() {
    super.onResume()

    val adapter = listViewProdutos.adapter as ProdutoAdapter
    database.use { this: SQLiteDatabase

        select( tableName: "produtos").exec { this: Cursor
            //criando o parser que montará o objeto produto
            val parser = rowParser { id: Int,
                                    nome: String,
                                    quantidade: Int,
                                    valor: Double,
                                    foto: ByteArray? ->

                                    //Colunas do banco de dados
                                    //Montagem do objeto Produto com as colunas do banco
                                    Produto(id, nome, quantidade, valor, foto?.toBitmap())
            }
        }
    }
}
```



```

//criando a lista de produtos com dados do banco
var listaProdutos = parseList(parser)

//limpando os dados da lista e carregando as novas informações
adapter.clear()
adapter.addAll(listaProdutos)

//efetuando a multiplicado e soma da quantidade e valor
var soma = 0.0
//val soma: Double = listaProdutos.sumByDouble { it.valor * it.quantidade }
for (item in listaProdutos) {
    soma += item.valor * item.quantidade
}

//formando em formato moeda
val f = NumberFormat.getCurrencyInstance(Locale(language: "pt", country: "br"))
txtTotal.text = "TOTAL: ${f.format(soma)}"

```

Fatec
Ferraz

40

package com.example.myapplication

import android.content.Intent

import androidx.appcompat.app.AppCompatActivity

import android.os.Bundle

import android.widget.*

import kotlinx.android.synthetic.main.activity_main.*

import org.w3c.dom.Text

import org.jetbrains.anko.startActivity

import java.text.NumberFormat

import java.util.*

import org.jetbrains.anko.db.parseList

import org.jetbrains.anko.db.rowParser

import org.jetbrains.anko.db.select

class MainActivity : AppCompatActivity() {

override fun onCreate(savedInstanceState: Bundle?) {


```

super.onCreate(savedInstanceState)
setContentView(R.layout.activity_main)

val produtosAdapter = ProdutoAdapter(this)
val listViewProdutos = findViewById<ListView>(R.id.listViewProdutos)

listViewProdutos.adapter = produtosAdapter

btnAdicionar.setOnClickListener {
    //val intent = Intent(this, CadastroActivity::class.java)
    //startActivity(intent)

    startActivity<CadastroActivity>()
}

listViewProdutos.setOnItemClickListener { adapterView: AdapterView<*>, view,
position: Int, id ->
    val item = produtosAdapter.getItem(position)
    produtosAdapter.remove(item)

}
}

override fun onResume() {
    super.onResume()

    val adapter = listViewProdutos.adapter as ProdutoAdapter
    database.use {

        select("produtos").exec {
            //criando o parser que montará o objeto produto
            val parser = rowParser { id: Int,
                                    nome: String,
                                    quantidade: Int,
                                    valor: Double,
                                    foto: ByteArray? ->
                                //Colunas do banco de dados

```

```

        //Montagem do objeto Produto com as colunas do banco
        Produto(id, nome, quantidade, valor, foto?.toBitmap())
    }

    //criando a lista de produtos com dados do banco
    var listaProdutos = parseList(parser)

    //limpando os dados da lista e carregando as novas informações
    adapter.clear()
    adapter.addAll(listaProdutos)

    //efetuando a multiplicado e soma da quantidade e valor
    var soma = 0.0
    //val soma: Double = listaProdutos.sumByDouble { it.valor * it.quantidade }
    for (item in listaProdutos) {
        soma += item.valor * item.quantidade
    }
    //formando em formato moeda
    val f = NumberFormat.getCurrencyInstance(Locale("pt", "br"))
    txtTotal.text = "TOTAL: ${f.format(soma)}"



    }
    }
}

```

GitHub



- https://github.com/marcosvoliveira/PPDM_1_PROJETO_06

- 
- 
- Atividade 10 – Android Studio – Projeto 5 –
Lista de Compra com classe de Dados

Referências

- GLAUBER, Nelson. Dominando o Android com Kotlin, 3ª Ed. São Paulo: Editora Novatec, 2019
- LECHETA, Ricardo R. **Android Essencial com Kotlin**. 2ª ed. São Paulo: Editora Novatec, 2018





Obrigado!

Prof. Marcos Vasconcelos de Oliveira