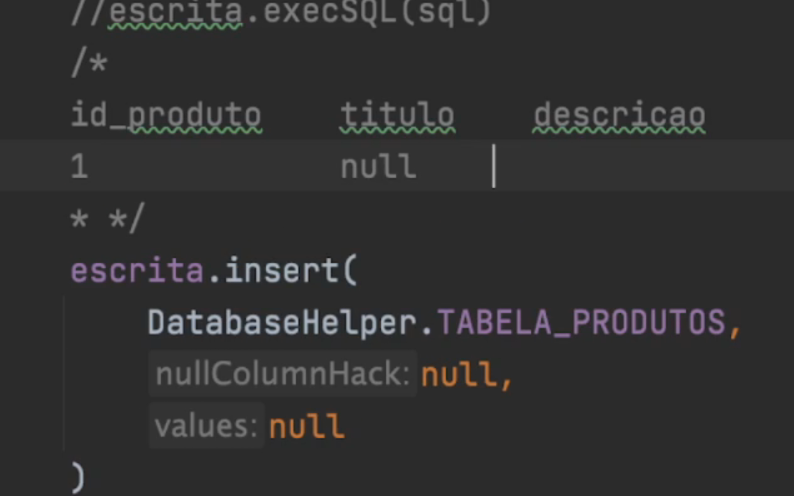
No 201 utilizamos o projeto DAO com nossos métodos, agora vamos utilizar os próprios métodos do SQLiteOpenHelper

# INSERIR PRODUTO

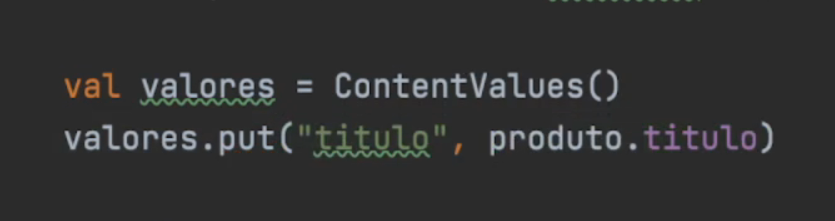
Imagina que queremos colocar todos os valores como nulo, então fazemos assim



Agora se a gente quiser passar valores, temos que o usar a estratégia de chave valor, falando um HashMap

Onde a variável valor recebe uma classe de ContentValues

E em valores usamos o put para passar a chave e também o valor



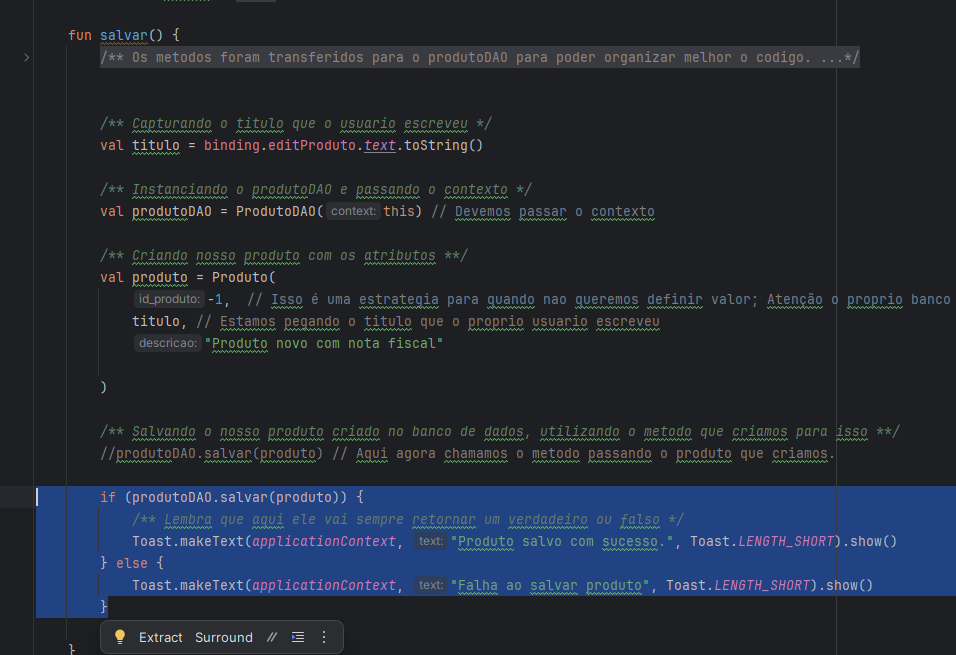
No código ficou dessa forma

val valores = ContentValues()  
// Aqui seria o nome da coluna e o valor.  
valores.put("${DatabaseHelper.TITULO}", produto.titulo)  
// Imagina que temos mais de um valor, então podemos fazer mais chaves e valores  
valores.put("${DatabaseHelper.DESCRICAO}", produto.descricao)

// Dessa forma podemos ir salvando vários valores no banco de dados.

try {  
 // escrita.execSQL(comandoSQL) // writable para escrita  
  
  
 escrita.insert(  
 */\*\* Aqui ele espera receber 3 parametros, Tabela, coluna e valor \*/* DatabaseHelper.TABELA\_PRODUTOS,  
 null, // Aqui só vamos precisar configurar quando não tiver nenhum valor sendo passado.  
 valores  
  
 )

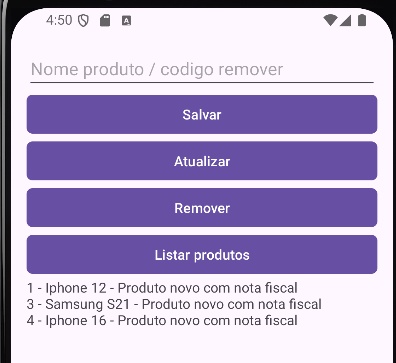
Agora dentro do método salvar que passa os dados para ProdutoDAO podemos fazer um if para verificar se deu tudo certo



Ficando dessa forma

# Listar produtos

Lembra que antes ele listava dentro do log, mas agora queremos listar ele diretamente na tela.



Fizemos apenas essas alterações para exibir os itens na tela, se caso não tivesse nenhum item ai sim ele exibiria a mensagem de nenhum item cadastrado.

private fun listar() {  
 */\*\* Cursor = Onde nosso mouse esta* val produtoDAO = ProdutoDAO(this)  
 val listaProdutos = produtoDAO.listar()  
  
 var texto = ""  
  
 if (listaProdutos.*isNotEmpty*()) {  
 */\*\* Conferindo se a lista esta retornando alguma coisa, pq ela pode vir vazia \*/*

*// Lembra que a listaProdutos é uma lista e por isso do ForEach, para percorrer toda a lista.* listaProdutos.*forEach* **{** texto += "${**it**.id\_produto} - ${**it**.titulo} - ${**it**.descricao}\n" // Aqui estamos configurando para ele ir acumulando os produtos dentro da variavel  
  
  
  
  
 Log.i("info\_db", "${**it**.id\_produto} - ${**it**.titulo}")  
 **}** binding.textResultado.*text* = texto // Imprimindo na tela.  
  
 }else {  
 binding.textResultado.*text* = "Nenhum item cadastrado."  
 }  
 }

# Atualizar produto

Vamos la em produtoDAO e vamos configurar tudo la

Alguns códigos você pode ver no método salvar, la vai ter os comentários explicando os códigos que estão repetidos.

Ficando dessa forma, lembrar também que o produto que estamos atualizando esta fixo, estamos atualizando o produto com ID = 1, que é oque configuramos no método la em MainActivity

val valores = ContentValues()  
valores.put("${DatabaseHelper.TITULO}", produto.titulo)  
valores.put("${DatabaseHelper.DESCRICAO}", produto.descricao)  
// Aqui precisamos do nosso argumento, que seria o id que vamos atualizar  
val args = *arrayOf*("${produto.id\_produto}") */\*\* Aqui sao os parametros que vai ser passado como argumento, eles vai entrar no lugar do ? \*/*try {  
 //escrita.execSQL(codigoSQL)  
  
  
  
  
 escrita.update(  
 */\*\* Aqui é um metodo do SQLiteOpenHelper, mas ele espera receber 4 parametros  
 \* Tabela, ContentValues, Where e uma Where args  
 \*/* DatabaseHelper.TABELA\_PRODUTOS,  
 valores,  
 "id\_produto = ?", // condiçao de oque vamos atualizar. Poderiamos passar 2 valores tambem ex: "id\_produto = 1 AND qnt = 10",  
 */\*\* Seria como se fosse uma funçao com seus metodos, imagina que no args vamos ter 3 parametros, então o primeiro parametro vai ficar no primeiro ?  
 \* O segundo parametro vai ficar no segundo ?  
 \* O terceiro parametro vai ficar no 3 ?  
 \*/* args  
  
  
  
 )

La podemos futuramente colocar dinâmico também

private fun atualizar() {  
 */\*\* Caso tenha duvida olhe o metodo de salvar \*\*/* val titulo = binding.editProduto.*text*.toString()  
 val produtoDAO = ProdutoDAO(this)  
 val produto = Produto (  
 1, titulo, "descricao produto..."  
 )  
  
 produtoDAO.atualizar(produto)  
  
}

# REMOVER PRODUTO

O resto ficou tudo igual, fizemos apenas essas alterações

override fun remover(idProduto: Int): Boolean {  
 */\*\* Remover um produto onde o ID seja igual a 01  
 \* Futuramente vamos fazer um recyclerView para poder remover um produto. \*\*/* //val codigoSQL = "DELETE FROM produtos WHERE id\_produto = $idProduto"  
  
 val args = *arrayOf*( idProduto.toString() ) // Argumento de qual id vai ser removido.  
  
 try {  
 // escrita.execSQL(codigoSQL)  
  
  
 escrita.delete(  
 */\*\* Também espera receber alguns parametros  
 \* 1 Tabela  
 \* 2 Condição  
 \*/* DatabaseHelper.TABELA\_PRODUTOS,  
 "${DatabaseHelper.ID\_PRODUTO} = ?", // Lembrar que so temos uma condição la em cima, então la que vamos definir qual produto estamos removendo.  
 args // Argumento de qual produto vamos remover.  
 )  
  
  
 Log.i("info\_db", "Sucesso ao remover")  
 }catch (e: Exception) {  
 e.printStackTrace()  
 Log.i("info\_db", "Erro ao remover")  
 return false  
 }  
 return true  
}