МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА» (БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №6

по дисциплине: Базы данных

тема: «Организация взаимодействия с базой данных через приложение с графическим интерфейсом»

Выполнил: ст. группы ПВ-223

Дмитриев Андрей Александрович

Проверил:

Панченко Максим Владимирович

Белгород 2024 г.

**Вариант 2.**

**Цель работы:** получение навыков разработки приложений для взаимодействия с базой данных, содержащих графический интерфейс пользователя.

**Задание к работе:**

1. Изучить библиотеку для реализации приложения с графическим интерфейсом на выбранном языке программирования.

2. Разработать приложение с графическим интерфейсом, которое обеспечит подключение к базе данных, разработанной на основе предыдущих лабораторных работ, а также обеспечит выполнение запросов.

**Ход работы:**

Приложение с графическим интерфейсом реализовано с ипользованием библиотеки android-compose. Как СУБД используется SQLite в место PostgreSQL из-за наличия сложностей в установке соединения.

Полный код находится по ссылке: <https://github.com/AnDreV133/SimpleStore/tree/master-lab6>.

Некоторые части исходного кода:

object StateManager {  
 open class State {  
 class NoConnection : State()  
 class ChangeStore : State()  
 class History(val storeId: Long) : State()  
 class Menu(val storeId: Long) : State()  
 class Shopping(val storeId: Long) : State()  
 class Rating(val storeId: Long) : State()  
 }  
  
 private val stackState = Stack<State>()  
  
 @Composable  
 fun Screen(conn: SQLiteDatabase?) {  
 val state = remember **{** *mutableStateOf*(  
 if (conn != null) State.ChangeStore()  
 else State.NoConnection()  
 )  
 **}** val activity = (*LocalContext*.current as? Activity)  
 BackHandler **{** rollback(state, activity)  
 **}** when (state.value) {  
 is State.NoConnection -> {  
 stackState.push(state.value as State.NoConnection)  
 Text(  
 modifier = Modifier  
 .*fillMaxSize*()  
 .*padding*(16.*dp*),  
 textAlign = TextAlign.Center,  
 fontSize = 24.*sp*,  
 text = "No connection"  
 )  
 }  
  
 is State.ChangeStore -> {  
 stackState.push(state.value as State.ChangeStore)  
 ChangeStore.Screen(conn!!, state)  
 }  
  
 is State.Menu -> {  
 val castedState = state.value as State.Menu  
 stackState.push(castedState)  
 Menu.Screen(castedState.storeId, state)  
 }  
  
 is State.History -> {  
 val castedState = state.value as State.History  
 stackState.push(castedState)  
 History.Screen(conn!!, castedState.storeId)  
 }  
  
 is State.Shopping -> {  
 val castedState = state.value as State.Shopping  
 stackState.push(castedState)  
 Shopping.Screen(conn!!, castedState.storeId) **{** rollback(state, activity) **}** }  
  
 is State.Rating -> {  
 val castedState = state.value as State.Rating  
 stackState.push(castedState)  
 Rating.Screen(conn!!, castedState.storeId)  
 }  
 }  
 }  
  
 private fun rollback(state: MutableState<State>, activity: Activity? = null) {  
 if (stackState.size >= 1) {  
 stackState.pop()  
 state.value = stackState.pop()  
 } else  
 activity?.finish()  
 }  
  
}

object ChangeStore {  
 class Model(val id: Long, val address: String)  
  
 @Composable  
 fun Screen(conn: SQLiteDatabase, state: MutableState<StateManager.State>) {  
 DrawerMenu(  
 menuItems = conn.*executeStores*().*map* **{** model **->** "id: ${model.id} by address: ${model.address}" *to* StateManager.State.Menu(model.id)  
 **}**,  
 state = state  
 )  
 }  
  
 private fun SQLiteDatabase.executeStores(): MutableList<Model> =  
 *mutableListOf*<Model>().*apply* **{** this@executeStores.*query*("select \* from ${Table.Store.T\_NAME};") **{** if (**it** == null) return@query  
  
 while (**it**.moveToNext()) {  
 add(  
 Model(  
 **it**.getLong(0),  
 **it**.getString(1)  
 )  
 )  
 }  
 **}  
 }**}

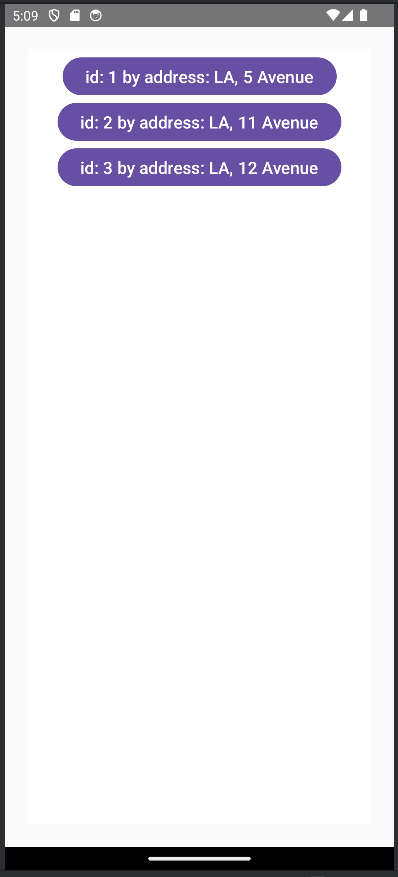
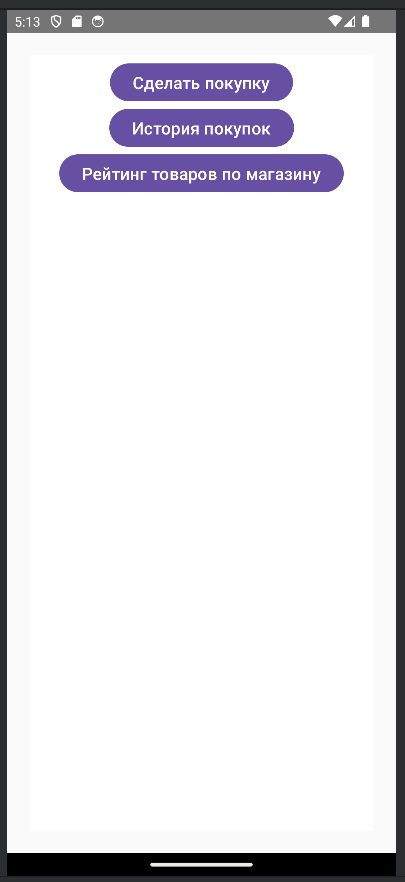
object Menu {  
 @Composable  
 fun Screen(storeId: Long, state: MutableState<StateManager.State>) {  
 DrawerMenu(  
 menuItems = *listOf*(  
 "Сделать покупку" *to* StateManager.State.Shopping(storeId),  
 "История покупок" *to* StateManager.State.History(storeId),  
 "Рейтинг товаров по магазину" *to* StateManager.State.Rating(storeId),  
 ),  
 state = state  
 )  
 }  
}

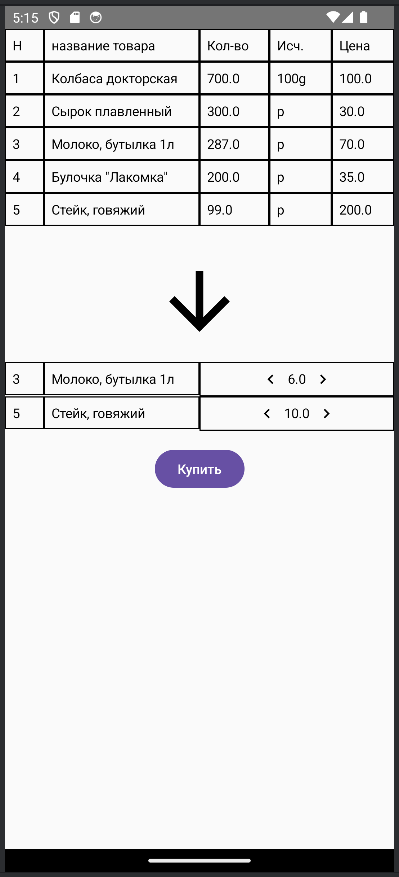
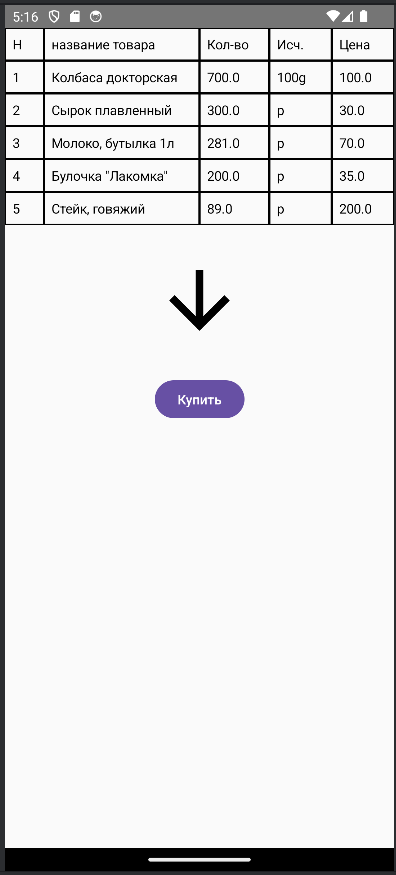
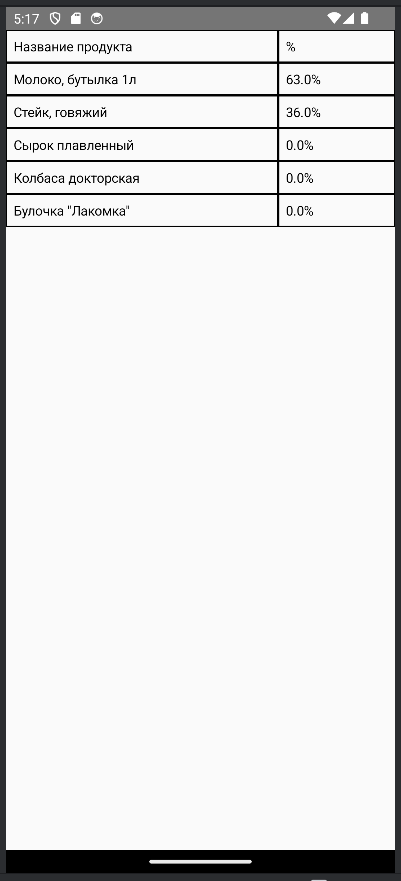
object Shopping {  
 private class Model(  
 val article: Long,  
 val productName: String,  
 val amount: Double,  
 val quan: String,  
 val price: Double  
 )  
  
 class Purchase(  
 val article: Long,  
 val productName: String,  
 var amount: Double,  
 val maxAmount: Double  
 )  
  
 @Composable  
 fun Screen(conn: SQLiteDatabase, storeId: Long, rollback: ()-> Unit) {…}  
  
 private fun SQLiteDatabase.queryAssortment(storeId: Long): MutableList<Model> {  
 val res = *mutableListOf*<Model>()  
  
 *query*(  
 """  
 select t0.${Table.Accounting.PRODUCT\_ARTICLE},  
 t1.${Table.Product.NAME},  
 t0.${Table.Accounting.AMOUNT},   
 t1.${Table.Product.QUANTITY\_TO\_ASSESS},  
 t0.${Table.Accounting.COST}  
 from ${Table.Accounting.T\_NAME} as t0  
 inner join ${Table.Product.T\_NAME} as t1   
 on t0.${Table.Accounting.PRODUCT\_ARTICLE}=t1.${Table.Product.ARTICLE}  
 where ${Table.Accounting.STORE\_ID}=$storeId;  
 """  
 ) **{** model **->** if (model == null) return@query  
  
 while (model.moveToNext()) {  
 res.add(  
 Model(  
 model.getLong(0),  
 model.getString(1),  
 model.getDouble(2),  
 model.getString(3),  
 model.getDouble(4)  
 )  
 )  
 }  
 **}** return res  
 }  
  
 private fun SQLiteDatabase.executeBuy(  
 storeId: Long,  
 purchases: List<Purchase>,  
 ): Boolean {  
 var res = false  
 try {  
 beginTransaction()  
 val checkId = insert(  
 Table.CheckList.T\_NAME,  
 null,  
 ContentValues().*apply* **{** put(Table.CheckList.STORE\_ID, storeId)  
 put(Table.CheckList.TIME, Calendar.getInstance().*time*.*time*)  
 **}** )  
  
 purchases.*forEach* **{** purchase **->** insert(  
 Table.Purchase.T\_NAME, null,  
 ContentValues().*apply* **{** put(Table.Purchase.CHECK\_LIST\_ID, checkId)  
 put(Table.Purchase.PRODUCT\_ARTICLE, purchase.article)  
 put(Table.Purchase.AMOUNT, purchase.amount)  
 **}**)  
  
 *execute*(  
 """  
 UPDATE ${Table.Accounting.T\_NAME}  
 SET ${Table.Accounting.AMOUNT}=${Table.Accounting.AMOUNT}-${purchase.amount}  
 WHERE ${Table.Accounting.STORE\_ID}=$storeId   
 AND ${Table.Accounting.PRODUCT\_ARTICLE}=${purchase.article};  
   
 """  
 )  
 **}** res = true  
 setTransactionSuccessful()  
 } catch (e: SQLiteException) {  
 Log.e("ShoppingState", e.toString())  
 } finally {  
 endTransaction()  
 }  
  
 return res  
 }  
}

object History {  
 class PurchaseModel(val name: String, val count: Double, val quan: String, val price: Double)  
 class PurchaseHistoryModel(val checkId: Long, val purchases: List<PurchaseModel>)  
  
 @Composable  
 fun Screen(conn: SQLiteDatabase, storeId: Long) {  
 LazyColumn(  
 modifier = Modifier  
 .*fillMaxWidth*()  
 ) **{** *items*(conn.*queryGetHistory*(storeId)) **{** model **->** val modifier = Modifier.*padding*(bottom = 10.*dp*)  
 Text(  
 modifier = modifier,  
 text = "Check Id: ${model.checkId}"  
 )  
 for (purchase in model.purchases) {  
 Text(  
 modifier = modifier,  
 text = "Name: ${purchase.name}, Count: ${purchase.count} in ${purchase.quan}, Price: ${purchase.price}"  
 )  
 }  
 **}  
 }** }  
  
 private fun SQLiteDatabase.queryGetHistory(storeId: Long): List<PurchaseHistoryModel> {  
 val res = *mutableListOf*<PurchaseHistoryModel>()  
 *query*(  
 """  
 select t0.${Table.CheckList.ID},  
 t2.${Table.Product.NAME},   
 t1.${Table.Purchase.AMOUNT},  
 t2.${Table.Product.QUANTITY\_TO\_ASSESS},  
 t1.${Table.Purchase.AMOUNT}\*t3.${Table.Accounting.COST}  
 from ${Table.CheckList.T\_NAME} as t0   
 inner join ${Table.Purchase.T\_NAME} as t1  
 on t0.${Table.CheckList.ID}=t1.${Table.Purchase.CHECK\_LIST\_ID}  
 and t0.${Table.CheckList.STORE\_ID}=$storeId  
 inner join ${Table.Product.T\_NAME} as t2  
 on t1.${Table.Purchase.T\_NAME}=t2.${Table.Product.ARTICLE}   
 inner join ${Table.Accounting.T\_NAME} as t3  
 on t1.${Table.Purchase.PRODUCT\_ARTICLE}=t3.${Table.Accounting.PRODUCT\_ARTICLE}  
 and t3.${Table.Accounting.STORE\_ID}=$storeId  
 order by t0.${Table.CheckList.ID} asc;  
 """  
 ) **{** cursor **->** if (cursor == null) return@query  
  
 val checkListMap = *mutableMapOf*<Long, List<PurchaseModel>>()  
  
 while (cursor.moveToNext()) {  
 checkListMap.compute(  
 cursor.getLong(0)  
 ) **{** k, v **->** (v?.*toMutableList*() ?: *mutableListOf*())  
 .*apply* **{** add(  
 PurchaseModel(  
 cursor.getString(1),  
 cursor.getDouble(2),  
 cursor.getString(3),  
 cursor.getDouble(4)  
 )  
 )  
 **}  
 }** }  
  
 for (checkList in checkListMap) {  
 res.add(PurchaseHistoryModel(checkList.key, checkList.value))  
 }  
 **}** return res  
 }  
}

object Rating {  
 class Model(val name: String, val partInPercent: Double)  
  
 @Composable  
 fun Screen(conn: SQLiteDatabase, storeId: Long) {  
  
 val products = conn.*executeStatistic*(storeId)  
  
 val nameWeight = 0.7f  
 val partInPercentWeight = 0.3f  
 LazyColumn(modifier = Modifier.*padding*(bottom = 16.*dp*)) **{** item(products.size) **{** Row **{** TableCellText(weight = nameWeight, text = "Название продукта")  
 TableCellText(weight = partInPercentWeight, text = "%")  
 **}  
 }** *items*(products) **{** model **->** Row **{** TableCellText(weight = nameWeight, text = model.name)  
 TableCellText(weight = partInPercentWeight, text = "${model.partInPercent}%")  
 **}  
 }  
 }** }  
  
 fun SQLiteDatabase.executeStatistic(storeId: Long): List<Model> {  
 val res = *mutableListOf*<Model>()  
 *query*(  
 """  
 select t2.${Table.Product.NAME},   
 round(coalesce(sum(t1.${Table.Purchase.AMOUNT})\*100/(select sum(amount) from ${Table.Purchase.T\_NAME}),0),2)   
 as amount\_in\_percent   
 from check\_list as t0   
 inner join ${Table.Purchase.T\_NAME} as t1  
 on t0.id=t1.${Table.Purchase.CHECK\_LIST\_ID}   
 and t0.${Table.CheckList.STORE\_ID}=$storeId  
 right join ${Table.Product.T\_NAME} as t2  
 on t1.${Table.Purchase.PRODUCT\_ARTICLE}=t2.${Table.Product.ARTICLE}  
 group by t2.${Table.Product.NAME}  
 order by amount\_in\_percent desc;  
 """  
 ) **{** cursor **->** if (cursor == null) return@query  
  
 while (cursor.moveToNext()) {  
 res.add(  
 Model(  
 name = cursor.getString(0),  
 partInPercent = cursor.getDouble(1)  
 )  
 )  
 }  
 **}** return res  
 }  
}

Скриншоты приложения:

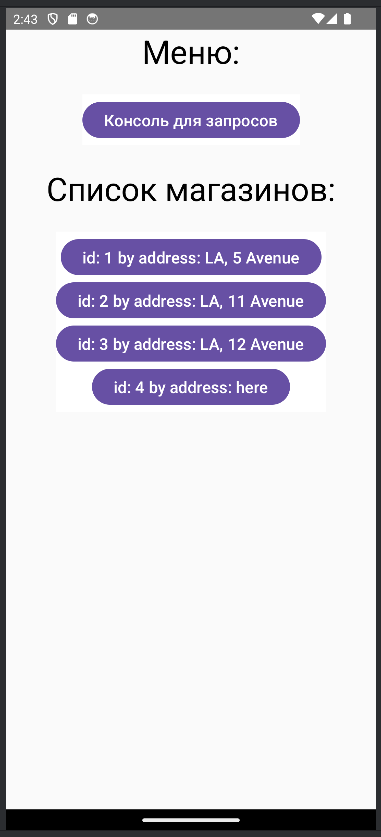
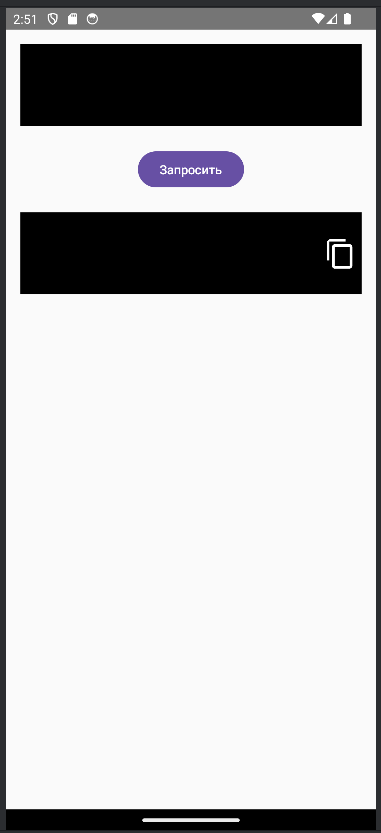
**Дополнительное задание.**

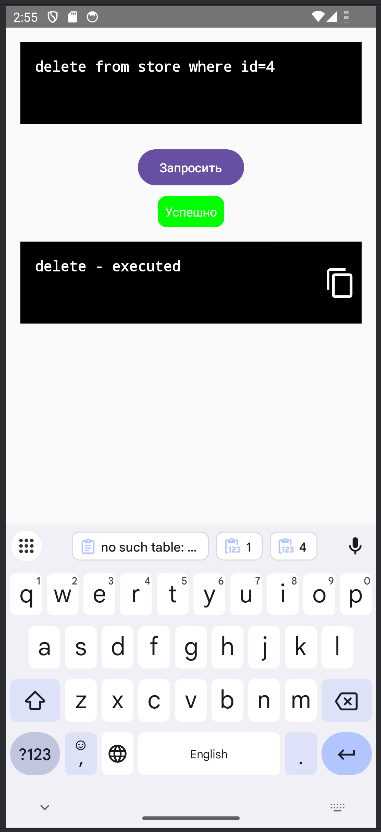
Реализовать экран, на котором будет отображена консоль, через которую можно осуществлять запросы к базе данных.

Код решения:

object Console {  
 @Composable  
 fun Screen(conn: SQLiteDatabase) {  
 var textQuery by remember **{** *mutableStateOf*("") **}** var resultFromQuery by remember **{** *mutableStateOf*("") **}** var isExecuteSuccess by remember **{** *mutableStateOf*<Boolean?>(null) **}** val scope = rememberCoroutineScope()  
 Column(  
 modifier = Modifier  
 .*fillMaxSize*(),  
 horizontalAlignment = Alignment.CenterHorizontally,  
 ) **{** CommandLineTextField(value = textQuery, onValueChange = **{** textQuery = **it }**)  
  
 Button(  
 modifier = Modifier.*padding*(8.*dp*),  
 onClick = **{** scope.*launch* **{** conn.sendExecuteAndGet(textQuery).*fold*(  
 onSuccess = **{** isExecuteSuccess = true  
 resultFromQuery = **it  
 }**,  
 onFailure = **{** isExecuteSuccess = false  
 resultFromQuery = **it**.message ?: "Ошибка запроса"  
 **}** )  
 **}  
 }** ) **{** Text("Запросить")  
 **}** if (isExecuteSuccess == true) {  
 TextInColoredRoundedRect(  
 backgroundColor = Color.Green,  
 textColor = Color.White,  
 text = "Успешно"  
 )  
 } else if (isExecuteSuccess == false) {  
 TextInColoredRoundedRect(  
 backgroundColor = Color.Red,  
 textColor = Color.Black,  
 text = "Ошибка"  
 )  
 }  
  
 CommandLineTextField(  
 value = resultFromQuery,  
 onValueChange = **{}**,  
 readOnly = true,  
 copyButton = true  
 )  
 **}** }  
  
 @Composable  
 fun TextInColoredRoundedRect(  
 backgroundColor: Color,  
 textColor: Color,  
 text: String,  
 modifier: Modifier = Modifier  
 ) {  
 Text(  
 modifier = modifier  
 .*drawWithCache* **{** onDrawBehind **{** drawRoundRect(  
 color = backgroundColor,  
 cornerRadius = *CornerRadius*(10.*dp*.*toPx*())  
 )  
 **}  
 }** .*padding*(8.*dp*),  
 color = textColor,  
 text = text  
 )  
 }  
  
 @Composable  
 fun CommandLineTextField(  
 value: String,  
 onValueChange: (String) -> Unit,  
 modifier: Modifier = Modifier,  
 readOnly: Boolean = false,  
 copyButton: Boolean = false,  
 ) {  
 val context = *LocalContext*.current  
  
 Box(  
 modifier = Modifier  
 .*fillMaxWidth*()  
 .*padding*(16.*dp*)  
 .*background*(Color.Black)  
 ) **{** Row(  
 modifier = modifier,  
 horizontalArrangement = Arrangement.SpaceBetween  
 ) **{** TextField(  
 value = value,  
 onValueChange = onValueChange,  
 modifier = Modifier  
 .*weight*(1f)  
 .*padding*(end = 8.*dp*),  
 readOnly = readOnly,  
 textStyle = TextStyle(  
 color = Color.White,  
 fontFamily = FontFamily.Monospace,  
 fontSize = 16.*sp*,  
 fontWeight = FontWeight.Bold  
 ),  
 minLines = 3,  
 colors = TextFieldDefaults.colors(  
 focusedContainerColor = Color.Black,  
 unfocusedContainerColor = Color.Black,  
 focusedIndicatorColor = Color.Transparent,  
 unfocusedIndicatorColor = Color.Transparent,  
 )  
 )  
  
 if (copyButton)  
 IconButton(  
 onClick = **{** copyToClipboard(context, value)  
 **}**,  
 modifier = Modifier.*align*(Alignment.CenterVertically)  
 ) **{** Icon(  
 imageVector = Icons.Default.ContentCopy,  
 tint = Color.White,  
 contentDescription = "copy button"  
 )  
 **}  
 }  
 }** }  
  
 private fun copyToClipboard(context: Context, text: String) {  
 val clipboardManager =  
 context.getSystemService(Context.*CLIPBOARD\_SERVICE*) as ClipboardManager  
 val clipData = ClipData.newPlainText("command", text)  
 clipboardManager.setPrimaryClip(clipData)  
 Toast.makeText(context, "Command copied to clipboard", Toast.*LENGTH\_SHORT*).show()  
 }  
  
 private suspend fun SQLiteDatabase.sendExecuteAndGet(  
 sql: String  
 ): Result<String> = sql.*trim*().*let* **{** try {  
 if (**it**.*startsWith*("SELECT", true)) {  
 val sb = StringBuilder()  
 val formatOutput = **{** list: List<String> **->** sb.append("| ")  
 for (item in list)  
 sb.append(item)  
 .append("\t | ")  
 sb.append("\n")  
 **}** *query*(**it**) **{** cursor **->** if (cursor == null) return@query  
  
 formatOutput(cursor.*columnNames*.*toList*())  
  
 while (cursor.moveToNext())  
 formatOutput(  
 (0..<cursor.*columnCount*)  
 .*map* **{** i **->** cursor.getString(i) **}** )  
 **}** Result.success(sb.toString())  
 } else {  
 *execute*(**it**)  
 Result.success("${**it**.*substringBefore*(' ')} - executed")  
 }  
 } catch (e: SQLException) {  
 Result.failure(e)  
 }  
 **}**}

Результат работы:

****

****

**Вывод:** в ходе работы получены навыки разработки приложений для взаимодействия с базой данных, содержащих графический интерфейс пользователя.