

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 №5**

по дисциплине: Базы данных

тема: «Организация взаимодействия с базой данных через консольное приложение»

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 использованием 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),

```



```

        }
    }

    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}%")
                }
            }
        }

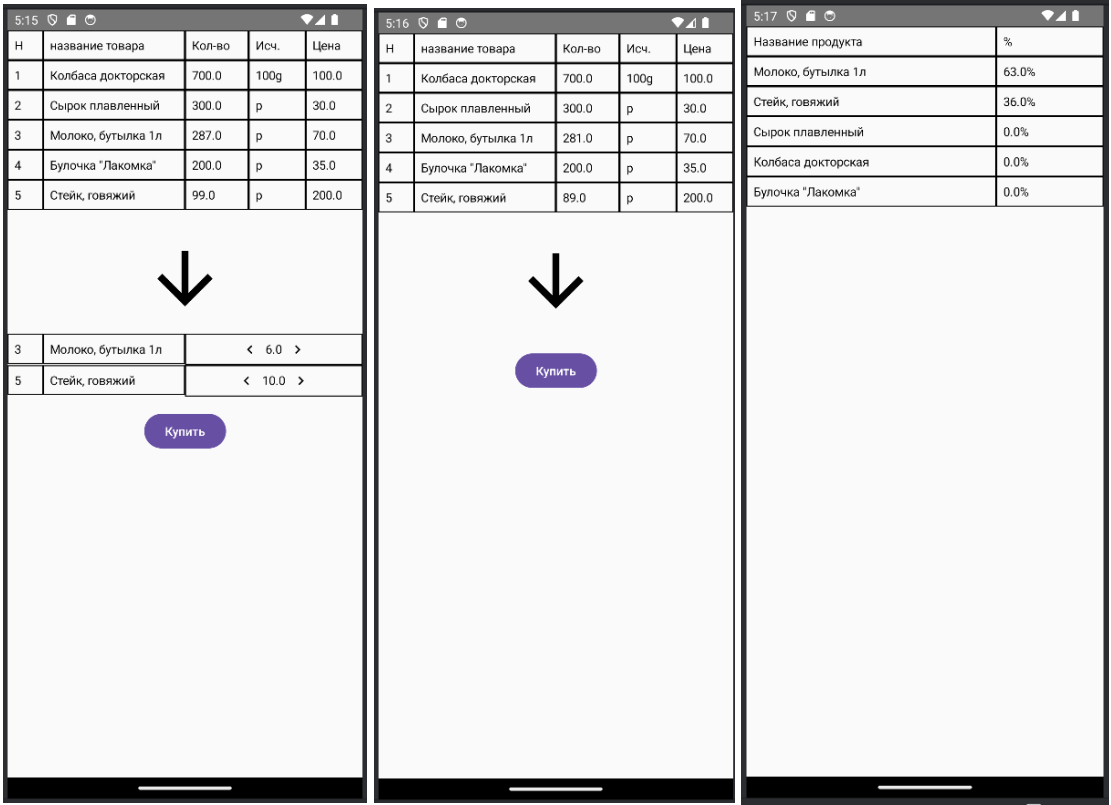
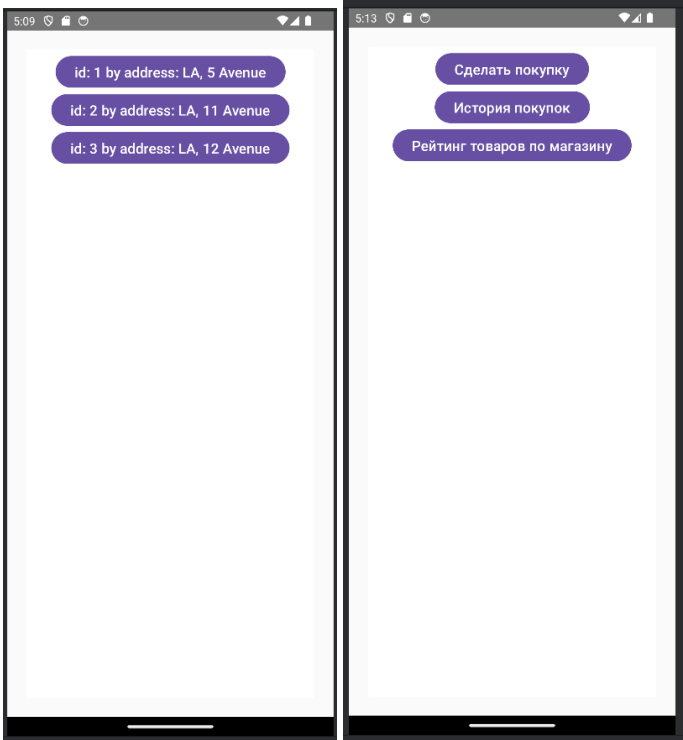
        fun SQLiteDatabase.executeStatistic(storeId: Long): List<Model> {
            val res = mutableListOf<Model>()
            query(
                """
                select  t2.${Table.Product.NAME},
                        round(coalesce(sum(t1.${Table.Purchase.AMOUNT})*100/(select
sum(amount) from ${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:





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