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 №7

по дисциплине: Базы данных

тема: «Организация взаимодействия с базой данных через приложение, использующее технологию ORM»

Выполнил: ст. группы ПВ-223

Дмитриев Андрей Александрович

Проверил:

Панченко Максим Владимирович

Белгород 2024 г.

**Вариант 2.**

**Цель работы:** разработать приложение, использующее технологию ORM, для взаимодействия с базой данных.

**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. Как ORM используется Room, классическая библиотека от Google.

Полный код находится по ссылке: <https://github.com/AnDreV133/SimpleStore/tree/master-lab7>.

Некоторые части исходного кода с пояснениями:

Соединение с базой данных:

private var *appDb*: AppDatabase? = null  
  
private class AppDbCallback: RoomDatabase.Callback() {  
 override fun onCreate(db: SupportSQLiteDatabase) {  
 super.onCreate(db)  
 Log.d(*TAG*, "onCreate db called")  
  
 if (*appDb* == null) {  
 Log.e(*TAG*, "db not be created")  
 return  
 }  
  
 *CoroutineScope*(Dispatchers.IO).*launch* **{** *appDb*!!.initDao().initializeDatabase()  
 **}** }  
}  
  
  
fun connect(applicationContext: Context) =  
 try {  
 *appDb* ?: Room.databaseBuilder(  
 applicationContext,  
 AppDatabase::class.*java*,  
 "app\_db"  
 )  
 .addCallback(AppDbCallback())  
 .setQueryCallback(  
 **{** sqlQuery, \_ **->** Log.d(*TAG*, "SQL Query: $sqlQuery") **}**,  
 Executors.newSingleThreadExecutor()  
 )  
 .build()  
 .*also* **{** *appDb* = **it }** } catch (e: SQLException) {  
 Log.e(*TAG*, e.toString())  
 null  
 }

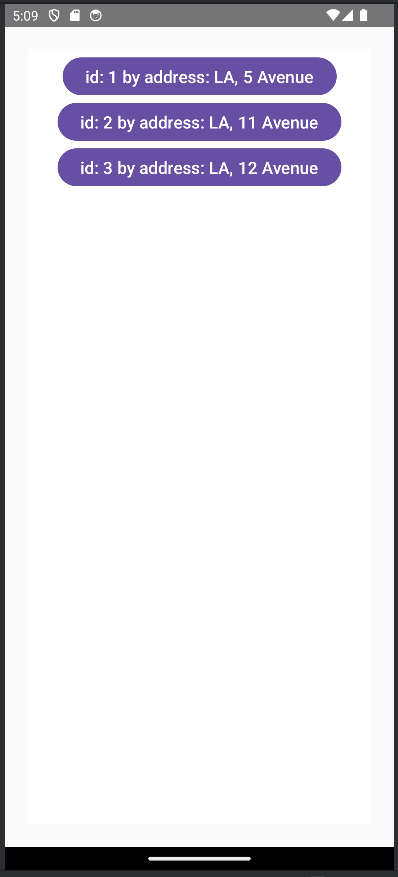
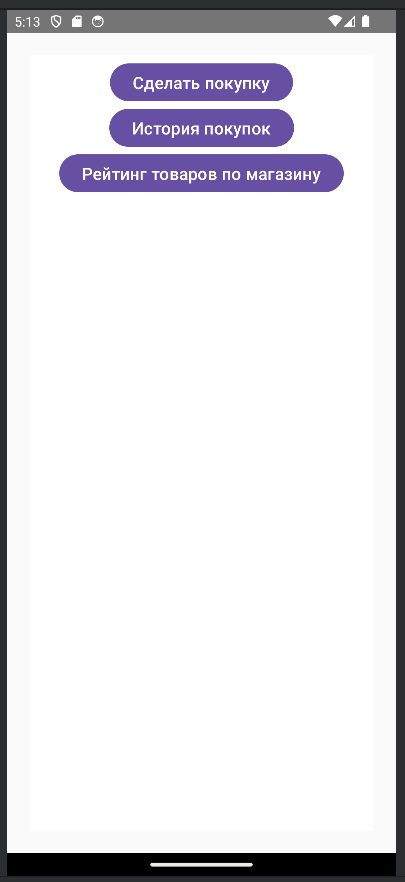
Описание сущностей:

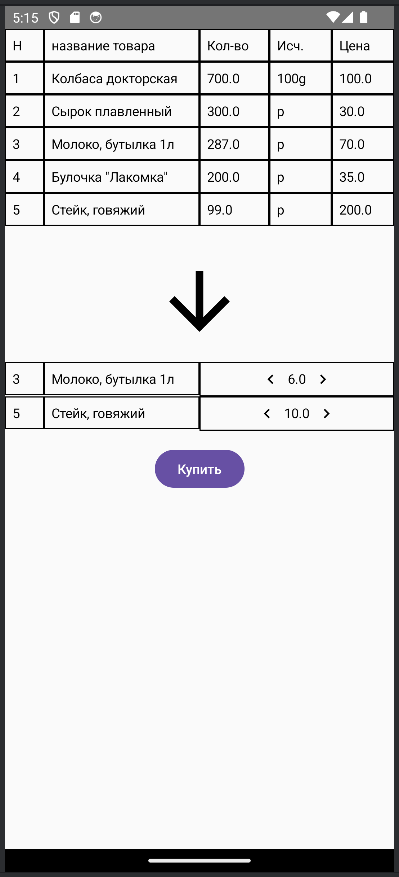
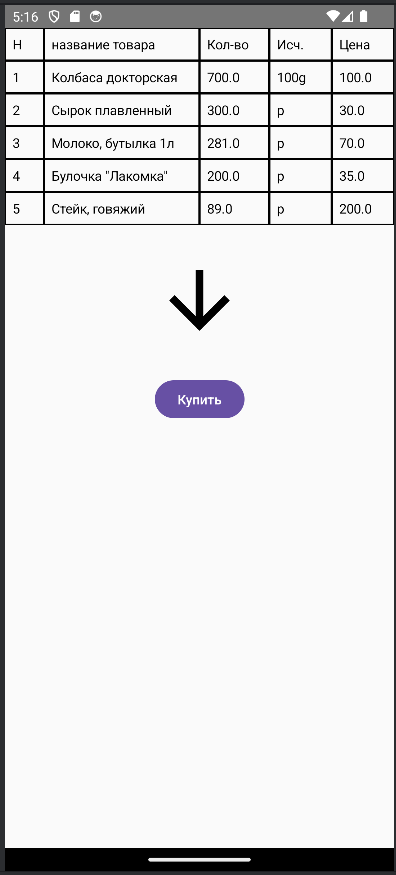
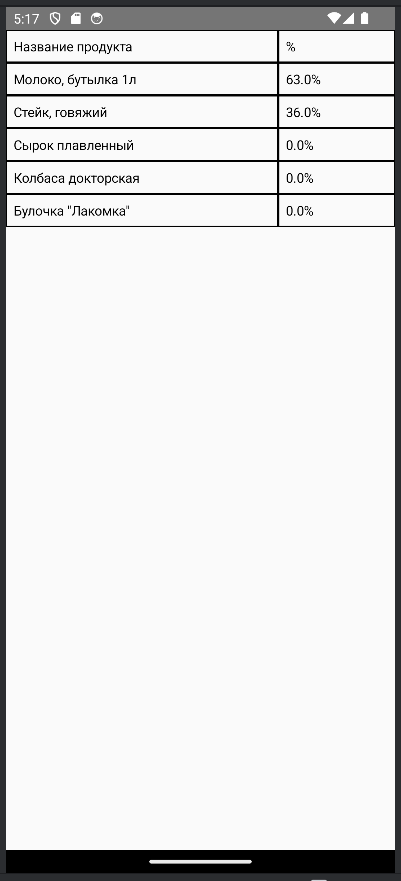
@Entity(tableName = Table.Store.T\_NAME)  
data class StoreEntity(  
 @PrimaryKey(autoGenerate = true)  
 val id: Long = -1,  
 @ColumnInfo(name = Table.Store.ADDRESS)  
 val address: String  
)  
  
@Entity(tableName = Table.Product.T\_NAME)  
data class ProductEntity(  
 @PrimaryKey(autoGenerate = true)  
 val article: Long = -1,  
 @ColumnInfo(name = Table.Product.NAME)  
 val name: String,  
 @ColumnInfo(name = Table.Product.CATEGORY)  
 val category: String?,  
 @ColumnInfo(name = Table.Product.QUANTITY\_TO\_ASSESS)  
 val quantityToAssess: String  
)  
  
@Entity(tableName = Table.CheckList.T\_NAME)  
data class CheckListEntity(  
 @PrimaryKey(autoGenerate = true)  
 val id: Long = -1,  
 @ColumnInfo(name = Table.CheckList.STORE\_ID)  
 val storeId: Long?,  
 @ColumnInfo(name = Table.CheckList.TIME)  
 val time: Long  
)  
  
@Entity(  
 tableName = Table.Accounting.T\_NAME,  
 primaryKeys = [  
 Table.Accounting.STORE\_ID,  
 Table.Accounting.PRODUCT\_ARTICLE  
 ]  
)  
data class AccountingEntity(  
 @ColumnInfo(name = Table.Accounting.STORE\_ID)  
 val storeId: Long,  
 @ColumnInfo(name = Table.Accounting.PRODUCT\_ARTICLE)  
 val productArticle: Long,  
 @ColumnInfo(name = Table.Accounting.COST)  
 val cost: Double?,  
 @ColumnInfo(name = Table.Accounting.AMOUNT)  
 val amount: Double  
)  
  
@Entity(tableName = Table.Purchase.T\_NAME)  
data class PurchaseEntity(  
 @PrimaryKey(autoGenerate = true)  
 val id: Long = -1,  
 @ColumnInfo(name = Table.Purchase.CHECK\_LIST\_ID)  
 val checkListId: Long? = null,  
 @ColumnInfo(name = Table.Purchase.PRODUCT\_ARTICLE)  
 val productArticle: Long,  
 @ColumnInfo(name = Table.Purchase.AMOUNT)  
 val amount: Double  
)

Dao объекты:

@Dao  
interface InitDao {  
 @Insert(onConflict = OnConflictStrategy.IGNORE)  
 suspend fun insertStores(stores: List<StoreEntity>)  
  
 @Insert(onConflict = OnConflictStrategy.IGNORE)  
 suspend fun insertProducts(products: List<ProductEntity>)  
  
 @Insert(onConflict = OnConflictStrategy.IGNORE)  
 suspend fun insertAccounting(accounting: List<AccountingEntity>)  
  
 @Transaction  
 suspend fun initializeDatabase() {  
 val stores = *listOf*(  
 StoreEntity(id = 1, address = "LA, 5 Avenue"),  
 StoreEntity(id = 2, address = "LA, 11 Avenue"),  
 StoreEntity(id = 3, address = "LA, 12 Avenue")  
 )  
  
 val products = *listOf*(  
 ProductEntity(  
 article = 1,  
 name = "Колбаса докторская",  
 category = "meat",  
 quantityToAssess = "100g"  
 ),  
 ProductEntity(  
 article = 2,  
 name = "Сырок плавленный",  
 category = "milk",  
 quantityToAssess = "p"  
 ),  
 ProductEntity(  
 article = 3,  
 name = "Молоко, бутылка 1л",  
 category = "milk",  
 quantityToAssess = "p"  
 ),  
 ProductEntity(  
 article = 4,  
 name = "Булочка \"Лакомка\"",  
 category = "bake",  
 quantityToAssess = "p"  
 ),  
 ProductEntity(  
 article = 5,  
 name = "Стейк, говяжий",  
 category = "meat",  
 quantityToAssess = "p"  
 )  
 )  
  
 val accounting = *listOf*(  
 AccountingEntity(storeId = 1, productArticle = 1, cost = 100.0, amount = 700.0),  
 AccountingEntity(storeId = 1, productArticle = 2, cost = 30.0, amount = 300.0),  
 AccountingEntity(storeId = 1, productArticle = 3, cost = 70.0, amount = 300.0),  
 AccountingEntity(storeId = 1, productArticle = 4, cost = 35.0, amount = 200.0),  
 AccountingEntity(storeId = 1, productArticle = 5, cost = 200.0, amount = 100.0),  
 AccountingEntity(storeId = 2, productArticle = 1, cost = 190.0, amount = 700.0),  
 AccountingEntity(storeId = 2, productArticle = 2, cost = 39.0, amount = 300.0),  
 AccountingEntity(storeId = 2, productArticle = 3, cost = 79.0, amount = 300.0),  
 AccountingEntity(storeId = 2, productArticle = 4, cost = 39.0, amount = 200.0),  
 AccountingEntity(storeId = 2, productArticle = 5, cost = 290.0, amount = 100.0)  
 )  
  
 insertStores(stores)  
 insertProducts(products)  
 insertAccounting(accounting)  
 }  
}  
  
@Dao  
abstract class ProductDao {  
 @Query(  
 """  
 INSERT INTO ${Table.CheckList.T\_NAME}   
 (${Table.CheckList.STORE\_ID}, ${Table.CheckList.TIME})  
 VALUES (:storeId, CURRENT\_TIMESTAMP);  
 """  
 )  
 protected abstract suspend fun insertCheckList(storeId: Long): Long  
  
 @Insert  
 protected abstract suspend fun insertPurchase(purchase: PurchaseEntity): Long  
  
 @Query(  
 """  
 UPDATE ${Table.Accounting.T\_NAME}  
 SET ${Table.Accounting.AMOUNT}=${Table.Accounting.AMOUNT}-:amount  
 WHERE ${Table.Accounting.STORE\_ID}=:storeId   
 AND ${Table.Accounting.PRODUCT\_ARTICLE}=:article;  
 """  
 )  
 protected abstract suspend fun updateAccounting(  
 storeId: Long,  
 article: Long,  
 amount: Double  
 ): Int  
  
 @Transaction  
 open suspend fun executeBuy(  
 storeId: Long,  
 purchases: List<PurchaseEntity>  
 ) {  
 val checkListId = insertCheckList(storeId)  
 purchases  
 .*map* **{ it**.copy(checkListId = checkListId) **}** .*forEach* **{** insertPurchase(**it**)  
 updateAccounting(  
 storeId,  
 **it**.productArticle,  
 **it**.amount  
 )  
 **}** }  
}  
  
@Dao  
interface StoreDao {  
 @Query("SELECT \* FROM ${Table.Store.T\_NAME}")  
 suspend fun getStores(): List<StoreEntity>  
}  
  
@Dao  
abstract class BigQueryDao {  
 @Query(  
 """  
 select t0.${Table.CheckList.ID} as ${Models.History.ColName.CHECK\_LIST\_ID},  
 t2.${Table.Product.NAME} as ${Models.History.ColName.PRODUCT\_NAME},  
 t1.${Table.Purchase.AMOUNT} as ${Models.History.ColName.AMOUNT},  
 t2.${Table.Product.QUANTITY\_TO\_ASSESS} as ${Models.History.ColName.QUANTITY\_TO\_ASSESS},  
 t1.${Table.Purchase.AMOUNT}\*t3.${Table.Accounting.COST} as ${Models.History.ColName.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.PRODUCT\_ARTICLE}=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;  
 """  
 )  
 abstract suspend fun getHistory(storeId: Long): List<Models.History>  
  
 @Query(  
 """  
 select t0.${Table.Accounting.PRODUCT\_ARTICLE} as ${Models.Assortment.ColName.ARTICLE},  
 t1.${Table.Product.NAME} as ${Models.Assortment.ColName.PRODUCT\_NAME},  
 t0.${Table.Accounting.AMOUNT} as ${Models.Assortment.ColName.AMOUNT},  
 t1.${Table.Product.QUANTITY\_TO\_ASSESS} as ${Models.Assortment.ColName.QUANTITY\_TO\_ASSESS},  
 t0.${Table.Accounting.COST} as ${Models.Assortment.ColName.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;  
 """  
 )  
 abstract suspend fun getAssortment(storeId: Long): List<Models.Assortment>  
  
 @Query(  
 """  
 select t2.${Table.Product.NAME} as ${Models.Rating.ColName.PRODUCT\_NAME},  
 round(coalesce(sum(t1.${Table.Purchase.AMOUNT})\*100/(select sum(amount) from ${Table.Purchase.T\_NAME}),0),2)  
 as ${Models.Rating.ColName.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 ${Models.Rating.ColName.AMOUNT\_IN\_PERCENT} desc;  
 """  
 )  
 abstract suspend fun getRating(storeId: Long): List<Models.Rating>  
}

Скриншоты приложения:

**Дополнительное задание**

Произвести миграцию базы данных. В примере к таблице добавляется новая колонка с номерами телефонов магазинов:

val *MIGRATION\_1\_2* = object : Migration(1, 2) {  
 override fun migrate(db: SupportSQLiteDatabase) {  
 db.execSQL("ALTER TABLE ${Table.Store.T\_NAME} ADD COLUMN ${Table.Store.PHONE\_NUMBER} TEXT")  
 }  
}

**Вывод:** в ходе работы получены навыки разработки приложения, использующее технологию ORM, для взаимодействия с базой данных.