Лабораторная работа № 5
Работа с базой данных Room.

1. Работа с БД в Compose.

Для работы с SQLite Jetpack Compose имееет специальную библиотеку Room, которая предоставляет высокоуровневый интерфейс поверх системы базы данных SQLite и которая значительно упрощает взаимодействие с базами данными.

Прежде всего для получения данных из базы данных обычно определяется отдельный модуль репозитория. Однако код приложения, в частности, репозиторий не обращается напрямую к базе данных. Вместо этого все операции с базой данных выполняются с использованием комбинации базы данных Room, DAO и сущностей. DAO (Data Access Object - объект доступа к данным) содержит операторы SQL, необходимые репозиторию для добавления, извлечения и удаления данных в базе данных SQLite.

Объект базы данных Room предоставляет интерфейс к базовой базе данных SQLite. Он также предоставляет репозиторию доступ к объекту DAO. Приложение должно иметь только один экземпляр базы данных Room, который можно использовать для доступа к нескольким таблицам базы данных.

Для определения данных, которые хранятся в отдельной таблице, применяются специальные классы - сущности (entity). Сущности определяют схему таблицы в базе данных, имя таблицы, ее столбцы и сопутствующую информацию. Когда репозиторий получает данные базы данных через DAO, то эти данные как раз представляют объекты классов сущностей. Аналогично, когда репозиторию необходимо добавить в базу данных новые данные, он создает объект сущности и затем вызывает методы добавления, определенные в DAO.

Определение сущности начинается с аннотации @Entity.

Таблице БД необходим столбец, который будет выступать в качестве первичного ключа. Для установки свойства как первичного ключа применяется аннотация @PrimaryKey.

Свойства класса сопоставляются с определенными столбцами БД. По умолчанию сопоставление между свойствами классов и столбцами в таблице

выполняется по имени. С помощью аннотации @ColumnInfo можно переопределить имя задав параметр name.

DAO объявляется как стандартный интерфейс Kotlin с помощью аннотации @Dao.

Интерфейс DAO может содержать функции, которые с помощью дополнительных аннотаций сопоставляются с конкретными инструкциями SQL.

Методы интерфейсов также могут принимать параметры, на которые затем можно ссылаться в выражениях SQL в качестве аргументов. Для вставки значения параметра в SQL-выражение применяется двоеточие «:».

Для сопоставления метода интерфейса с SQL-выражением добавления данных может применяться специальная аннотация @Insert. @Insert аннотация позволяет добавлять сразу несколько объектов в рамках одной транзакции.

Для удаления (в том числе для удаления набора данных) используется аннотация @Delete.

Для обновления используется аннотация @Update.

```
@Dao
interface UserDao {
    @Query("SELECT * FROM users")
    fun getUsers(): LiveData<List<User>>
    @Query("SELECT * FROM users WHERE name = :userName")
    fun getUser(userName: String): List<User>
    @Insert
    fun addUser(user: User)
    @Insert
    fun insertUsers(User... userList)
    @Query("DELETE FROM users WHERE userName = :name")
    fun deleteUser(name: String)
    @Delete
    fun deleteUsers(User... users)
    @Update
    fun updateUsers(User... users)
}
```

База данных Room представляет слой поверх фактической базы данных SQLite, который отвечает за предоставление доступа к экземплярам DAO, связанным с базой данных. Каждое приложение Android должно иметь только один экземпляр базы данных Room. К классу базы данных Room применяется аннотация

@Database, которая объявляет сущности, с которыми должна работать база данных.

```
@Database(entities = [(User::class)], version = 1)
abstract class UserRoomDatabase: RoomDatabase() {

    abstract fun UserDao(): UserDao

    // реализуем синглтон
    companion object {
        private var INSTANCE: UserRoomDatabase? = null
        fun getInstance(context: Context): UserRoom-
Database {

            synchronized(this) {
                var instance = INSTANCE
                if (instance == null) {
                    instance = Room.databaseBuilder(
                        context.applicationContext,
                        UserRoomDatabase::class.java,
                        "User_database"

                    ).fallbackToDestructiveMigration().bu
ild()
                    INSTANCE = instance
                }
                return instance
            }
        }
    }
}
```

Репозиторий содержит код, который вызывает методы DAO для выполнения операций с базой данных. При вызове методов DAO важно отметить, что если метод не возвращает экземпляр LiveData (который автоматически выполняет запросы в отдельном потоке), операцию нельзя выполнить в основном потоке приложения.

```
class UserRepository(private val userDao: UserDao) {
    private val coroutineScope = CoroutineScope(Dispatch-
ers.Main)

    val userList: LiveData<List<User>>? = userDao.ge-
tUsers()

    fun addUser(User: User) {
        coroutineScope.launch(Dispatchers.IO) {
```

```
            userDao.addUser(User)
        }
    }

    fun deleteUser(id:Int) {
        coroutineScope.launch(Dispatchers.IO) {
            userDao.deleteUser(id)
        }
    }
}
```

После объявления всех классов необходимо создать и инициализировать экземпляры базы данных, DAO и репозитория

2. Задание для самостоятельной реализации

Изменить проект из лабораторной работы № 8, так чтобы работа с базой данных осуществлялась с помощью библиотеки Room.

3. Вопросы для самопроверки

- Для чего используется библиотека Room?
- Как использовать библиотеку Room?
- В чем преимущества использования библиотеки Room?

4. Ссылки

- https://metanit.com/kotlin/jetpack/16.1.php
- https://metanit.com/kotlin/jetpack/16.2.php
- https://metanit.com/kotlin/jetpack/16.3.php