



Android Developer Basic

Persistent storage (Database)



Проверить, идет ли запись

Меня хорошо видно && слышно?



Ставим "+", если все хорошо "-", если есть проблемы



Тема вебинара

Persistent storage (Database)



Кирьяков Максим

Android разработчик

Об опыте :

4+ лет опыта коммерческой разработки

@Crafto



Правила вебинара



Активно участвуем



Off-topic обсуждаем в TG



Задаем вопрос
в чат или голосом



Вопросы вижу в чате, могу ответить
не сразу

Условные обозначения



Индивидуально



Время, необходимое на
активность



Пишем в чат



Говорим голосом

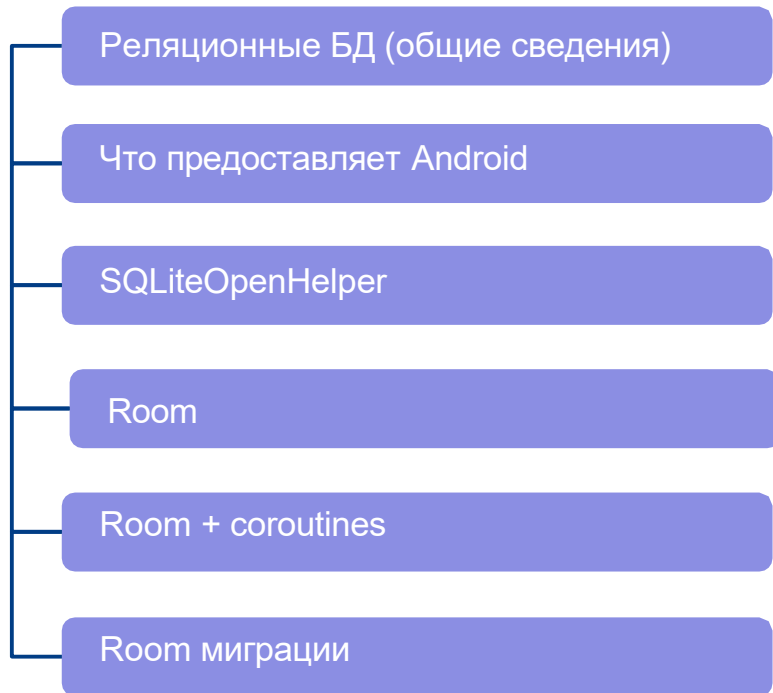


Документ



Ответьте себе или задайте
вопрос

Маршрут вебинара



Цели вебинара

1. Познакомиться с реляционными базами данных
2. Узнать способы работы с БД в Android
3. Научиться использовать Room для работы с БД



Кто работал с базами данных?



Реляционные базы данных



Что такое реляционные базы данных?



Реляционные базы данных

- данные хранятся в таблицах
- число столбцов и их названия заранее известны
- число строк может меняться
- одна строка - одна сущность
- основной интерфейс доступа - SQL (Structured Query Language)



Типы данных в SQLite

- **NULL**: указывает фактически на отсутствие значения
- **INTEGER**: целое число, которое может быть положительным и отрицательным и может занимать 1 -8 байт
- **REAL**: число с плавающей точкой, занимает 8 байт в памяти
- **TEXT**: строка текста в одинарных кавычках
- **BLOB**: бинарные данные



SQLite в Android

Описываем таблицу

```
object NotesDbContract {  
    const val TABLE_NAME = "notes"  
    const val COLUMN_NAME_TITLE = "title"  
    const val COLUMN_NAME_NOTE = "note"  
    const val COLUMN_NAME_DATE = "date"  
}
```

Создаем базу данных

```
class NotesDBHelper (context: Context) :
    SQLiteOpenHelper (context, DB_NAME, null, DB_VERSION) {

    override fun onCreate (db: SQLiteDatabase) {
        db.execSQL (DB_CREATE_NOTES )
    }

    override fun onUpgrade (db: SQLiteDatabase?, oldVersion: Int, newVersion: Int) {
        /* no migrations */
    }

    companion object {
        private const val DB_NAME = "notes_db"
        private const val DB_VERSION = 1

        private const val DB_CREATE_NOTES =
            "CREATE TABLE ${NotesDbContract.TABLE_NAME} (" +
                "${BaseColumns._ID} INTEGER PRIMARY KEY AUTOINCREMENT," +
                "${NotesDbContract.COLUMN_NAME_TITLE} TEXT," +
                "${NotesDbContract.COLUMN_NAME_NOTE} TEXT," +
                "${NotesDbContract.COLUMN_NAME_DATE} INTEGER)"

    }
}
```

Получаем Cursor с данными из БД

```
suspend fun getNote(id: Long): Note? = withContext (ioDispatcher) {  
    val projection = arrayOf(  
        BaseColumns._ID,  
        NotesDbContract.COLUMN_NAME_TITLE ,  
        NotesDbContract.COLUMN_NAME_NOTE ,  
        NotesDbContract.COLUMN_NAME_DATE ,  
    )  
    val selection = "${BaseColumns._ID} = ?"  
    val selectionArgs = arrayOf("$id")  
  
    val cursor = db.query(  
        NotesDbContract.TABLE_NAME ,  
        projection ,  
        selection ,  
        selectionArgs, null, null, null  
    )  
  
    return@withContext cursor?.use { cursor ->  
        if (cursor.moveToFirst ()) parceNote (cursor) else null  
    }  
}
```

Получаем объект из Cursor'a

```
private fun parceNote(cursor: Cursor): Note {  
    val timeStamp =  
        cursor.getLong(cursor.getColumnIndexOrThrow(NotesDbContract.COLUMN_NAME_DATE))  
  
    return Note(  
        id = cursor.getLong(  
            cursor.getColumnIndexOrThrow(BaseColumns._ID)),  
        title = cursor.getString(  
            cursor.getColumnIndexOrThrow(NotesDbContract.COLUMN_NAME_TITLE)),  
        text = cursor.getString(  
            cursor.getColumnIndexOrThrow(NotesDbContract.COLUMN_NAME_NOTE)),  
        date = Date(timeStamp)  
    )  
}
```


Запись данных в БД

```
suspend fun addNote(title: String, note: String) = withContext(ioDispatcher) {  
    val values = ContentValues().apply {  
        put(NotesDbContract.COLUMN_NAME_TITLE, title)  
        put(NotesDbContract.COLUMN_NAME_NOTE, note)  
        put(NotesDbContract.COLUMN_NAME_DATE, Date().time)  
    }  
  
    db.insert(NotesDbContract.TABLE_NAME, null, values)  
}
```

Недостатки

- много кода для работы с базой данных
- нужно конвертировать объекты из курсора
- сложно работать с миграциями
- нет компайл-тайм проверки запросов, о неправильном запросе можно узнать только в рантайме (а можно и не узнать)

Room persistence library



Что такое Room persistence library?



Room -ORM для упрощения работы с SQLite.

ORM (Object-Relational Mapping) — технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных»

Основные компоненты Room

- **Entity** - класс/модель с аннотациями, описывающий таблицу базы данных
- **DAO** (data access object) – это класс, содержащий CRUD (create/read/update/delete) методы для конкретной сущности
- **Database** - абстракция над SQLite скрывающая работу с БД

Entity

```
@Entity(tableName = NoteEntity.TABLE_NAME)
data class NoteEntity(
    @PrimaryKey(autoGenerate = true)
    val id: Long = 0,
    val title: String,
    val note: String,
    @ColumnInfo(name = "date", defaultValue = "CURRENT_TIMESTAMP")
    val date: Date = Date()
) {
    companion object {
        const val TABLE_NAME = "notes"
    }
}
```



Вложенные классы

```
@Entity(tableName = NoteEntity.TABLE_NAME)
data class NoteEntity(
    @PrimaryKey(autoGenerate = true)
    val id: Long = 0,
    val title: String,
    val note: String,
    @ColumnInfo(name = "date", defaultValue = "CURRENT_TIMESTAMP")
    val date: Date = Date(),
    @Embedded
    val image: Image? = null
)

data class Image(
    val fileName: String,
    val fileUri: Uri
)
```

Поля вложенного класса
добавляются в таблицу



Что делать с типом Date, БД не может хранить такой тип?

TypeConver

```
class TypeConverters {  
    @TypeConverter  
    fun fromTimestamp(timestamp: Long): Date = Date(timestamp)  
  
    @TypeConverter  
    fun fromDate(date: Date): Long = date.time  
}
```



Dao

```
@Dao
interface NotesDao {
    @Query("SELECT * FROM ${NoteEntity.TABLE_NAME}")
    fun getAllNotes(): Flow<List<NoteEntity>>

    @Query("SELECT * FROM ${NoteEntity.TABLE_NAME} WHERE id = :id")
    suspend fun getNote(id: Long): NoteEntity?

    @Insert
    suspend fun insert(note: NoteEntity)

    @Update
    suspend fun update(note: NoteEntity)
}
```



Database

```
@Database(version = 1, entities = [NoteEntity::class])
@androidx.room.TypeConverters (TypeConverters::class)
abstract class NotesDB : RoomDatabase () {
    abstract fun notesDao(): NotesDao
}
```

```
private val db = Room.databaseBuilder (
    context,
    NotesDB::class.java,
    "notes_db"
).build()
```



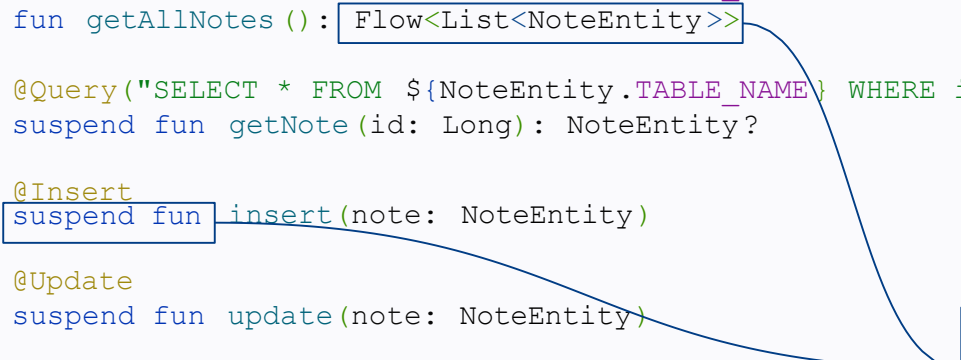
Coroutines

```
@Dao
interface NotesDao {
    @Query("SELECT * FROM ${NoteEntity.TABLE_NAME}")
    fun getAllNotes(): Flow<List<NoteEntity>>

    @Query("SELECT * FROM ${NoteEntity.TABLE_NAME} WHERE id = :id")
    suspend fun getNote(id: Long): NoteEntity?

    @Insert
    suspend fun insert(note: NoteEntity)

    @Update
    suspend fun update(note: NoteEntity)
}
```

A diagram consisting of three blue lines. The first line starts from the `Flow<List<NoteEntity>>` return type of the `getAllNotes` method and points to the top of a box. The second line starts from the `suspend fun insert` method signature and points to the middle of the box. The third line starts from the `suspend fun update` method signature and points to the bottom of the box.

метод автоматически
выполняется на IO

RxJava(2, 3)

```
@Dao
interface NotesDao {
    @Query("SELECT * FROM ${NoteEntity.TABLE_NAME}")
    fun getAllNotes(): Observable<List<NoteEntity>>

    @Query("SELECT * FROM ${NoteEntity.TABLE_NAME} WHERE id = :id")
    fun getNote(id: Long): Single<NoteEntity?>

    @Insert
    fun insert(note: NoteEntity): Completable

    @Update
    fun update(note: NoteEntity): Completable
}
```

метод автоматически
выполняется на IO

LiveData

a

```
@Dao
interface NotesDao {
    @Query("SELECT * FROM ${NoteEntity.TABLE_NAME}")
    fun getAllNotes (): LiveData<List<NoteEntity>>
}
```

метод автоматически
выполняется на IO



Room миграции



Что такое миграции ДБ?



Миграции

```
val MIGRATION_1_2 = object : Migration(1, 2) {  
    override fun migrate(database: SupportSQLiteDatabase) {  
        database.execSQL("ALTER TABLE ${NoteEntity.TABLE_NAME} ADD COLUMN fileUri TEXT")  
        database.execSQL("ALTER TABLE ${NoteEntity.TABLE_NAME} ADD COLUMN fileName TEXT")  
    }  
}  
  
private val db = Room.databaseBuilder(  
    context,  
    NotesDB::class.java,  
    "notes_db"  
)  
.addMigrations(MIGRATION_1_2)  
.build()
```

Автоматические миграции

```
@Database (
    version = 2,
    entities = [NoteEntity::class],
    autoMigrations = [AutoMigration (from = 1, to = 2)]
)
@androidx.room.TypeConverters (TypeConverters::class)
abstract class NotesDB : RoomDatabase () {
    abstract fun notesDao (): NotesDao
}
```



Автоматические миграции

```
build.gradle

android {

    . . .

    defaultConfig {

        . . .

        javaCompileOptions {
            annotationProcessorOptions {
                arguments = ["room.schemaLocation": "$projectDir/schemas".toString()]
            }
        }
    }
}
```

Root связи между таблицами



Зачем связывать таблицы?



One to one

One to one -связь при которой родительский объект связан только с одним дочерним, так же как и дочерний связан только с одним родительским

One to one

```
@Entity(tableName = NoteEntity.TABLE_NAME)
data class NoteEntity(
    @PrimaryKey(autoGenerate = true)
    val noteId: Long = 0,
    val title: String,
    val note: String,
    @ColumnInfo(defaultValue = "CURRENT_TIMESTAMP" )
    val date: Date = Date()
)
```

```
@Entity
data class Image(
    @PrimaryKey
    val imageId: Long,
    val fileName: String,
    val fileUri: Uri,
    val parentNoteId: Long
)
```

поле по которому
связываются объекты

One to one

```
class NoteWithImage (  
    @Embedded  
    val note: NoteEntity,  
  
    // обозначаем отношение с помощью аннотации  
    @Relation (  
        parentColumn = "noteId", // ссылка на primary key родителя  
        entityColumn = "parentNoteId"  
    )  
    val image: Image  
)
```

One to one

Dao

```
@Transaction
@Query("SELECT * FROM ${NoteEntity.TABLE_NAME}")
fun getNotesWithImage(): List<NoteWithImage>
```



One to many

One to many - связь при которой родительский объект связан с 0 или более дочерними объектами, тогда как каждый дочерний связан только с одним родительским

One to many

```
class NoteWithImages (  
    @Embedded  
    val note: NoteEntity,  
  
    // обозначаем отношение с помощью аннотации  
    @Relation (  
        parentColumn = "noteId",  
        entityColumn = "parentNoteId"  
    )  
    val image: List<Image>  
)
```

список дочерних
объектов

One to many

Dao

```
@Transaction
@Query("SELECT * FROM ${NoteEntity.TABLE_NAME}")
fun getNotesWithImage(): List<NoteWithImages>
```



Many to many

Many to many -связь при которой родительский объект связан с 0 или более дочерними объектами, и каждый дочерний может быть связан с 0 и более родительскими

Many to many

```
@Entity(tableName = NoteEntity.TABLE_NAME)
data class NoteEntity(
    @PrimaryKey(autoGenerate = true)
    val noteId: Long = 0,
    val title: String,
    val note: String,
    @ColumnInfo(defaultValue = "CURRENT_TIMESTAMP" )
    val date: Date = Date()
)
```

```
@Entity
data class Image(
    @PrimaryKey
    val imageId: Long,
    val fileName: String,
    val fileUri: Uri
)
```

Many to many

```
// дополнительная таблица обозначающая связь

@Entity(primaryKeys = ["noteId", "imageId"])
data class NoteImageCrossRef (
    val noteId: Long,
    val imageId: Long
)
```


Many to many

```
class NoteWithImages (  
    @Embedded  
    val note: NoteEntity,  
  
    @Relation (  
        parentColumn = "noteId",  
        entityColumn = "imageId",  
        associateBy = Junction (NoteImageCrossRef :: class) // обозначает объединяющий класс  
    )  
    val images: List<Image>  
)  
  
class ImageWithNotes (  
    @Embedded  
    val image: Image,  
  
    @Relation (  
        parentColumn = "imageId",  
        entityColumn = "noteId",  
        associateBy = Junction (NoteImageCrossRef :: class)  
    )  
    val notes: List<NoteEntity>  
)
```

Many to many

Dao

```
@Transaction
@Query("SELECT * FROM ${NoteEntity.TABLE_NAME}")
fun getNoteWithImages (): List<NoteWithImages >
```

```
@Transaction
@Query("SELECT * FROM Image" )
fun getImageWithNotes (): List<ImageWithNotes >
```

Вопросы?



Ставим “+”,
если вопросы есть



Ставим “-”,
если вопросов нет

Цели вебинара

1. Познакомиться с реляционными базами данных
2. Узнать способы работы с БД в Android
3. Научиться использовать Room для работы с БД

Список материалов для изучения

1. [Учебник по SQLite](#)
2. [Google о Room](#)

**Заполните, пожалуйста,
опрос о занятии
по ссылке в чате**

Спасибо за внимание