BIBLIOTECA DE PERSISTENCIA ROOM DATABASE CON KOTLIN

OBJETIVO

En esta guía aprenderemos a almacenar datos en una aplicación Android usando la libreria room para que a futuro puedas implementarlo en tus aplicaciones.

ROOM

Room es una librería de Android que facilita el manejo de una Base de datos (SQLite). Room está compuesto de 3 componentes:

- Base de Datos, contiene el punto de acceso principal para la conexión a la Base de datos de tu app.
- Entidad, Representa una tabla de la Base de Datos.
- DAO, contiene los métodos CRUD para acceder a la Base de Datos.

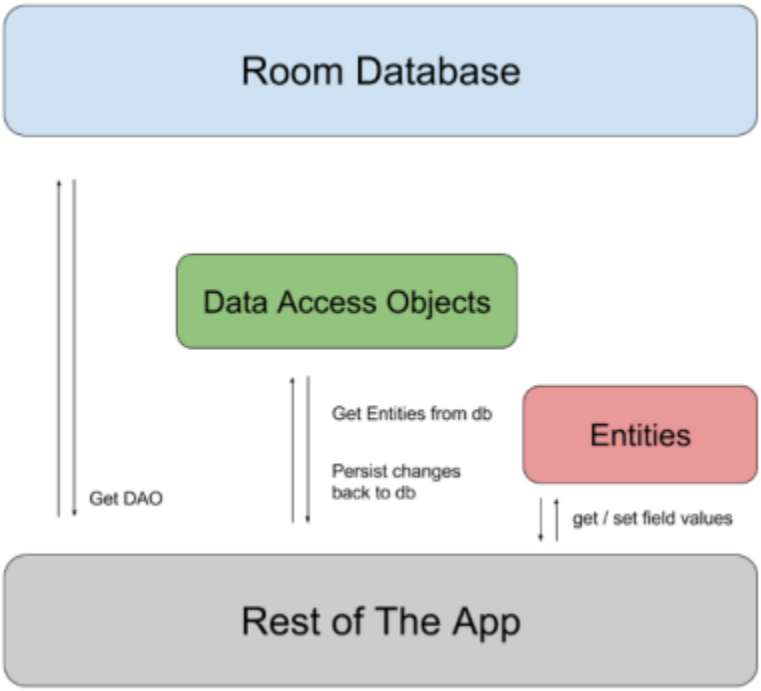

Figure 1: Arquitectura de Room

Una de las principales ventajas de usar Room es la verificación de consultas en tiempo real. Es decir que cada `@Entidad` y `@Query` se verificará al momento de su compilación, de esta forma podremos trabajar con SQLite de una forma más fuida.

Anotaciones

Room se basa en anotaciones para realizar ciertas operaciones, las principales anotaciones que se usa en ROOM

son las siguientes:

- **@Database:** Se utiliza para indicar que la clase actual será la definición de una Base de Datos. Además, dicha clase debería ser abstracta y heredar de RoomDatabase.
- **@Dao:** se utiliza para las interfaces de los DAOs.
- **@Entity:** indica que la clase es una entidad.
- **@PrimaryKey:** indica que el atributo al que acompaña será la clave primaria de la tabla. También podemos establecer que se asigne automáticamente si la incluimos así: @PrimaryKey(autoGenerate = true)».
- **@ColumnInfo:** sirve para personalizar la columna de la base de datos del atributo asociado. Podemos indicar, entre otras cosas, un nombre para la columna diferente al del atributo.
- **@Ignore:** previene que el atributo se almacene como campo en la base de datos.
- **@Index:** para indicar el índice de la entidad.
- **@ForeingKey**: indica que el atributo es una clave foránea relacionada con la clave primaria de otra entidad.
- **@Embedded:** para incluir una entidad dentro de otra.
- **@Insert:** anotación para los métodos de los DAOs que inserten en la base de datos.
- **@Delete:** anotación para los métodos de los DAOs que borren en la base de datos.
- **@Update:** anotación para los métodos de los DAOs que actualicen una entidad en la base de datos.
- **@Query:** anotación para un método del DAO que realice una consulta en la base de datos, la cual deberemos especificar.

Instalación

Para instalar la biblioteca ROOM en nuestro proyecto se debe agregar las siguientes lineas en el achivo build.gradle.kts: del proyecto:

```
plugins {

    //plugin ksp
    id("com.google.devtools.ksp") version "1.9.22-1.0.17" apply false

}
```

Del módulo(App)

```
plugins {

    id("com.google.devtools.ksp") //plugin ksp

}


dependencies {
        //dependencias de la libreria Room
    val room_version = "2.6.1"

    implementation("androidx.room:room-runtime:$room_version")
    annotationProcessor("androidx.room:room-compiler:$room_version")
```

```
    implementation("com.google.devtools.ksp:symbol-processing-api:1.9.22-1.0.17")
    // To use Kotlin Symbol Processing (KSP)
    ksp("androidx.room:room-compiler:$room_version")
}
```

Entidad

Una entidad (@Entity) representa a una tabla de la Base de datos. Internamente Room creará las tablas correspondientes para todas las clases que tienen la anotación @Entity en nuestro proyecto. Por ejemplo si nuestra base de datos tiene la tabla Persona, debemos crear el `data class Persona` de la siguiente forma:

Persona.kt

```
@Entity
data class Persona(

    @PrimaryKey val id: Long,
    var nombre: String? = null,
    var apellido: String? = null,
    var telefono: String? = null,
    var imagen: String? = null
){

    override fun toString(): String {
        return nombre ?: ""
    }

}
```

DAO

El Data Access Object (DAO) es una interfaz donde se especifica todos los métodos con los que accederemos a las tablas de nuestra DB

PersonaDao.kt

```
@Dao
public interface PersonaDao {
    @Query("select * from persona")
    fun selectPersonas(): List<Persona>

    @Insert
    fun insertPersona(vararg per: Persona)

    @Update
    fun updatePersona(p: Persona)
```

```
    @Query("select * from persona where id=:id")
    fun selecPersonaBydId(id:Long):Persona

    @Query("update persona set nombre=:nombre where id=:id")
    fun updateNombre(id:Long,nombre:String)


}
```

La Base de Datos y su instancia

Se lo define en un clase kotlin (AppDataBase.kt) que debe contener además el DAO

AppDataBase.kt

```
@Database(entities =  [Persona::class],version = 1)
abstract class AppDatabase: RoomDatabase() {
    abstract fun personasDao(): PersonaDao
}
```

La instancia del acceso a la Base de Datos debe estar disponible en todo el ciclo de vida de la aplicación por tal motivo se lo define en una clase que extenda a la clase `Application`.

PersonasApplication.kt

```
class PersonasApplication:Application () {
    companion object{
        lateinit var db :AppDatabase
    }

    override fun onCreate() {
        super.onCreate()
        PersonasApplication.db =
Room.databaseBuilder(applicationContext,AppDatabase::class.java,
            "personas.db"
        )
            .fallbackToDestructiveMigration() // para destruir todos los datos al
migrar de una version a otra
            .allowMainThreadQueries() // para habilitar consultas a la db en el
thread principal
            .build()
    }
}
```

Finalmente agregamos en el atributo `android:name` del archivo `AndroidManifest.xml` la referencia a la clase

anterios.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <application
        android:name=".PersonasApplication"

        ...
  />
```

Accediendo a Room

Para acceder a los datos se debe obtener la instancia de la clase que representa a la Base de Datos (PersonasApplication.db) dentro de la Actividad Principal.

```
PersonasApplication.db.personasDao()
```

Realizando consultas

```
var personasList : List<Persona> =
PersonasApplication.db.personasDao().selectPersonas()
        for (p in personasList){
            println(p.nombre)
        }
```

Consulta tipo Insert

```
   val p = Persona(Date().time,"Teofilo","copa","75486932","no tiene")
   PersonasApplication.db.personasDao().insertPersona(p)
```

Update

```
val p = Persona(22)
p.nombre = "NOMBRE MODIFICADO"
p.apellido = "APELLIDO MODIFICADO"
PersonasApplication.db.personasDao().updatePersona(p)
```

Delete

```
db.getDao().deleteCategoria(catEl);
```

Visualizar los Datos desde la consola

Es posible conectarse al archivo de la Base de Datos directamente usando la herramienta adb de Android Studio desde una terminal. Para ello debemos listar primero los dispositivos conectados:

```
adb devices
List of devices attached
emulator-5554   device
```

Para conectarnos a un dispositivo usamos el siguiente comando:

```
adb -s emulator-5554 shell
```

Donde **emulator-5554** es el nombre del emulador al cual nos conectaremos.

Una vez conectado ejecutamos el comando `sqlite3` y la ruta de nuestro archivo de Base de Datos, que generalmente es `/data/data/<nombre del paquete de nuestro proyecto android>/databases/NombreDb`. Ejemplo:

```
sqlite3 /data/data/com.teofilo.myapp/databases/DbPublicaciones
```

Una vez conectado a la Base de Datos SQLite, usando el comando **.tables** podremos ver las tablas de nuestra Base de Datos.

```
sqlite> .tables
android_metadata  publicaciones
```

Desde la consola de SQLite se puede ejecutar todas los comandos SQL para operar con nuestras tablas:

```
sqlite> select * from publicaciones;
2|titulo actualizado|descripcion actualizado
3|Primera publicacion|Descripcion actualizada en 3
4|Primera publicacion|Descripcion de primera publicacion
7|titulo2|Descripcion 2
```

Es posible descargar el archivo de Base de Datos desde el emulador a nuestra maquina, para ello podemos recurrir al comando **adb pull**

o usar la Herramienta **Device File Explorer** de Android Studio.

```
adb -s emulator-5554 pull /data/data/com.example.lab4ejemplo/databases/personas.db
/tmp/personas.db
```

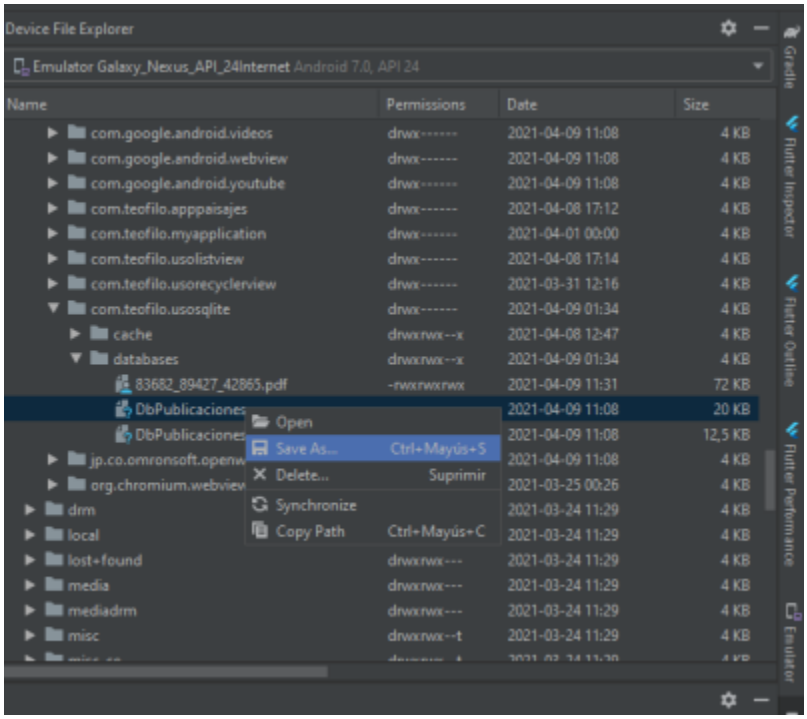

Figure 2: Herramienta Android Device Explorer de Android Studio

~~===== Práctico Nro. 14 ===== Usando el modelo de datos del práctico 13 (menú de productos de un restaurant) agregar las operaciones de CRUD (Create, Update, Delete) y un filtro buscador, usando ROOM.~~