1. To **Change the Size of ImageView**:

Ex: in the XML File You Add Under the ImageView

<ImageView  
 android:id="@+id/imageView"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent"  
 app:srcCompat="@drawable/background"  
 android:scaleType="fitXY"/>

**Note**: This Will Make **the Image Expand** to **Take All the Screen**

EX: to make the image **Take All the Screen** **Without Expanding**

<ImageView  
 android:id="@+id/imageView"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:scaleType="fitXY"  
 android:adjustViewBounds="true"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent"  
 app:srcCompat="@drawable/art\_background" />

**Note**: For **More Ways** Visit this [WebSite](https://guides.codepath.com/android/Working-with-the-ImageView)

1. **Save in SQLite**:
2. Create **DBHelper class**:

class DBHelper(  
 context: Context?,  
 name: String?= "details.db",  
 factory: SQLiteDatabase.CursorFactory?= null,  
 version: Int= 1,  
 private val tableName: String= "notes"  
)

: SQLiteOpenHelper(context, name, factory, version) {  
 override fun onCreate(p0: SQLiteDatabase?) {  
 p0?.execSQL("create table $tableName (Note Text)")  
 }  
  
 override fun onUpgrade(p0: SQLiteDatabase?, p1: Int, p2: Int) {}  
  
 fun saveNotes(note: String): Long {  
 val cv= ContentValues()  
 cv.put("Note",note)  
 return *writableDatabase*.insert(tableName,null,cv)  
 }  
}

1. **In Main Class**:

val dbHelper= DBHelper(this)

val check= dbHelper.saveNotes(noteEntry.*text*.toString())  
val wrongCode: Long= -1  
if (check != wrongCode) {  
 StyleableToast.makeText(this, "Saved Successfully!!\n$check", R.style.*mytoast*).show()  
}  
else  
 StyleableToast.makeText(this,"Something Went Wrong!!\n$check",R.style.*mytoast*).show()

1. The **Deference** between (**this** and **applicationContext**) when we need to use Context:

If we are going to use it **inside the same activity** it’s ok to use (**this**)

**But** if we need to **send it to another function** or **another class**, better to use (**applicationContext**).

Because if we pass the activity context and that activity got destroyed the **app will crash**.

1. **Read from SQLite**
2. In DBHelper class

fun gettingNotes(): ArrayList<String>{  
 return try{  
 val notes= *arrayListOf*<String>()  
 val cursor =  
 sqLiteDatabase.query(tableName, *arrayOf*("Note"), "Note=?", *arrayOf*("Good"), null, null, null)  
 cursor.moveToFirst()  
 while (!cursor.*isAfterLast*){  
 notes.add(cursor.getString(0))  
 cursor.moveToNext()  
 }  
 notes  
 } catch (e:Exception){  
 *arrayListOf*("Error")  
 }  
}

**Note: SQL Query** = select Note from tablenName where Note= Good

Note: If you want to **select all the table** you can replace all with null (Except table name)

Ex: val cursor =  
 sqLiteDatabase.query(tableName, null ,null, null, null, null, null)

Note: You can Replace the index from **set number to index that the program locate**

EX: notes.add(cursor.getString(cursor.getColumnIndexOrThrow("Note")))

1. **In Main Class**

private lateinit var notes: ArrayList<String>

notes= dpHlpr.gettingNotes()

**Note: Another Way to Read The Entire Table**

val selectQuery = "SELECT \* FROM $tableName "var cursor: Cursor? = null  
try {  
 cursor = sqLiteDatabase.rawQuery(selectQuery, null)  
} catch (e: SQLiteException){  
 return ArrayList()  
}  
var noteText: String  
if(cursor.moveToFirst()){  
 do {  
 noteText = cursor.getString(cursor.getColumnIndexOrThrow("Note"))  
 noteList.add(noteText)  
 } while (cursor.moveToNext())  
}

1. **Delete From SQLite**

fun deleteNotes(pk: Int): Int{  
 return sqLiteDatabase.delete(tableName,"PK=?", *arrayOf*("$pk"))  
}

1. **Edit To SQLite**

fun updateNotes(pk:Int, newNote: String): Int{  
 val contentValue= ContentValues()  
 contentValue.put("Note",newNote)  
 return sqLiteDatabase.update(tableName,contentValue,"PK=?", *arrayOf*("$pk"))  
}

1. If we **Upgrade the version** to change the table columns or add new columns we make sure to **change onUpgrade:**

override fun onCreate(p0: SQLiteDatabase?) {  
 p0?.execSQL("create table $tableName (PK INTEGER PRIMARY KEY AUTOINCREMENT, Note Text)")  
}  
  
override fun onUpgrade(p0: SQLiteDatabase?, p1: Int, p2: Int) {  
 p0?.execSQL("DROP TABLE IF EXISTS $tableName")  
 onCreate(p0)  
}

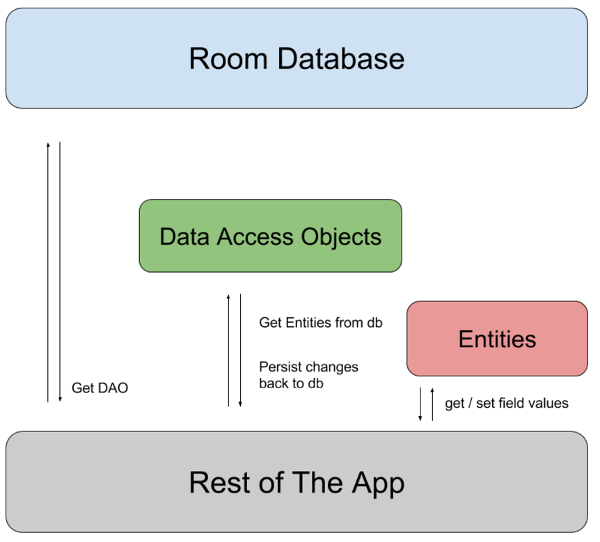
1. **Room**

Room is a persistence library and an abstraction layer over SQLite.

Room annotations (@Entity, @Dao, @Database ) make it easier to save data to SQLite.

Room verifies SQL queries during compile time, which means there will never be any runtime errors due to query syntax.

While it may take some time to get used to the setup, once you are familiar with it, you will notice that room makes using SQL much easier.  It also makes integration with other Android architectural components easier.  This will become more evident when we look at View Models and Live Data.



Source Code: <https://github.com/KPRanjithKumar/SQLiteRoomNameLocation>

Note: to Use Room We Need To Add To Our Gradle File

After this:

plugins **{** id 'com.android.application'  
 id 'kotlin-android'  
**}**apply plugin: 'kotlin-kapt'

and the implementations:

implementation 'androidx.room:room-common:2.3.0'  
implementation 'androidx.room:room-ktx:2.3.0'  
implementation "org.jetbrains.kotlinx:kotlinx-coroutines-android:1.4.1"  
implementation "androidx.room:room-runtime:2.3.0"  
implementation 'android.arch.persistence.room:runtime:1.1.1'  
kapt "androidx.room:room-compiler:2.3.0"