**Разработка базы данных с помощью Room**

Все сталкиваются рано или поздно с созданием локальной базы данных в своих приложениях. Часто, после нескольких часов гугления у нас остается список из кучи ORM которые могут нам помочь с разработкой БД. Я уже писал про самостоятельную разработку БД, еще очень давно, с того времени много чего поменялось, мой уровень знаний вырос, и я сейчас бы не советовал использовать тот способ, так как я писал уже про работу с БД с помощью Realm который разы удобней и проще чем написание и поддержка базы на стандартных методах и классах андроида. Тем более Room является библиотекой которую сам Google советует использовать как БД.  
  
Вот эти две статьи по базам данных. Первая и очень старая. Этот вариант предпочитают использовать только хардкорные трукодеры которые не признают библиотеки, и которым нравится прям контролировать все все. Второй вариант представляет собой библиотеку которая делает все за вас, а вам нужно просто создать модели по которым будет строится БД и дальше уже просто стучаться в БД на чтение, запись, апдейт или удаление данных.

* [Работа с базой данных Android (31 января 2013 г.)](http://dajver.blogspot.com/2013/01/android.html)
* [Разработка базы данных с помощью Realm (15 июня 2017 г.)](http://dajver.blogspot.com/2017/06/realm.html)

А сегодня я расскажу про еще одну ORM которая помогает реализовывать базу данных, и при чем помогает это сделать красиво и без лишних движений. Начнем мы как всегда с настройки проекта, так как нам нужно подключить кучу библиотек без которых наша жизнь была бы скучна и грустна.  
  
Приложение будет очень простое, оно будет уметь отображать данные в списке на главном экране, и уметь удалять из списка по одному элементу. И у нас будет отдельный экран для добавления данных в БД, в котором будут два поля и кнопка добавить. Вот и вся функциональность, но этого я думаю будет достаточно что бы понять основы.  
  
Добавляем в app/build.gradle наши библиотеки. А еще по недавней традиции добавляем java 8 в проект, так будет красивее.   
  
**app/build.gradle**

android {

...

compileOptions {

sourceCompatibility JavaVersion.VERSION\_1\_8

targetCompatibility JavaVersion.VERSION\_1\_8

}

}

dependencies {

implementation fileTree(dir: 'libs', include: ['\*.jar'])

implementation 'com.android.support:appcompat-v7:26.1.0'

implementation 'com.android.support:recyclerview-v7:26.1.+'

implementation 'com.jakewharton:butterknife:8.8.1'

annotationProcessor 'com.jakewharton:butterknife-compiler:8.8.1'

implementation "android.arch.persistence.room:runtime:1.0.0"

annotationProcessor "android.arch.persistence.room:compiler:1.0.0"

}

Как видно из списка dependencies, у нас подключен RecyclerView для отображения списка, Butter Knife для простого доступа к вьюхам, и сам Room для создания БД. Вот и все, у нас есть все что нам нужно для создания красоты.   
  
Начнем мы с того что создадим модель которая будет иметь поля в которые мы будем сохранять данные.   
  
**DataModel.java**

import android.arch.persistence.room.Entity;

import android.arch.persistence.room.PrimaryKey;

import android.support.annotation.NonNull;

@Entity

public class DataModel {

@NonNull

@PrimaryKey

private String title;

private String description;

@NonNull

public String getTitle() {

return title;

}

public void setTitle(@NonNull String title) {

this.title = title;

}

public String getDescription() {

return description;

}

public void setDescription(String description) {

this.description = description;

}

}

Вот такая моделька у нас будет. В ней у нас будет тайтл который в то же время будет у нас @PrimaryKey для сохранения связей между таблицами, но так как у нас таблица одна нам пока связывать ничего особо не нужно, и еще у нас будет поле описание. Так же у нас вверху над классом стоит аннотация @Entity, она значит для Room что этот класс будет использовать как таблица в БД.  
  
Дальше нам нужно создать интерфейс Dao который будет расказывать Room что мы будем делать с нашей моделью (таблицей). Для этого мы создаем интерфейс и определяем в нем методы которые нам нужны для работы с БД, в моем случае это будет запись, удаление и чтение.  
  
**DataDao.java**

import android.arch.persistence.room.Dao;

import android.arch.persistence.room.Delete;

import android.arch.persistence.room.Insert;

import android.arch.persistence.room.Query;

import com.project.dajver.roomdatabaseexample.db.model.DataModel;

import java.util.List;

@Dao

public interface DataDao {

@Insert

void insert(DataModel dataModel);

@Delete

void delete(DataModel dataModel);

@Query("SELECT \* FROM DataModel")

List<DataModel> getAllData();

//пример запроса с выборкой

@Query("SELECT \* FROM DataModel WHERE title LIKE :title")

List<DataModel> getByTitle(String title);

}

Как видно из этого интерфейса, мы определили этот интерфейс аннотацией @Dao, она объясняет Room что мы будем делать с таблицей, мы в нее можем записать данные, удалить их и получить список всех данных. Каждый метод мы инициализируем аннотациями которые указывают то или иное действие. @Insert — очевидно значит запись в БД. @Delete — очевидно удаление. и @Query() — у нас выполняет действия по выполнению SQL запросов к БД, если захотите закостамизировать какие-то реквесты, на пример поиск по БД, вам достаточно просто вписать SQL в эту аннотацию и в зависимости от параметров которые вы там укажите, Room вернет вам ваши данные из БД.  
  
Дальше нам нужно создать расширение для нашего интерфейса БД, который будет хранить в себе абстрактный метод для обращения к интерфейсу, что бы мы могли вызвать наши методы по записи, удалению и получению данных.  
  
**DatabaseHelper.java**

@Database(entities = { DataModel.class }, version = 1, exportSchema = false)

public abstract class DatabaseHelper extends RoomDatabase {

public abstract DataDao getDataDao();

@Override

protected SupportSQLiteOpenHelper createOpenHelper(DatabaseConfiguration config) {

return null;

}

@Override

protected InvalidationTracker createInvalidationTracker() {

return null;

}

}

В этом классе мы указали в аннотации @Database, что у нас будет использоваться класс DataModel как таблица в которой будут храниться данные. version = 1 — у нас значит версию базы данных, при обновлении БД нужно будет только увеличивать версию и никаких сложных действий больше делать не придется, все остальное Room сделает сам. exportSchema = false — я использовал для того что бы не было постоянных ворнингов что схема не может быть построена или сохраненна. По сути каждый раз когда вы создаете БД создается файл схемы БД в JSON, и каждый раз при обновлении БД оно создает ее бекап что бы можно было видеть что было в старой и что появилось в новой. Детальней можно прочесть [тут на стеке](https://stackoverflow.com/a/44645943/520349), может кому-то эта функция понадобится.  
  
Ну и собственно наш единственный абстрактный метод abstract DataDao getDataDao() который возвращает все методы по БД которые у нас созданны в интерфейсе Dao.  
  
Теперь нам нужно создать инстанс БД в синглтоне, не создавать же нам его каждый раз. По этому я выбрал класс Application который создается во время первого запуска приложения, и живет все время пока приложение работает. В нем я создал инстанс Room, а точней инстанс нашего DatabaseHelper который мы создали ранее.  
  
**App.java**

import android.app.Application;

import android.arch.persistence.room.Room;

import com.project.dajver.roomdatabaseexample.db.DatabaseHelper;

public class App extends Application {

private static App instance;

private DatabaseHelper db;

public static App getInstance() {

return instance;

}

@Override

public void onCreate() {

super.onCreate();

instance = this;

db = Room.databaseBuilder(getApplicationContext(), DatabaseHelper.class, "data-database")

.allowMainThreadQueries()

.build();

}

public DatabaseHelper getDatabaseInstance() {

return db;

}

}

Все что происходит вокруг этого класса я думаю можно не описывать, хочу только остановиться на методе onCreate() в котором у нас создается объект класса DatabaseHelper. А точнее мы создаем его экземпляр с помощью Room.databaseBuilder, и называем его каким-то своим произвольным названием которое вам будет хотеться его назвать в моем случае это data-database. Приставка database не обязательна, это просто для примера. Ну и allowMainThreadQueries() разрешает нам делать запросы сразу в UI потоке без лишних обработчиков.  
  
Далее создадим адаптер в котором будем отображать данные из БД.  
  
**SomeDataRecyclerAdapter.java**  
import android.content.Context;

import android.support.v7.widget.RecyclerView;

import android.view.LayoutInflater;

import android.view.View;

import android.view.ViewGroup;

import android.widget.TextView;

import com.project.dajver.roomdatabaseexample.R;

import com.project.dajver.roomdatabaseexample.db.model.DataModel;

import java.util.ArrayList;

import java.util.List;

import butterknife.BindView;

import butterknife.ButterKnife;

public class SomeDataRecyclerAdapter extends RecyclerView.Adapter<RecyclerView.ViewHolder>{

private List<DataModel> dataModels = new ArrayList<>();

private OnDeleteListener onDeleteListener;

private Context context;

public SomeDataRecyclerAdapter(Context context, List<DataModel> dataModels) {

this.context = context;

this.dataModels = dataModels;

}

@Override

public RecyclerView.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {

View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.item\_some\_data, parent, false);

return new NewsViewHolder(view);

}

@Override

public void onBindViewHolder(final RecyclerView.ViewHolder holder, final int position) {

final NewsViewHolder viewHolder = (NewsViewHolder) holder;

viewHolder.title.setText(dataModels.get(position).getTitle());

viewHolder.description.setText(dataModels.get(position).getDescription());

}

@Override

public int getItemCount() {

return dataModels.size();

}

public class NewsViewHolder extends RecyclerView.ViewHolder {

@BindView(R.id.title)

public TextView title;

@BindView(R.id.description)

public TextView description;

@BindView(R.id.delete)

public TextView delete;

public NewsViewHolder(View itemView) {

super(itemView);

ButterKnife.bind(this, itemView);

delete.setOnClickListener(view -> {

onDeleteListener.onDelete(dataModels.get(getAdapterPosition()));

dataModels.remove(getAdapterPosition());

notifyItemRemoved(getAdapterPosition());

});

}

}

public void setOnDeleteListener(OnDeleteListener onDeleteListener) {

this.onDeleteListener = onDeleteListener;

}

public interface OnDeleteListener {

void onDelete(DataModel dataModel);

}

}

Вполне себе стандартный адаптер. В него мы передаем список с DataModel и дальше в onBindViewHolder биндим эти данные во вьюхи. Так же у нас есть колбек который возвращает клик по крестику и вовзращает этот колбек в активити для удаления айтема из списка. Я даже не знаю что тут еще описывать еще. По идее все уже должны быть знакомы с этим адаптером, это вроде как стандарт.  
  
Ну и файл разметки для адаптера.  
  
**item\_some\_data.xml**

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:orientation="vertical"

android:padding="10dp">

<LinearLayout

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:gravity="center\_vertical"

android:orientation="horizontal">

<LinearLayout

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:layout\_weight="0.1"

android:orientation="vertical">

<TextView

android:id="@+id/title"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:text="TextView"

android:textColor="@android:color/black"

android:textSize="18sp" />

<TextView

android:id="@+id/description"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:layout\_marginTop="10dp"

android:text="TextView" />

</LinearLayout>

<TextView

android:id="@+id/delete"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:padding="10dp"

android:text="[X]"

android:textColor="@android:color/black"

android:textSize="18sp" />

</LinearLayout>

</LinearLayout>

А теперь осталось написать активити с списком и активити добавления. Начнем мы с главного экрана со списком.   
  
**MainActivity.java**

import android.content.Intent;

import android.os.Bundle;

import android.support.v7.app.AppCompatActivity;

import android.support.v7.widget.LinearLayoutManager;

import android.support.v7.widget.RecyclerView;

import android.view.Menu;

import android.view.MenuItem;

import com.project.dajver.roomdatabaseexample.App;

import com.project.dajver.roomdatabaseexample.R;

import com.project.dajver.roomdatabaseexample.db.DatabaseHelper;

import com.project.dajver.roomdatabaseexample.db.model.DataModel;

import com.project.dajver.roomdatabaseexample.ui.AddDataActivity;

import com.project.dajver.roomdatabaseexample.ui.main.adapter.SomeDataRecyclerAdapter;

import butterknife.BindView;

import butterknife.ButterKnife;

public class MainActivity extends AppCompatActivity implements SomeDataRecyclerAdapter.OnDeleteListener {

@BindView(R.id.recyclerView)

RecyclerView recyclerView;

private DatabaseHelper databaseHelper;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

ButterKnife.bind(this);

recyclerView.setLayoutManager(new LinearLayoutManager(this, LinearLayoutManager.VERTICAL,false));

databaseHelper = App.getInstance().getDatabaseInstance();

}

@Override

public boolean onCreateOptionsMenu(Menu menu) {

getMenuInflater().inflate(R.menu.menu\_add\_button, menu);

return true;

}

@Override

public boolean onOptionsItemSelected(MenuItem item) {

switch (item.getItemId()) {

case R.id.action\_add: {

startActivity(new Intent(this, AddDataActivity.class));

break;

}

}

return false;

}

@Override

protected void onResume() {

super.onResume();

SomeDataRecyclerAdapter recyclerAdapter = new SomeDataRecyclerAdapter(this, databaseHelper.getDataDao().getAllData());

recyclerAdapter.setOnDeleteListener(this);

recyclerView.setAdapter(recyclerAdapter);

}

@Override

public void onDelete(DataModel dataModel) {

databaseHelper.getDataDao().delete(dataModel);

}

}

В onCreate() мы создали инстанс DatabaseHelper что бы можно было получать данные из БД и удалять их, и указали RecyclerView какой LayoutManager ему использовать. В onCreateOptionsMenu() и onOptionsItemSelected() мы делаем менюшку в тулбаре. В onResume() создаем адаптер, и каждый раз когда мы удем возвращаться с экрана добавления у нас будет обновленный адаптер с внесенными туда данными. Ну и onDelete() который по клику удаляет айтем из списка и БД.  
  
Разметка активити и файл с пунктом меню для тулбара будет выглядеть так:  
  
**activity\_main\_xml**

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:app="http://schemas.android.com/apk/res-auto"

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

tools:context="com.project.dajver.roomdatabaseexample.ui.main.MainActivity">

<android.support.v7.widget.RecyclerView

android:id="@+id/recyclerView"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent" />

</LinearLayout>

**menu\_add\_button.xml**

<?xml version="1.0" encoding="utf-8"?>

<menu

xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:app="http://schemas.android.com/apk/res-auto"

xmlns:tools="http://schemas.android.com/tools"

tools:context=".activity.MainActivity">

<item

android:id="@+id/action\_add"

android:title="Add"

app:showAsAction="always"/>

</menu>

Как видно из кода сверху у нас все тривиально. Список в мейн активити, и кнопка в меню которая видна всегда по умолчанию.  
  
Дальше давайте сделаем экран добавления. В нем у нас будет как я говорил ранее — два поля и кнопка для добавления. После добавления экран закрывается.  
  
**AddDataActivity.java**

import android.os.Bundle;

import android.support.v7.app.AppCompatActivity;

import android.widget.EditText;

import com.project.dajver.roomdatabaseexample.App;

import com.project.dajver.roomdatabaseexample.R;

import com.project.dajver.roomdatabaseexample.db.DatabaseHelper;

import com.project.dajver.roomdatabaseexample.db.model.DataModel;

import butterknife.BindView;

import butterknife.ButterKnife;

import butterknife.OnClick;

public class AddDataActivity extends AppCompatActivity {

@BindView(R.id.title)

EditText title;

@BindView(R.id.description)

EditText description;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_add);

ButterKnife.bind(this);

}

@OnClick(R.id.save)

public void onSaveClick() {

DatabaseHelper databaseHelper = App.getInstance().getDatabaseInstance();

DataModel model = new DataModel();

model.setTitle(title.getText().toString());

model.setDescription(description.getText().toString());

databaseHelper.getDataDao().insert(model);

finish();

}

}

Единственное что нас тут интересует это onSaveClick(), в нем мы создаем инстанс DatabaseHelper, а дальше заполняем нашу модель DataModel, и передаем этот объект на запись в insert. И завершаем активити.   
  
Разметка для класса добавления.  
  
**activity\_add.xml**

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical"

android:padding="10dp">

<EditText

android:id="@+id/title"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:layout\_marginTop="10dp"

android:ems="10"

android:hint="Title"

android:inputType="textPersonName" />

<EditText

android:id="@+id/description"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:layout\_marginTop="10dp"

android:ems="10"

android:hint="Description"

android:inputType="textPersonName" />

<Button

android:id="@+id/save"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:layout\_marginTop="10dp"

android:onClick="onSaveClick"

android:text="Save" />

</LinearLayout>

Осталось не забыть добавить активити в манифест, и начать компиляцию.   
  
Ну и это вроде бы все что нужно для создания небольшого приложения с базой данных. По сути теперь после того как вы все это собрали в кучу, у вас должно запустится приложение с пустым экраном, и кнопкой вверху на тулбаре ADD, после нажатия на которую вы сможете перейти на экран добавления, и после добавления увидите все что вы ввели в списке на главном экране.