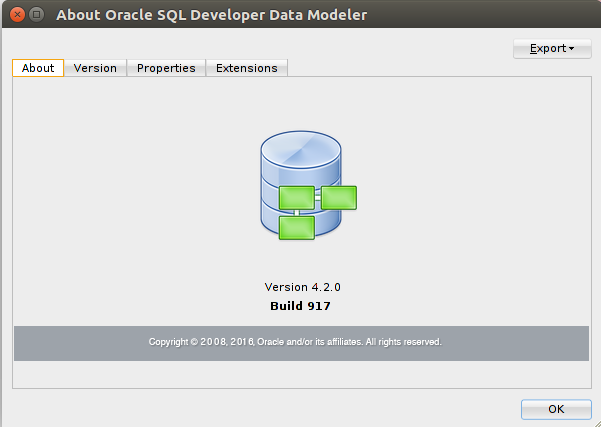
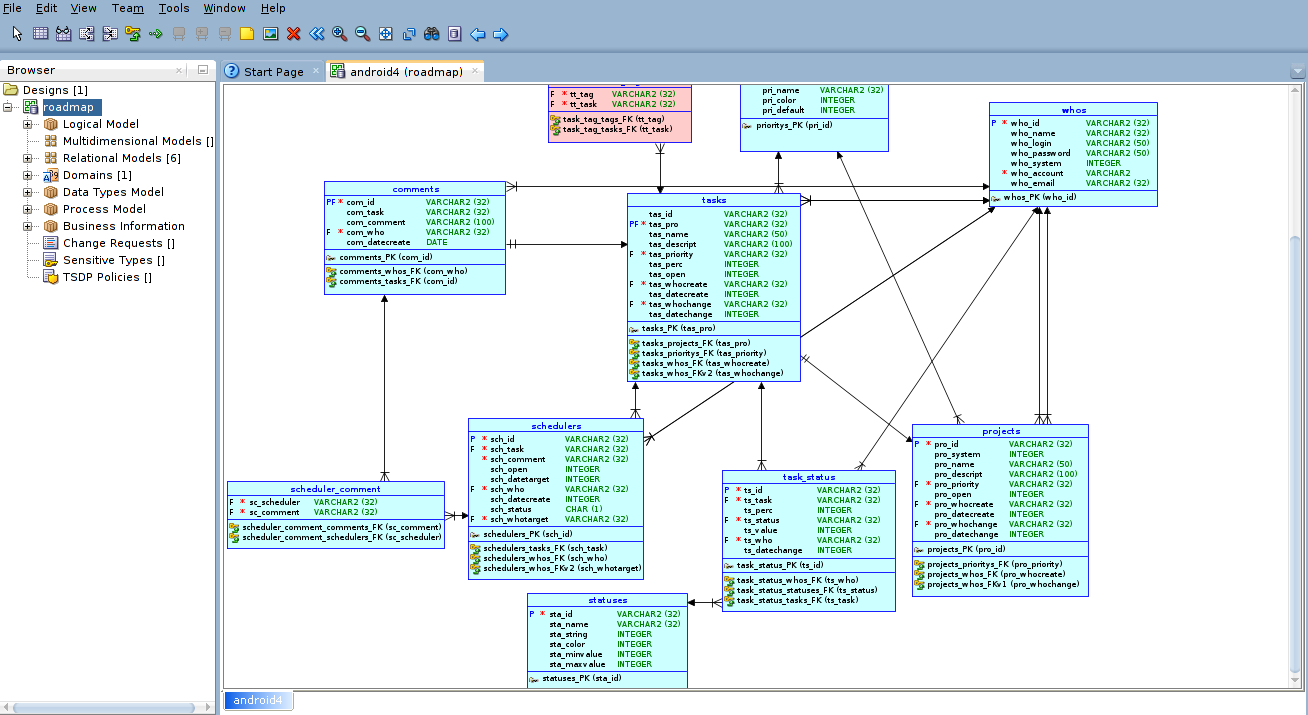
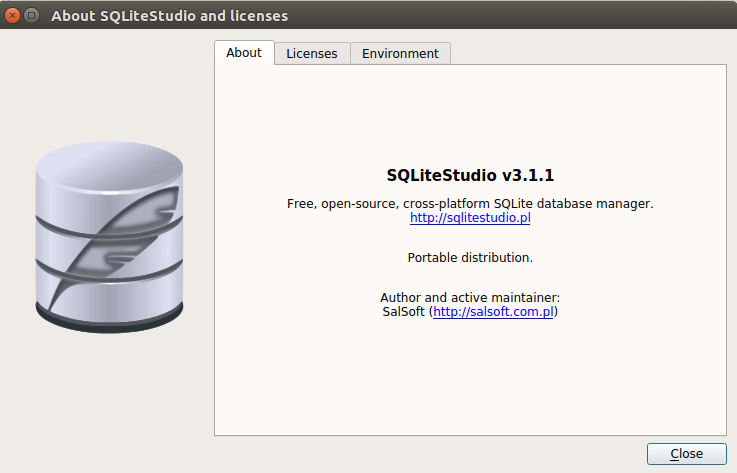
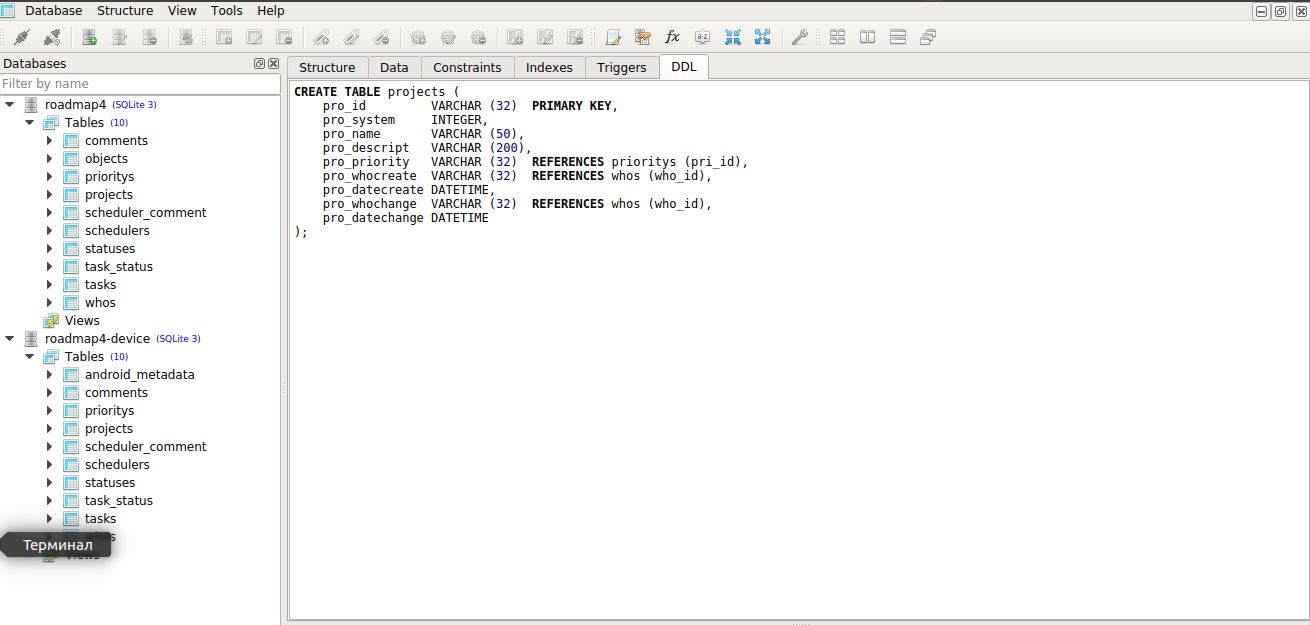
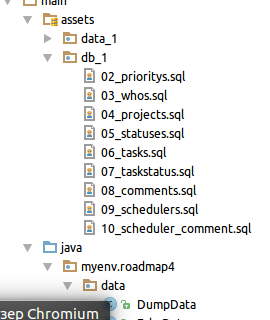
Как я создаю базу данных для своих приложений

О вечном вопросе: почему?

Если в нашем приложении больше 5 таблиц, то уже было бы не плохо использовать какой-нибудь инструмент для визуального проектирования архитектуры БД. Поскольку для меня это хобби, то и использую я абсолютно бесплатный инструмент под названием Oracle SQL Developer Data Modeler (скачать его можно [тут](http://www.oracle.com/technetwork/indexes/downloads/index.html#tools)).



Данная программа позволяет визуально рисовать таблицы, и строить взаимосвязи с ними. Многие ошибки проектирования архитектуры БД можно избежать при таком подходе проектирования (это я уже вам говорю как профессиональный программист БД). Выглядит это примерно так:  
  
  
  
Спроектировав саму архитектуру, приступаем к более нудной части, заключающийся в созданий sql кода для создания таблиц. Для помощи в этом вопросе, я уже использую инструмент под названием SQLiteStudio (его в свою очередь можно скачать тут [тут](https://sqlitestudio.pl/)).  
  
  
  
Данный инструмент является аналогом таких известных продуктов как SQL Naviagator, Toad etc. Но как следует из названия, заточен он под работу с SQLite. Он позволяет визуально создать БД и получить DDL код создаваемых таблиц. Кстати, он также позволяет создавать представления (View), которые вы тоже при желании можете использовать в своем приложении. Не знаю насколько правильный подход использования представлений в программах для Android, но в одном из своих приложений я использовал их.  
  
  
  
Собственно говоря я больше не каких сторонних инструментов не использую, и дальше начинается магия с Android Studio. Как я уже писал выше, если начать внедрять SQL код в Java код, то на выходе мы получим плохочитаемый, а значит и плохо расширяемый код. Поэтому я выношу все SQL инструкции во внешние файлы, которые у меня находятся в директории **assets**. В Android Studio выглядит это примерно так:



О директориях db и data

Внутри директории **assets** я создал две директории **db\_01** и **data\_01**. Цифры в названиях директорий соответствуют номеру версии моей БД с которой я работаю. В директории **db**у меня хранятся сами SQL скрипты создания таблиц. А в директории **data** хранятся данные необходимые для начального заполнения таблиц.

Теперь давайте посмотрим на код внутри моего *DBHelper* который я использую в своих проектах. Сначала переменные класса и конструктор (тут без каких либо неожиданностей):

private static final String TAG = "RoadMap4.DBHelper";

String mDb = "db\_";

String mData = "data\_";

Context mContext;

int mVersion;

public DBHelper(Context context, String name, int version) {

super(context, name, null, version);

mContext = context;

mVersion = version;

}

Теперь метод *onCreate* и тут становится уже интереснее:

@Override

public void onCreate(SQLiteDatabase db) {

ArrayList<String> tables = getSQLTables();

for (String table: tables){

db.execSQL(table);

}

ArrayList<HashMap<String, ContentValues>> dataSQL = getSQLDatas();

for (HashMap<String, ContentValues> hm: dataSQL){

for (String table: hm.keySet()){

Log.d(TAG, "insert into " + table + " " + hm.get(table));

long rowId = db.insert(table, null, hm.get(table));

}

}

}

Логически он разделен на два цикла, в первом цикле я получаю список SQL — инструкций для создания БД и затем выполняю их, во втором цикле я уже заполняю созданные ранее таблицы начальными данными. И так, шаг первый:

private ArrayList<String> getSQLTables() {

ArrayList<String> tables = new ArrayList<>();

ArrayList<String> files = new ArrayList<>();

AssetManager assetManager = mContext.getAssets();

String dir = mDb + mVersion;

try {

String[] listFiles = assetManager.list(dir);

for (String file: listFiles){

files.add(file);

}

Collections.sort(files, new QueryFilesComparator());

BufferedReader bufferedReader;

String query;

String line;

for (String file: files){

Log.d(TAG, "file db is " + file);

bufferedReader = new BufferedReader(new InputStreamReader(assetManager.open(dir + "/" + file)));

query = "";

while ((line = bufferedReader.readLine()) != null){

query = query + line;

}

bufferedReader.close();

tables.add(query);

}

} catch (IOException e) {

e.printStackTrace();

}

return tables;

}

Тут все достаточно просто, мы просто читаем содержимое файлов, и конкатенируем содержимое каждого файла в элемент массива. Обратите внимание, что я произвожу сортировку списка файлов, так как таблицы могут иметь внешние ключи, а значит таблицы должны создаваться в определенном порядке. Я использую нумерацию в название файлов, и с помощью нею и произвожу сортировку.

private class QueryFilesComparator implements Comparator<String>{

@Override

public int compare(String file1, String file2) {

Integer f2 = Integer.parseInt(file1.substring(0, 2));

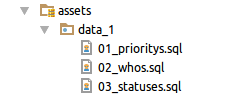
Integer f1 = Integer.parseInt(file2.substring(0, 2));

return f2.compareTo(f1);

}

}

С заполнением таблиц все веселей. Таблицы у меня заполняются не только жестко заданными значениями, но также значениями из ресурсов и UUID ключами (я надеюсь когда-нибудь прийти к сетевой версии своей программы, что бы мои пользователи могли работать с общими данными). Сама структура файлов с начальными данными выглядит так:



Несмотря на то, что файлы у меня имеют расширение sql, внутри не sql код а вот такая штука:  
  
prioritys  
pri\_id:UUID:UUID  
pri\_object:string:object\_task  
pri\_name:string:normal  
pri\_color:color:colorGreen  
pri\_default:int:1  
prioritys  
pri\_id:UUID:UUID  
pri\_object:string:object\_task  
pri\_name:string:hold  
pri\_color:color:colorBlue  
pri\_default:int:0  
prioritys  
pri\_id:UUID:UUID  
pri\_object:string:object\_task  
pri\_name:string:important  
pri\_color:color:colorRed  
pri\_default:int:0  
prioritys  
pri\_id:UUID:UUID  
pri\_object:string:object\_project  
pri\_name:string:normal  
pri\_color:color:colorGreen  
pri\_default:int:1  
prioritys  
pri\_id:UUID:UUID  
pri\_object:string:object\_project  
pri\_name:string:hold  
pri\_color:color:colorBlue  
pri\_default:int:0  
prioritys  
pri\_id:UUID:UUID  
pri\_object:string:object\_project  
pri\_name:string:important  
pri\_color:color:colorRed  
pri\_default:int:0  
  
Структура файла такая: я выполняю вызов функции *split(":")* применительно к строчке и если получаю что ее размер равен 1 то значит это название таблицы, куда надо записать данные. Иначе это сами данные. Первое поле это название поля в таблице. Второе поле тип, по которому я определяю что мне надо в это самое поле записать. Если это UUID — это значит мне надо сгенерировать уникальное значение UUID. Если string значит мне надо из ресурсов вытащить строковое значение. Если color, то опять-таки, из ресурсов надо вытащить код цвета. Если int или text, то я просто преобразую данное значение в int или String без каких либо телодвижений. Сам код выглядит вот так:

private ArrayList<HashMap<String, ContentValues>> getSQLDatas() {

ArrayList<HashMap<String, ContentValues>> data = new ArrayList<>();

ArrayList<String> files = new ArrayList<>();

AssetManager assetManager = mContext.getAssets();

String dir = mData + mVersion;

try {

String[] listFiles = assetManager.list(dir);

for (String file: listFiles){

files.add(file);

}

Collections.sort(files, new QueryFilesComparator());

BufferedReader bufferedReader;

String line;

int separator = 0;

ContentValues cv = null;

String[] fields;

String nameTable = null;

String packageName = mContext.getPackageName();

boolean flag = false;

HashMap<String, ContentValues> hm;

for (String file: files){

Log.d(TAG, "file db is " + file);

bufferedReader = new BufferedReader(new InputStreamReader(assetManager.open(dir + "/" + file)));

while ((line = bufferedReader.readLine()) != null){

fields = line.trim().split(":");

if (fields.length == 1){

if (flag == true){

hm = new HashMap<>();

hm.put(nameTable, cv);

data.add(hm);

}

*// наименование таблицы*

nameTable = line.trim();

cv = new ContentValues();

continue;

} else {

if (fields[1].equals("UUID")){

cv.put(fields[0], UUID.randomUUID().toString());

} else if (fields[1].equals("color") || fields[1].equals("string")){

int resId = mContext.getResources().getIdentifier(fields[2], fields[1], packageName);

Log.d(TAG, fields[1] + " " + resId);

switch (fields[1]){

case "color":

cv.put(fields[0], resId);

break;

case "string":

cv.put(fields[0], mContext.getString(resId));

break;

default:

break;

}

} else if (fields[1].equals("text")){

cv.put(fields[0], fields[2]);

} else if (fields[1].equals("int")){

cv.put(fields[0], Integer.parseInt(fields[2]));

}

}

flag = true;

}

bufferedReader.close();

}

} catch (IOException e) {

e.printStackTrace();

}

return data;

}

Внезапно

Уже вставив код в данную статью заметил две проблемы. Во первых если в конце файла не будет пустой строчке то я не добавлю ContentValues в свой массив. Во вторых, задумался о проблеме, если мне потребуется вставить данные в таблицу, в которой есть внешний ключ. С ходу нечего не придумал, буду думать на досуге как это лучше реализовать.

Ну и в качестве постскриптума: я повторюсь сказав что я любитель в программировании под Android, что пол-беды. Вторая беда, что в моем окружении нет программистов под Android и собственно говоря не с кем не посоветоваться не устроить мозговой штурм как лучше что-то сделать. Приходится идти методом научного тыка, по пути наступая на грабли. Иногда бывает больно, но в целом круто. Проект над которым я сейчас работаю, уже переживает 4 реинкарнацию. Поэтому просьба не стреляйте в пианиста, я играю как умею. Если напишите как сделать лучше, буду благодарен и рад.