Работа с готовой базой данных SQLite под Android средствами Java

В предыдущей статье посвящённой работе с SQLite в Android приложениях на Java была рассмотрена, если её можно так назвать, «классическая» схема. Когда создаётся пустая база данных, которая впоследствии заполняется в коде программы.

Это вполне оправдано в случае небольших баз данных. Однако работать с крупными объёмами информации или данными сложной структуры таким способом неудобно. В подобных ситуациях имеет смысл создать и заполнить базу данных отдельно от приложения, а затем включить уже готовую базу данных в его состав, несмотря на то, что подобный подход более сложен в отношении реализации.

Способ использования готовой базы данных приведённый в данной статье является универсальным и не требует каких-либо сторонних библиотек и т.д., так как реализован на «чистом» Android SDK.

**Важное замечание!**

Если вы ранее не работали с SQLite под Android средствами Java, настоятельно рекомендуется вначале ознакомиться со статьёй «[Работа с СУБД SQLite под Android на Java](http://streletzcoder.ru/rabota-s-subd-sqlite-pod-android-na-java/)», где помимо упомянутой «классической» схемы подробно описаны основы работы с данной СУБД.

**Создание базы данных**

Создать базу данных можно в любой программе, которая предназначена для администрирования с баз данных СУБД SQLite.

При создании базы данных необходимо учитывать один нюанс.

В Android в базе данных SQLite должна обязательно присутствовать специальная таблица *android\_metadata*. Эта таблица создаётся непосредственно Android и используется в служебных целях.

При работе с базой данных по «классической» схеме, данная таблица создаётся автоматически без участия разработчика. Если же речь идёт об использовании уже готовой базы данных, таблицу *android\_metadata* необходимо создать самостоятельно.

Таблица *android\_metadata* содержит одно единственное поле *locale*, которое имеет тип данных текст и значение по умолчанию «en\_US».

|  |  |
| --- | --- |
| 1 | CREATE TABLE android\_metadata ( locale  text DEFAULT en\_US ); |

После создания таблицы необходимо добавить в неё строку со значением по умолчанию.

|  |  |
| --- | --- |
| 1 | INSERT INTO android\_metadata VALUES ('en\_US'); |

Когда указанная запись будет добавлена, можно приступить к созданию и, если необходимо, заполнению других таблиц базы данных.

**Добавление базы данных в приложение**

После того как база данных будет полностью готова её необходимо включить в состав приложения.

Для этого требуется создать в приложении системную папку *assets* (если её ещё нет) и поместить в неё файл базы данных.

Больше никаких действий выполнять не требуется. При сборке база данных будет автоматически включена в приложение.

**Работа с базой данных в приложении**

Для того чтобы в самом приложении можно было работать с присоединённой базой данных требуется выполнить ряд операций:

1. Копирование файла базы данных из ресурсов приложения в системный каталог Android для баз данных;
2. Реализовать подключение к базе данных и необходимые операции по работе с ней.

Важно отметить, что работа с готовой базой данных отличается от классической схемы только необходимостью копирования и реализацией подключения к базе данных. В остальном всё идентично.

***1.Копирование файла базы данных и подключение к ней***

Для доступа к базе данных создадим класс-хэлпер.

Вначале включим в него только необходимые константы, поля, конструктор и заглушки методов *onCreate* и *OnUpgrade*.

Java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | public class DbHelper extends SQLiteOpenHelper {      private static String DB\_PATH = "/data/data/your\_package\_name/databases/";      private static String DB\_NAME = "test\_db.db";      private SQLiteDatabase dataBase;      private final Context fContext;      public DbHelper(Context context) {          super(context, DB\_NAME, null, 1);          this.fContext = context;      }      @Override      public void onCreate(SQLiteDatabase db) {      }      @Override      public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {      }  } |

Назначение констант:

* DB\_PATH – путь к системному каталогу баз данных.  
  Вместо «your\_package\_name» должно быть указано имя пакета приложения;
* DB\_NAME – имя файлы базы данных приложения.

**Примечание автора от 02.03.2019**

Приведённое значение DB\_PATH актуально только для устаревших версий Android. В частности, исходные примеры для данной статьи были написаны под Android 4.2.2.

В новых версиях Android настоятельно рекомендуется получать путь к файлу базы данных динамически, как показано ниже.

Java

|  |  |
| --- | --- |
| 1 | String db\_path = fContext.getDatabasePath(DB\_NAME); |

Несмотря на то, что методы *onCreate* и *OnUpgrade*. являются обязательными, при работе с готовой базой данных в их реализации нет необходимости. Потому в дальнейшем они так и останутся в виде заглушек.

Вся работа по копированию и подключению возлагается на пять методов:

* **createDataBase**  
  Проверка существования файла базы данных в системном каталоге и его копирование туда из ресурсов в случае отсутствия;
* **checkDataBase**  
  Собственно алгоритм проверки существования файла базы данных в системном каталоге;
* **copyDataBase**  
  Собственно алгоритм копирования файла базы данных в системный каталог;
* **openDataBase**Подключение к базе данных;
* **close**  
  Закрытие подключения к базе данных.

Ниже представлен полный код класса-хэлпера со всеми пятью методами:

Java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74 | import android.content.Context;  import android.database.SQLException;  import android.database.sqlite.SQLiteDatabase;  import android.database.sqlite.SQLiteException;  import android.database.sqlite.SQLiteOpenHelper;  import java.io.FileOutputStream;  import java.io.IOException;  import java.io.InputStream;  import java.io.OutputStream;  public class DbHelper extends SQLiteOpenHelper {      private static String DB\_PATH = "/data/data/com.example.dbreadytest/databases/";      private static String DB\_NAME = "test\_db.db";      private SQLiteDatabase dataBase;      private final Context fContext;      public DbHelper(Context context) {          super(context, DB\_NAME, null, 1);          this.fContext = context;      }      public void createDataBase() throws IOException {          boolean dbExist = checkDataBase();          if (dbExist) {              //ничего не делаем – файл базы данных уже есть          } else {              this.getReadableDatabase();              try {                  copyDataBase();              } catch (IOException e) {                  throw new Error("Error copying database");              }          }      }      private boolean checkDataBase() {          SQLiteDatabase checkDB = null;          try {              String myPath = DB\_PATH + DB\_NAME;              checkDB = SQLiteDatabase.openDatabase(myPath, null, SQLiteDatabase.OPEN\_READONLY);          } catch (SQLiteException e) {              //файл базы данных отсутствует          }          if (checkDB != null) {              checkDB.close();          }          return checkDB != null ? true : false;      }      private void copyDataBase() throws IOException {          InputStream input = fContext.getAssets().open(DB\_NAME);          String outFileName = DB\_PATH + DB\_NAME;          OutputStream output = new FileOutputStream(outFileName);          byte[] buffer = new byte[1024];          int length;          while ((length = input.read(buffer)) > 0) {              output.write(buffer, 0, length);          }          output.flush();          output.close();          input.close();      }      public void openDataBase() throws SQLException {          String path = DB\_PATH + DB\_NAME;          dataBase = SQLiteDatabase.openDatabase(path, null, SQLiteDatabase.OPEN\_READONLY);      }      @Override      public synchronized void close() {          if (dataBase != null)              dataBase.close();          super.close();      }      @Override      public void onCreate(SQLiteDatabase db) {      }      @Override      public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {      }  } |

***2.Инициализация базы данных в приложении***

Инициализация базы данных состоит в создании при запуске приложения экземпляра класса-хэлпера и последующим вызовом его методов *createDataBase* и *openDataBase* с обработкой соответствующих исключений.

Java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | DbHelper helper = new DbHelper(this);  try {      helper.createDataBase();  } catch (IOException ioe) {      throw new Error("Не возможно инициализировать базу данных");  }  try {      helper.openDataBase();  } catch (SQLException sqle) {      throw sqle;  } |

***3.Работа с инициализированной базой данных***

Работа с базой данных после её инициализации уже ничем не отличается от классической схемы.

В данной статье в качестве примера использована простая база данных с единственной таблицей, в которой перечислены виды животных.

Создадим перечисление с полями таблицы.

Java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | public enum Fields {      ID(0),      ANIMALNAME(1);      Fields(int i) {          this.fieldCode =i;      }      public int getFieldCode()      {          return fieldCode;      }      private int fieldCode;  } |

Создадим класс-репозиторий, который будет получать из этой таблицы данные.

Java

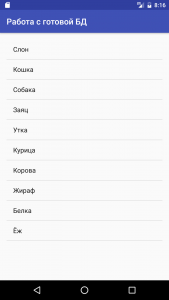
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | import android.content.Context;  import android.database.Cursor;  import android.database.sqlite.SQLiteDatabase;  import java.util.ArrayList;  public class DbRepository {      private SQLiteDatabase db;      private Context cont;      public DbRepository(Context context) {          //Подключение к базе данных          db = new DbHelper(context).getWritableDatabase();          cont=context;      }      public ArrayList<String> getData()      {          Fields field = Fields.ANIMALNAME;          ArrayList<String> list = new ArrayList<String>();          Cursor cursor = db.query("Animals", null, null, null, null, null, null);          if ((cursor != null) && (cursor.getCount() > 0)) {              cursor.moveToFirst();              do {                  list.add(cursor.getString(field.getFieldCode()));              } while (cursor.moveToNext());          }          return list;      }  } |

После этого в приложении получим данные из таблицы и выведем их в ListView.

Java

|  |  |
| --- | --- |
| 1  2  3  4 | final DbRepository repository = new DbRepository(this.getApplicationContext());  ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, android.R.layout.simple\_list\_item\_1, repository.getData());  adapter.setDropDownViewResource(android.R.layout.simple\_list\_item\_1);  dataList.setAdapter(adapter); |

В результате данные из таблицы будут отображены на экране устройства.

[](http://streletzcoder.ru/wp-content/uploads/2016/10/Dannyie-iz-tablitsyi.png)