[Хранение данных. SQLite](https://startandroid.ru/ru/uroki/vse-uroki-spiskom/74-urok-34-hranenie-dannyh-sqlite.html" \o "Урок 34. Хранение данных. SQLite)

В этом уроке:

- хранение данных с помощью SQLite

На прошлом уроке мы рассмотрели самый простой способ хранения данных - **Preferences**. Но способ этот достаточно ограничен и для хранения большого количества структурированных данных неудобен. На этом уроке рассмотрим [SQLite](http://www.sqlite.org/index.html). Это база данных с таблицами и запросами - все как в обычных БД.

Для начала, немного теории по взаимодействию приложения и БД.

В приложении, при подключении к БД мы указываем **имя**БД и **версию**. При этом могут возникнуть следующие ситуации:

1) БД **не существует**. Это может быть например в случае первичной установки программы. В этом случае приложение должно само **создать**БД и все таблицы в ней. И далее оно уже работает с только что созданной БД.

2) БД **существует**, но ее версия **устарела**. Это может быть в случае обновления программы. Например новой версии программы нужны дополнительные поля в старых таблицах или новые таблицы. В этом случае приложение должно **апдейтить**существующие таблицы и создать новые, если это необходимо.

3) БД **существует**и ее версия **актуальна**. В этом случае приложение успешно **подключается**к БД и работает.

Как вы понимаете, фраза "приложение должно" равнозначна фразе "разработчик должен", т.е. это наша задача. Для обработки описанных выше ситуаций нам надо создать **класс**, являющийся наследником для [SQLiteOpenHelper](http://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html). Назовем его DBHelper. Этот класс предоставит нам методы для **создания**или **обновления**БД в случаях ее **отсутствия**или **устаревания**.

**onCreate**- метод, который будет вызван, если БД, к которой мы хотим подключиться – не существует

**onUpgrade**- будет вызван в случае, если мы пытаемся подключиться к БД более новой версии, чем существующая

Давайте накидаем простое приложение – справочник контактов, которое будет хранить **имя**и **email**. **Вводить**данные будем на **экране**приложения, а для **отображения**информации используем **логи**. Обычно для этого используется List (список) – но мы эту тему пока не знаем. Да и не хочется перегружать приложение. Главное – освоить приемы работы с БД.

Создадим проект:

**Project name**: P0341\_SimpleSQLite  
**Build Target**: Android 2.3.3  
**Application name**: SimpleSQLite  
**Package name**: ru.startandroid.develop.p0341simplesqlite  
**Create Activity**: MainActivity

Нарисуем экран для ввода записей и очистки таблицы. Открываем**main.xml** и пишем:

|  |  |
| --- | --- |
| 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 | <?xml version="1.0" encoding="utf-8"?>  <LinearLayout      xmlns:android="http://schemas.android.com/apk/res/android"      android:layout\_width="fill\_parent"      android:layout\_height="fill\_parent"      android:orientation="vertical">      <LinearLayout          android:id="@+id/linearLayout1"          android:layout\_width="match\_parent"          android:layout\_height="wrap\_content">          <TextView              android:layout\_width="wrap\_content"              android:layout\_height="wrap\_content"              android:text="Name"              android:layout\_marginLeft="5dp"              android:layout\_marginRight="5dp">          </TextView>          <EditText              android:id="@+id/etName"              android:layout\_width="wrap\_content"              android:layout\_height="wrap\_content"              android:layout\_weight="1">              <requestFocus>              </requestFocus>          </EditText>      </LinearLayout>      <LinearLayout          android:id="@+id/linearLayout3"          android:layout\_width="match\_parent"          android:layout\_height="wrap\_content">          <TextView              android:id="@+id/textView2"              android:layout\_width="wrap\_content"              android:layout\_height="wrap\_content"              android:text="Email"              android:layout\_marginLeft="5dp"              android:layout\_marginRight="5dp">          </TextView>          <EditText              android:id="@+id/etEmail"              android:layout\_width="wrap\_content"              android:layout\_height="wrap\_content"              android:layout\_weight="1">          </EditText>      </LinearLayout>      <LinearLayout          android:id="@+id/linearLayout2"          android:layout\_width="match\_parent"          android:layout\_height="wrap\_content">          <Button              android:id="@+id/btnAdd"              android:layout\_width="wrap\_content"              android:layout\_height="wrap\_content"              android:text="Add">          </Button>          <Button              android:id="@+id/btnRead"              android:layout\_width="wrap\_content"              android:layout\_height="wrap\_content"              android:text="Read">          </Button>          <Button              android:id="@+id/btnClear"              android:layout\_width="wrap\_content"              android:layout\_height="wrap\_content"              android:text="Clear">          </Button>      </LinearLayout>  </LinearLayout> |

Пара полей для ввода и кнопки добавления записи, вывода существующих записей и очистки таблицы.

Открываем **MainActivity.java** и пишем:

|  |  |
| --- | --- |
| 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  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136 | package ru.startandroid.develop.p0341simplesqlite;    import android.app.Activity;  import android.content.ContentValues;  import android.content.Context;  import android.database.Cursor;  import android.database.sqlite.SQLiteDatabase;  import android.database.sqlite.SQLiteOpenHelper;  import android.os.Bundle;  import android.util.Log;  import android.view.View;  import android.view.View.OnClickListener;  import android.widget.Button;  import android.widget.EditText;    public class MainActivity extends Activity implements OnClickListener {      final String LOG\_TAG = "myLogs";      Button btnAdd, btnRead, btnClear;    EditText etName, etEmail;      DBHelper dbHelper;      /\*\* Called when the activity is first created. \*/    @Override    public void onCreate(Bundle savedInstanceState) {      super.onCreate(savedInstanceState);      setContentView(R.layout.main);        btnAdd = (Button) findViewById(R.id.btnAdd);      btnAdd.setOnClickListener(this);        btnRead = (Button) findViewById(R.id.btnRead);      btnRead.setOnClickListener(this);        btnClear = (Button) findViewById(R.id.btnClear);      btnClear.setOnClickListener(this);        etName = (EditText) findViewById(R.id.etName);      etEmail = (EditText) findViewById(R.id.etEmail);        // создаем объект для создания и управления версиями БД      dbHelper = new DBHelper(this);    }        @Override    public void onClick(View v) {        // создаем объект для данных      ContentValues cv = new ContentValues();        // получаем данные из полей ввода      String name = etName.getText().toString();      String email = etEmail.getText().toString();        // подключаемся к БД      SQLiteDatabase db = dbHelper.getWritableDatabase();          switch (v.getId()) {      case R.id.btnAdd:        Log.d(LOG\_TAG, "--- Insert in mytable: ---");        // подготовим данные для вставки в виде пар: наименование столбца - значение          cv.put("name", name);        cv.put("email", email);        // вставляем запись и получаем ее ID        long rowID = db.insert("mytable", null, cv);        Log.d(LOG\_TAG, "row inserted, ID = " + rowID);        break;      case R.id.btnRead:        Log.d(LOG\_TAG, "--- Rows in mytable: ---");        // делаем запрос всех данных из таблицы mytable, получаем Cursor        Cursor c = db.query("mytable", null, null, null, null, null, null);          // ставим позицию курсора на первую строку выборки        // если в выборке нет строк, вернется false        if (c.moveToFirst()) {            // определяем номера столбцов по имени в выборке          int idColIndex = c.getColumnIndex("id");          int nameColIndex = c.getColumnIndex("name");          int emailColIndex = c.getColumnIndex("email");            do {            // получаем значения по номерам столбцов и пишем все в лог            Log.d(LOG\_TAG,                "ID = " + c.getInt(idColIndex) +                ", name = " + c.getString(nameColIndex) +                ", email = " + c.getString(emailColIndex));            // переход на следующую строку            // а если следующей нет (текущая - последняя), то false - выходим из цикла          } while (c.moveToNext());        } else          Log.d(LOG\_TAG, "0 rows");        c.close();        break;      case R.id.btnClear:        Log.d(LOG\_TAG, "--- Clear mytable: ---");        // удаляем все записи        int clearCount = db.delete("mytable", null, null);        Log.d(LOG\_TAG, "deleted rows count = " + clearCount);        break;      }      // закрываем подключение к БД      dbHelper.close();    }          class DBHelper extends SQLiteOpenHelper {        public DBHelper(Context context) {        // конструктор суперкласса        super(context, "myDB", null, 1);      }        @Override      public void onCreate(SQLiteDatabase db) {        Log.d(LOG\_TAG, "--- onCreate database ---");        // создаем таблицу с полями        db.execSQL("create table mytable ("            + "id integer primary key autoincrement,"            + "name text,"            + "email text" + ");");      }        @Override      public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {        }    }    } |

Куча новых незнакомых слов в коде. Давайте разбираться.

В методе Activity - **onCreate**мы определяем объекты, присваиваем обработчики и создаем объект dbHelper класса **DBHelper**для управления БД. Сам класс будет описан ниже.

Далее смотрим метод Activity – **onClick**, в котором мы обрабатываем нажатия на кнопки.

Класс [ContentValues](http://developer.android.com/reference/android/content/ContentValues.html) используется для указания **полей**таблицы и **значений**, которые мы в эти поля будем вставлять. Мы создаем объект **cv**, и позже его используем. Далее мы записываем в переменные значения из **полей**ввода. Затем, с помощью метода [getWritableDatabase](http://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html#getWritableDatabase()) подключаемся к БД и получаем объект [SQLiteDatabase](http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html). Он позволит нам работать с БД. Мы будем использовать его методы **insert**– вставка записи, **query**– чтение, **delete**– удаление. У них много разных параметров на вход, но мы пока используем самый минимум.

Далее смотрим, какая кнопка была нажата:

**btnAdd**– добавление записи в таблицу *mytable*.  Мы заполняем объект **cv** парами: **имя** поля и **значение**. И (при вставке записи в таблицу) в указанные поля будут вставлены соответствующие значения. Мы заполняем поля *name*и *email*. *id* у нас заполнится автоматически (primary key autoincrement). Вызываем метод [insert](http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html#insert(java.lang.String,%20java.lang.String,%20android.content.ContentValues)) – передаем ему **имя таблицы**и объект **cv**с вставляемыми значениями. Второй аргумент метода используется, при вставке в таблицу пустой строки. Нам это сейчас не нужно, поэтому передаем null. Метод **insert**возвращает **ID**вставленной строки, мы его сохраняем в **rowID**и выводим в **лог**.

**btnRead**– чтение всех записей из таблицы *mytable*. Для чтения используется метод [query](http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html#query(java.lang.String,%20java.lang.String[],%20java.lang.String,%20java.lang.String[],%20java.lang.String,%20java.lang.String,%20java.lang.String)). На вход ему подается **имя таблицы**, список запрашиваемых полей, условия выборки, группировка, сортировка. Т.к. нам нужны все данные во всех полях без сортировок и группировок - мы используем везде **null**. Только имя таблицы указываем. Метод возвращает нам объект класса [Cursor](http://developer.android.com/reference/android/database/Cursor.html). Его можно рассматривать как таблицу с данными. Метод [moveToFirst](http://developer.android.com/reference/android/database/Cursor.html#moveToFirst()) – делает **первую**запись в Cursor **активной**и заодно проверяет, есть ли вообще записи в нем (т.е. выбралось ли что-либо в методе **query**). Далее мы получаем порядковые номера столбцов в Cursor по их именам с помощью метода [getColumnIndex](http://developer.android.com/reference/android/database/Cursor.html#getColumnIndex(java.lang.String)). Эти номера потом используем для чтения данных в методах [getInt](http://developer.android.com/reference/android/database/Cursor.html#getInt(int)) и [getString](http://developer.android.com/reference/android/database/Cursor.html#getString(int)) и выводим данные в лог. С помощью метода [moveToNext](http://developer.android.com/reference/android/database/Cursor.html#moveToNext()) мы перебираем все строки в Cursor пока не добираемся до последней. Если же записей не было, то выводим в лог соответствующее сообщение – *0 rows*. В конце закрываем курсор (освобождаем занимаемые им ресурсы) методом [close](http://developer.android.com/reference/android/database/Cursor.html#close()), т.к. далее мы его нигде не используем.

**btnClear**– очистка таблицы. Метод [delete](http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html#delete(java.lang.String,%20java.lang.String,%20java.lang.String[])) удаляет записи. На вход передаем **имя таблицы**и **null**в качестве условий для удаления, а значит удалится все. Метод возвращает **кол-во** **удаленных**записей.

После этого закрываем соединение с БД методом [close](http://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html#close()).

Класс **DBHelper**является **вложенным**в **MainActivity**и описан в конце кода. Как я уже писал выше, этот класс должен наследовать класс **SQLiteOpenHelper**.

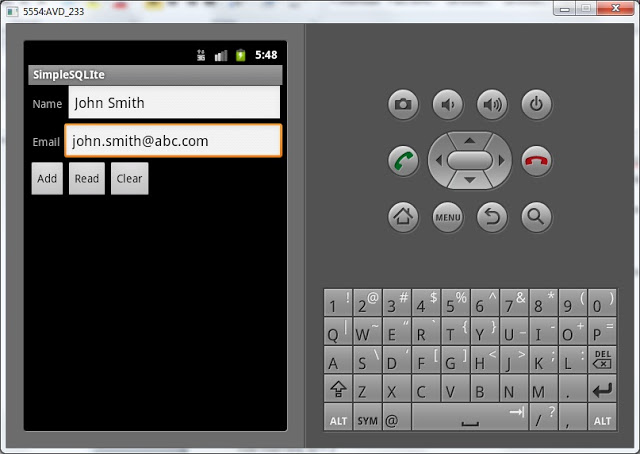
В **конструкторе**мы вызываем конструктор суперкласса и передаем ему:   
**context**- контекст  
*mydb*- название базы данных  
**null**– объект для работы с курсорами, нам пока не нужен, поэтому null  
*1* – версия базы данных

В методе **onCreate**этого класса мы используем метод execSQL объекта SQLiteDatabase для выполнения SQL-запроса, который создает таблицу. Напомню – этот метод вызывается, если БД не существует и ее надо создавать. По запросу видно, что мы создаем таблицу *mytable* с полями *id*, *name* и *email*.

Метод **onUpgrade**пока не заполняем, т.к. используем одну версию БД и менять ее не планируем.

Все сохраним и запустим приложение. Будем работать с БД и смотреть логи, которые покажут, какие методы выполняются, и что в них происходит.

Введем чего-нить в поля ввода и нажмем **Add**.



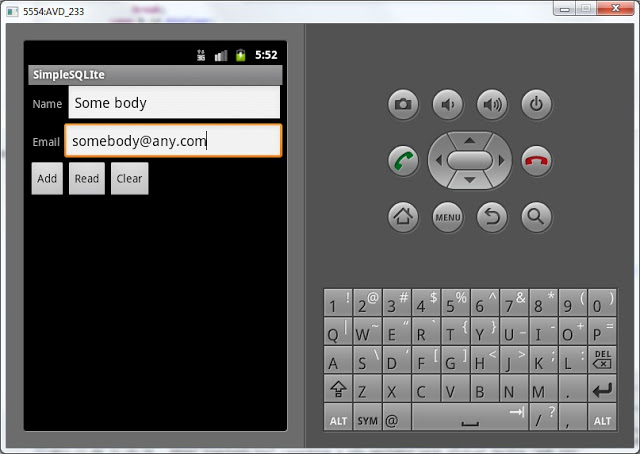
Смотрим лог:

*--- onCreate database ---  
--- Insert in mytable: ---  
row inserted, ID = 1*

Мы видим, что вызывался метод **onCreate**в классе **DBHelper**, а значит выполнялся скрипт по **созданию**таблицы. Это произошло потому, что это **первый**запуск приложения и БД еще **не была создана**. Теперь БД существует и с ней можно работать.

Далее видим, что вызывался метод вставки записи и вернул **ID**= 1.

Вставим еще какую-нибудь запись.



Смотрим лог:

*--- Insert in mytable: ---  
row inserted, ID = 2*

На этот раз **onCreate не вызывался**, т.к. БД уже **существует**. Вставилась запись с ID = 2.

Давайте посмотрим содержимое таблицы - нажмем кнопку **Read**и посмотрим лог:

*--- Rows in mytable: ---  
ID = 1, name = John Smith, email =*[*john.smith@abc.com*](mailto:john.smith@abc.com) *ID = 2, name = Some body, email =*[*somebody@any.com*](mailto:somebody@any.com)

Мы видим записи, которые вставляли. Тут все верно.

Теперь очистим таблицу - нажмем **Clear**. Смотрим лог:

*--- Clear mytable: ---  
deleted rows count = 2*

Удалено две записи, все верно. Если теперь посмотрим содержимое таблицы – кнопка Read:

*--- Rows in mytable: ---  
0 rows*

Записей нет.

В этой теме важно понять, что для работы с БД мы использовали два класса:

- **DBHelper**, наследующий **SQLiteOpenHelper**. В его **конструкторе**мы вызываем конструктор супер-класса и указываем имя и версию БД. Метод **getWritableDatabase**выполняет подключение к базе данных и возвращает нам объект **SQLiteDatabase**для работы с ней. Метод **close**закрывает подключение к БД. В случае, когда БД отсутствует или устарела, класс предоставляет нам самим реализовать создание или обновление в методах **onCreate**и **onUpgrate**.

- **SQLiteDatabase**. Содержит методы для работы с данными – т.е. **вставка**, **обновление**, **удаление**и **чтение**.

Файл базы можно найти в **File Explorer**, как и на прошлом уроке. Путь к нему *data/data/ru.startandroid.develop.p0341simpelsqlite/databases/myDB*.

[SQLite. Методы update и delete с указанием условия](https://startandroid.ru/ru/uroki/vse-uroki-spiskom/75-urok-35-metody-query-i-delete-s-ukazaniem-uslovija.html)

В этом уроке:

- используем методы update и delete с указанием условия

На прошлом уроке мы разобрали, как **вставить**запись, **считать**все записи из таблицы и **очистить**таблицу. Теперь посмотрим, как **обновить**и **удалить**конкретную запись.

Новый проект создавать не будем, используем **P0341\_SimpleSQLite** с прошлого урока. Немного поменяем экран, добавим **поле**для ввода **ID** и **кнопки**для **обновления**и **удаления**.

Перепишем **main.xml**:

|  |  |
| --- | --- |
| 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  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100 | <?xml version="1.0" encoding="utf-8"?>  <LinearLayout      xmlns:android="http://schemas.android.com/apk/res/android"      android:layout\_width="fill\_parent"      android:layout\_height="fill\_parent"      android:orientation="vertical">      <LinearLayout          android:id="@+id/linearLayout4"          android:layout\_width="match\_parent"          android:layout\_height="wrap\_content"          android:layout\_marginTop="5dp">          <TextView              android:layout\_width="wrap\_content"              android:layout\_height="wrap\_content"              android:text="ID"              android:layout\_marginLeft="5dp"              android:layout\_marginRight="25dp">          </TextView>          <EditText              android:id="@+id/etID"              android:layout\_width="70dp"              android:layout\_height="wrap\_content"              android:layout\_marginTop="2dp">          </EditText>          <Button              android:id="@+id/btnUpd"              android:layout\_width="wrap\_content"              android:layout\_height="wrap\_content"              android:text="Update">          </Button>          <Button              android:id="@+id/btnDel"              android:layout\_width="wrap\_content"              android:layout\_height="wrap\_content"              android:text="Delete">          </Button>      </LinearLayout>      <LinearLayout          android:id="@+id/linearLayout1"          android:layout\_width="match\_parent"          android:layout\_height="wrap\_content">          <TextView              android:layout\_width="wrap\_content"              android:layout\_height="wrap\_content"              android:text="Name"              android:layout\_marginLeft="5dp"              android:layout\_marginRight="5dp">          </TextView>          <EditText              android:id="@+id/etName"              android:layout\_width="wrap\_content"              android:layout\_height="wrap\_content"              android:layout\_weight="1">              <requestFocus>              </requestFocus>          </EditText>      </LinearLayout>      <LinearLayout          android:id="@+id/linearLayout3"          android:layout\_width="match\_parent"          android:layout\_height="wrap\_content">          <TextView              android:id="@+id/textView2"              android:layout\_width="wrap\_content"              android:layout\_height="wrap\_content"              android:text="Email"              android:layout\_marginLeft="5dp"              android:layout\_marginRight="5dp">          </TextView>          <EditText              android:id="@+id/etEmail"              android:layout\_width="wrap\_content"              android:layout\_height="wrap\_content"              android:layout\_weight="1">          </EditText>      </LinearLayout>      <LinearLayout          android:id="@+id/linearLayout2"          android:layout\_width="match\_parent"          android:layout\_height="wrap\_content">          <Button              android:id="@+id/btnAdd"              android:layout\_width="wrap\_content"              android:layout\_height="wrap\_content"              android:text="Add">          </Button>          <Button              android:id="@+id/btnRead"              android:layout\_width="wrap\_content"              android:layout\_height="wrap\_content"              android:text="Read">          </Button>          <Button              android:id="@+id/btnClear"              android:layout\_width="wrap\_content"              android:layout\_height="wrap\_content"              android:text="Clear">          </Button>      </LinearLayout>  </LinearLayout> |

По нажатию кнопки **Update**мы будем читать содержимое полей **Name**и **Email**, и обновлять запись в таблице, для которой **id** = значению из поля **ID**. По нажатию кнопки **Delete**будем удалять запись из таблицы по **id**= значению из поля **ID**. Экран получился, конечно, не самый лучший с точки зрения эргономики и юзабилити, но у нас тут не кружок юных дизайнеров, а серьезное изучение БД.

Подредактируем **MainActivity.java**.Добавим описание и определение новых экранных элементов, присвоение обработчиков для кнопок.

|  |  |
| --- | --- |
| 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 | final String LOG\_TAG = "myLogs";    Button btnAdd, btnRead, btnClear, btnUpd, btnDel;  EditText etName, etEmail, etID;    ...    public void onCreate(Bundle savedInstanceState) {      ...      btnClear = (Button) findViewById(R.id.btnClear);    btnClear.setOnClickListener(this);      btnUpd = (Button) findViewById(R.id.btnUpd);    btnUpd.setOnClickListener(this);      btnDel = (Button) findViewById(R.id.btnDel);    btnDel.setOnClickListener(this);      etName = (EditText) findViewById(R.id.etName);    etEmail = (EditText) findViewById(R.id.etEmail);    etID = (EditText) findViewById(R.id.etID);      // создаем объект для создания и управления версиями БД    dbHelper = new DBHelper(this);  } |

(Обращайте внимание только на выделенные строки)

Теперь дополним реализацию **onClick**:

|  |  |
| --- | --- |
| 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 | public void onClick(View v) {       // создаем объект для данных     ContentValues cv = new ContentValues();       // получаем данные из полей ввода     String name = etName.getText().toString();     String email = etEmail.getText().toString();     String id = etID.getText().toString();       // подключаемся к БД     SQLiteDatabase db = dbHelper.getWritableDatabase();       switch (v.getId()) {     case R.id.btnAdd:       ...     case R.id.btnRead:       ...     case R.id.btnClear:       ...     case R.id.btnUpd:       if (id.equalsIgnoreCase("")) {         break;       }       Log.d(LOG\_TAG, "--- Update mytable: ---");       // подготовим значения для обновления       cv.put("name", name);       cv.put("email", email);       // обновляем по id       int updCount = db.update("mytable", cv, "id = ?",           new String[] { id });       Log.d(LOG\_TAG, "updated rows count = " + updCount);       break;     case R.id.btnDel:       if (id.equalsIgnoreCase("")) {         break;       }       Log.d(LOG\_TAG, "--- Delete from mytable: ---");       // удаляем по id       int delCount = db.delete("mytable", "id = " + id, null);       Log.d(LOG\_TAG, "deleted rows count = " + delCount);       break;     }     // закрываем подключение к БД     dbHelper.close();   } |

(Добавляете только выделенный текст)

Мы добавляем переменную **id**, пишем в нее значение поля **etID**. В **switch**добавляем две новые ветки:

**btnUpd**– **обновление**записи в mytable. Проверяем, что значение **id**не пустое, заполняем **cv**данными для апдейта и **обновляем**запись. Для этого используется метод **update**. На вход ему подается **имя** таблицы, заполненный **ContentValues**с значениями для обновления, строка **условия**(Where) и массив **аргументов**для строки условия. В строке условия я использовал знак *?*. При запросе к БД вместо этого знака будет подставлено значение из массива **аргументов**, в нашем случае это – значение переменной **id**. Если знаков *?* в строке условия несколько, то им будут сопоставлены значения из массива по порядку. Метод **update**возвращает нам **кол-во обновленных** записей, которое мы выводим в лог.

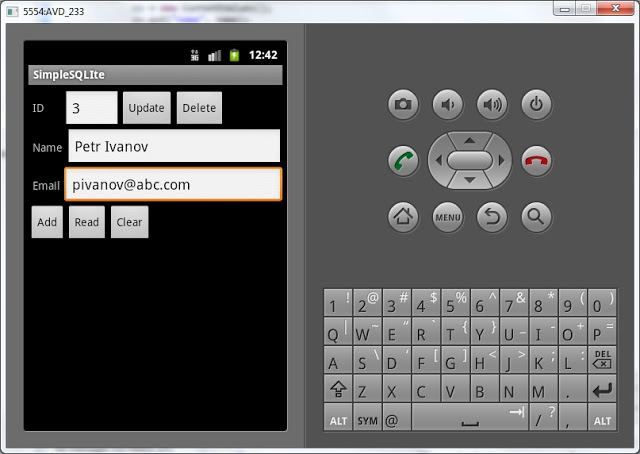
**btnDel**– **удаление**записи из mytable. Проверяем, что **id**не пустое и вызываем метод **delete**. На вход передаем **имя**таблицы, строку **условия**и массив **аргументов**для условия. Метод delete возвращает кол-во удаленных строк, которое мы выводим в лог.

Обратите внимание, что **условия**и для **update**и для **delete**у меня одинаковые, а именно ***id****= значение из поля****etID***. Но реализовал я их немного по-разному. Для **update**использовал символ *?* в строке условия и **массив**аргументов. А для **delete**вставил значение сразу в строку **условия**. Таким образом, я просто показал**способы формирования условия**. А вы уже используйте тот, что больше нравится или лучше в конкретной ситуации.

Все сохраним и запустим. На прошлом уроке мы добавляли пару записей, но потом их удалили. Так что я буду добавлять снова. **Добавлю**пару записей, нажму **Read**, вижу в логе:

*ID = 3, name = Ivan Petrov, email = ipetrov @abc.com  
ID = 4, name = Anton Sidorov, email = asidorov @def.com*

Теперь попробуем обновить запись с **ID**=3. Для этого вводим 3 в поле **ID** и новые данные в поля **Name**и **Email**:



Жмем **Update**, смотрим лог:

*-- Update mytable: ---  
updated rows count = 1*

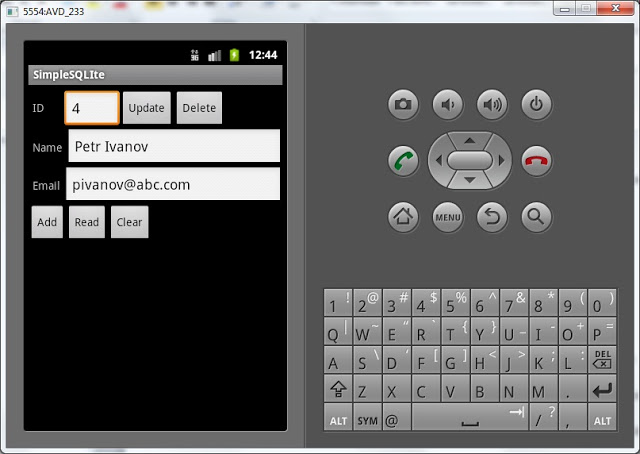
обновилась одна запись, все верно.

Нажмем **Read**и убедимся в этом. Лог:

*ID = 3, name = Petr Ivanov, email = pivanov @abc.com  
ID = 4, name = Anton Sidorov, email = asidorov @def.com*

Запись действительно обновилась.

Теперь давайте удалим запись с **ID**= 4. Вводим 4 в поле **ID**



Жмем **Delete**, смотрим лог:

*--- Delete from mytable: ---  
deleted rows count = 1*

одна запись удалена.

Жмем **Read**, смотрим лог:

*--- Rows in mytable: ---  
ID = 3, name = Petr Ivanov, email = pivanov @abc.com*

осталась одна запись.

Если попробовать удалить запись с **пустым**полем **ID**, то ничего не будет, т.к. мы реализовали проверку. Если же попробовать удалить запись с несуществующим **ID**, то метод **delete**вернет **0**. Т.е. ничего не было удалено, т.к. не нашлось записей для удаления.

Теперь вы умеете **читать**, **вставлять**, **удалять**и **изменять**записи в SQLite.

Полный код **MainActivity.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  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160 | package ru.startandroid.develop.p0341simplesqlite;    import ru.startandroid.develop.p0341simpelsqlite.R;  import android.app.Activity;  import android.content.ContentValues;  import android.content.Context;  import android.database.Cursor;  import android.database.sqlite.SQLiteDatabase;  import android.database.sqlite.SQLiteOpenHelper;  import android.os.Bundle;  import android.util.Log;  import android.view.View;  import android.view.View.OnClickListener;  import android.widget.Button;  import android.widget.EditText;    public class MainActivity extends Activity implements OnClickListener {      final String LOG\_TAG = "myLogs";      Button btnAdd, btnRead, btnClear, btnUpd, btnDel;    EditText etName, etEmail, etID;      DBHelper dbHelper;      /\*\* Called when the activity is first created. \*/    public void onCreate(Bundle savedInstanceState) {      super.onCreate(savedInstanceState);      setContentView(R.layout.main);        btnAdd = (Button) findViewById(R.id.btnAdd);      btnAdd.setOnClickListener(this);        btnRead = (Button) findViewById(R.id.btnRead);      btnRead.setOnClickListener(this);        btnClear = (Button) findViewById(R.id.btnClear);      btnClear.setOnClickListener(this);        btnUpd = (Button) findViewById(R.id.btnUpd);      btnUpd.setOnClickListener(this);        btnDel = (Button) findViewById(R.id.btnDel);      btnDel.setOnClickListener(this);        etName = (EditText) findViewById(R.id.etName);      etEmail = (EditText) findViewById(R.id.etEmail);      etID = (EditText) findViewById(R.id.etID);        // создаем объект для создания и управления версиями БД      dbHelper = new DBHelper(this);    }      public void onClick(View v) {        // создаем объект для данных      ContentValues cv = new ContentValues();        // получаем данные из полей ввода      String name = etName.getText().toString();      String email = etEmail.getText().toString();      String id = etID.getText().toString();        // подключаемся к БД      SQLiteDatabase db = dbHelper.getWritableDatabase();        switch (v.getId()) {      case R.id.btnAdd:        Log.d(LOG\_TAG, "--- Insert in mytable: ---");        // подготовим данные для вставки в виде пар: наименование столбца -        // значение        cv.put("name", name);        cv.put("email", email);        // вставляем запись и получаем ее ID        long rowID = db.insert("mytable", null, cv);        Log.d(LOG\_TAG, "row inserted, ID = " + rowID);        break;      case R.id.btnRead:        Log.d(LOG\_TAG, "--- Rows in mytable: ---");        // делаем запрос всех данных из таблицы mytable, получаем Cursor        Cursor c = db.query("mytable", null, null, null, null, null, null);          // ставим позицию курсора на первую строку выборки        // если в выборке нет строк, вернется false        if (c.moveToFirst()) {            // определяем номера столбцов по имени в выборке          int idColIndex = c.getColumnIndex("id");          int nameColIndex = c.getColumnIndex("name");          int emailColIndex = c.getColumnIndex("email");            do {            // получаем значения по номерам столбцов и пишем все в лог            Log.d(LOG\_TAG,                "ID = " + c.getInt(idColIndex) + ", name = "                    + c.getString(nameColIndex) + ", email = "                    + c.getString(emailColIndex));            // переход на следующую строку            // а если следующей нет (текущая - последняя), то false -            // выходим из цикла          } while (c.moveToNext());        } else          Log.d(LOG\_TAG, "0 rows");        c.close();        break;      case R.id.btnClear:        Log.d(LOG\_TAG, "--- Clear mytable: ---");        // удаляем все записи        int clearCount = db.delete("mytable", null, null);        Log.d(LOG\_TAG, "deleted rows count = " + clearCount);        break;      case R.id.btnUpd:        if (id.equalsIgnoreCase("")) {          break;        }        Log.d(LOG\_TAG, "--- Update mytable: ---");        // подготовим значения для обновления        cv.put("name", name);        cv.put("email", email);        // обновляем по id        int updCount = db.update("mytable", cv, "id = ?",            new String[] { id });        Log.d(LOG\_TAG, "updated rows count = " + updCount);        break;      case R.id.btnDel:        if (id.equalsIgnoreCase("")) {          break;        }        Log.d(LOG\_TAG, "--- Delete from mytable: ---");        // удаляем по id        int delCount = db.delete("mytable", "id = " + id, null);        Log.d(LOG\_TAG, "deleted rows count = " + delCount);        break;      }      // закрываем подключение к БД      dbHelper.close();    }      class DBHelper extends SQLiteOpenHelper {        public DBHelper(Context context) {        // конструктор суперкласса        super(context, "myDB", null, 1);      }        public void onCreate(SQLiteDatabase db) {        Log.d(LOG\_TAG, "--- onCreate database ---");        // создаем таблицу с полями        db.execSQL("create table mytable ("            + "id integer primary key autoincrement,"            + "name text,"            + "email text" + ");");      }        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {        }    }    } |

[SQLite. Подробнее про метод query. Условие, сортировка, группировка](https://startandroid.ru/ru/uroki/vse-uroki-spiskom/76-urok-36-sqlite-podrobnee-pro-metod-query-uslovie-sortirovka-gruppirovka.html)

В этом уроке:

- подробно разбираем метод чтения данных query  
- используем сортировку, группировку, условия, having

На прошлых уроках мы использовали метод **query** для чтения всех данных из таблицы. Мы использовали только имя таблицы в качестве входного параметра и получали все записи. Но у query есть и другие параметры:

**columns**– список полей, которые мы хотим получить  
**selection**– строка условия WHERE  
**selectionArgs**– массив аргументов для selection. В selection можно использовать знаки *?*, которые будут заменены этими значениями.  
**groupBy**- группировка   
**having**– использование условий для агрегатных функций   
**orderBy**- сортировка

Попробуем на примере их использовать. Создадим приложение – **справочник стран**. Возьмем **десять стран** и сохраним в БД их **наименование**, **количество**населения и **регион**.  Реализуем в приложении следующие функции:

- вывод всех записей  
- вывод значения агрегатной функции (SUM, MIN, MAX, COUNT)  
- вывод стран с населением, больше чем указано  
- группировка стран по региону  
- вывод регионов с населением больше, чем указано  
- сортировка стран по наименованию, населению или региону

Выводить все данные снова будем в лог.

Создадим проект:

**Project name**: P0361\_SQLiteQuery  
**Build Target**: Android 2.3.3  
**Application name**: SQLiteQuery  
**Package name**: ru.startandroid.develop.p0361sqlitequery  
**Create Activity**: MainActivity

Открываем layout-файл **main.xml** и пишем:

|  |  |
| --- | --- |
| 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  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124 | <?xml version="1.0" encoding="utf-8"?>  <LinearLayout      xmlns:android="http://schemas.android.com/apk/res/android"      android:layout\_width="fill\_parent"      android:layout\_height="fill\_parent"      android:orientation="vertical">      <TextView          android:layout\_width="fill\_parent"          android:layout\_height="wrap\_content"          android:text="Справочник стран"          android:textSize="14sp"          android:gravity="center\_horizontal"          android:layout\_marginBottom="5dp"          android:layout\_marginTop="5dp">      </TextView>      <Button          android:id="@+id/btnAll"          android:layout\_width="wrap\_content"          android:layout\_height="wrap\_content"          android:text="Все записи"          android:layout\_marginTop="5dp">      </Button>      <LinearLayout          android:id="@+id/linearLayout1"          android:layout\_width="match\_parent"          android:layout\_height="wrap\_content"          android:layout\_marginTop="5dp">          <Button              android:id="@+id/btnFunc"              android:layout\_width="wrap\_content"              android:layout\_height="wrap\_content"              android:text="Функция">          </Button>          <EditText              android:id="@+id/etFunc"              android:layout\_width="wrap\_content"              android:layout\_height="wrap\_content"              android:layout\_weight="1">              <requestFocus>              </requestFocus>          </EditText>      </LinearLayout>      <LinearLayout          android:id="@+id/linearLayout2"          android:layout\_width="match\_parent"          android:layout\_height="wrap\_content"          android:layout\_marginTop="5dp">          <Button              android:id="@+id/btnPeople"              android:layout\_width="wrap\_content"              android:layout\_height="wrap\_content"              android:text="Население">          </Button>          <EditText              android:id="@+id/etPeople"              android:layout\_width="wrap\_content"              android:layout\_height="wrap\_content"              android:layout\_weight="1"              android:inputType="number">          </EditText>      </LinearLayout>      <Button          android:id="@+id/btnGroup"          android:layout\_width="wrap\_content"          android:layout\_height="wrap\_content"          android:text="Население по региону"          android:layout\_marginTop="5dp">      </Button>      <LinearLayout          android:id="@+id/linearLayout4"          android:layout\_width="match\_parent"          android:layout\_height="wrap\_content"          android:layout\_marginTop="5dp">          <Button              android:id="@+id/btnHaving"              android:layout\_width="wrap\_content"              android:layout\_height="wrap\_content"              android:text="Население по региону >">          </Button>          <EditText              android:id="@+id/etRegionPeople"              android:layout\_width="wrap\_content"              android:layout\_height="wrap\_content"              android:layout\_weight="1"              android:inputType="number">          </EditText>      </LinearLayout>      <LinearLayout          android:id="@+id/linearLayout3"          android:layout\_width="match\_parent"          android:layout\_height="wrap\_content"          android:layout\_marginTop="5dp">          <Button              android:id="@+id/btnSort"              android:layout\_width="wrap\_content"              android:layout\_height="wrap\_content"              android:text="Сортировка">          </Button>          <RadioGroup              android:id="@+id/rgSort"              android:layout\_width="wrap\_content"              android:layout\_height="wrap\_content">              <RadioButton                  android:id="@+id/rName"                  android:layout\_width="wrap\_content"                  android:layout\_height="wrap\_content"                  android:checked="true"                  android:text="Наименование">              </RadioButton>              <RadioButton                  android:id="@+id/rPeople"                  android:layout\_width="wrap\_content"                  android:layout\_height="wrap\_content"                  android:text="Население">              </RadioButton>              <RadioButton                  android:id="@+id/rRegion"                  android:layout\_width="wrap\_content"                  android:layout\_height="wrap\_content"                  android:text="Регион">              </RadioButton>          </RadioGroup>      </LinearLayout>  </LinearLayout> |

6 кнопок – 6 функций, которые мы планируем реализовать. Поля для ввода значений, где это необходимо. Для **сортировки**используем **RadioGroup**.

Код для **MainActivity.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  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209 | package ru.startandroid.develop.p0361sqlitequery;    import android.app.Activity;  import android.content.ContentValues;  import android.content.Context;  import android.database.Cursor;  import android.database.sqlite.SQLiteDatabase;  import android.database.sqlite.SQLiteOpenHelper;  import android.os.Bundle;  import android.util.Log;  import android.view.View;  import android.view.View.OnClickListener;  import android.widget.Button;  import android.widget.EditText;  import android.widget.RadioGroup;    public class MainActivity extends Activity implements OnClickListener {      final String LOG\_TAG = "myLogs";      String name[] = { "Китай", "США", "Бразилия", "Россия", "Япония",        "Германия", "Египет", "Италия", "Франция", "Канада" };    int people[] = { 1400, 311, 195, 142, 128, 82, 80, 60, 66, 35 };    String region[] = { "Азия", "Америка", "Америка", "Европа", "Азия",        "Европа", "Африка", "Европа", "Европа", "Америка" };      Button btnAll, btnFunc, btnPeople, btnSort, btnGroup, btnHaving;    EditText etFunc, etPeople, etRegionPeople;    RadioGroup rgSort;      DBHelper dbHelper;    SQLiteDatabase db;      /\*\* Called when the activity is first created. \*/      public void onCreate(Bundle savedInstanceState) {      super.onCreate(savedInstanceState);      setContentView(R.layout.main);        btnAll = (Button) findViewById(R.id.btnAll);      btnAll.setOnClickListener(this);        btnFunc = (Button) findViewById(R.id.btnFunc);      btnFunc.setOnClickListener(this);        btnPeople = (Button) findViewById(R.id.btnPeople);      btnPeople.setOnClickListener(this);        btnSort = (Button) findViewById(R.id.btnSort);      btnSort.setOnClickListener(this);        btnGroup = (Button) findViewById(R.id.btnGroup);      btnGroup.setOnClickListener(this);        btnHaving = (Button) findViewById(R.id.btnHaving);      btnHaving.setOnClickListener(this);        etFunc = (EditText) findViewById(R.id.etFunc);      etPeople = (EditText) findViewById(R.id.etPeople);      etRegionPeople = (EditText) findViewById(R.id.etRegionPeople);        rgSort = (RadioGroup) findViewById(R.id.rgSort);        dbHelper = new DBHelper(this);      // подключаемся к базе      db = dbHelper.getWritableDatabase();        // проверка существования записей      Cursor c = db.query("mytable", null, null, null, null, null, null);      if (c.getCount() == 0) {        ContentValues cv = new ContentValues();        // заполним таблицу        for (int i = 0; i < 10; i++) {          cv.put("name", name[i]);          cv.put("people", people[i]);          cv.put("region", region[i]);          Log.d(LOG\_TAG, "id = " + db.insert("mytable", null, cv));        }      }      c.close();      dbHelper.close();      // эмулируем нажатие кнопки btnAll      onClick(btnAll);      }      public void onClick(View v) {        // подключаемся к базе      db = dbHelper.getWritableDatabase();        // данные с экрана      String sFunc = etFunc.getText().toString();      String sPeople = etPeople.getText().toString();      String sRegionPeople = etRegionPeople.getText().toString();        // переменные для query      String[] columns = null;      String selection = null;      String[] selectionArgs = null;      String groupBy = null;      String having = null;      String orderBy = null;        // курсор      Cursor c = null;        // определяем нажатую кнопку      switch (v.getId()) {      // Все записи      case R.id.btnAll:        Log.d(LOG\_TAG, "--- Все записи ---");        c = db.query("mytable", null, null, null, null, null, null);        break;      // Функция      case R.id.btnFunc:        Log.d(LOG\_TAG, "--- Функция " + sFunc + " ---");        columns = new String[] { sFunc };        c = db.query("mytable", columns, null, null, null, null, null);        break;      // Население больше, чем      case R.id.btnPeople:        Log.d(LOG\_TAG, "--- Население больше " + sPeople + " ---");        selection = "people > ?";        selectionArgs = new String[] { sPeople };        c = db.query("mytable", null, selection, selectionArgs, null, null,            null);        break;      // Население по региону      case R.id.btnGroup:        Log.d(LOG\_TAG, "--- Население по региону ---");        columns = new String[] { "region", "sum(people) as people" };        groupBy = "region";        c = db.query("mytable", columns, null, null, groupBy, null, null);        break;      // Население по региону больше чем      case R.id.btnHaving:        Log.d(LOG\_TAG, "--- Регионы с населением больше " + sRegionPeople            + " ---");        columns = new String[] { "region", "sum(people) as people" };        groupBy = "region";        having = "sum(people) > " + sRegionPeople;        c = db.query("mytable", columns, null, null, groupBy, having, null);        break;      // Сортировка      case R.id.btnSort:        // сортировка по        switch (rgSort.getCheckedRadioButtonId()) {        // наименование        case R.id.rName:          Log.d(LOG\_TAG, "--- Сортировка по наименованию ---");          orderBy = "name";          break;        // население        case R.id.rPeople:          Log.d(LOG\_TAG, "--- Сортировка по населению ---");          orderBy = "people";          break;        // регион        case R.id.rRegion:          Log.d(LOG\_TAG, "--- Сортировка по региону ---");          orderBy = "region";          break;        }        c = db.query("mytable", null, null, null, null, null, orderBy);        break;      }        if (c != null) {        if (c.moveToFirst()) {          String str;          do {            str = "";            for (String cn : c.getColumnNames()) {              str = str.concat(cn + " = "                  + c.getString(c.getColumnIndex(cn)) + "; ");            }            Log.d(LOG\_TAG, str);            } while (c.moveToNext());        }        c.close();      } else        Log.d(LOG\_TAG, "Cursor is null");        dbHelper.close();    }      class DBHelper extends SQLiteOpenHelper {        public DBHelper(Context context) {        // конструктор суперкласса        super(context, "myDB", null, 1);      }        public void onCreate(SQLiteDatabase db) {        Log.d(LOG\_TAG, "--- onCreate database ---");        // создаем таблицу с полями        db.execSQL("create table mytable ("            + "id integer primary key autoincrement," + "name text,"            + "people integer," + "region text" + ");");      }        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {        }    }    } |

Никаких проверок на вводимые с экрана значения я не делал, чтобы не перегружать код. Он и так немаленький получился, но ничего сложного в нем нет.

Три массива данных **name**, **people**, **region**. Это **наименования**стран, их **население**(в млн.) и **регионы**, к которым страны относятся. По этим данным мы будем заполнять таблицу.

В методе **onCreate**мы определяем и находим экранные элементы, присваиваем обработчики, создаем объект **dbHelper**для управления БД, подключаемся к базе и получаем объект **db**для работы с БД, проверяем наличие записей в таблице, если нет ничего – заполняем ее данными, закрываем соединение и эмулируем нажатие кнопки **Все записи** - чтобы сразу вывести весь список.

В методе **onClick**– подключаемся к базе, читаем данные с экранных полей в переменные, описываем переменные, которые будем использовать в методе **query**, и курсор, и смотрим, какая кнопка была нажата.

**btnAll**– вывод всех записей. Вызываем метод query с **именем**таблицы и **null**для остальных параметров. Это уже знакомо, делали на прошлом уроке.

**btnFunc**– вывод значения агрегатной функции (или любого поля). Использую параметр **columns**, в который надо записать **поля**, которые я хотел бы получить из таблицы, т.е. то, что обычно перечисляется после слова **SELECT**в SQL-запросе. **columns** имеет тип**String[]** – массив строк. Создаем массив из одного значения, которое считано с поля **etFunc**на экране. Запускаем query.

**btnPeople**– вывод стран с населением больше введенного на экране количества. Используем **selection** для формирования условия. При этом используем один аргумент -*?*. Значение аргумента задаем в **selectionArgs**– это **sPeople**– содержимое поля **etPeople**. Запускаем query.

**btnGroup**– группировка стран по регионам и вывод общее количество населения. Используем **columns**для указания столбцов, которые хотели бы получить – **регион** и **сумма**населения. В **groupBy** указываем, что **группировка**будет по **региону**. Запускаем query.

**btnHaving**– вывод регионов с населением больше указанного числа. Полностью аналогично случаю с группировкой, но добавляется условие в параметре **having**– сумма населения региона должна быть меньше **sRegionPeople**(значение **etRegionPeople**с экрана).

**btnSort**– сортировка стран. Определяем какой **RadioButton**включен и соответственно указываем в **orderBy**поле для сортировки данных. Запускаем query.

В выше описанных случаях мы запускали **query**и получали объект c класса **Cursor**. Далее мы проверяем, что он существует и в нем есть записи (**moveToFirst**). Если все ок, то мы запускаем перебор записей в цикле **do … while (c.moveToNext())**. Для каждой записи перебираем названия полей (**getColumnNames**), получаем по каждому полю его номер и извлекаем данные методом **getString**. Формируем список полей и значений в переменную **str**, которую потом выводим в лог. После всего этого закрываем соединение.

Ну и в конце кода идет описание вложенного класса **DBHelper**. Тут ничего не изменилось с прошлых уроков. Только при **создании таблицы**используются другие **поля**.

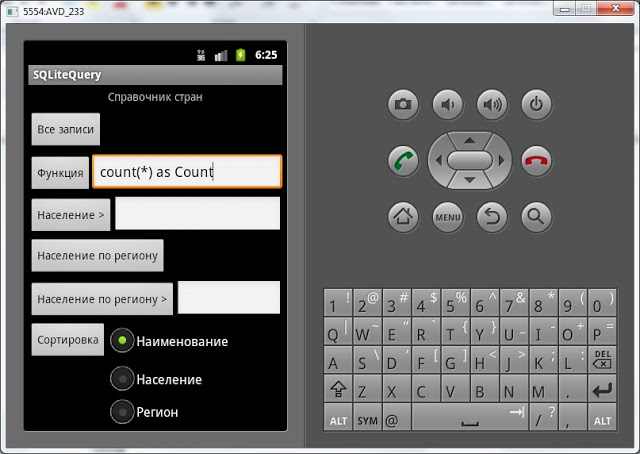
Сохраняем все и запускаем приложение.

В лог при запуске вывелись все записи, как если бы мы нажали кнопку «Все записи».

*--- Все записи ---  
id = 1; name = Китай; people = 1400; region = Азия;   
id = 2; name = США; people = 311; region = Америка;   
id = 3; name = Бразилия; people = 195; region = Америка;   
id = 4; name = Россия; people = 142; region = Европа;   
id = 5; name = Япония; people = 128; region = Азия;   
id = 6; name = Германия; people = 82; region = Европа;   
id = 7; name = Египет; people = 80; region = Африка;   
id = 8; name = Италия; people = 60; region = Европа;   
id = 9; name = Франция; people = 66; region = Европа;   
id = 10; name = Канада; people = 35; region = Америка;*

Т.е. таблица заполнена данными, можно работать.

Попробуем использовать агрегатную функцию. Например – получим**кол-во записей**. Вводим значение:

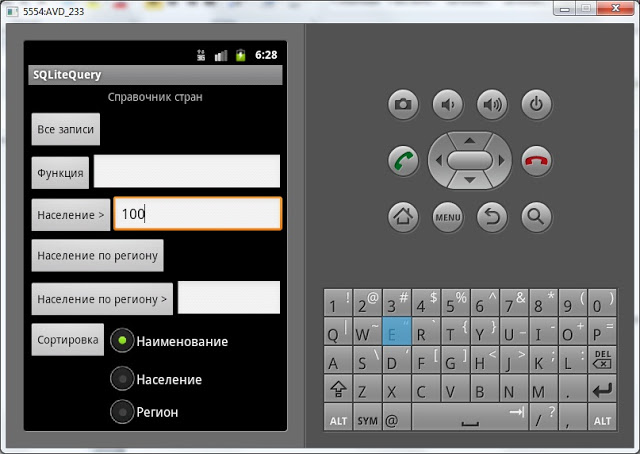


жмем кнопку **Функция**. Смотрим лог:

*--- Функция count(\*) as Count ---  
Count = 10;*

Все верно, 10 записей в таблице.

Покажем страны с населением больше 100 млн. Вводим **100** и жмем **Население >**



Лог:

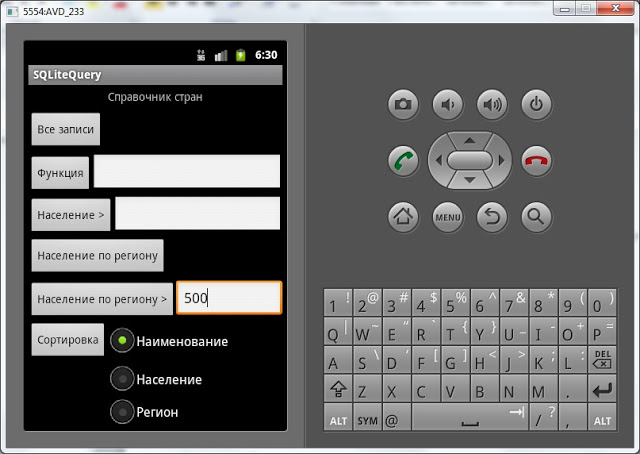
*--- Население больше 100 ---  
id = 1; name = Китай; people = 1400; region = Азия;   
id = 2; name = США; people = 311; region = Америка;   
id = 3; name = Бразилия; people = 195; region = Америка;   
id = 4; name = Россия; people = 142; region = Европа;   
id = 5; name = Япония; people = 128; region = Азия;*

Сгруппируем страны по региону и покажем население регионов. Нажмем кнопку **Население по региону**

Лог:

*--- Население по региону ---  
region = Азия; people = 1528;   
region = Америка; people = 541;   
region = Африка; people = 80;   
region = Европа; people = 350;*

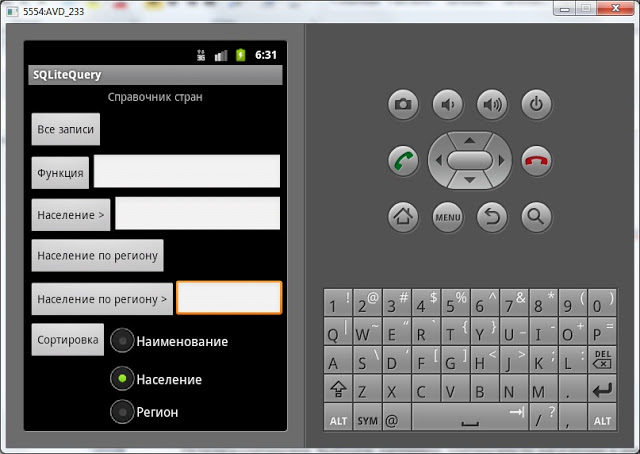
Теперь отобразим только те регионы, в которых население выше 500 млн.чел.  Вводим **500**и жмем **Население по региону >**



Лог:

*--- Регионы с населением больше 500 ---  
region = Азия; people = 1528;   
region = Америка; people = 541;*

Осталась сортировка. Выберем, например, сортировку по **населению**и жмем кнопку **Сортировка**



Лог:

*--- Сортировка по населению ---  
id = 10; name = Канада; people = 35; region = Америка;   
id = 8; name = Италия; people = 60; region = Европа;   
id = 9; name = Франция; people = 66; region = Европа;   
id = 7; name = Египет; people = 80; region = Африка;   
id = 6; name = Германия; people = 82; region = Европа;   
id = 5; name = Япония; people = 128; region = Азия;   
id = 4; name = Россия; people = 142; region = Европа;   
id = 3; name = Бразилия; people = 195; region = Америка;   
id = 2; name = США; people = 311; region = Америка;   
id = 1; name = Китай; people = 1400; region = Азия;*

По умолчанию **сортировка** идет по **возрастанию**.

Все работает так, как и должно. На этих примерах мы использовали все основные **параметры**метода **query**. Кроме описанных параметров, у метода query есть также реализации с использованием параметров **limit**и **distinct**. Я не стал их здесь отдельно показывать. Расскажу на словах:

**limit**– **строковый**параметр, указывается в формате *[offset], rows*. Т.е. если в **query**в качестве **limit**передать строку "*5*" - то запрос выдаст только **пять**первых записей. Если же передать "*3,5*", то запрос выдаст **пять**записей, начиная с **четвертой**(НЕ с третьей).

**distinct**– это **boolean**-параметр, удаление дубликатов. Может быть **true**или **false**.

Надеюсь, что метод query, который сначала казался большим скоплением параметров, стал понятен и прост.

[Запросы из связанных таблиц. INNER JOIN в SQLite. Метод rawQuery.](https://startandroid.ru/ru/uroki/vse-uroki-spiskom/77-urok-37-zaprosy-iz-svjazannyh-tablits-inner-join-v-sqlite-metod-rawquery.html)

28 ноября 2011

В этом уроке:

- читаем данные из связанных таблиц  
- используем rawQuery

Мы достаточно подробно разобрали метод **query**. Но не рассмотрели, как с его помощью выполнять запросы для **связанных таблиц**. Создадим простое приложение, которое будет делать запрос из двух таблиц и выводить результат в лог. Таблицы будут **people**и **position**. В первую (people) запишем список **людей**, во вторую (position) – список **должностей**. И для каждого **человека**в **people**будет прописан **id должности**из **position**.

Создадим проект:

**Project name:** P0371\_SQLiteInnerJoin  
**Build Target:** Android 2.3.3  
**Application name:** SQLiteInnerJoin  
**Package name:** ru.startandroid.develop.p0371sqliteinnerjoin  
**Create Activity:** MainActivity

Экран вообще использовать не будем, поэтому **main.xml** даже не трогаем. Открываем **MainActivity.java** и пишем код:

|  |  |
| --- | --- |
| 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  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145 | package ru.startandroid.develop.p0371sqliteinnerjoin;    import android.app.Activity;  import android.content.ContentValues;  import android.content.Context;  import android.database.Cursor;  import android.database.sqlite.SQLiteDatabase;  import android.database.sqlite.SQLiteOpenHelper;  import android.os.Bundle;  import android.util.Log;    public class MainActivity extends Activity {      final String LOG\_TAG = "myLogs";      // данные для таблицы должностей    int[] position\_id = { 1, 2, 3, 4 };    String[] position\_name = { "Директор", "Программер", "Бухгалтер", "Охранник" };    int[] position\_salary = { 15000, 13000, 10000, 8000 };      // данные для таблицы людей    String[] people\_name = { "Иван", "Марья", "Петр", "Антон", "Даша", "Борис", "Костя", "Игорь" };    int[] people\_posid = { 2, 3, 2, 2, 3, 1, 2, 4 };      /\*\* Called when the activity is first created. \*/    public void onCreate(Bundle savedInstanceState) {      super.onCreate(savedInstanceState);      setContentView(R.layout.main);        // Подключаемся к БД      DBHelper dbh = new DBHelper(this);      SQLiteDatabase db = dbh.getWritableDatabase();        // Описание курсора      Cursor c;        // выводим в лог данные по должностям      Log.d(LOG\_TAG, "--- Table position ---");      c = db.query("position", null, null, null, null, null, null);      logCursor(c);      c.close();      Log.d(LOG\_TAG, "--- ---");        // выводим в лог данные по людям      Log.d(LOG\_TAG, "--- Table people ---");      c = db.query("people", null, null, null, null, null, null);      logCursor(c);      c.close();      Log.d(LOG\_TAG, "--- ---");        // выводим результат объединения      // используем rawQuery      Log.d(LOG\_TAG, "--- INNER JOIN with rawQuery---");      String sqlQuery = "select PL.name as Name, PS.name as Position, salary as Salary "          + "from people as PL "          + "inner join position as PS "          + "on PL.posid = PS.id "          + "where salary > ?";      c = db.rawQuery(sqlQuery, new String[] {"12000"});      logCursor(c);      c.close();      Log.d(LOG\_TAG, "--- ---");        // выводим результат объединения      // используем query      Log.d(LOG\_TAG, "--- INNER JOIN with query---");      String table = "people as PL inner join position as PS on PL.posid = PS.id";      String columns[] = { "PL.name as Name", "PS.name as Position", "salary as Salary" };      String selection = "salary < ?";      String[] selectionArgs = {"12000"};      c = db.query(table, columns, selection, selectionArgs, null, null, null);      logCursor(c);      c.close();      Log.d(LOG\_TAG, "--- ---");        // закрываем БД      dbh.close();    }      // вывод в лог данных из курсора    void logCursor(Cursor c) {      if (c != null) {        if (c.moveToFirst()) {          String str;          do {            str = "";            for (String cn : c.getColumnNames()) {              str = str.concat(cn + " = " + c.getString(c.getColumnIndex(cn)) + "; ");            }            Log.d(LOG\_TAG, str);          } while (c.moveToNext());        }      } else        Log.d(LOG\_TAG, "Cursor is null");    }      // класс для работы с БД    class DBHelper extends SQLiteOpenHelper {        public DBHelper(Context context) {        super(context, "myDB", null, 1);      }        public void onCreate(SQLiteDatabase db) {        Log.d(LOG\_TAG, "--- onCreate database ---");          ContentValues cv = new ContentValues();          // создаем таблицу должностей        db.execSQL("create table position ("            + "id integer primary key,"            + "name text," + "salary integer"            + ");");          // заполняем ее        for (int i = 0; i < position\_id.length; i++) {          cv.clear();          cv.put("id", position\_id[i]);          cv.put("name", position\_name[i]);          cv.put("salary", position\_salary[i]);          db.insert("position", null, cv);        }          // создаем таблицу людей        db.execSQL("create table people ("            + "id integer primary key autoincrement,"            + "name text,"            + "posid integer"            + ");");          // заполняем ее        for (int i = 0; i < people\_name.length; i++) {          cv.clear();          cv.put("name", people\_name[i]);          cv.put("posid", people\_posid[i]);          db.insert("people", null, cv);        }      }        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {        }    }    } |

Разбираем код. Сначала идут несколько **массивов**с **данными**для таблиц. Обратите внимание, для **должностей**мы будем указывать **id**при заполнении таблиц. Это сделано для того, чтобы мы знали эти номера и могли их использовать в таблице **людей**для указания id должности.

В методе Activity **onCreate**мы создаем объект для **управления**БД и **подключаемся**к БД. Далее используя знакомый нам **query**выводим в лог данные из таблиц **position**и **people**.

Для вывода объединения таблиц используем [rawQuery](http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html#rawQuery(java.lang.String,%20java.lang.String[])). Это несложный метод, который принимает на вход **SQL-запрос** и список **аргументов**для условия WHERE (если необходимо). Мы сформировали запрос на объединение двух таблиц и вывода **имени**, **должности**и **зарплаты**человека. Условие выборки: **ЗП**должна быть **больше 12000**. Мы использовали аргументы для формирования условия.

Далее снова выводим объединение таблиц, но используем обычный **query**. В **table**записываем все таблицы, их алиасы и условие **JOIN**. В **columns**– все нужные поля с использованием алиасов. Ну и в **selection** и **selectionArgs**пишем условие выборки – **ЗП меньше 12000**.

Наш метод **logCursor** получает на вход **Cursor**и выводит в лог все **содержимое**. Тут все знакомо с прошлых уроков.

В классе **DBHelper**кодим **создание** таблиц и, на этот раз, здесь же их **наполнение**данными.

Все сохраним и запустим. Смотрим лог:

*--- onCreate database ---  
--- Table position ---  
id = 1; name = Директор; salary = 15000;  
id = 2; name = Программер; salary = 13000;  
id = 3; name = Бухгалтер; salary = 10000;  
id = 4; name = Охранник; salary = 8000;  
--- ---  
--- Table people ---  
id = 1; name = Иван; posid = 2;  
id = 2; name = Марья; posid = 3;  
id = 3; name = Петр; posid = 2;  
id = 4; name = Антон; posid = 2;  
id = 5; name = Даша; posid = 3;  
id = 6; name = Борис; posid = 1;  
id = 7; name = Костя; posid = 2;  
id = 8; name = Игорь; posid = 4;  
--- ---*

Вывели данные по таблицам раздельно.

*--- INNER JOIN with rawQuery---  
Name = Иван; Position = Программер; Salary = 13000;  
Name = Петр; Position = Программер; Salary = 13000;  
Name = Антон; Position = Программер; Salary = 13000;  
Name = Борис; Position = Директор; Salary = 15000;  
Name = Костя; Position = Программер; Salary = 13000;  
--- ---*

Вывели данные из rawQuery. ЗП > 12000

*--- INNER JOIN with query---  
Name = Марья; Position = Бухгалтер; Salary = 10000;  
Name = Даша; Position = Бухгалтер; Salary = 10000;  
Name = Игорь; Position = Охранник; Salary = 8000;  
--- ---*

Вывели данные из query. ЗП < 12000

Как видите, запросы из связанных таблиц в **SQLite**– не проблема и ничем не отличается от обычных БД.

Что использовать **rawQuery**или **query**– решать вам и зависит от ситуации. Хотя, навскидку я не могу придумать преимущества одного над другим в той или иной ситуации. Но наверно они есть.

Данные по ЗП и должностям являются выдуманными, любое совпадение – случайно. И, разумеется, ничего против бухгалтеров и охранников я не имею ))

# [Транзакции в SQLite. Небольшой FAQ по SQLite.](https://startandroid.ru/ru/uroki/vse-uroki-spiskom/78-urok-38-tranzaktsii-v-sqlite.html)

В этом уроке:

- используем транзакции при работе с БД

Что такое БД-**транзакция**, думаю объяснять особо не надо. Она используется при работе с данными по принципу «**все или ничего**». Т.е., например, вам нужно вставить пачку данных. Но вставить надо так, чтобы или все вставилось или ничего не вставилось.  И если в процессе половина записей прошла, а другая нет – должна быть возможность **откатить**изменения.

Напишем простое приложение и исследуем возможности **SQLite**в этом плане.

Создадим проект:

**Project name**: P0381\_SQLiteTransaction  
**Build Target**: Android 2.3.3  
**Application name**: SQLiteTransaction  
**Package name**: ru.startandroid.develop.p0381sqlitetransaction  
**Create Activity**: MainActivity

Открываем**MainActivity.java** и пишем:

|  |  |
| --- | --- |
| 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  75  76  77  78  79  80  81  82  83 | package ru.startandroid.develop.p0381sqlitetransaction;    import android.app.Activity;  import android.content.ContentValues;  import android.content.Context;  import android.database.Cursor;  import android.database.sqlite.SQLiteDatabase;  import android.database.sqlite.SQLiteOpenHelper;  import android.os.Bundle;  import android.util.Log;    public class MainActivity extends Activity {      final String LOG\_TAG = "myLogs";      DBHelper dbh;    SQLiteDatabase db;      /\*\* Called when the activity is first created. \*/    public void onCreate(Bundle savedInstanceState) {      super.onCreate(savedInstanceState);      setContentView(R.layout.main);      Log.d(LOG\_TAG, "--- onCreate Activity ---");      dbh = new DBHelper(this);      myActions();    }      void myActions() {      db = dbh.getWritableDatabase();      delete(db, "mytable");      insert(db, "mytable", "val1");      read(db, "mytable");      dbh.close();    }      void insert(SQLiteDatabase db, String table, String value) {      Log.d(LOG\_TAG, "Insert in table " + table + " value = " + value);      ContentValues cv = new ContentValues();      cv.put("val", value);      db.insert(table, null, cv);    }      void read(SQLiteDatabase db, String table) {      Log.d(LOG\_TAG, "Read table " + table);      Cursor c = db.query(table, null, null, null, null, null, null);      if (c != null) {        Log.d(LOG\_TAG, "Records count = " + c.getCount());        if (c.moveToFirst()) {          do {            Log.d(LOG\_TAG, c.getString(c.getColumnIndex("val")));          } while (c.moveToNext());        }        c.close();      }    }      void delete(SQLiteDatabase db, String table) {      Log.d(LOG\_TAG, "Delete all from table " + table);      db.delete(table, null, null);    }      // класс для работы с БД    class DBHelper extends SQLiteOpenHelper {        public DBHelper(Context context) {        super(context, "myDB", null, 1);      }        public void onCreate(SQLiteDatabase db) {        Log.d(LOG\_TAG, "--- onCreate database ---");          db.execSQL("create table mytable ("            + "id integer primary key autoincrement,"            + "val text"            + ");");      }        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {        }    }    } |

Разбираем код. Я создал несколько методов, где сгруппировал операции: **insert**– набор операций для вставки записи, **read**– чтение всех записей, **delete**– удаление всех записей. Класс **DBHelper**– для управления БД. Интересовать нас будет метод **myActions**. Сейчас в нем мы подключаемся к БД, очищаем таблицу mytable, вставляем строку с значением val1, выводим в лог все записи из таблицы и отключаемся.

Все сохраним, запустим приложение. Смотрим лог:

--- onCreate Activity ------ onCreate database ---Delete all from table mytableInsert in table mytable value = val1Read table mytableRecords count = 1val1

Все верно, запись вставилась и отобразилась.

Теперь попробуем использовать транзакцию. Поправим код **myActions** на этот:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | void myActions() {    db = dbh.getWritableDatabase();    delete(db, "mytable");    db.beginTransaction();    insert(db, "mytable", "val1");    db.endTransaction();    insert(db, "mytable", "val2");    read(db, "mytable");    dbh.close();  } |

Мы подключаемся к базе, чистим таблицу, открываем транзакцию методом [beginTransaction](http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html#beginTransaction()), вставляем val1, закрываем транзакцию методом [endTransaction](http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html#endTransaction()), вставляем val2, выводим содержимое в лог и отключаемся. Все сохраняем, запускаем и смотрим лог:

--- onCreate Activity ---Delete all from table mytableInsert in table mytable value = val1Insert in table mytable value = val2Read table mytableRecords count = 1val2

По логу видно, что вставляли мы **две**записи, но прошла только **вторая**. Та, которая была в **транзакции**– не записалась. Это произошло потому, что мы явно не указали, что транзакция должна быть успешно **закрыта**. Если этого не сделать, то при закрытии транзакции все операции **отменятся**. Давайте исправимся. Снова перепишем **myActions**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | void myActions() {    db = dbh.getWritableDatabase();    delete(db, "mytable");    db.beginTransaction();    insert(db, "mytable", "val1");    db.setTransactionSuccessful();    insert(db, "mytable", "val2");    db.endTransaction();    insert(db, "mytable", "val3");    read(db, "mytable");    dbh.close();  } |

Подключаемся к БД, чистим таблицу, открываем транзакцию, вставляем val1, подтверждаем успешность транзакции методом [setTransactionSuccessful](http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html#setTransactionSuccessful()), вставляем val2, закрываем транзакцию, вставляем val3, выводим содержимое и отключаемся.

Сохраняем, запускаем, смотрим лог:

--- onCreate Activity ---Delete all from table mytableInsert in table mytable value = val1Insert in table mytable value = val2Insert in table mytable value = val3Read table mytableRecords count = 3val1val2val3

Вставились все три записи. Обратите внимание - несмотря на то, что val2 мы вставляли уже после подтверждения успешности транзакции, запись вставилась, вошла в эту транзакцию. Но проводить операции после подтверждения транзакции не рекомендуется хелпом.

**Транзакция**при открытии ставит **блокировку**на базу. Убедимся в этом, попробуем создать новое подключение к БД во время транзакции. Перепишем **myActions**:

|  |  |
| --- | --- |
| 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 | void myActions() {    try {      db = dbh.getWritableDatabase();      delete(db, "mytable");        db.beginTransaction();      insert(db, "mytable", "val1");        Log.d(LOG\_TAG, "create DBHelper");      DBHelper dbh2 = new DBHelper(this);      Log.d(LOG\_TAG, "get db");      SQLiteDatabase db2 = dbh2.getWritableDatabase();      read(db2, "mytable");      dbh2.close();        db.setTransactionSuccessful();      db.endTransaction();        read(db, "mytable");      dbh.close();      } catch (Exception ex) {      Log.d(LOG\_TAG, ex.getClass() + " error: " + ex.getMessage());    }  } |

Подключаемся к базе, чистим таблицу, открываем транзакцию, вставляем запись, создаем новое подключение к БД - **db2**, читаем содержимое вторым подключением, закрываем второе подключение, успешно закрываем транзакцию, читаем содержимое первым подключением, закрываем первое подключение.

Все сохраним и запустим. Смотрим лог:

--- onCreate Activity ---Delete all from table mytableInsert in table mytable value = val1create DBHelperget dbclass android.database.sqlite.SQLiteException error: database is locked

Мы видим, что при попытке создать второе подключение к базе произошла ошибка **SQLiteException**– база **заблокирована**открытой транзакцией. Если вы теперь закоментите или удалите строки управления транзакцией и снова выполните код, то все пройдет успешно, т.к. никаких блокировок не будет.

Наверняка есть некоторые вопросы по этой теме. Попробую здесь же ответить на некоторые.

### close

Метод **close**есть и у **SQLiteDatabase**и у **SQLiteOpenHelper**. Какая между ними разница? Каким из них пользоваться для закрытия подключения?

Тут надо понимать один момент – объект **SQLiteOpenHelper**всегда предоставляет только **одно**подключение. Попробую объяснить этот механизм. У объекта **SQLiteOpenHelper**есть внутренний атрибут **mDatabase**типа **SQLiteDatabase**. Когда мы вызываем метод **getWritableDatabase**, объект **SQLiteOpenHelper**проверяет: если **mDatabase**не null и не закрыт, то он и идет в качестве **return**. Иначе  **SQLiteOpenHelper**выполняет подключение к БД, записывает новый **SQLiteDatabase**-объект в **mDatabase**и возвращает нам его. Т.е. метод **getWritableDatabase**либо возвращает **существующее подключение**к БД, либо создает новое в случае отсутствия подключения. Когда же выполняется метод **close**для **SQLiteOpenHelper**, то происходит вызов **close**для **mDatabase**и выполняется код **mDatabase = null**.

Рассмотрим на примере. Снова меняем метод myActions:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | void myActions() {    db = dbh.getWritableDatabase();    SQLiteDatabase db2 = dbh.getWritableDatabase();    Log.d(LOG\_TAG, "db = db2 - " + db.equals(db2));    Log.d(LOG\_TAG, "db open - " + db.isOpen() + ", db2 open - " + db2.isOpen());    db2.close();    Log.d(LOG\_TAG, "db open - " + db.isOpen() + ", db2 open - " + db2.isOpen());  } |

Сначала мы получаем **db**. При этом **dbh**проверяет свой внутренний атрибут **mDatabase**. Т.к. это первая попытка подключения, то **mDatabase**пуст, поэтому внутри **dbh**производится подключение и в **mDatabase**записывается свежесозданный **SQLiteDatabase**, и он же и возвращается в **db**из метода **getWritableDatabse**.

Затем мы из того же **dbh**получаем **db2**. **dbh**снова проверяет свой внутренний **mDatabse**, видит, что он уже **не null и не закрыт**, и возвращает нам его в наш **db2**. В итоге **db**и **db2**равны и ссылаются на один и тот же объект. Проверяем это с помощью метода **equals**. Далее проверим, что **db**и **db2 открыты**. Потом **закроем**только **db2**, и еще раз проверим на **открытость оба**объекта.

Сохраняем, запускаем, смотрим лог:

--- onCreate Activity ---db = db2 – truedb open - true, db2 open – truedb open - false, db2 open - false

Видим, что **equals**вернул **true**. Затем видно, что **db** и **db2 открыты**. А после закрытия **db2**видим, что закрыты оба объекта. Все оттого, что «оба объекта» – это всего лишь **две ссылки**на **один объект**.

Если в коде вместо **db2.close()** поставить**dbh.close()** - эффект будет тот же. **dbh**вызовет метод **close**для **mDatabase**и обнулит его -**mDatabase = null**. А **db**и **db2**будут ссылаться на закрытый **SQLiteDatabase**.

Я думаю, что правильнее вызывать **close**для **SQLiteOpenHelper**, а не для **SQLiteDatabase**. Т.к. гарантировано **закрывается**текущее открытое соединение и **обнуляется**внутренняя ссылка на объект.

Если вам надо получить второе открытое подключение к БД, то надо создавать новый экземпляр **DBHelper**и вызывать **getWritableDatabase**. Мы так делали чуть выше в примере с блокировкой транзакции.

### read write

В чем разница между **getWritableDatabase**и **getReadableDatabase**? Судя по хелпу, в обычной ситуации **оба**метода возвращают**одно и то же**. И оба позволят **читать**и **менять**БД. В случае же, например, проблемы**отсутствия свободного места** на устройстве, метод **getReadableDatabase**вернет БД только **для чтения**, а **getWritableDatabase**выдаст **ошибку**.

### \_id, как имя поля-идентификатора

В различных источниках при работе с БД в качестве наименования поля-идентификатора в таблице используют не просто id, а \_id. Почему?

Ответ нашелся в доках по Cursor-адаптерам. Цитата: "The Cursor must include a column named "\_id" or this class will not work.". Т.е. если вы планируете использовать Cursor-адаптеры, то необходимо, чтобы таблица содержала поле \_id, иначе адаптер не будет работать.

### Блокировка

Метод открытия транзакции **beginTransaction**ставит блокировку в режиме **EXCLUSIVE**. Т.е. БД блокируется и на чтение и на запись для других подключений. В SDK Android версии старше 2.3.3 появился метод **beginTransactionNonExclusive**, который ставит блокировку в режиме **IMMEDIATE**. Я подозреваю, что это позволит читать данные другим подключениям.

Если есть желание подробнее погрузиться в тему, вам [сюда](http://www.enterra.ru/blog/android_issues_with_sqlite/).

### Синтаксис

И кстати, рекомендуемая форма для использования транзакций такая:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | db.beginTransaction();  try {    ...    db.setTransactionSuccessful();  } finally {    db.endTransaction();  } |

Это очень важно! Т.е. если вы **открыли**транзакцию, **выполнили**какие-либо действия и**не закрыли** транзакцию, то все операции будут считаться **неуспешными**и изменения не будут внесены в БД. Поэтому **закрытие** транзакции **необходимо**выполнять и **finally**нам это гарантирует.

[onUpgrade. Обновляем БД в SQLite](https://startandroid.ru/ru/uroki/vse-uroki-spiskom/79-urok-39-onupgrade-obnovljaem-bd-v-sqlite.html)

В этом уроке:

- меняем версию и обновляем структуру БД в onUpgrade

С **развитием**приложения может возникнуть необходимость **изменения структуры БД**, которую оно использует. На одном из прошлых уроков я упоминал, что для этого используется метод **onUpgrade**класса **SQLiteOpenHelper**. Этот метод вызывается, если **существующая**версия БД **отличается**от той, к которой мы пытаемся **подключиться**. **Версию**мы обычно **указывали**при вызове **конструктора** супер-класса **SQLiteOpenHelper**в конструкторе DBHelper.

Попробуем воспользоваться методом onUpgrade и посмотреть, как происходит переход на новую версию БД. Для этого напишем небольшое приложение, аналогичное одному из приложений с прошлых уроков – про сотрудников и должности.

**Первая**версия БД будет содержать только таблицу **people**с именем сотрудника и его должностью. Но такая таблица будет не совсем корректна. Если вдруг у нас изменится название должности, придется обновлять все соответствующие записи в people. Поэтому мы решаем изменить БД и организовать данные немного по-другому.

Во **второй**версии добавим таблицу **position**с названием должности и зарплатой. И в таблице **people**вместо названия должности пропишем соответствующий **ID**из **position**.

Создадим проект:

**Project name**: P0391\_SQLiteOnUpgradeDB  
**Build Target**: Android 2.3.3  
**Application name**: SQLiteOnUpgradeDB  
**Package name**: ru.startandroid.develop.p0391sqliteonupgradedb  
**Create Activity**: MainActivity

Экран снова не используем, будем выводить все в лог.

Открываем **MainActivity.java** и кодим:

|  |  |
| --- | --- |
| 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  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95 | package ru.startandroid.develop.p0391sqliteonupgradedb;    import android.app.Activity;  import android.content.ContentValues;  import android.content.Context;  import android.database.Cursor;  import android.database.sqlite.SQLiteDatabase;  import android.database.sqlite.SQLiteOpenHelper;  import android.os.Bundle;  import android.util.Log;    public class MainActivity extends Activity {      final String LOG\_TAG = "myLogs";      final String DB\_NAME = "staff"; // имя БД    final int DB\_VERSION = 1; // версия БД      /\*\* Called when the activity is first created. \*/    @Override    public void onCreate(Bundle savedInstanceState) {        super.onCreate(savedInstanceState);      setContentView(R.layout.main);        DBHelper dbh = new DBHelper(this);      SQLiteDatabase db = dbh.getWritableDatabase();      Log.d(LOG\_TAG, " --- Staff db v." + db.getVersion() + " --- ");      writeStaff(db);      dbh.close();    }      // запрос данных и вывод в лог    private void writeStaff(SQLiteDatabase db) {      Cursor c = db.rawQuery("select \* from people", null);      logCursor(c, "Table people");      c.close();    }      // вывод в лог данных из курсора    void logCursor(Cursor c, String title) {      if (c != null) {        if (c.moveToFirst()) {          Log.d(LOG\_TAG, title + ". " + c.getCount() + " rows");          StringBuilder sb = new StringBuilder();          do {            sb.setLength(0);            for (String cn : c.getColumnNames()) {              sb.append(cn + " = "                  + c.getString(c.getColumnIndex(cn)) + "; ");            }            Log.d(LOG\_TAG, sb.toString());          } while (c.moveToNext());        }      } else        Log.d(LOG\_TAG, title + ". Cursor is null");    }      // класс для работы с БД    class DBHelper extends SQLiteOpenHelper {        public DBHelper(Context context) {        super(context, DB\_NAME, null, DB\_VERSION);      }        public void onCreate(SQLiteDatabase db) {        Log.d(LOG\_TAG, " --- onCreate database --- ");          String[] people\_name = { "Иван", "Марья", "Петр", "Антон", "Даша",            "Борис", "Костя", "Игорь" };        String[] people\_positions = { "Программер", "Бухгалтер",            "Программер", "Программер", "Бухгалтер", "Директор",            "Программер", "Охранник" };          ContentValues cv = new ContentValues();          // создаем таблицу людей        db.execSQL("create table people ("            + "id integer primary key autoincrement,"            + "name text, position text);");          // заполняем ее        for (int i = 0; i < people\_name.length; i++) {          cv.clear();          cv.put("name", people\_name[i]);          cv.put("position", people\_positions[i]);          db.insert("people", null, cv);        }      }        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {        }    }  } |

Код несложен. Я сгруппировал операции по**выводу в лог данных** из Cursor – метод **logCursor**. Метод **writeStaff**– выбирает данные из таблицы people и вызывает метод для вывода данных в лог. В методе Activity **onCreate**мы создаем объект **DBHelper**, **подключаемся**к БД, выводим в лог **версию**БД, вызываем **writeStaff**и **отключаемся**.

В **DBHelper**все как обычно. В конструкторе вызываем конструктор супер-класса. Обратите внимание, **DB\_VERSION**= 1 – мы будем подключаться к базе версии 1. В методе **onCreate**создаем таблицу и заполняем ее.

Все сохраним и запустим приложение. Смотрим лог:

*--- onCreate database ---   
--- Staff db v.1 ---   
Table people. 8 rows  
id = 1; name = Иван; position = Программер;   
id = 2; name = Марья; position = Бухгалтер;   
id = 3; name = Петр; position = Программер;   
id = 4; name = Антон; position = Программер;   
id = 5; name = Даша; position = Бухгалтер;   
id = 6; name = Борис; position = Директор;   
id = 7; name = Костя; position = Программер;   
id = 8; name = Игорь; position = Охранник;*

БД **создалась**, версия = **1**и данные из таблицы вывелись в лог. Приложение работает, все ок. Но тут мы (внезапно!) понимаем, что при проектировании структуры БД была допущена ошибка. Записывать название должности в поле таблицы people – неправильно. К тому же у нас еще добавляются данные по зарплатам. Надо создать таблицу должностей - position, и использовать из нее id в таблице people. Тем самым структура нашей БД меняется и мы присваиваем ей версию – **2**.

Но наше приложение уже установлено у пользователей. Оно уже создало БД версии 1, и в этой БД уже есть данные. Мы не можем просто удалить существующие таблицы и создать новые, т.к. возможно пользователь уже хранит там свои данные. Нам надо будет написать скрипты для обновления без потери данных.

План обновления такой:

- создаем и заполняем данными таблицу **position**  
- добавляем в таблицу **people**столбец – **posid**для хранения **id**из **position**  
- заполняем**people.posid** данными из **position**в зависимости от значения **people.position**  
- удаляем столбец **people.position**

Давайте менять **MainActivity.java**. Наше приложение теперь будет ориентировано на БД версии 2. Укажем это, изменив значение константы **DB\_VERSION**на 2:

|  |  |
| --- | --- |
| 1 | final int DB\_VERSION = 2; // версия БД |

Метод writeStaff перепишем таким образом:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | private void writeStaff(SQLiteDatabase db) {      Cursor c = db.rawQuery("select \* from people", null);      logCursor(c, "Table people");      c.close();        c = db.rawQuery("select \* from position", null);      logCursor(c, "Table position");      c.close();        String sqlQuery = "select PL.name as Name, PS.name as Position, salary as Salary "        + "from people as PL "        + "inner join position as PS "        + "on PL.posid = PS.id ";      c = db.rawQuery(sqlQuery, null);      logCursor(c, "inner join");      c.close();    } |

Будем выводить в лог данные из таблиц people, position и их объединения.

Реализуем метод обновления - **onUpgrade**в **DBHelper**:

|  |  |
| --- | --- |
| 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 | public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {    Log.d(LOG\_TAG, " --- onUpgrade database from " + oldVersion        + " to " + newVersion + " version --- ");      if (oldVersion == 1 && newVersion == 2) {        ContentValues cv = new ContentValues();        // данные для таблицы должностей      int[] position\_id = { 1, 2, 3, 4 };      String[] position\_name = { "Директор", "Программер",          "Бухгалтер", "Охранник" };      int[] position\_salary = { 15000, 13000, 10000, 8000 };        db.beginTransaction();      try {        // создаем таблицу должностей        db.execSQL("create table position ("            + "id integer primary key,"            + "name text, salary integer);");          // заполняем ее        for (int i = 0; i < position\_id.length; i++) {          cv.clear();          cv.put("id", position\_id[i]);          cv.put("name", position\_name[i]);          cv.put("salary", position\_salary[i]);          db.insert("position", null, cv);        }          db.execSQL("alter table people add column posid integer;");          for (int i = 0; i < position\_id.length; i++) {          cv.clear();          cv.put("posid", position\_id[i]);          db.update("people", cv, "position = ?",              new String[] { position\_name[i] });        }          db.execSQL("create temporary table people\_tmp ("            + "id integer, name text, position text, posid integer);");          db.execSQL("insert into people\_tmp select id, name, position, posid from people;");        db.execSQL("drop table people;");          db.execSQL("create table people ("            + "id integer primary key autoincrement,"            + "name text, posid integer);");          db.execSQL("insert into people select id, name, posid from people\_tmp;");        db.execSQL("drop table people\_tmp;");          db.setTransactionSuccessful();      } finally {        db.endTransaction();      }    }  } |

Все в соответствии с планом обновления, который я приводил выше. Есть пара нюансов.

Во-первых, используем**БД-транзакцию**. Т.е. нам надо чтобы на БД накатились **все**наши обновления. А в случае **ошибки**в процессе обновления - **все изменения должны быть отменены** и БД должна остаться прежней. Тут транзакции очень выручают.

Во-вторых, в SQLite **нельзя**просто так **удалить**столбец, приходится создавать временную таблицу, перекидывать туда данные, удалять оригинал, создавать его снова с нужной структурой, скидывать в него данные из временной таблицы и удалять временную таблицу. Подробнее об этом можно почитать тут - [How do I add or delete columns from an existing table in SQLite](http://www.sqlite.org/faq.html#q11).

Наше приложение **обновилось**. И теперь, при запуске, оно попытается подключиться к БД версии **2**, но увидит, что существующая версия = **1**и вызовет метод **onUpgrade**, дав нам возможность внести необходимые изменения в структуру БД. Но это произойдет в случае обновления приложения. А что будет если пользователь поставит наше новое приложение на свежий смартфон первый раз?

В этом случае приложение также попытается подключиться к БД версии **2**. Но т.к. приложение только что установлено, то БД еще не существует. Приложение **создаст**БД и присвоит ей версию номер **2**, т.к. оно умеет работать именно с такой версией. При создании будет вызван метод **onCreate**в **DBHelper**. Значит, в нем мы должны прописать код, который будет создавать нам БД версии **2**– т.е. обновленную таблицу **people**и новую таблицу **position**.

Пишем **onCreate**в DBHelper:

|  |  |
| --- | --- |
| 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 | public void onCreate(SQLiteDatabase db) {    Log.d(LOG\_TAG, " --- onCreate database --- ");      String[] people\_name = { "Иван", "Марья", "Петр", "Антон", "Даша",        "Борис", "Костя", "Игорь" };    int[] people\_posid = { 2, 3, 2, 2, 3, 1, 2, 4 };      // данные для таблицы должностей    int[] position\_id = { 1, 2, 3, 4 };    String[] position\_name = { "Директор", "Программер", "Бухгалтер",        "Охранник" };    int[] position\_salary = { 15000, 13000, 10000, 8000 };      ContentValues cv = new ContentValues();      // создаем таблицу должностей    db.execSQL("create table position (" + "id integer primary key,"        + "name text, salary integer" + ");");      // заполняем ее    for (int i = 0; i < position\_id.length; i++) {      cv.clear();      cv.put("id", position\_id[i]);      cv.put("name", position\_name[i]);      cv.put("salary", position\_salary[i]);      db.insert("position", null, cv);    }      // создаем таблицу людей    db.execSQL("create table people ("        + "id integer primary key autoincrement,"        + "name text, posid integer);");      // заполняем ее    for (int i = 0; i < people\_name.length; i++) {      cv.clear();      cv.put("name", people\_name[i]);      cv.put("posid", people\_posid[i]);      db.insert("people", null, cv);    }  } |

Создание и заполнение данными двух таблиц. Все понятно.

Теперь можно все сохранить и запустить приложение.

Смотрим лог:

*--- onUpgrade database from 1 to 2 version ---   
--- Staff db v.2 ---   
Table people. 8 rows  
id = 1; name = Иван; posid = 2;   
id = 2; name = Марья; posid = 3;   
id = 3; name = Петр; posid = 2;   
id = 4; name = Антон; posid = 2;   
id = 5; name = Даша; posid = 3;   
id = 6; name = Борис; posid = 1;   
id = 7; name = Костя; posid = 2;   
id = 8; name = Игорь; posid = 4;   
Table position. 4 rows  
id = 1; name = Директор; salary = 15000;   
id = 2; name = Программер; salary = 13000;   
id = 3; name = Бухгалтер; salary = 10000;   
id = 4; name = Охранник; salary = 8000;   
inner join. 8 rows  
Name = Иван; Position = Программер; Salary = 13000;   
Name = Марья; Position = Бухгалтер; Salary = 10000;   
Name = Петр; Position = Программер; Salary = 13000;   
Name = Антон; Position = Программер; Salary = 13000;   
Name = Даша; Position = Бухгалтер; Salary = 10000;   
Name = Борис; Position = Директор; Salary = 15000;   
Name = Костя; Position = Программер; Salary = 13000;   
Name = Игорь; Position = Охранник; Salary = 8000;*

Видим, что вызывался **onUpgrade**и **обновил**нам БД с версии**1** на **2**. Далее выводим все данные, чтобы убедиться, что обновление прошло корректно.

Можно также убедиться, что новый onCreate в DBHelper корректно отработает. Для этого надо удалить файл БД и запустить приложение. Приложение не найдет БД и создаст ее сразу в новом формате и с версией 2.

Сценарий выдуманный, есть к чему придраться и о чем поспорить, но смысл не в этом. Смысл в том, что мы увидели, как происходит **обновление**БД, если приложение запросило новую версию. Поначалу, возможно, покажется запутанным этот механизм создания и обновления. Но сложного реально ничего нет. С опытом придет полное понимание.

Еще хочу отметить, что у объекта Cursor есть метод close(), который освобождает занимаемые им ресурсы. Не забывайте про него.

Думаю, теперь можно смело сказать, что работу с **SQLite** в Android мы изучили достаточно основательно. И в дальнейших уроках сможем свободно использовать эти знания.

Полный код **MainActivity.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  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180 | package ru.startandroid.develop.p0391sqliteonupgradedb;    import android.app.Activity;  import android.content.ContentValues;  import android.content.Context;  import android.database.Cursor;  import android.database.sqlite.SQLiteDatabase;  import android.database.sqlite.SQLiteOpenHelper;  import android.os.Bundle;  import android.util.Log;    public class MainActivity extends Activity {      final String LOG\_TAG = "myLogs";      final String DB\_NAME = "staff"; // имя БД    final int DB\_VERSION = 2; // версия БД      /\*\* Called when the activity is first created. \*/    @Override    public void onCreate(Bundle savedInstanceState) {        super.onCreate(savedInstanceState);      setContentView(R.layout.main);        DBHelper dbh = new DBHelper(this);      SQLiteDatabase db = dbh.getWritableDatabase();      Log.d(LOG\_TAG, " --- Staff db v." + db.getVersion() + " --- ");      writeStaff(db);      dbh.close();    }      // запрос данных и вывод в лог    private void writeStaff(SQLiteDatabase db) {      Cursor c = db.rawQuery("select \* from people", null);      logCursor(c, "Table people");      c.close();        c = db.rawQuery("select \* from position", null);      logCursor(c, "Table position");      c.close();        String sqlQuery = "select PL.name as Name, PS.name as Position, salary as Salary "          + "from people as PL "          + "inner join position as PS "          + "on PL.posid = PS.id ";      c = db.rawQuery(sqlQuery, null);      logCursor(c, "inner join");      c.close();    }      // вывод в лог данных из курсора    void logCursor(Cursor c, String title) {      if (c != null) {        if (c.moveToFirst()) {          Log.d(LOG\_TAG, title + ". " + c.getCount() + " rows");          StringBuilder sb = new StringBuilder();          do {            sb.setLength(0);            for (String cn : c.getColumnNames()) {              sb.append(cn + " = "                  + c.getString(c.getColumnIndex(cn)) + "; ");            }            Log.d(LOG\_TAG, sb.toString());          } while (c.moveToNext());        }      } else        Log.d(LOG\_TAG, title + ". Cursor is null");    }      // класс для работы с БД    class DBHelper extends SQLiteOpenHelper {        public DBHelper(Context context) {        super(context, DB\_NAME, null, DB\_VERSION);      }        public void onCreate(SQLiteDatabase db) {        Log.d(LOG\_TAG, " --- onCreate database --- ");          String[] people\_name = { "Иван", "Марья", "Петр", "Антон", "Даша",            "Борис", "Костя", "Игорь" };        int[] people\_posid = { 2, 3, 2, 2, 3, 1, 2, 4 };          // данные для таблицы должностей        int[] position\_id = { 1, 2, 3, 4 };        String[] position\_name = { "Директор", "Программер", "Бухгалтер",            "Охранник" };        int[] position\_salary = { 15000, 13000, 10000, 8000 };          ContentValues cv = new ContentValues();          // создаем таблицу должностей        db.execSQL("create table position (" + "id integer primary key,"            + "name text, salary integer" + ");");          // заполняем ее        for (int i = 0; i < position\_id.length; i++) {          cv.clear();          cv.put("id", position\_id[i]);          cv.put("name", position\_name[i]);          cv.put("salary", position\_salary[i]);          db.insert("position", null, cv);        }          // создаем таблицу людей        db.execSQL("create table people ("            + "id integer primary key autoincrement,"            + "name text, posid integer);");          // заполняем ее        for (int i = 0; i < people\_name.length; i++) {          cv.clear();          cv.put("name", people\_name[i]);          cv.put("posid", people\_posid[i]);          db.insert("people", null, cv);        }      }        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {        Log.d(LOG\_TAG, " --- onUpgrade database from " + oldVersion            + " to " + newVersion + " version --- ");          if (oldVersion == 1 && newVersion == 2) {            ContentValues cv = new ContentValues();            // данные для таблицы должностей          int[] position\_id = { 1, 2, 3, 4 };          String[] position\_name = { "Директор", "Программер",              "Бухгалтер", "Охранник" };          int[] position\_salary = { 15000, 13000, 10000, 8000 };            db.beginTransaction();          try {            // создаем таблицу должностей            db.execSQL("create table position ("                + "id integer primary key,"                + "name text, salary integer);");              // заполняем ее            for (int i = 0; i < position\_id.length; i++) {              cv.clear();              cv.put("id", position\_id[i]);              cv.put("name", position\_name[i]);              cv.put("salary", position\_salary[i]);              db.insert("position", null, cv);            }              db.execSQL("alter table people add column posid integer;");              for (int i = 0; i < position\_id.length; i++) {              cv.clear();              cv.put("posid", position\_id[i]);              db.update("people", cv, "position = ?",                  new String[] { position\_name[i] });            }              db.execSQL("create temporary table people\_tmp ("                + "id integer, name text, position text, posid integer);");              db.execSQL("insert into people\_tmp select id, name, position, posid from people;");            db.execSQL("drop table people;");              db.execSQL("create table people ("                + "id integer primary key autoincrement,"                + "name text, posid integer);");              db.execSQL("insert into people select id, name, posid from people\_tmp;");            db.execSQL("drop table people\_tmp;");              db.setTransactionSuccessful();          } finally {            db.endTransaction();          }        }      }      }  } |