**Размещение файлов во внешнем хранилище**

В прошлой теме мы рассмотрели сохранение и чтение файлов из каталога приложения. По умолчанию такие файлы доступны только самому приложения. Однако мы можем помещать и работать с файлами из внешнего хранилища. Это также позволит другим программам открывать данные файлы и при необходимости изменять.

Весь механизм работы с файлами будет таким же, как и при работе с хранилищем приложения. Ключевым отличием здесь будет получение и использование пути к внешнему хранилищу через метод **Environment.getExternalStorageDirectory()**

Итак, пусть в файле **activity\_main.xml** будет такая же разметка интерфейса.

А код класса **MainActivity** будет выглядеть следующим образом:

package com.example.eugene.filesapp;

import android.Manifest;

import android.content.pm.PackageManager;

import android.os.Environment;

import android.support.annotation.NonNull;

import android.support.v4.app.ActivityCompat;

import android.support.v4.content.ContextCompat;

import android.support.v7.app.AppCompatActivity;

import android.os.Bundle;

import android.view.View;

import android.widget.EditText;

import android.widget.TextView;

import android.widget.Toast;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.IOException;

public class MainActivity extends AppCompatActivity {

    private final static String FILE\_NAME = "content.txt";

    private static final int REQUEST\_PERMISSION\_WRITE = 1001;

    private boolean permissionGranted;

    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity\_main);

    }

**private File getExternalPath() {**

**return(new File(Environment.getExternalStorageDirectory(), FILE\_NAME));**

    }

    // сохранение файла

    public void saveText(View view){

        if(!permissionGranted){

            checkPermissions();

            return;

        }

        FileOutputStream fos = null;

        try {

            EditText textBox = (EditText) findViewById(R.id.save\_text);

            String text = textBox.getText().toString();

            fos = new FileOutputStream(getExternalPath());

            fos.write(text.getBytes());

            Toast.makeText(this, "Файл сохранен", Toast.LENGTH\_SHORT).show();

        }

        catch(IOException ex) {

            Toast.makeText(this, ex.getMessage(), Toast.LENGTH\_SHORT).show();

        }

        finally{

            try{

                if(fos!=null)

                    fos.close();

            }

            catch(IOException ex){

                Toast.makeText(this, ex.getMessage(), Toast.LENGTH\_SHORT).show();

            }

        }

    }

    // открытие файла

    public void openText(View view){

        if(!permissionGranted){

            checkPermissions();

            return;

        }

        FileInputStream fin = null;

        TextView textView = (TextView) findViewById(R.id.open\_text);

        File file = getExternalPath();

        // если файл не существует, выход из метода

        if(!file.exists()) return;

        try {

            fin =  new FileInputStream(file);

            byte[] bytes = new byte[fin.available()];

            fin.read(bytes);

            String text = new String (bytes);

            textView.setText(text);

        }

        catch(IOException ex) {

            Toast.makeText(this, ex.getMessage(), Toast.LENGTH\_SHORT).show();

        }

        finally{

            try{

                if(fin!=null)

                    fin.close();

            }

            catch(IOException ex){

                Toast.makeText(this, ex.getMessage(), Toast.LENGTH\_SHORT).show();

            }

        }

    }

    // проверяем, доступно ли внешнее хранилище для чтения и записи

    public boolean isExternalStorageWriteable(){

        String state = Environment.getExternalStorageState();

        return  Environment.MEDIA\_MOUNTED.equals(state);

    }

    // проверяем, доступно ли внешнее хранилище хотя бы только для чтения

    public boolean isExternalStorageReadable(){

        String state = Environment.getExternalStorageState();

        return  (Environment.MEDIA\_MOUNTED.equals(state) ||

                Environment.MEDIA\_MOUNTED\_READ\_ONLY.equals(state));

    }

    private boolean checkPermissions(){

        if(!isExternalStorageReadable() || !isExternalStorageWriteable()){

            Toast.makeText(this, "Внешнее хранилище не доступно", Toast.LENGTH\_LONG).show();

            return false;

        }

        int permissionCheck = ContextCompat.checkSelfPermission(this, Manifest.permission.WRITE\_EXTERNAL\_STORAGE);

        if(permissionCheck!= PackageManager.PERMISSION\_GRANTED){

            ActivityCompat.requestPermissions(this, new String[]{Manifest.permission.WRITE\_EXTERNAL\_STORAGE}, REQUEST\_PERMISSION\_WRITE);

            return false;

        }

        return true;

    }

    @Override

    public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults){

        switch (requestCode){

            case REQUEST\_PERMISSION\_WRITE:

                if(grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION\_GRANTED){

                    permissionGranted = true;

                    Toast.makeText(this, "Разрешения получены", Toast.LENGTH\_LONG).show();

                }

                else{

                    Toast.makeText(this, "Необходимо дать разрешения", Toast.LENGTH\_LONG).show();

                }

                break;

        }

    }

}

С помощью выражения Environment.getExternalStorageDirectory() получаем доступ к папке приложения во внешнем хранилище и устанавливаем объект файла:

|  |  |
| --- | --- |
|  | private File getExternalPath() {      return(new File(Environment.getExternalStorageDirectory(), FILE\_NAME));  } |

Поскольку для чтения/записи во внешнее хранилище необходимы разрешения, то перед операциями сохранения и записи файла необходимо проверить наличие разрешений. Для этого определен метод checkPermissions(). При установке разрешений срабатывает метод onRequestPermissionsResult(), в котором в случае удачной установки разрешений для переменной permissionGranted задается значение true.

Чтобы использовать внешнее хранилище, также надо установить разрешения в файле манифеста AndroidManifest.xml:

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

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

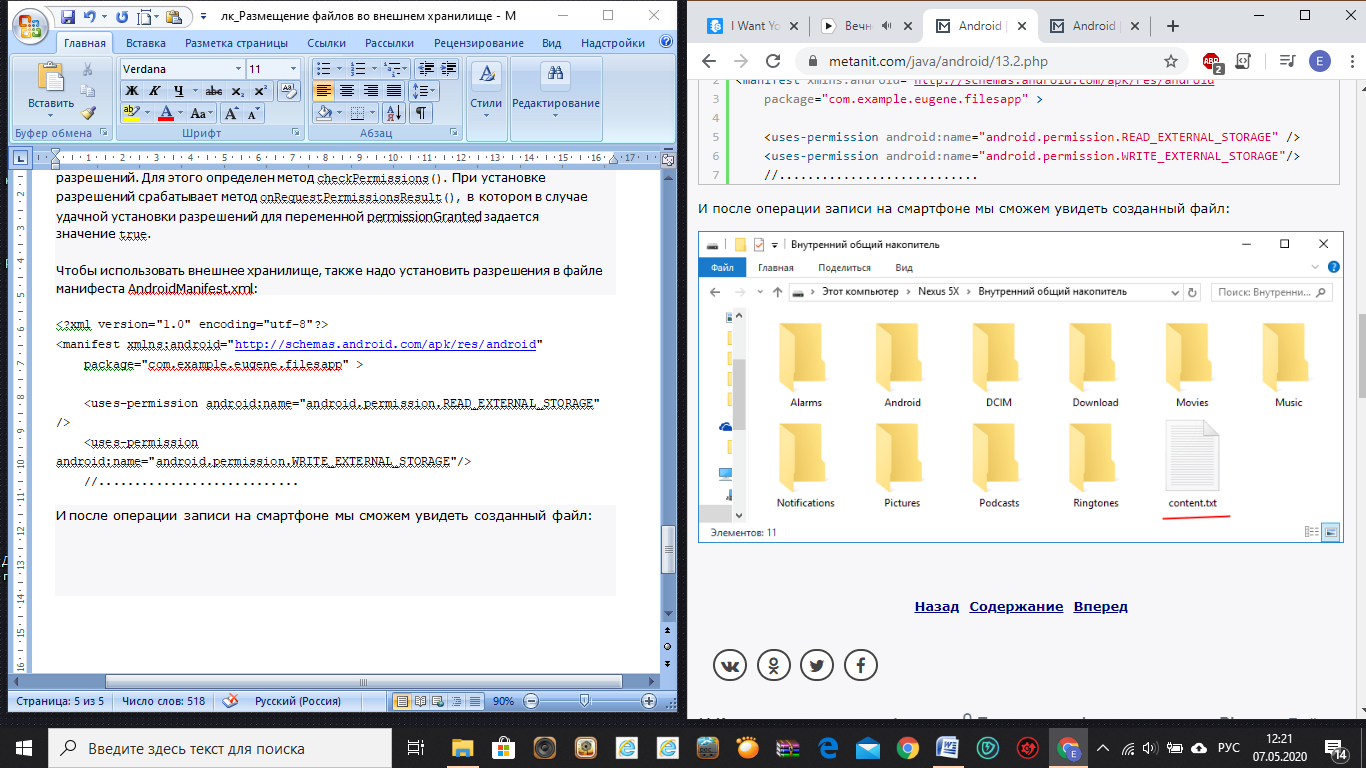
    package="com.example.eugene.filesapp" >

    <uses-permission android:name="android.permission.READ\_EXTERNAL\_STORAGE" />

    <uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE"/>

    //............................

И после операции записи на смартфоне мы сможем увидеть созданный файл:



## Работа с json

ля работы с форматом json нет встроенных средств, но есть куча библиотек и пакетов, которые можно использовать для данной цели. Одним из наиболее популярных из них является пакет **com.google.code.gson**.

Для его использования в проекте Android, необходимо добавить соответствующую зависимость в файл **guild.gradle**, который относится к модулю app:

|  |  |
| --- | --- |
|  | compile 'com.google.code.gson:gson:2.8.0' |

То есть после добавления секция зависимостей в файле build.gradle может выглядеть следующим образом:

dependencies {

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

    androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {

        exclude group: 'com.android.support', module: 'support-annotations'

    })

    compile 'com.android.support:appcompat-v7:25.3.0'

    compile 'com.android.support.constraint:constraint-layout:1.0.2'

    compile 'com.google.code.gson:gson:2.8.0'

    testCompile 'junit:junit:4.12'

}

После добавления пакета в проект добавим новый класс **Phone**, который будет представлять данные:

package com.example.eugene.jsonapp;

public class Phone {

    private String name;

    private int price;

    Phone(String name, int price){

        this.name = name;

        this.price = price;

    }

    public String getName() {

        return name;

    }

    public void setName(String name) {

        this.name = name;

    }

    public int getPrice() {

        return price;

    }

    public void setPrice(int price) {

        this.price = price;

    }

    @Override

    public  String toString(){

        return name + " " + String.valueOf(price);

    }

}

Объекты этого класса мы будем сериализовать в формат json и наоборот десериализовать из файла.

Для работы с json добавим следующий класс **JSONHelper**:

package com.example.eugene.jsonapp;

import android.content.Context;

import com.google.gson.Gson;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.List;

class JSONHelper {

    private static final String FILE\_NAME = "data.json";

    static boolean exportToJSON(Context context, List<Phone> dataList) {

        Gson gson = new Gson();

        DataItems dataItems = new DataItems();

        dataItems.setPhones(dataList);

        String jsonString = gson.toJson(dataItems);

        FileOutputStream fileOutputStream = null;

        try {

            fileOutputStream = context.openFileOutput(FILE\_NAME, Context.MODE\_PRIVATE);

            fileOutputStream.write(jsonString.getBytes());

            return true;

        } catch (Exception e) {

            e.printStackTrace();

        } finally {

            if (fileOutputStream != null) {

                try {

                    fileOutputStream.close();

                } catch (IOException e) {

                    e.printStackTrace();

                }

            }

        }

        return false;

    }

    static List<Phone> importFromJSON(Context context) {

        InputStreamReader streamReader = null;

        FileInputStream fileInputStream = null;

        try{

            fileInputStream = context.openFileInput(FILE\_NAME);

            streamReader = new InputStreamReader(fileInputStream);

            Gson gson = new Gson();

            DataItems dataItems = gson.fromJson(streamReader, DataItems.class);

            return  dataItems.getPhones();

        }

        catch (IOException ex){

            ex.printStackTrace();

        }

        finally {

            if (streamReader != null) {

                try {

                    streamReader.close();

                } catch (IOException e) {

                    e.printStackTrace();

                }

            }

            if (fileInputStream != null) {

                try {

                    fileInputStream.close();

                } catch (IOException e) {

                    e.printStackTrace();

                }

            }

        }

        return null;

    }

    private static class DataItems {

        private List<Phone> phones;

        List<Phone> getPhones() {

            return phones;

        }

        void setPhones(List<Phone> phones) {

            this.phones = phones;

        }

    }

}

Для работы с json создается объект **Gson**. Для сериализации данных в формат json у этого объекта вызывается метод **toJson()**, в который передаются сериализуемые данные.

Для упрощения работы с данными применяется вспомогательный класс DataItems. На выходе метод toJson() возвращает строку, которая затем сохраняется в текстовый файл.

Для десериализации выполняется метод **fromJson()**, в который передается объект Reader с сериализованными данными и тип, к которому надо десериализиовать данные.

Теперь определим основной функционал для взаимодействия с пользователем. Изменим файл **activity\_main.xml** следующим образом:

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

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

    android:layout\_width="match\_parent"

    android:layout\_height="match\_parent"

    android:orientation="vertical">

    <EditText

        android:id="@+id/nameText"

        android:layout\_width="match\_parent"

        android:layout\_height="wrap\_content"

        android:hint="Введите название"/>

    <EditText

        android:id="@+id/priceText"

        android:layout\_width="match\_parent"

        android:layout\_height="wrap\_content"

        android:hint="Введите цену"/>

    <Button

        android:layout\_width="match\_parent"

        android:layout\_height="wrap\_content"

        android:text="Добавить"

        android:onClick="addPhone"/>

    <LinearLayout

        android:layout\_width="match\_parent"

        android:layout\_height="wrap\_content"

        android:orientation="horizontal">

        <Button

            android:layout\_width="0dp"

            android:layout\_weight="1"

            android:layout\_height="wrap\_content"

            android:text="Сохранить"

            android:onClick="save"/>

        <Button

            android:layout\_width="0dp"

            android:layout\_weight="1"

            android:layout\_height="wrap\_content"

            android:text="Открыть"

            android:onClick="open"/>

    </LinearLayout>

    <ListView

        android:id="@+id/list"

        android:layout\_width="match\_parent"

        android:layout\_height="wrap\_content" />

</LinearLayout>

Здесь определены два текстовых поля для ввода названия модели и цены объекта Phone и одна кнопка для добавления данных в список. Еще одна кнопка выполняет сериализацию данных из списка в файл, а третья кнопка - восстановление данных из файла.

Для вывода сами данных определен элемент ListView.

И изменим класс **MainActivity**:

package com.example.eugene.jsonapp;

import android.support.v7.app.AppCompatActivity;

import android.os.Bundle;

import android.view.View;

import android.widget.ArrayAdapter;

import android.widget.EditText;

import android.widget.ListView;

import android.widget.Toast;

import java.util.ArrayList;

import java.util.List;

public class MainActivity extends AppCompatActivity {

    private ArrayAdapter<Phone> adapter;

    private EditText nameText, priceText;

    private List<Phone> phones;

    ListView listView;

    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity\_main);

        nameText = (EditText) findViewById(R.id.nameText);

        priceText = (EditText) findViewById(R.id.priceText);

        phones = new ArrayList<>();

        listView = (ListView) findViewById(R.id.list);

        adapter = new ArrayAdapter<>(this, android.R.layout.simple\_list\_item\_1, phones);

        listView.setAdapter(adapter);

    }

    public void addPhone(View view){

        String name = nameText.getText().toString();

        int price = Integer.parseInt(priceText.getText().toString());

        Phone phone = new Phone(name, price);

        phones.add(phone);

        adapter.notifyDataSetChanged();

    }

    public void save(View view){

        boolean result = JSONHelper.exportToJSON(this, phones);

        if(result){

            Toast.makeText(this, "Данные сохранены", Toast.LENGTH\_LONG).show();

        }

        else{

            Toast.makeText(this, "Не удалось сохранить данные", Toast.LENGTH\_LONG).show();

        }

    }

    public void open(View view){

        phones = JSONHelper.importFromJSON(this);

        if(phones!=null){

            adapter = new ArrayAdapter<>(this, android.R.layout.simple\_list\_item\_1, phones);

            listView.setAdapter(adapter);

            Toast.makeText(this, "Данные восстановлены", Toast.LENGTH\_LONG).show();

        }

        else{

            Toast.makeText(this, "Не удалось открыть данные", Toast.LENGTH\_LONG).show();

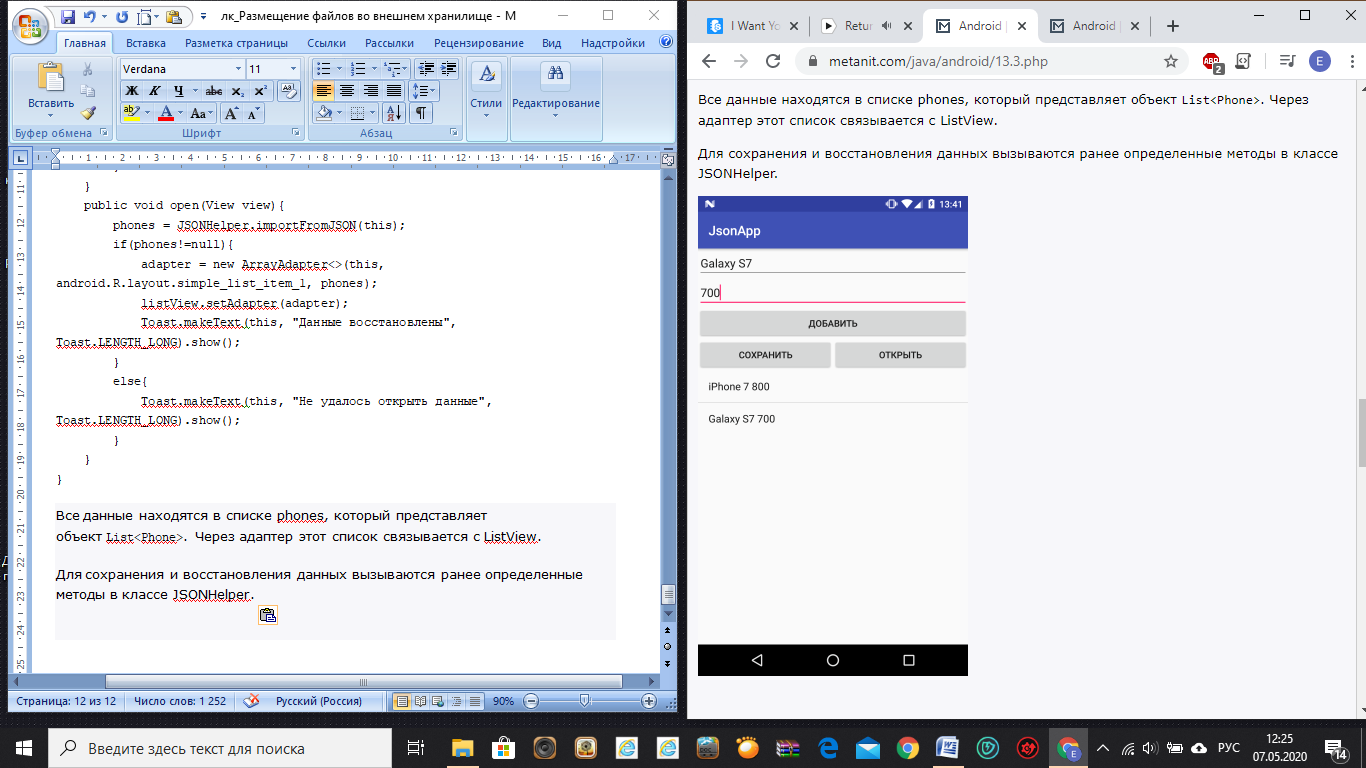
        }

    }

}

Все данные находятся в списке phones, который представляет объект List<Phone>. Через адаптер этот список связывается с ListView.

Для сохранения и восстановления данных вызываются ранее определенные методы в классе JSONHelper.



# Работа с базами данных SQLite

## Подключение к базе данных SQLite

В Android имеется встроенная поддержка одной из распространенных систем управления базами данных - SQLite. Для этого в пакете **android.database.sqlite** определен набор классов, которые позволяют работать с базами данных SQLite. И каждое приложение может создать свою базу данных.

Чтобы использовать SQLite в Android, надо создать базу данных с помощью выражение на языке SQL. После этого база данных будет храниться в каталоге приложения по пути:

|  |  |
| --- | --- |
|  | DATA/data/[Название\_приложения]/databases/[Название\_файла\_базы\_данных] |

ОС Android по умолчанию уже содержит ряд встроенных бад SQLite, которые используются стандартными программами - для списка контактов, для хранения фотографий с камеры, музыкальных альбомов и т.д.

Основную функциональность по работе с базами данных предоставляет пакет **android.database**. Функциональность непосредственно для работы с SQLite находится в пакете **android.database.sqlite**.

База данных в SQLite представлена классом **android.database.sqlite.SQLiteDatabase**. Он позволяет выполнять запросы к бд, выполнять с ней различные манипуляции.

Класс **android.database.sqlite.SQLiteCursor** предоставляет запрос и позволяет возвращать набор строк, которые соответствуют этому запросу.

Класс **android.database.sqlite.SQLiteQueryBuilder** позволяет создавать SQL-запросы.

Сами sql-выражения представлены классом **android.database.sqlite.SQLiteStatement**, которые позволяют с помощью плейсхолдеров вставлять в выражения динамические данные.

Класс **android.database.sqlite.SQLiteOpenHelper** позволяет создать базу данных со всеми таблицами, если их еще не существует.

В SQLite применяется следующая система типов данных:

* **INTEGER**: представляет целое число, аналог типу int в java
* **REAL**: представляет число с плавающей точкой, аналог float и double в java
* **TEXT**: представляет набор символов, аналог String и char в java
* **BLOB**: представляет массив бинарных данных, например, изображение, аналог типу int в java

Сохраняемые данные должны представлять соответствующие типы в java.

### Создание и открытие базы данных

Для создания или открытия новой базы данных из кода Activity в Android мы можем вызвать метод **openOrCreateDatabase()**. Этот метод может принимать три параметра:

* название для базы данных
* числовое значение, которое определяет режим работы (как правило, в виде константы MODE\_PRIVATE)
* необязательный параметр в виде объекта SQLiteDatabase.CursorFactory, который представляет фабрику создания курсора для работы с бд

Например, создание базы данных app.db:

|  |  |
| --- | --- |
|  | SQLiteDatabase db = getBaseContext().openOrCreateDatabase("app.db", MODE\_PRIVATE, null); |

Для выполнения запроса к базе данных можно использовать метод **execSQL** класса SQLiteDatabase. В этот метод передается SQL-выражение. Например, создание в базе данных таблицы users:

|  |  |
| --- | --- |
|  | SQLiteDatabase db = getBaseContext().openOrCreateDatabase("app.db", MODE\_PRIVATE, null);  db.execSQL("CREATE TABLE IF NOT EXISTS users (name TEXT, age INTEGER)"); |

Если нам надо не просто выполнить выражение, но и получить из бд какие-либо данные, то используется метод **rawQuery()**. Этот метод в качестве параметра принимает SQL-выражение, а также набор значений для выражения sql. Например, получение всех объектов из базы данных:

SQLiteDatabase db = getBaseContext().openOrCreateDatabase("app.db", MODE\_PRIVATE, null);

db.execSQL("CREATE TABLE IF NOT EXISTS users (name TEXT, age INTEGER)");

Cursor query = db.rawQuery("SELECT \* FROM users;", null);

if(query.moveToFirst()){

    String name = query.getString(0);

    int age = query.getInt(1);

}

Метод db.rawQuery() возвращает объект Cursor, с помощью которого мы можем извлечь полученные данные.

Возможна ситуация, когда в базе данных не будет объектов, и для этого методом query.moveToFirst() пытаемся переместиться к первому объекту, полученному из бд. Если этот метод возвратит значение false, значит запрос не получил никаких данных из бд.

Теперь для работы с базой данных сделаем простейшее приложение. Для этого создадим новый проект.

В файле **activity\_main.xml** определим простейший графический интерфейс:

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

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

    android:id="@+id/activity\_main"

    android:layout\_width="match\_parent"

    android:layout\_height="match\_parent"

    android:padding="16dp"

    android:orientation="vertical">

    <Button

        android:layout\_width="wrap\_content"

        android:layout\_height="wrap\_content"

        android:text="Click"

        android:onClick="onClick"/>

    <TextView

        android:id="@+id/textView"

        android:layout\_width="wrap\_content"

        android:layout\_height="wrap\_content"

        android:textSize="18sp" />

</LinearLayout>

А в классе **MainActivity** определим взаимодействие с базой данных:

package com.example.eugene.sqliteapp;

import android.database.Cursor;

import android.database.sqlite.SQLiteDatabase;

import android.support.v7.app.AppCompatActivity;

import android.os.Bundle;

import android.view.View;

import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity\_main);

    }

    public void onClick(View view){

        SQLiteDatabase db = getBaseContext().openOrCreateDatabase("app.db", MODE\_PRIVATE, null);

        db.execSQL("CREATE TABLE IF NOT EXISTS users (name TEXT, age INTEGER)");

        db.execSQL("INSERT INTO users VALUES ('Tom Smith', 23);");

        db.execSQL("INSERT INTO users VALUES ('John Dow', 31);");

        Cursor query = db.rawQuery("SELECT \* FROM users;", null);

        TextView textView = (TextView) findViewById(R.id.textView);

        if(query.moveToFirst()){

            do{

                String name = query.getString(0);

                int age = query.getInt(1);

                textView.append("Name: " + name + " Age: " + age + "\n");

            }

            while(query.moveToNext());

        }

        query.close();

        db.close();

    }

}

По нажатию на кнопку здесь вначале создается в базе данных app.db новая таблица users, а затем в нее добавляются два объекта в базу данных с помощью SQL-выражения INSERT.

Далее с помощью выражения SELECT получаем всех добавленных пользователей из базы данных в виде курсора Cursor.

Вызовом query.moveToFirst() перемещаемся в курсоре к первому объекту, и так как у нас может быть больше одного объекта, то проходим по всему курсору в цикле do...while.

Для получения данных из курсора применяются методы query.getString(0) и query.getInt(1). В скобках в методы передается номер столбца, из которого мы получаем данные. Например, выше мы добавили вначале имя пользователя в виде строки, а затем возраст в виде числа. Значит, нулевым столбцом будет идти строкое значение, которое получаем с помощью метода getString(), а следующим - первым столбцом идет числовое значение, для которого применяется метод getInt().

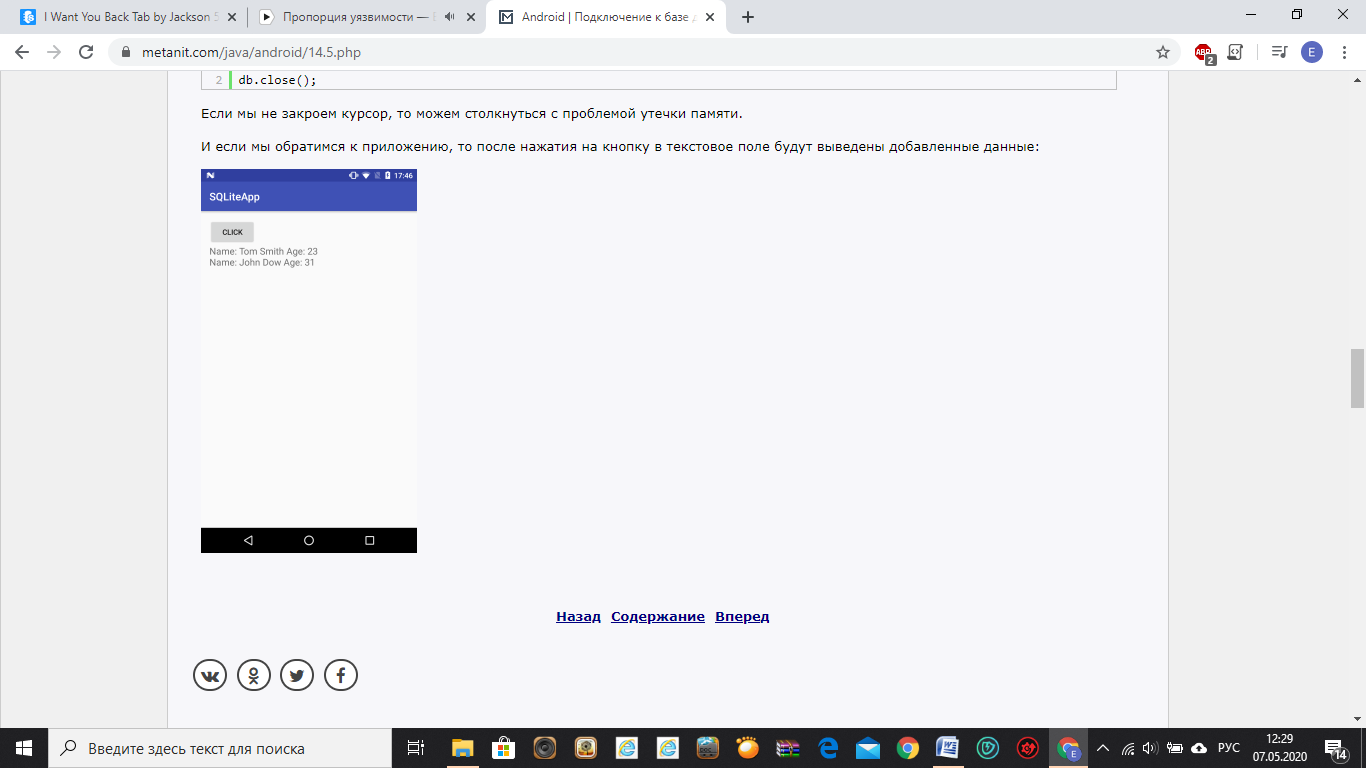
После завершения работы с курсором и базой данных мы закрываем все связанные объекты:

query.close();

db.close();

Если мы не закроем курсор, то можем столкнуться с проблемой утечки памяти.

И если мы обратимся к приложению, то после нажатия на кнопку в текстовое поле будут выведены добавленные данные:



## SimpleCursorAdapter и получение данных

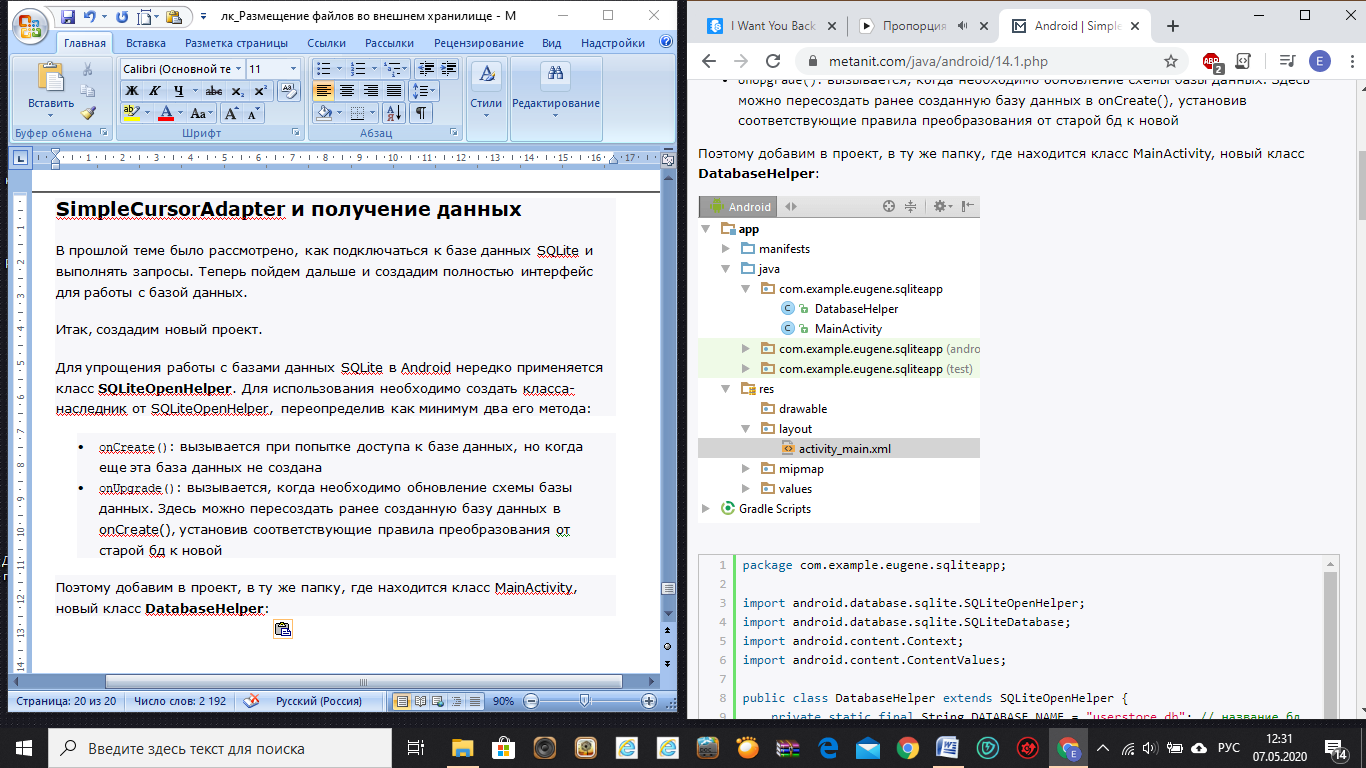
В прошлой теме было рассмотрено, как подключаться к базе данных SQLite и выполнять запросы. Теперь пойдем дальше и создадим полностью интерфейс для работы с базой данных.

Итак, создадим новый проект.

Для упрощения работы с базами данных SQLite в Android нередко применяется класс **SQLiteOpenHelper**. Для использования необходимо создать класса-наследник от SQLiteOpenHelper, переопределив как минимум два его метода:

* onCreate(): вызывается при попытке доступа к базе данных, но когда еще эта база данных не создана
* onUpgrade(): вызывается, когда необходимо обновление схемы базы данных. Здесь можно пересоздать ранее созданную базу данных в onCreate(), установив соответствующие правила преобразования от старой бд к новой

Поэтому добавим в проект, в ту же папку, где находится класс MainActivity, новый класс **DatabaseHelper**:



package com.example.eugene.sqliteapp;

import android.database.sqlite.SQLiteOpenHelper;

import android.database.sqlite.SQLiteDatabase;

import android.content.Context;

import android.content.ContentValues;

public class DatabaseHelper extends SQLiteOpenHelper {

    private static final String DATABASE\_NAME = "userstore.db"; // название бд

    private static final int SCHEMA = 1; // версия базы данных

    static final String TABLE = "users"; // название таблицы в бд

    // названия столбцов

    public static final String COLUMN\_ID = "\_id";

    public static final String COLUMN\_NAME = "name";

    public static final String COLUMN\_YEAR = "year";

    public DatabaseHelper(Context context) {

        super(context, DATABASE\_NAME, null, SCHEMA);

    }

    @Override

    public void onCreate(SQLiteDatabase db) {

        db.execSQL("CREATE TABLE users (" + COLUMN\_ID

                + " INTEGER PRIMARY KEY AUTOINCREMENT," + COLUMN\_NAME

                + " TEXT, " + COLUMN\_YEAR + " INTEGER);");

        // добавление начальных данных

        db.execSQL("INSERT INTO "+ TABLE +" (" + COLUMN\_NAME

                + ", " + COLUMN\_YEAR  + ") VALUES ('Том Смит', 1981);");

    }

    @Override

    public void onUpgrade(SQLiteDatabase db, int oldVersion,  int newVersion) {

        db.execSQL("DROP TABLE IF EXISTS "+TABLE);

        onCreate(db);

    }

}

Если база данных отсутствует или ее версия (которая задается в переменной SCHEMA) выше текущей, то срабатывает метод onCreate().

Для выполнения запросов к базе данных нам потребуется объект **SQLiteDatabase**, который представляет базу данных. Метод onCreate() получает в качестве параметра базу данных приложения.

Для выполнения запросов к SQLite используется метод **execSQL()**. Он принимает sql-выражение CREATE TABLE, которое создает таблицу. Здесь также при необходимости мы можем выполнить и другие запросы, например, добавить какие-либо начальные данные. Так, в данном случае с помощью того же метода и выражения sql INSERT добавляется один объект в таблицу.

В методе onUpgrade() происходит обновление схемы БД. В данном случае для примера использован примитивный поход с удалением предыдущей базы данных с помощью sql-выражения DROP и последующим ее созданием. Но в реальности если вам будет необходимо сохранить данные, этот метод может включать более сложную логику - добавления новых столбцов, удаление ненужных, добавление дополнительных данных и т.д.

Далее определим в файле **activity\_main.xml** следующую разметку:

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

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

    android:orientation="vertical"

    android:layout\_width="match\_parent"

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

    <TextView

        android:id="@+id/header"

        android:layout\_width="match\_parent"

        android:layout\_height="wrap\_content"

        android:textSize="18dp"/>

    <ListView

        android:id="@+id/list"

        android:layout\_width="match\_parent"

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

</LinearLayout>

Здесь определен список ListView, для отображения полученных данных, с заголовком, который будет выводить число полученных объектов.

И изменим код класса **MainActivity** следующим образом:

package com.example.eugene.sqliteapp;

import android.database.Cursor;

import android.database.sqlite.SQLiteDatabase;

import android.support.v7.app.AppCompatActivity;

import android.os.Bundle;

import android.view.View;

import android.widget.AdapterView;

import android.widget.ListView;

import android.widget.SimpleCursorAdapter;

import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    ListView userList;

    TextView header;

    DatabaseHelper databaseHelper;

    SQLiteDatabase db;

    Cursor userCursor;

    SimpleCursorAdapter userAdapter;

    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity\_main);

        header = (TextView)findViewById(R.id.header);

        userList = (ListView)findViewById(R.id.list);

        databaseHelper = new DatabaseHelper(getApplicationContext());

    }

    @Override

    public void onResume() {

        super.onResume();

        // открываем подключение

**db = databaseHelper.getReadableDatabase();**

        //получаем данные из бд в виде курсора

        userCursor =  db.rawQuery("select \* from "+ DatabaseHelper.TABLE, null);

        // определяем, какие столбцы из курсора будут выводиться в ListView

        String[] headers = new String[] {DatabaseHelper.COLUMN\_NAME, DatabaseHelper.COLUMN\_YEAR};

        // создаем адаптер, передаем в него курсор

        userAdapter = new SimpleCursorAdapter(this, android.R.layout.two\_line\_list\_item,

                userCursor, headers, new int[]{android.R.id.text1, android.R.id.text2}, 0);

        header.setText("Найдено элементов: " + String.valueOf(userCursor.getCount()));

        userList.setAdapter(userAdapter);

    }

    @Override

    public void onDestroy(){

        super.onDestroy();

        // Закрываем подключение и курсор

        db.close();

        userCursor.close();

    }

}

В методе onCreate() происходит создание объекта SQLiteOpenHelper. Сама инициализация объектов для работы с базой данных происходит в методе onResume(), который срабатывает после метода onCreate().

Чтобы получить объект базы данных, надо использовать метод getReadableDatabase() (получение базы данных для чтения) или getWritableDatabase(). Так как в данном случае мы будет только считывать данные из бд, то воспользуемся первым методом:

|  |  |
| --- | --- |
|  | db = sqlHelper.getReadableDatabase(); |

### Получение данных и Cursor

Android предоставляет различные способы для осуществления запросов к объекту SQLiteDatabase. В большинстве случаев мы можем применять метод **rawQuery()**, который принимает два параметра: SQL-выражение SELECT и дополнительный параметр, задающий параметры запроса.

После выполнения запроса rawQuery() возвращает объект **Cursor**, который хранит результат выполнения SQL-запроса:

|  |  |
| --- | --- |
|  | userCursor =  db.rawQuery("select \* from "+ DatabaseHelper.TABLE, null); |

Класс Cursor предлагает ряд методов для управления выборкой, в частности:

* getCount(): получает количество извлеченных из базы данных объектов
* Методы moveToFirst() и moveToNext() позволяют переходить к первому и к следующему элементам выборки. Метод isAfterLast() позволяет проверить, достигнут ли конец выборки.
* Методы get\*(columnIndex) (например, getLong(), getString()) позволяют по индексу столбца обратиться к данному столбцу текущей строки

### CursorAdapter

Дополнительно для управления курсором в Android имеется класс CursorAdapter. Он позволяет адаптировать полученный с помощью курсора набор к отображению в списковых элементах наподобие ListView. Как правило, при работе с курсором используется подкласс CursorAdapter - **SimpleCursorAdapter**. Хотя можно использовать и другие адаптеры, типа ArrayAdapter.

userAdapter = new SimpleCursorAdapter(this, android.R.layout.two\_line\_list\_item,

                userCursor, headers, new int[]{android.R.id.text1, android.R.id.text2}, 0);

userList.setAdapter(userAdapter);

Конструктор класса SimpleCursorAdapter принимает шесть параметров:

1. Первым параметром выступает контекст, с которым ассоциируется адаптер, например, текущая activity
2. Второй параметр - ресурс разметки интерфейса, который будет использоваться для отображения результатов выборки
3. Третий параметр - курсор
4. Четвертый параметр - список столбцов из выборки, которые будут отображаться в разметке интерфейса
5. Пятый параметр - элементы внутри ресурса разметки, которые будут отображать значения столбцов из четвертого параметра
6. Шестой параметр - флаги, задающие поведения адаптера

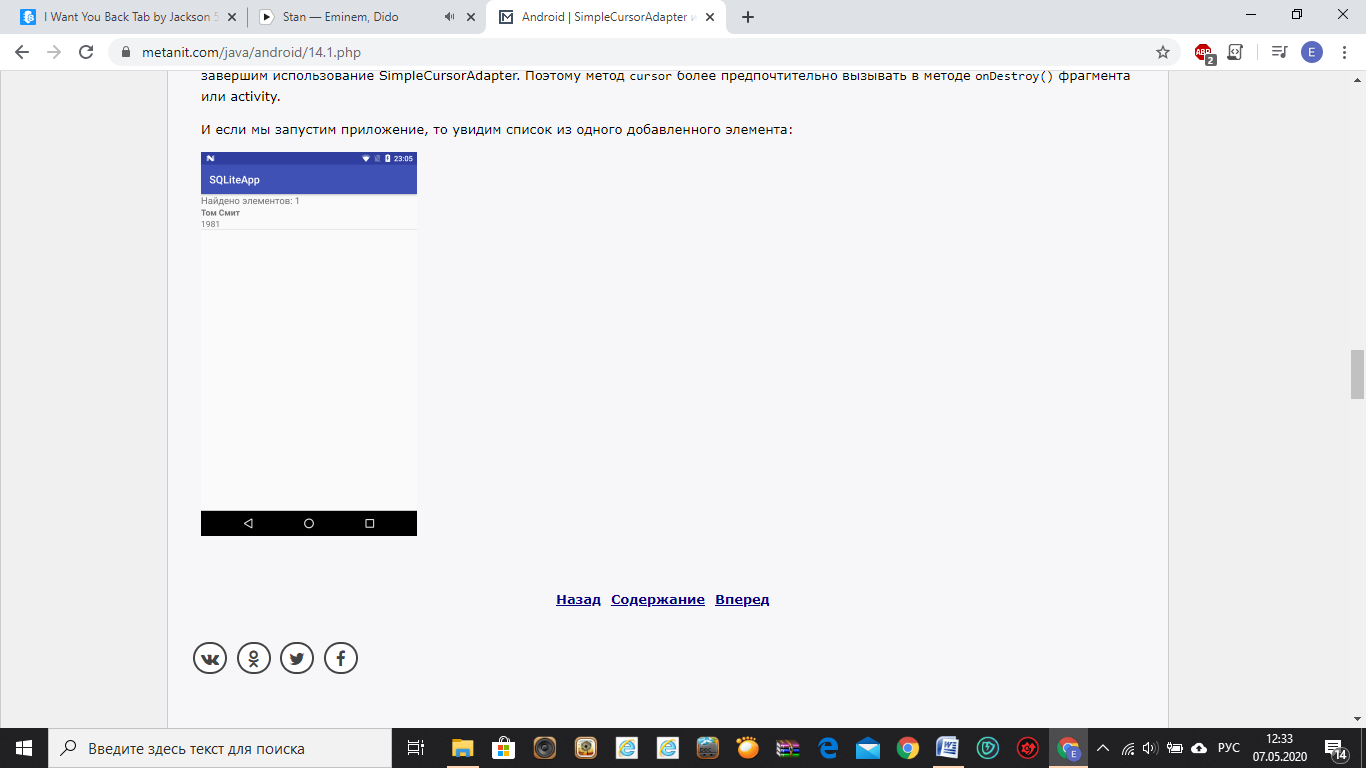
При использовании CursorAdapter и его подклассов следует учитывать, что выборка курсора должна включать целочисленный столбец с названием **\_id**, который должен быть уникальным для каждого элемента выборки. Значение этого столбца при нажатии на элемент списка затем передается в метод обработки onListItemClick(), благодаря чему мы можем по id идентифицировать нажатый элемент.

В данном случае у нас первый столбец как раз называется "\_id".

После завершения работу курсор должен быть закрыт методом close()

И также надо учитывать, что если мы используем курсор в SimpleCursorAdapter, то мы не можем использовать метод close(), пока не завершим использование SimpleCursorAdapter. Поэтому метод cursor более предпочтительно вызывать в методе onDestroy() фрагмента или activity.

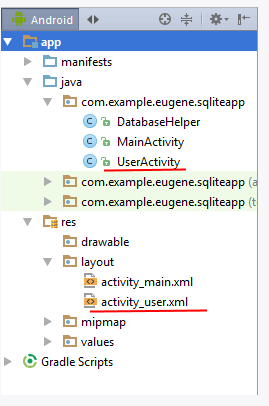
И если мы запустим приложение, то увидим список из одного добавленного элемента:



**Добавление, удаление и обновление данных в SQLite**

Продолжим работу с проектом из прошлой темы, где мы получаем данные. Теперь добавим в него стандартную CRUD-логику (создание, обновление, удаление).

Чтобы не нагромождать форму с главной activity, все остальные действия по работе с данными будут происходить на другом экране. Добавим в проект новый класс activity, который назовем UserActivity:



В файле **activity\_user.xml** определим универсальую форму для добавления/обновления/удаления данных:

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

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

    android:orientation="vertical"

    android:layout\_width="match\_parent"

    android:layout\_height="match\_parent"

    android:padding="16dp">

    <EditText

        android:id="@+id/name"

        android:layout\_width="match\_parent"

        android:layout\_height="wrap\_content"

        android:hint="Введите имя"/>

    <EditText

        android:id="@+id/year"

        android:layout\_width="match\_parent"

        android:layout\_height="wrap\_content"

        android:hint="Введите год рождения"/>

    <LinearLayout

        android:layout\_width="match\_parent"

        android:layout\_height="wrap\_content"

        android:orientation="horizontal">

        <Button

            android:id="@+id/saveButton"

            android:layout\_width="0dp"

            android:layout\_weight="1"

            android:layout\_height="wrap\_content"

            android:text="Сохранить"

            android:onClick="save"/>

        <Button

            android:id="@+id/deleteButton"

            android:layout\_width="0dp"

            android:layout\_weight="1"

            android:layout\_height="wrap\_content"

            android:text="Удалить"

            android:onClick="delete"/>

    </LinearLayout>

</LinearLayout>

И также изменим код **UserActivity**:

package com.example.eugene.sqliteapp;

import android.content.ContentValues;

import android.content.Intent;

import android.database.Cursor;

import android.database.sqlite.SQLiteDatabase;

import android.support.v7.app.AppCompatActivity;

import android.os.Bundle;

import android.view.View;

import android.widget.Button;

import android.widget.EditText;

public class UserActivity extends AppCompatActivity {

    EditText nameBox;

    EditText yearBox;

    Button delButton;

    Button saveButton;

    DatabaseHelper sqlHelper;

    SQLiteDatabase db;

    Cursor userCursor;

    long userId=0;

    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity\_user);

        nameBox = (EditText) findViewById(R.id.name);

        yearBox = (EditText) findViewById(R.id.year);

        delButton = (Button) findViewById(R.id.deleteButton);

        saveButton = (Button) findViewById(R.id.saveButton);

        sqlHelper = new DatabaseHelper(this);

        db = sqlHelper.getWritableDatabase();

        Bundle extras = getIntent().getExtras();

        if (extras != null) {

            userId = extras.getLong("id");

        }

        // если 0, то добавление

        if (userId > 0) {

            // получаем элемент по id из бд

            userCursor = db.rawQuery("select \* from " + DatabaseHelper.TABLE + " where " +

                    DatabaseHelper.COLUMN\_ID + "=?", new String[]{String.valueOf(userId)});

            userCursor.moveToFirst();

            nameBox.setText(userCursor.getString(1));

            yearBox.setText(String.valueOf(userCursor.getInt(2)));

            userCursor.close();

        } else {

            // скрываем кнопку удаления

            delButton.setVisibility(View.GONE);

        }

    }

    public void save(View view){

        ContentValues cv = new ContentValues();

        cv.put(DatabaseHelper.COLUMN\_NAME, nameBox.getText().toString());

        cv.put(DatabaseHelper.COLUMN\_YEAR, Integer.parseInt(yearBox.getText().toString()));

        if (userId > 0) {

            db.update(DatabaseHelper.TABLE, cv, DatabaseHelper.COLUMN\_ID + "=" + String.valueOf(userId), null);

        } else {

            db.insert(DatabaseHelper.TABLE, null, cv);

        }

        goHome();

    }

    public void delete(View view){

        db.delete(DatabaseHelper.TABLE, "\_id = ?", new String[]{String.valueOf(userId)});

        goHome();

    }

    private void goHome(){

        // закрываем подключение

        db.close();

        // переход к главной activity

        Intent intent = new Intent(this, MainActivity.class);

        intent.addFlags(Intent.FLAG\_ACTIVITY\_CLEAR\_TOP | Intent.FLAG\_ACTIVITY\_SINGLE\_TOP);

        startActivity(intent);

    }

}

При обновлении или удалении объекта из списка из главной activity в UserActivity будет передаваться id объекта:

long userId=0;

//.................

Bundle extras = getIntent().getExtras();

if (extras != null) {

    userId = extras.getLong("id");

}

Если из MainActivity не было передано id, то устанавливаем его значение 0, следовательно, у нас будет добавление, а не редактирование/удаление

Если id определен, то получаем по нему из базы данных объект для редактирования/удаления:

if (id < 0) {

    userCursor = db.rawQuery("select \* from " + DatabaseHelper.TABLE + " where " +

            DatabaseHelper.COLUMN\_ID + "=?", new String[]{String.valueOf(id)});

    userCursor.moveToFirst();

    nameBox.setText(userCursor.getString(1));

    yearBox.setText(String.valueOf(userCursor.getInt(2)));

    userCursor.close();

}

Иначе просто скрываем кнопку удаления.

Для выполнения операций по вставке, обновлению и удалению данных SQLiteDatabase имеет методы insert(), update() и delete(). Эти методы вызываются в обработчиках кнопок:

|  |  |
| --- | --- |
|  | db.delete(DatabaseHelper.TABLE, "\_id = ?", new String[]{String.valueOf(id)}); |

В метод delete() передается название таблицы, а также столбец, по которому происходит удаление, и его значение. В качестве критерия можно выбрать несколько столбцов, поэтому третьим параметром идет массив. Знак вопроса ? обозначает параметр, вместо которого подставляется значение из третьего параметра.

### ContentValues

Для добавления или обновления нам надо создать объект **ContentValues**. Данный объект представляет словарь, который содержит набор пар "ключ-значение". Для добавления в этот словарь нового объекта применяется метод **put**. Первый параметр метода - это ключ, а второй - значение, например:

|  |  |
| --- | --- |
|  | ContentValues cv = new ContentValues();  cv.put("NAME", "Tom");  cv.put("YEAR", 30); |

В качестве значений в метод put можно передавать строки, целые числа, числа с плавающей точкой

В данном же случае добавляются введенные в текстовое поля значения:

|  |  |
| --- | --- |
|  | ContentValues cv = new ContentValues();  cv.put(DatabaseHelper.COLUMN\_NAME, nameBox.getText().toString());  cv.put(DatabaseHelper.COLUMN\_YEAR, Integer.parseInt(yearBox.getText().toString())); |

При обновлении в метод update() передается название таблицы, объект ContentValues и критерий, по которому происходит обновление (в данном случае столбец id):

|  |  |
| --- | --- |
|  | db.update(DatabaseHelper.TABLE, cv, DatabaseHelper.COLUMN\_ID + "="  + String.valueOf(id), null); |

Метод insert() принимает название таблицы, объект ContentValues с добавляемыми значениями. Второй параметр является необязательным: он передает столбец, в который надо добавить значение NULL:

|  |  |
| --- | --- |
|  | db.insert(DatabaseHelper.TABLE, null, cv); |

Вместо этих методов, как в прошлой теме, можно использовать метод execSQL() с точным указанием выполняемого sql-выражения. В то же время методы delete/insert/update имеют преимущество - они возвращают id измененной записи, по которому мы можем узнать об успешности операции, или -1 в случае неудачной операции:

|  |  |
| --- | --- |
|  | long result = db.insert(DatabaseHelper.TABLE, null, cv);  if(result>0){        // действия  } |

После каждой операции выполняется метод goHome(), который возвращает на главную activity.

После этого нам надо исправить код MainActivity, чтобы она инициировала выполнение кода в UserActivity. Для этого изменим код **activity\_main.xml**:

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

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

    android:orientation="vertical"

    android:layout\_width="match\_parent"

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

    <Button

        android:layout\_width="match\_parent"

        android:layout\_height="wrap\_content"

        android:text="Добавить"

        android:onClick="add"

        android:textSize="18dp"/>

    <ListView

        android:id="@+id/list"

        android:layout\_width="match\_parent"

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

</LinearLayout>

В данном случае была добавлена кнопка для вызова UserActivity.

И также изменим код класса **MainActivity**:

package com.example.eugene.sqliteapp;

import android.content.Intent;

import android.database.Cursor;

import android.database.sqlite.SQLiteDatabase;

import android.support.v7.app.AppCompatActivity;

import android.os.Bundle;

import android.view.View;

import android.widget.AdapterView;

import android.widget.ListView;

import android.widget.SimpleCursorAdapter;

public class MainActivity extends AppCompatActivity {

    ListView userList;

    DatabaseHelper databaseHelper;

    SQLiteDatabase db;

    Cursor userCursor;

    SimpleCursorAdapter userAdapter;

    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity\_main);

        userList = (ListView)findViewById(R.id.list);

        userList.setOnItemClickListener(new AdapterView.OnItemClickListener() {

            @Override

            public void onItemClick(AdapterView<?> parent, View view, int position, long id) {

                Intent intent = new Intent(getApplicationContext(), UserActivity.class);

                intent.putExtra("id", id);

                startActivity(intent);

            }

        });

        databaseHelper = new DatabaseHelper(getApplicationContext());

    }

    @Override

    public void onResume() {

        super.onResume();

        // открываем подключение

        db = databaseHelper.getReadableDatabase();

        //получаем данные из бд в виде курсора

        userCursor =  db.rawQuery("select \* from "+ DatabaseHelper.TABLE, null);

        // определяем, какие столбцы из курсора будут выводиться в ListView

        String[] headers = new String[] {DatabaseHelper.COLUMN\_NAME, DatabaseHelper.COLUMN\_YEAR};

        // создаем адаптер, передаем в него курсор

        userAdapter = new SimpleCursorAdapter(this, android.R.layout.two\_line\_list\_item,

                userCursor, headers, new int[]{android.R.id.text1, android.R.id.text2}, 0);

        userList.setAdapter(userAdapter);

    }

    // по нажатию на кнопку запускаем UserActivity для добавления данных

    public void add(View view){

        Intent intent = new Intent(this, UserActivity.class);

        startActivity(intent);

    }

    @Override

    public void onDestroy(){

        super.onDestroy();

        // Закрываем подключение и курсор

        db.close();

        userCursor.close();

    }

}

При нажатии на кнопку запускается UserActivity, при этом не передается никакого id, то есть в UserActivity id будет равен нулю, значит будет идти добавление данных:

public void add(View view){

    Intent intent = new Intent(this, UserActivity.class);

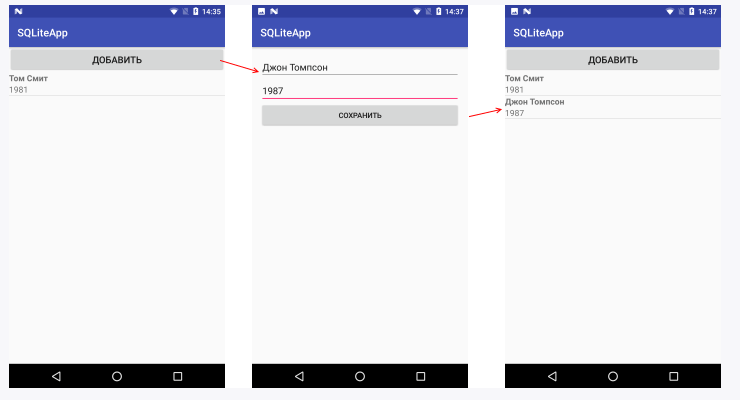
    startActivity(intent);

}

Другую ситуацию представляет обработчик нажатия на элемент списка - при нажатии также будет запускаться UserActivity, но теперь будет передаваться id выбранной записи:

|  |  |
| --- | --- |
|  | public void onItemClick(AdapterView<?> parent, View view, int position, long id)  {      Intent intent = new Intent(getApplicationContext(), UserActivity.class);      intent.putExtra("id", id);      startActivity(intent);  } |

Запустим приложение и нажмем на кнопку, которая должен перенаправлять на UserActivity:



При нажатии в MainActivity на элемент списка этот элемент попадет на UserActivity, где его можно будет удалить или подредактировать:

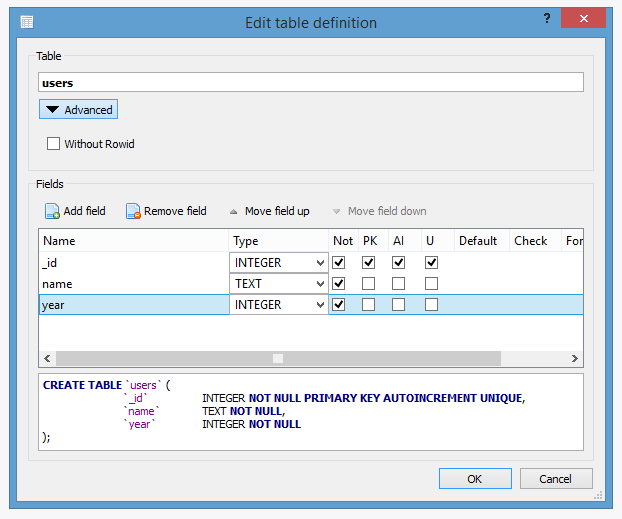


**Использование существующей БД SQLite**

Кроме создания новой базы данных мы также можем использовать уже существующую. Это может быть более предпочтительно, так как в этом случае база данных приложения уже будет содержать всю необходимую информацию.

Возьмем проект, созданный в предыдущих темах. Для начала создадим базу данных SQLite. В этом нам может помочь такой инструмент как **Sqlitebrowser**. Он бесплатный и доступен для различных операционных систем по адресу <http://sqlitebrowser.org/>. Хотя можно использовать и другие способы для создания начальной БД.

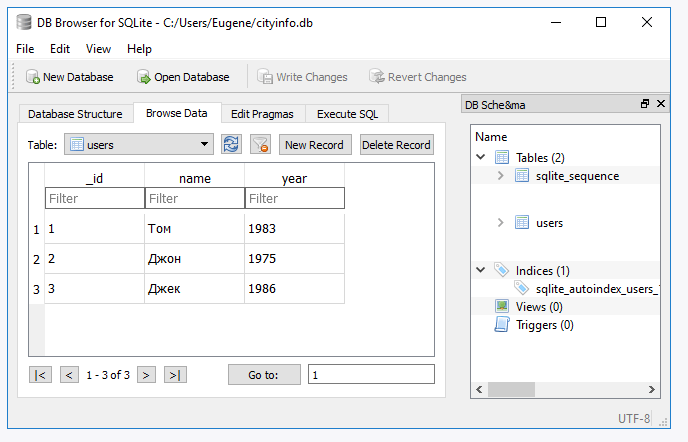
Sqlitebrowser представляет графический интерфейс для создания базы данных и определения в ней всех необходимых таблиц:

****

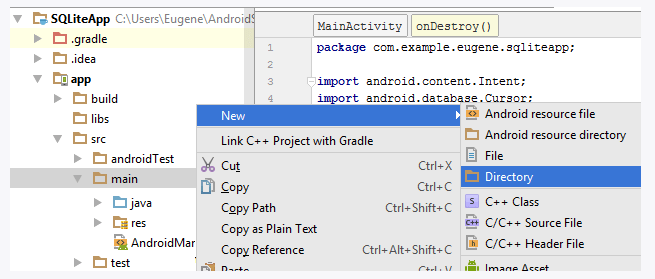
Как видно на скриншоте, я определяю таблицу users с тремя полями: \_id, name, age. Общая команда на создание таблицы будет следующей:

|  |  |
| --- | --- |
|  | CREATE TABLE `users` (      `\_id`   INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,      `name`  TEXT NOT NULL,      `year`  INTEGER NOT NULL  ); |

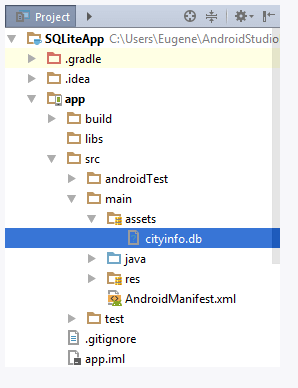
Там же в программе добавим несколько элементов в созданную таблицу:



После создания таблицы добавим в проект в Android Studio папку **assets**, а в папку assets - только что созданную базу данных. Для этого перейдем к полному опеределению проекта, нажмем на папку main правой кнопкой мыши и в меню выберем **New -> Directory**:



Назовем добавляемую папку assets и затем скопируем в нее нашу базу данных:



В моем случае база данных называется "cityinfo.db".

Изменим код DatabaseHelper следующим образом:

package com.example.eugene.sqliteapp;

import android.database.SQLException;

import android.database.sqlite.SQLiteOpenHelper;

import android.database.sqlite.SQLiteDatabase;

import android.content.Context;

import android.util.Log;

import java.io.File;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.InputStream;

import java.io.OutputStream;

class DatabaseHelper extends SQLiteOpenHelper {

    private static String DB\_PATH; // полный путь к базе данных

    private static String DB\_NAME = "cityinfo.db";

    private static final int SCHEMA = 1; // версия базы данных

    static final String TABLE = "users"; // название таблицы в бд

    // названия столбцов

    static final String COLUMN\_ID = "\_id";

    static final String COLUMN\_NAME = "name";

    static final String COLUMN\_YEAR = "year";

    private Context myContext;

    DatabaseHelper(Context context) {

        super(context, DB\_NAME, null, SCHEMA);

        this.myContext=context;

        DB\_PATH =context.getFilesDir().getPath() + DB\_NAME;

    }

    @Override

    public void onCreate(SQLiteDatabase db) {

    }

    @Override

    public void onUpgrade(SQLiteDatabase db, int oldVersion,  int newVersion) {

    }

    void create\_db(){

        InputStream myInput = null;

        OutputStream myOutput = null;

        try {

            File file = new File(DB\_PATH);

            if (!file.exists()) {

                this.getReadableDatabase();

                //получаем локальную бд как поток

                myInput = myContext.getAssets().open(DB\_NAME);

                // Путь к новой бд

                String outFileName = DB\_PATH;

                // Открываем пустую бд

                myOutput = new FileOutputStream(outFileName);

                // побайтово копируем данные

                byte[] buffer = new byte[1024];

                int length;

                while ((length = myInput.read(buffer)) > 0) {

                    myOutput.write(buffer, 0, length);

                }

                myOutput.flush();

                myOutput.close();

                myInput.close();

            }

        }

        catch(IOException ex){

            Log.d("DatabaseHelper", ex.getMessage());

        }

    }

    public SQLiteDatabase open()throws SQLException {

        return SQLiteDatabase.openDatabase(DB\_PATH, null, SQLiteDatabase.OPEN\_READWRITE);

    }

}

По умолчанию база данных будет размещаться во внешнем хранилище, выделяемом для приложения в папке **/data/data/[название\_пакета]/databases/**, и чтобы получить полный путь к базе данных в конструкторе используется выражение:

|  |  |
| --- | --- |
|  | DB\_PATH =context.getFilesDir().getPath() + DB\_NAME; |

Метод onCreate() нам не нужен, так как нам не требуется создание встроенной базы данных. Зато здесь определен дополнительный метод create\_db(), цель которого копирование базы данных из папки assets в то место, которое указано в переменной DB\_PATH.

Кроме этого здесь также определен метод открытия базы данных open() с помощью метода SQLiteDatabase.openDatabase()

Новый способ организации подключения изменит использование DatabaseHelper в activity. Так, обновим класс **MainActivity**:

package com.example.eugene.sqliteapp;

import android.content.Intent;

import android.database.Cursor;

import android.database.sqlite.SQLiteDatabase;

import android.support.v7.app.AppCompatActivity;

import android.os.Bundle;

import android.view.View;

import android.widget.AdapterView;

import android.widget.ListView;

import android.widget.SimpleCursorAdapter;

public class MainActivity extends AppCompatActivity {

    ListView userList;

    DatabaseHelper databaseHelper;

    SQLiteDatabase db;

    Cursor userCursor;

    SimpleCursorAdapter userAdapter;

    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity\_main);

        userList = (ListView)findViewById(R.id.list);

        userList.setOnItemClickListener(new AdapterView.OnItemClickListener() {

            @Override

            public void onItemClick(AdapterView<?> parent, View view, int position, long id) {

                Intent intent = new Intent(getApplicationContext(), UserActivity.class);

                intent.putExtra("id", id);

                startActivity(intent);

            }

        });

        databaseHelper = new DatabaseHelper(getApplicationContext());

        // создаем базу данных

        databaseHelper.create\_db();

    }

    @Override

    public void onResume() {

        super.onResume();

        // открываем подключение

        db = databaseHelper.open();

        //получаем данные из бд в виде курсора

        userCursor =  db.rawQuery("select \* from "+ DatabaseHelper.TABLE, null);

        // определяем, какие столбцы из курсора будут выводиться в ListView

        String[] headers = new String[] {DatabaseHelper.COLUMN\_NAME, DatabaseHelper.COLUMN\_YEAR};

        // создаем адаптер, передаем в него курсор

        userAdapter = new SimpleCursorAdapter(this, android.R.layout.two\_line\_list\_item,

                userCursor, headers, new int[]{android.R.id.text1, android.R.id.text2}, 0);

        userList.setAdapter(userAdapter);

    }

    // по нажатию на кнопку запускаем UserActivity для добавления данных

    public void add(View view){

        Intent intent = new Intent(this, UserActivity.class);

        startActivity(intent);

    }

    @Override

    public void onDestroy(){

        super.onDestroy();

        // Закрываем подключение и курсор

        db.close();

        userCursor.close();

    }

}

И также изменим класс **UserActivity**:

package com.example.eugene.sqliteapp;

import android.content.ContentValues;

import android.content.Intent;

import android.database.Cursor;

import android.database.sqlite.SQLiteDatabase;

import android.support.v7.app.AppCompatActivity;

import android.os.Bundle;

import android.view.View;

import android.widget.Button;

import android.widget.EditText;

public class UserActivity extends AppCompatActivity {

    EditText nameBox;

    EditText yearBox;

    Button delButton;

    Button saveButton;

    DatabaseHelper sqlHelper;

    SQLiteDatabase db;

    Cursor userCursor;

    long userId=0;

    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity\_user);

        nameBox = (EditText) findViewById(R.id.name);

        yearBox = (EditText) findViewById(R.id.year);

        delButton = (Button) findViewById(R.id.deleteButton);

        saveButton = (Button) findViewById(R.id.saveButton);

        sqlHelper = new DatabaseHelper(this);

        db = sqlHelper.open();

        Bundle extras = getIntent().getExtras();

        if (extras != null) {

            userId = extras.getLong("id");

        }

        // если 0, то добавление

        if (userId > 0) {

            // получаем элемент по id из бд

            userCursor = db.rawQuery("select \* from " + DatabaseHelper.TABLE + " where " +

                    DatabaseHelper.COLUMN\_ID + "=?", new String[]{String.valueOf(userId)});

            userCursor.moveToFirst();

            nameBox.setText(userCursor.getString(1));

            yearBox.setText(String.valueOf(userCursor.getInt(2)));

            userCursor.close();

        } else {

            // скрываем кнопку удаления

            delButton.setVisibility(View.GONE);

        }

    }

    public void save(View view){

        ContentValues cv = new ContentValues();

        cv.put(DatabaseHelper.COLUMN\_NAME, nameBox.getText().toString());

        cv.put(DatabaseHelper.COLUMN\_YEAR, Integer.parseInt(yearBox.getText().toString()));

        if (userId > 0) {

            db.update(DatabaseHelper.TABLE, cv, DatabaseHelper.COLUMN\_ID + "=" + String.valueOf(userId), null);

        } else {

            db.insert(DatabaseHelper.TABLE, null, cv);

        }

        goHome();

    }

    public void delete(View view){

        db.delete(DatabaseHelper.TABLE, "\_id = ?", new String[]{String.valueOf(userId)});

        goHome();

    }

    private void goHome(){

        // закрываем подключение

        db.close();

        // переход к главной activity

        Intent intent = new Intent(this, MainActivity.class);

        intent.addFlags(Intent.FLAG\_ACTIVITY\_CLEAR\_TOP | Intent.FLAG\_ACTIVITY\_SINGLE\_TOP);

        startActivity(intent);

    }

}

Вся остальная работа с данными будет той же, чтобы и в прошлых темах:

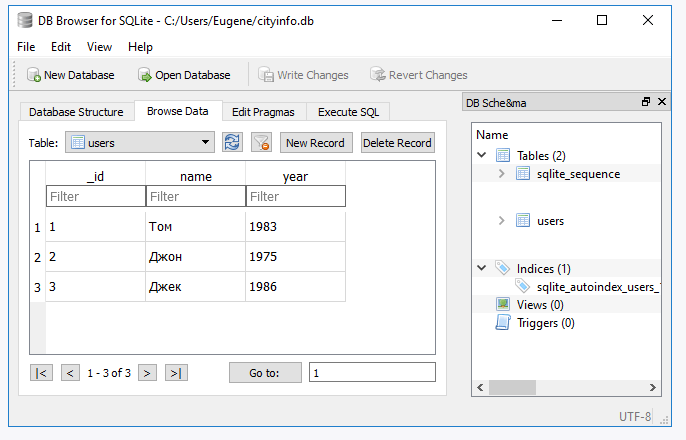


**Динамический поиск по базе данных SQLite**

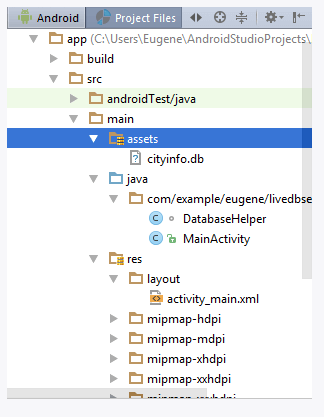
Рассмотрим, как мы можем создать в приложении на Android динамический поиск по базе данных SQLite.

Итак, создадим новый проект с пустой MainActivity. Для этого проекта возьмем базу данных из прошлой темы. Данная база данных называется *cityinfo* и имеет одну таблицу users с тремя полями \_id, name, age:

|  |  |
| --- | --- |
|  | CREATE TABLE `users` (      `\_id`   INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,      `name`  TEXT NOT NULL,      `year`  INTEGER NOT NULL  ); |



И также добавим в проект в Android Studio папку **assets**, а в папку assets - только что созданную базу данных:



В данном случае база данных называется "cityinfo.db".

Как показано выше на скриншоте, добавив в проект в одну папку с MainActivity новый класс DatabaseHelper:

package com.example.eugene.livedbsearchapp;

import android.database.SQLException;

import android.database.sqlite.SQLiteOpenHelper;

import android.database.sqlite.SQLiteDatabase;

import android.content.Context;

import android.util.Log;

import java.io.File;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.InputStream;

import java.io.OutputStream;

class DatabaseHelper extends SQLiteOpenHelper {

    private static String DB\_PATH; // полный путь к базе данных

    private static String DB\_NAME = "cityinfo.db";

    private static final int SCHEMA = 1; // версия базы данных

    static final String TABLE = "users"; // название таблицы в бд

    // названия столбцов

    static final String COLUMN\_ID = "\_id";

    static final String COLUMN\_NAME = "name";

    static final String COLUMN\_YEAR = "year";

    private Context myContext;

    DatabaseHelper(Context context) {

        super(context, DB\_NAME, null, SCHEMA);

        this.myContext=context;

        DB\_PATH =context.getFilesDir().getPath() + DB\_NAME;

    }

    @Override

    public void onCreate(SQLiteDatabase db) {

    }

    @Override

    public void onUpgrade(SQLiteDatabase db, int oldVersion,  int newVersion) {

    }

    void create\_db(){

        InputStream myInput = null;

        OutputStream myOutput = null;

        try {

            File file = new File(DB\_PATH);

            if (!file.exists()) {

                this.getReadableDatabase();

                //получаем локальную бд как поток

                myInput = myContext.getAssets().open(DB\_NAME);

                // Путь к новой бд

                String outFileName = DB\_PATH;

                // Открываем пустую бд

                myOutput = new FileOutputStream(outFileName);

                // побайтово копируем данные

                byte[] buffer = new byte[1024];

                int length;

                while ((length = myInput.read(buffer)) > 0) {

                    myOutput.write(buffer, 0, length);

                }

                myOutput.flush();

                myOutput.close();

                myInput.close();

            }

        }

        catch(IOException ex){

            Log.d("DatabaseHelper", ex.getMessage());

        }

    }

    SQLiteDatabase open()throws SQLException {

        return SQLiteDatabase.openDatabase(DB\_PATH, null, SQLiteDatabase.OPEN\_READWRITE);

    }

}

Перейдем к файлу **activity\_main.xml**, который определяет визуальный интерфейс, и изменим его следующим образом:

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

    android:layout\_width="match\_parent"

    android:layout\_height="match\_parent"

    android:orientation="vertical">

    <EditText android:id="@+id/userFilter"

        android:layout\_width="match\_parent"

        android:layout\_height="wrap\_content"

        android:hint="Поиск" />

    <ListView

        android:id="@+id/userList"

        android:layout\_width="match\_parent"

        android:layout\_height="wrap\_content">

    </ListView>

</LinearLayout>

Итак, у нас будет элемент ListView для отображения списка и текстовое поле для фильтрации.

Теперь изменим код **MainActivity**:

package com.example.eugene.livedbsearchapp;

import android.database.Cursor;

import android.database.SQLException;

import android.database.sqlite.SQLiteDatabase;

import android.support.v7.app.AppCompatActivity;

import android.os.Bundle;

import android.text.Editable;

import android.text.TextWatcher;

import android.widget.EditText;

import android.widget.FilterQueryProvider;

import android.widget.ListView;

import android.widget.SimpleCursorAdapter;

public class MainActivity extends AppCompatActivity {

    DatabaseHelper sqlHelper;

    SQLiteDatabase db;

    Cursor userCursor;

    SimpleCursorAdapter userAdapter;

    ListView userList;

    EditText userFilter;

    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity\_main);

        userList = (ListView)findViewById(R.id.userList);

        userFilter = (EditText)findViewById(R.id.userFilter);

        sqlHelper = new DatabaseHelper(getApplicationContext());

        // создаем базу данных

        sqlHelper.create\_db();

    }

    @Override

    public void onResume() {

        super.onResume();

        try {

            db = sqlHelper.open();

            userCursor = db.rawQuery("select \* from " + DatabaseHelper.TABLE, null);

            String[] headers = new String[]{DatabaseHelper.COLUMN\_NAME, DatabaseHelper.COLUMN\_YEAR};

            userAdapter = new SimpleCursorAdapter(this, android.R.layout.two\_line\_list\_item,

                    userCursor, headers, new int[]{android.R.id.text1, android.R.id.text2}, 0);

            // если в текстовом поле есть текст, выполняем фильтрацию

            // данная проверка нужна при переходе от одной ориентации экрана к другой

            if(!userFilter.getText().toString().isEmpty())

                userAdapter.getFilter().filter(userFilter.getText().toString());

            // установка слушателя изменения текста

            userFilter.addTextChangedListener(new TextWatcher() {

                public void afterTextChanged(Editable s) { }

                public void beforeTextChanged(CharSequence s, int start, int count, int after) { }

                // при изменении текста выполняем фильтрацию

                public void onTextChanged(CharSequence s, int start, int before, int count) {

                    userAdapter.getFilter().filter(s.toString());

                }

            });

            // устанавливаем провайдер фильтрации

            userAdapter.setFilterQueryProvider(new FilterQueryProvider() {

                @Override

                public Cursor runQuery(CharSequence constraint) {

                    if (constraint == null || constraint.length() == 0) {

                        return db.rawQuery("select \* from " + DatabaseHelper.TABLE, null);

                    }

                    else {

                        return db.rawQuery("select \* from " + DatabaseHelper.TABLE + " where " +

                                DatabaseHelper.COLUMN\_NAME + " like ?", new String[]{"%" + constraint.toString() + "%"});

                    }

                }

            });

            userList.setAdapter(userAdapter);

        }

        catch (SQLException ex){}

    }

    @Override

    public void onDestroy(){

        super.onDestroy();

        // Закрываем подключение и курсор

        db.close();

        userCursor.close();

    }

}

Прежде всего надо отметить, что для фильтрации данных в адаптере, нам надо получить фильтр адаптера, а у этого фильтра выполнить метод filter():

|  |  |
| --- | --- |
|  | userAdapter.getFilter().filter(s.toString()); |

В этот метод filter() передается ключ поиска.

Для текстового поля мы можем отслеживать изменения содержимого с помощью слушателя:

userFilter.addTextChangedListener(new TextWatcher() {

    public void afterTextChanged(Editable s) {

    }

    public void beforeTextChanged(CharSequence s, int start, int count, int after) {

    }

    // при изменении текста выполняем фильтрацию

    public void onTextChanged(CharSequence s, int start, int before, int count) {

        userAdapter.getFilter().filter(s.toString());

    }

});

В слушателе TextWatcher в методе onTextChanged как раз и вызывается метод filter(), в который передется введенная пользователем в текстовое поле последовательность символов.

Сам вызызов метода filter() мало на что влияет. Нам надо еще определить провайдер фильтрации адаптера, которые и будет инкапсулировать реальную логику фильтрации:

userAdapter.setFilterQueryProvider(new FilterQueryProvider() {

    @Override

    public Cursor runQuery(CharSequence constraint) {

        if (constraint == null || constraint.length() == 0) {

            return db.rawQuery("select \* from " + DatabaseHelper.TABLE, null);

        }

        else {

            return db.rawQuery("select \* from " + DatabaseHelper.TABLE + " where " +

                DatabaseHelper.COLUMN\_NAME + " like ?", new String[]{"%" + constraint.toString() + "%"});

            }

    }

});

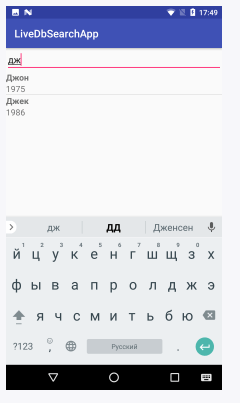
Сущность этого провайдера заключается в выполнении SQL-выражений к бд, а именно конструкций "select from" и "select from where like". Данные простейшие выражения выполняют регистрозависимую фильтрацию. В результате адаптаре получает отфильтрованные данные.

Следует также отметить следующий код:

|  |  |
| --- | --- |
|  | if(!userFilter.getText().toString().isEmpty())      userAdapter.getFilter().filter(userFilter.getText().toString()); |

Данный код нам нужен при смене ориентации (например, с портретной на альбомную). И если ориентация устройства изменена, но в текстовом поле все же есть некоторые текст-фильтр, то выполняется фильтрация. Иначе бы она не выполнялась.

И после запуска мы сможем насладиться фильтрацией данных:



# Перелистывание страниц и ViewPager

## ViewPager и разделение приложения на страницы

Нередко можно встретить приложения, которые реализуют систему перелистывания, а само приложение предстает в виде набора страниц, которые можно пролистывать влево и вправо. В приложении Android для создания подобного эффекта можно использовать элемент **ViewPager**. Для создания эффекта страниц он использует фрагменты.

Итак, создадим новый проект. Добавим в папку *res/layout* файл разметки для фрагмента, который будет представлять страницу. Назовем его **fragment\_page.xml** и определим в нем следующий код:

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

    android:layout\_width="match\_parent"

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

    <TextView

        android:id="@+id/displayText"

        android:layout\_width="wrap\_content"

        android:layout\_height="wrap\_content"

        android:textSize="22sp"

        android:layout\_gravity="center" />

</FrameLayout>

Фрагмент будет отображать текстовое поле с номером страницы.

Теперь добавим в проект сам класс фрагмента. Назовем его **PageFragment**:

package com.example.eugene.viewpagerapp;

import android.os.Bundle;

import android.support.v4.app.Fragment;

import android.view.LayoutInflater;

import android.view.View;

import android.view.ViewGroup;

import android.widget.TextView;

public class PageFragment extends Fragment {

    private int pageNumber;

    public static PageFragment newInstance(int page) {

        PageFragment fragment = new PageFragment();

        Bundle args=new Bundle();

        args.putInt("num", page);

        fragment.setArguments(args);

        return fragment;

    }

    public PageFragment() {

    }

    @Override

    public void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        pageNumber = getArguments() != null ? getArguments().getInt("num") : 1;

    }

    @Override

    public View onCreateView(LayoutInflater inflater, ViewGroup container,

                             Bundle savedInstanceState) {

        View result=inflater.inflate(R.layout.fragment\_page, container, false);

        TextView pageHeader=(TextView)result.findViewById(R.id.displayText);

        String header = String.format("Фрагмент %d", pageNumber+1);

        pageHeader.setText(header);

        return result;

    }

}

Переменная pageNumber указывает на номер текущей страницы. Номер страницы будет передаваться извне через фабричный метод newInstance(). Передача номера происходит путем добавления значения в аргумент "num"

Затем при создании фрагмента в методе onCreate() этот номер будет извлекаться из аргумента "num" (если аргументы определены):

|  |  |
| --- | --- |
|  | pageNumber = getArguments() != null ? getArguments().getInt("num") : 1; |

В методе onCreateView() полученный номер страницы будет отображаться в текстовом поле.

Сам по себе фрагмент еще не создает функциональность постраничной навигации. Для этого нам нужен один из классов **PagerAdapter**. Android SDK содержит две встроенных реализации PagerAdapter: классы **FragmentPagerAdapter** и **FragmentStatePagerAdapter**.

FragmentPagerAdapter хранит все количество используемых фрагментов в памяти и поэтому подходит, если в приложении небольшое количество страниц. А класс FragmentStatePagerAdapter хранит в памяти только текущую, предыдущую и следующую страницу (при их наличии). За счет этого он экономнее расходует память, но в то же время работает чуть медленнее. В данном случае воспользуемся классом FragmentPagerAdapter.

Однако оба класса являются абстрактными, поэтому напрямую мы их использовать не можем, и нам нужно создать класс-наследник. Для этого добавим в проект новый класс, который назовем **MyAdapter**:

package com.example.eugene.viewpagerapp;

import android.support.v4.app.Fragment;

import android.support.v4.app.FragmentManager;

import android.support.v4.app.FragmentPagerAdapter;

public class MyAdapter extends FragmentPagerAdapter {

    public MyAdapter(FragmentManager mgr) {

        super(mgr);

    }

    @Override

    public int getCount() {

        return(10);

    }

    @Override

    public Fragment getItem(int position) {

        return(PageFragment.newInstance(position));

    }

}

Класс FragmentPagerAdapter определяет два метода:

* getCount(): возвращает количество страниц, которые будут в ViewPager (в нашем случае 10)
* getItem(int position): по номеру страницы, передаваемому в качестве параметра position, возвращает объект фрагмента

В завершении установим в файле **activity\_main.xml** элемент ViewPager:

|  |
| --- |
| <android.support.v4.view.ViewPager      xmlns:android="<http://schemas.android.com/apk/res/android>"      android:id="@+id/pager"      android:layout\_width="match\_parent"      android:layout\_height="match\_parent">    </android.support.v4.view.ViewPager> |

И также изменим код **MainActivity**:

package com.example.eugene.viewpagerapp;

import android.support.v7.app.ActionBarActivity;

import android.os.Bundle;

import android.view.Menu;

import android.view.MenuItem;

import android.support.v4.view.ViewPager;

public class MainActivity extends ActionBarActivity {

    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity\_main);

        ViewPager pager=(ViewPager)findViewById(R.id.pager);

        pager.setAdapter(new MyAdapter(getSupportFragmentManager()));

    }

}

Чтобы перелистывание заработало, установим для ViewPager адаптер MyAdapter.

И запустив проект, мы сможем с помощью перелистывания перемещаться по страницам:

