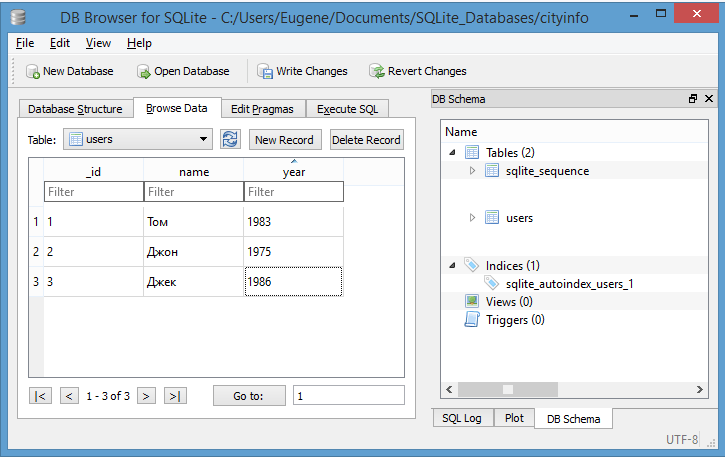
**Динамический поиск по базе данных SQLite**

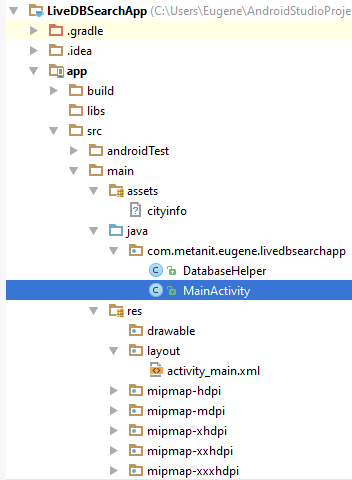
Рассмотрим, как мы можем создать в приложении на Android динамический поиск по базе данных SQLite.

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:

|  |  |
| --- | --- |
| 1  2  3  4  5 | 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 моем случае база данных называется "cityinfo".

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

|  |  |
| --- | --- |
| 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 | package com.metanit.eugene.livedbsearchapp;    import android.database.sqlite.SQLiteOpenHelper;  import android.database.sqlite.SQLiteDatabase;  import android.content.Context;  import java.io.IOException;  import java.io.InputStream;  import java.io.OutputStream;  import java.io.FileOutputStream;  import java.io.File;  import java.sql.SQLException;    public class DatabaseHelper extends SQLiteOpenHelper {        private static String DB\_PATH = "/data/data/com.metanit.eugene.livedbsearchapp/databases/";      private static String DB\_NAME = "cityinfo";      private static final int SCHEMA = 1; // версия базы данных      static final String TABLE = "users";        public static final String COLUMN\_ID = "\_id";      public static final String COLUMN\_NAME = "name";      public static final String COLUMN\_YEAR = "year";      public SQLiteDatabase database;      private Context myContext;        public DatabaseHelper(Context context) {          super(context, DB\_NAME, null, SCHEMA);          this.myContext=context;      }        @Override      public void onCreate(SQLiteDatabase db) {      }      @Override      public void onUpgrade(SQLiteDatabase db, int oldVersion,  int newVersion) {        }        public void create\_db(){          InputStream myInput = null;          OutputStream myOutput = null;          try {              File file = new File(DB\_PATH + DB\_NAME);              if (!file.exists()) {                  this.getReadableDatabase();                  //получаем локальную бд как поток                  myInput = myContext.getAssets().open(DB\_NAME);                  // Путь к новой бд                  String outFileName = DB\_PATH + DB\_NAME;                    // 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){            }      }      public void open() throws SQLException {          String path = DB\_PATH + DB\_NAME;          database = SQLiteDatabase.openDatabase(path, null,                      SQLiteDatabase.OPEN\_READWRITE);        }        @Override      public synchronized void close() {          if (database != null) {              database.close();          }          super.close();      }  } |

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

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | <LinearLayout xmlns:android="<http://schemas.android.com/apk/res/android>"      android:layout\_width="match\_parent"      android:layout\_height="match\_parent" android:orientation="vertical"      android:focusable="true"      android:focusableInTouchMode="true"      >      <EditText android:id="@+id/userFilter" android:layout\_width="match\_parent"          android:layout\_height="wrap\_content" android:ems="10"          android:hint="Поиск"          />        <ListView          android:id="@+id/userList"          android:layout\_width="match\_parent"          android:layout\_height="wrap\_content"          android:focusable="true"          android:focusableInTouchMode="true">      </ListView>    </LinearLayout> |

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

Теперь изменим код MainActivity:

|  |  |
| --- | --- |
| 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 | package com.metanit.eugene.livedbsearchapp;    import android.os.Bundle;  import android.support.v7.app.AppCompatActivity;  import android.text.Editable;  import android.text.TextWatcher;  import android.widget.EditText;  import android.widget.FilterQueryProvider;  import android.widget.ListView;  import android.database.Cursor;  import android.widget.SimpleCursorAdapter;  import java.sql.SQLException;    public class MainActivity extends AppCompatActivity {        ListView userList;      EditText userFilter;      DatabaseHelper sqlHelper;      Cursor userCursor;      SimpleCursorAdapter userAdapter;        @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 {              sqlHelper.open();              userCursor = sqlHelper.database.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 sqlHelper.database.rawQuery("select \* from " + DatabaseHelper.TABLE, null);                      }                      else {                          return sqlHelper.database.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 подключения          sqlHelper.database.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():

|  |  |
| --- | --- |
| 1 | userAdapter.getFilter().filter(s.toString()); |

В этот метод filter() передается ключ поиска.

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

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | 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 нало еще определить провайдер фильтрации адаптера, которые и будет инкапсулировать реальную логику фильтрации:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | userAdapter.setFilterQueryProvider(new FilterQueryProvider() {      @Override      public Cursor runQuery(CharSequence constraint) {            if (constraint == null || constraint.length() == 0) {                return sqlHelper.database.rawQuery("select \* from " + DatabaseHelper.TABLE, null);          }          else {              return sqlHelper.database.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:

|  |  |
| --- | --- |
| 1  2 | 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:

