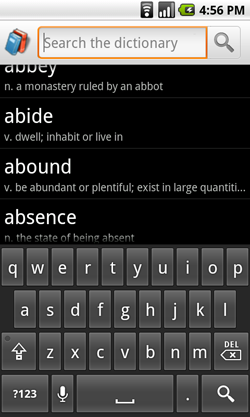
**Использование Android Search Dialog. Пример простого приложения**

  
  
Данная статья предназначена для тех, кто уже написал свой HelloWorld для Android и знает, что такое Activity и Intent, а так же где находится манифест, и зачем нужны layout'ы. В противном случае, можно ознакомиться с этим материалом, например, на [developer.android.com](http://developer.android.com/guide/index.html).   
  
В статье описывается создание несложного приложения, которое использует механизм реализации поиска, основанный на возможностях встроенного фреймворка. После прочтения вы также сможете настроить свое приложение таким образом, чтобы оно осуществляло поиск по данным, используя стандартный Android Search Dialog.

**Немного теории**

Android Search Dialog (далее — «диалог поиска») управляется с помощью поискового фреймворка. Это означает, что разработчику не нужно задумываться над тем как его нарисовать или как отловить поисковый запрос. За вас эту работу сделает [SearchManager](http://developer.android.com/reference/android/app/SearchManager.html).  
  
Итак, когда пользователь запускает поиск, SearchManager создает Intent, и направляет его к Activity, которое отвечает за поиск данных (при этом сам запрос помещается в экстры). То есть по сути в приложении должно быть хотя бы одно Activity, которое получает поисковые Intent'ы, выполняет поиск, и предоставляет пользователю результаты. Для реализации потребуется следующее:

* Конфигурационный xml файл (в нем содержится информация о диалоге)
* Activity, которое будет получать поисковые запросы, выполнять поиск и выводить результаты на экран
* Механизм вызова поискового диалога (так как не все устройства с Android на борту имеют на корпусе кнопку поиска)

**Конфигурационный файл**

Начнем с создания конфигурационного файла, потому что это самая простая часть. Конечно, вам нужно заранее приготовиться, создать новый проект, Activity (У меня используется Activity «Main» и имя проекта «SearchExample»). Всё, что нужно для конфигурации — это создать файл searchable.xml в поддиректории res/xml/ вашего проекта, со следующим содержанием.

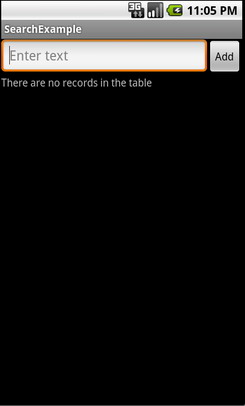
<?xml version="1.0" encoding="utf-8"?>  
<searchable xmlns:android="http://schemas.android.com/apk/res/android"  
android:label="@string/app\_name"  
android:hint="@string/search\_hint"   
>  
</searchable>  
  
\* This source code was highlighted with [Source Code Highlighter](http://virtser.net/blog/post/source-code-highlighter.aspx).

Обязательным атрибутом является только android:label, причем он должен ссылаться на строку, которая является такой же, что и название приложения. Второй атрибут, android:hint используется для отображения строки в пустом диалоге. Например, это может быть «Поиск по Видео» или «Поиск контактов» и т.п. Этот атрибут указывает на то, по каким данным осуществляется поиск. Также важно знать, что элемент searchable поддерживает множество других атрибутов, подробнее можно прочесть [Здесь](http://developer.android.com/guide/topics/search/searchable-config.html).

**Создаем Activity**

Минимально, всё что нам нужно от пользовательского интерфейса Activity — это список для вывода результатов поиска и механизм вызова поискового диалога. Так и сделаем, добавив только поле для ввода текста и кнопку, чтобы мы сами могли заполнять базу. Забегая вперед, скажу, что данные будем хранить в БД SQLite.  
  
Опишем интерфейс Activity следующим образом (файл находится в res/layout/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="fill\_parent"  
android:layout\_height="fill\_parent">  
<LinearLayout  
android:orientation="horizontal"  
android:layout\_width="fill\_parent"  
android:layout\_height="wrap\_content"  
android:gravity="top">  
<EditText   
android:id="@+id/text"   
android:layout\_width="wrap\_content"   
android:layout\_height="wrap\_content"   
android:hint="@string/text"   
android:layout\_weight="100.0" />  
<Button   
android:id="@+id/add"   
android:layout\_width="wrap\_content"   
android:layout\_height="wrap\_content"  
android:text="@string/add" />  
</LinearLayout>  
<ListView   
android:id="@android:id/list"   
android:layout\_width="fill\_parent"   
android:layout\_height="wrap\_content" />  
<TextView   
android:layout\_gravity="left"   
android:id="@android:id/empty"   
android:layout\_width="fill\_parent"   
android:layout\_height="fill\_parent"  
android:text="@string/no\_records"/>  
</LinearLayout>  
  
\* This source code was highlighted with [Source Code Highlighter](http://virtser.net/blog/post/source-code-highlighter.aspx).

Выглядит следующим образом:  
  
  
  
Также нам понадобится layout для вида элемента списка, опишем его простейшим образом (файл находится в res/layout/record.xml)

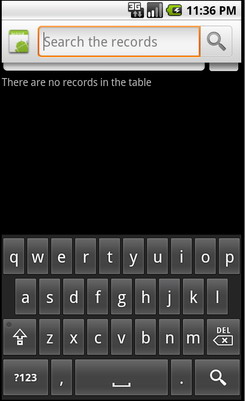
<?xml version="1.0" encoding="utf-8" ?>   
<TextView   
android:id="@+id/text1"   
xmlns:android="http://schemas.android.com/apk/res/android"   
android:layout\_width="wrap\_content"  
android:layout\_height="wrap\_content"  
/>   
  
\* This source code was highlighted with [Source Code Highlighter](http://virtser.net/blog/post/source-code-highlighter.aspx).

Также, не забываем про файл ресурсов, где хранятся наши строки (файл в res/values/strings.xml)

<?xml version="1.0" encoding="utf-8"?>  
<resources>  
<string name="app\_name">SearchExample</string>  
<string name="add">Add</string>  
<string name="text">Enter text</string>  
<string name="no\_records">There are no records in the table</string>  
<string name="search\_hint">Search the records</string>  
<string name="search">Search</string>  
</resources>  
  
\* This source code was highlighted with [Source Code Highlighter](http://virtser.net/blog/post/source-code-highlighter.aspx).

Напоминаю, что когда пользователь выполняет поиск, то нужно, чтобы SearchManager отправлял Intent вместе с поисковым запросом к нашему Activity. Чтобы SearchManager точно знал, куда ему отсылать Intent, нужно объявить об этом в манифесте. Итак, идем в AndroidManifest.xml, и изменяем его так, чтобы Activity могло принимать Intent'ы типа Search. Кроме этого, применим наш конфигурационный файл searchable.xml. Итак, манифест будет выглядеть примерно так:

<?xml version="1.0" encoding="utf-8"?>  
<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
package="com.example.search"  
android:versionCode="1"  
android:versionName="1.0">  
<application android:icon="@drawable/icon" android:label="@string/app\_name">  
<activity android:name=".Main"  
android:label="@string/app\_name">  
<intent-filter>  
<action android:name="android.intent.action.MAIN" />  
<category android:name="android.intent.category.LAUNCHER" />  
</intent-filter>  
<intent-filter>  
<action android:name="android.intent.action.SEARCH" />  
</intent-filter>  
<meta-data   
android:name="android.app.searchable"  
android:resource="@xml/searchable"  
/>  
</activity>  
  
</application>  
<uses-sdk android:minSdkVersion="5" />  
  
</manifest>   
  
\* This source code was highlighted with [Source Code Highlighter](http://virtser.net/blog/post/source-code-highlighter.aspx).

Сейчас, вы уже можете проверить, все ли вы сделали правильно. Вызвать диалог на эмуляторе можно, например, нажав кнопку поиска. Ну или если вы проверяете на девайсе, то зажав «Меню». Выглядеть должно примерно так:  
  


**Выполнение поиска**

После того, как мы объявили наше Activity и внесли необходимые изменения в AndroidManifest.xml, можно рассмотреть само выполнение поиска. Оно включает в себя три этапа:

* Получение запроса
* Поиск данных
* вывод результатов

**Получение запроса**

Так как SearchManager посылает Intent типа Search нашему Activity, то всё что нужно сделать это проверить на Intent этого типа при старте Activity. Тогда, если мы получаем нужный Intent, то можно извлекать из него экстру и выполнять поиск.

**Поиск данных**

Так как тип структуры хранения данных для разных приложений может различаться, то и методы для них свои. В нашем случае, проще всего выполнить запрос по таблице БД SQLite запросом LIKE. Конечно, лучше использовать FTS3, он значительно быстрее, подробнее о FTS3 можно прочесть на сайте [SQLite.org](http://www.sqlite.org/fts3.html). В идеале, также нужно всегда рассчитывать, что поиск может занять продолжительное время, поэтому можно создать какой-нибудь ProgressDialog, чтобы у нас не завис интерфейс, и чтобы пользователь знал, что приложение работает.

**Вывод результатов**

Вообще вывод результатов — это проблема UI, но так как мы используем ListView, то для нас проблема решается простым обновлением адаптера.

**Исходный код**

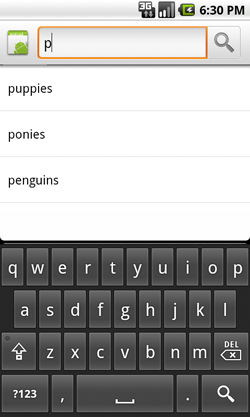
Наконец, привожу полный исходный код двух классов с комментариями. Первый — Main, наследник ListActivity, он используется для наполнения БД и вывода результатов. Второй класс — RecordsDbHelper, он реализует интерфейс для взаимодействия с БД. Самые важные методы — добавление записей и поиск совпадений, с помощью запроса LIKE.  
  
**Файл Main.java**

package com.example.search;  
  
import android.app.ListActivity;  
import android.app.SearchManager;  
import android.content.Intent;  
import android.database.Cursor;  
import android.os.Bundle;  
import android.view.Menu;  
import android.view.MenuInflater;  
import android.view.MenuItem;  
import android.view.View;  
import android.widget.Button;  
import android.widget.EditText;  
import android.widget.SimpleCursorAdapter;  
  
public class Main extends ListActivity {  
private EditText text;  
private Button add;  
private RecordsDbHelper mDbHelper;  
  
@Override  
public void onCreate(Bundle savedInstanceState) {  
super.onCreate(savedInstanceState);  
setContentView(R.layout.main);  
//Создаем экземпляр БД  
mDbHelper = new RecordsDbHelper(this);  
//Открываем БД для записи  
mDbHelper.open();  
//Получаем Intent  
Intent intent = getIntent();  
//Проверяем тип Intent  
if (Intent.ACTION\_SEARCH.equals(intent.getAction())) {   
//Берем строку запроса из экстры  
String query = intent.getStringExtra(SearchManager.QUERY);  
//Выполняем поиск  
showResults(query);  
}  
  
add = (Button) findViewById(R.id.add);  
text = (EditText) findViewById(R.id.text);  
add.setOnClickListener(new View.OnClickListener() {  
public void onClick(View view) {  
String data = text.getText().toString();  
if (!data.equals("")) {  
saveTask(data);  
text.setText("");  
}  
}  
});  
}  
  
private void saveTask(String data) {  
mDbHelper.createRecord(data);  
}  
  
private void showResults(String query) {  
//Ищем совпадения  
Cursor cursor = mDbHelper.fetchRecordsByQuery(query);  
startManagingCursor(cursor);  
String[] from = new String[] { RecordsDbHelper.KEY\_DATA };  
int[] to = new int[] { R.id.text1 };  
  
SimpleCursorAdapter records = new SimpleCursorAdapter(this,  
R.layout.record, cursor, from, to);  
//Обновляем адаптер  
setListAdapter(records);  
}  
//Создаем меню для вызова поиска (интерфейс в res/menu/main\_menu.xml)  
public boolean onCreateOptionsMenu(Menu menu) {  
MenuInflater inflater = getMenuInflater();  
inflater.inflate(R.menu.main\_menu, menu);  
return true;   
}  
  
public boolean onOptionsItemSelected(MenuItem item) {  
switch (item.getItemId()) {  
case R.id.search\_record:  
onSearchRequested();  
return true;  
default:  
return super.onOptionsItemSelected(item);  
}  
}  
}  
  
\* This source code was highlighted with [Source Code Highlighter](http://virtser.net/blog/post/source-code-highlighter.aspx).

**Файл RecordsDbHelper.java**

package com.example.search;  
  
import android.content.ContentValues;  
import android.content.Context;  
import android.database.Cursor;  
import android.database.SQLException;  
import android.database.sqlite.SQLiteDatabase;  
import android.database.sqlite.SQLiteOpenHelper;  
import android.util.Log;  
  
public class RecordsDbHelper {  
  
public static final String KEY\_DATA = "data";  
public static final String KEY\_ROWID = "\_id";  
  
private static final String TAG = "RecordsDbHelper";  
private DatabaseHelper mDbHelper;  
private SQLiteDatabase mDb;  
  
private static final String DATABASE\_CREATE = "CREATE TABLE records(\_id INTEGER PRIMARY KEY AUTOINCREMENT, "  
+ "data TEXT NOT NULL);";  
  
private static final String DATABASE\_NAME = "data";  
private static final String DATABASE\_TABLE = "records";  
private static final int DATABASE\_VERSION = 1;  
  
private final Context mCtx;  
  
private static class DatabaseHelper extends SQLiteOpenHelper {  
  
DatabaseHelper(Context context) {  
super(context, DATABASE\_NAME, null, DATABASE\_VERSION);  
}  
  
@Override  
public void onCreate(SQLiteDatabase db) {  
  
db.execSQL(DATABASE\_CREATE);  
}  
  
@Override  
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {  
Log.w(TAG, "Upgrading database from version " + oldVersion + " to "  
+ newVersion + ", which will destroy all old data");  
db.execSQL("DROP TABLE IF EXISTS tasks");  
onCreate(db);  
}  
}  
  
public RecordsDbHelper(Context ctx) {  
this.mCtx = ctx;  
}  
  
public RecordsDbHelper open() throws SQLException {  
mDbHelper = new DatabaseHelper(mCtx);  
mDb = mDbHelper.getWritableDatabase();  
return this;  
}  
  
public void close() {  
mDbHelper.close();  
}  
  
//Добавляем запись в таблицу  
public long createRecord(String data) {  
ContentValues initialValues = new ContentValues();  
initialValues.put(KEY\_DATA, data);  
return mDb.insert(DATABASE\_TABLE, null, initialValues);  
}  
  
//Поиск запросом LIKE  
public Cursor fetchRecordsByQuery(String query) {  
return mDb.query(true, DATABASE\_TABLE, new String[] { KEY\_ROWID,  
KEY\_DATA }, KEY\_DATA + " LIKE" + "'%" + query + "%'", null,  
null, null, null, null);  
}  
}  
  
\* This source code was highlighted with [Source Code Highlighter](http://virtser.net/blog/post/source-code-highlighter.aspx).

ЧАСТЬ-2



В статье объясняется, как модифицировать ваше приложение так, чтобы к диалогу добавлялись подсказки по недавним поисковым запросам. Для понимания кода и теории (помимо той, что требовалась в первой части) требуется знание контент-провайдеров. Его можно почерпнуть из [официального гайда](http://developer.android.com/guide/topics/providers/content-providers.html).

**Немного теории**

По сути, подсказки по недавним поисковым запросам представляют собой просто сохраненные запросы. Когда пользователь выбирает одну из подсказок, то Activity, которое отвечает за поиск, получает Intent типа Search с подсказкой в качестве строки, которую оно уже обрабатывало ранее. За отображение подсказок, так же как и за весь диалог, отвечает Search Manager, а для хранения используется контент-провайдер.  
  
Когда Search Manager определяет наше Activity как отвечающее за поиск и обеспечивающее подсказки к поиску, то происходит следующая последовательность действий:

1. Когда Search Manager получает текст поискового запроса, то он отправляет свой запрос к контент-провайдеру, обеспечивающему подсказки.
2. Контент-провайдер возвращает курсор, указывающий на подсказки, которые совпадают с текстом поискового запроса.
3. Search Manager отображает подсказки, используя курсор

После того как список подсказок был отображен, может случиться следующее:

* Если пользователь изменяет текст запроса, то все вышеперечисленные шаги повторятся.
* Если пользователь запускает поиск, то подсказки игнорируются.
* Если пользователь выбирает подсказку, то к Activity доставляется Intent с текстом этой подсказки в качестве запроса.

Итак, для реализации подсказок нам потребуется следующее:

* Создать контент-провайдер, который будет наследником класса [SearchRecentSuggestionsProvider](http://developer.android.com/reference/android/content/SearchRecentSuggestionsProvider.html) и объявить его в манифесте
* Изменить конфигурационный xml файл диалога, добавив в него информацию о контент-провайдере
* Изменить Activty так, чтобы оно сохраняло запросы каждый раз, когда запускается поиск

**Создаем контент-провайдер**

Всё что от нас требуется, это контент-провайдер, который является наследником класса [SearchRecentSuggestionsProvider](http://developer.android.com/reference/android/content/SearchRecentSuggestionsProvider.html). Этот класс практически делает за разработчика все действия и всё, что от нас требуется — это написать конструктор.  
  
**Файл SuggestionProvider.java**

package com.example.search;  
  
import android.content.SearchRecentSuggestionsProvider;  
  
public class SuggestionProvider extends SearchRecentSuggestionsProvider {  
public final static String AUTHORITY = "com.example.search.SuggestionProvider";  
public final static int MODE = DATABASE\_MODE\_QUERIES;  
  
public SuggestionProvider() {  
setupSuggestions(AUTHORITY, MODE);  
}  
}  
  
\* This source code was highlighted with [Source Code Highlighter](http://virtser.net/blog/post/source-code-highlighter.aspx).

Методу [setupSuggestions()](http://developer.android.com/reference/android/content/SearchRecentSuggestionsProvider.html#setupSuggestions(java.lang.String,%20int)) передается строка авторизации и режим работы БД контент-провайдера. Строка авторизации может быть любой, единственное требование — уникальность. Однако, в официальной документации рекомендуется использовать полное имя контент-провайдера, включая название пакета. Режим работы БД должен включать в себя [DATABASE\_MODE\_QUERIES](http://developer.android.com/reference/android/content/SearchRecentSuggestionsProvider.html#DATABASE_MODE_QUERIES), также опционально можно добавить [DATABASE\_MODE\_2LINES](http://developer.android.com/reference/android/content/SearchRecentSuggestionsProvider.html#DATABASE_MODE_2LINES). Во втором случае к таблице подсказок добавляется столбец, позволяющий отображать для каждой подсказки вторую строку. Выглядеть в коде будет так:

public final static int MODE = DATABASE\_MODE\_QUERIES | DATABASE\_MODE\_2LINES;  
  
\* This source code was highlighted with [Source Code Highlighter](http://virtser.net/blog/post/source-code-highlighter.aspx).

Теперь не забываем, что нужно объявить наш контент-провайдер в манифесте.

**Файл AndroidManifest.xml**

<?xml version="1.0" encoding="utf-8"?>  
<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
package="com.example.search"  
android:versionCode="1"  
android:versionName="1.0">  
<application android:icon="@drawable/icon" android:label="@string/app\_name">  
<activity android:name=".Main"  
android:label="@string/app\_name">  
<intent-filter>  
<action android:name="android.intent.action.MAIN" />  
<category android:name="android.intent.category.LAUNCHER" />  
</intent-filter>  
<intent-filter>  
<action android:name="android.intent.action.SEARCH" />  
</intent-filter>  
<meta-data   
android:name="android.app.searchable"  
android:resource="@xml/searchable"  
/>  
</activity>  
<provider android:name=".SuggestionProvider"  
android:authorities="com.example.search.SuggestionProvider" />   
</application>  
<uses-sdk android:minSdkVersion="5" />  
  
</manifest>   
  
\* This source code was highlighted with [Source Code Highlighter](http://virtser.net/blog/post/source-code-highlighter.aspx).

**Изменение конфигурационного файла**  
Для того, чтобы диалог использовал наш контент-провайдер для подсказок нужно добавить в него параметры android:searchSuggestAuthority и android:searchSuggestSelection

**Файл searchable.xml**

<?xml version="1.0" encoding="utf-8"?>  
<searchable xmlns:android="http://schemas.android.com/apk/res/android"  
android:label="@string/app\_name"  
android:hint="@string/search\_hint"   
android:searchSuggestAuthority="com.example.search.SuggestionProvider"  
android:searchSuggestSelection=" ?">  
</searchable>  
  
\* This source code was highlighted with [Source Code Highlighter](http://virtser.net/blog/post/source-code-highlighter.aspx).

Значение параметра android:searchSuggestAuthority должно полностью совпадать со строкой авторизации контент-провайдера.  
Значение параметра android:searchSuggestSelection должно быть представлять знак вопроса, поставленный после пробела, потому что это аргумент выборки из БД и знак вопроса автоматически заменяется на текст, введенный пользователем.

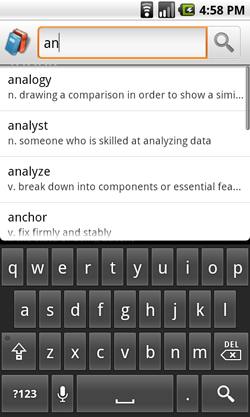
**Изменение Activity**

Всё что нам нужно — это сохранить текст запроса, для этого создается экземпляр класса [SearchRecentSuggestions](http://developer.android.com/reference/android/provider/SearchRecentSuggestions.html) и вызывается метод [saveRecentQuery()](http://developer.android.com/reference/android/provider/SearchRecentSuggestions.html#saveRecentQuery(java.lang.String,%20java.lang.String)). Это происходит каждый раз, когда в Activity приходит Intent с запросом на поиск данных. В метод [saveRecentQuery()](http://developer.android.com/reference/android/provider/SearchRecentSuggestions.html#saveRecentQuery(java.lang.String,%20java.lang.String)) передается два параметра, первый является обязательным и представляет собой строку поискового запроса, второй — опциональный, требуется если вы используете [DATABASE\_MODE\_2LINES](http://developer.android.com/reference/android/content/SearchRecentSuggestionsProvider.html#DATABASE_MODE_2LINES) для отображения второй строки текста в подсказке.  
  
В официальной документации рекомендуется также предусмотреть интерфейс для очистки всей таблицы подсказок. По всей видимости, это нужно для обеспечения user's privacy. Мы просто добавим еще один пункт меню, при нажатии на который будет вызываться очистка всей истории запросов.

**Файл Main.java**

package com.example.search;  
  
import android.app.ListActivity;  
import android.app.SearchManager;  
import android.content.Intent;  
import android.database.Cursor;  
import android.os.Bundle;  
import android.provider.SearchRecentSuggestions;  
import android.view.Menu;  
import android.view.MenuInflater;  
import android.view.MenuItem;  
import android.view.View;  
import android.widget.Button;  
import android.widget.EditText;  
import android.widget.SimpleCursorAdapter;  
  
public class Main extends ListActivity {  
private EditText text;  
private Button add;  
private RecordsDbHelper mDbHelper;  
  
@Override  
public void onCreate(Bundle savedInstanceState) {  
super.onCreate(savedInstanceState);  
setContentView(R.layout.main);  
//Создаем экземпляр БД  
mDbHelper = new RecordsDbHelper(this);  
//Открываем БД для записи  
mDbHelper.open();  
//Получаем Intent  
Intent intent = getIntent();  
//Проверяем тип Intent  
if (Intent.ACTION\_SEARCH.equals(intent.getAction())) {   
//Берем строку запроса из экстры  
String query = intent.getStringExtra(SearchManager.QUERY);  
//Создаем экземпляр SearchRecentSuggestions  
SearchRecentSuggestions suggestions = new SearchRecentSuggestions(this,  
SuggestionProvider.AUTHORITY, SuggestionProvider.MODE);  
//Сохраняем запрос  
suggestions.saveRecentQuery(query, null);   
//Выполняем поиск  
showResults(query);  
}  
  
add = (Button) findViewById(R.id.add);  
text = (EditText) findViewById(R.id.text);  
add.setOnClickListener(new View.OnClickListener() {  
public void onClick(View view) {  
String data = text.getText().toString();  
if (!data.equals("")) {  
saveTask(data);  
text.setText("");  
}  
}  
});  
}  
  
private void saveTask(String data) {  
mDbHelper.createRecord(data);  
}  
  
private void showResults(String query) {  
//Ищем совпадения  
Cursor cursor = mDbHelper.fetchRecordsByQuery(query);  
startManagingCursor(cursor);  
String[] from = new String[] { RecordsDbHelper.KEY\_DATA };  
int[] to = new int[] { R.id.text1 };  
  
SimpleCursorAdapter records = new SimpleCursorAdapter(this,  
R.layout.record, cursor, from, to);  
//Обновляем адаптер  
setListAdapter(records);  
}  
//Создаем меню для вызова поиска (интерфейс в res/menu/main\_menu.xml)  
public boolean onCreateOptionsMenu(Menu menu) {  
MenuInflater inflater = getMenuInflater();  
inflater.inflate(R.menu.main\_menu, menu);  
return true;   
}  
  
public boolean onOptionsItemSelected(MenuItem item) {  
switch (item.getItemId()) {  
case R.id.search\_record:  
//Вызываем поиск  
onSearchRequested();  
return true;  
case R.id.clear\_recent\_suggestions:  
//Очищаем историю  
SearchRecentSuggestions suggestions = new SearchRecentSuggestions(this,  
SuggestionProvider.AUTHORITY, SuggestionProvider.MODE);  
suggestions.clearHistory();  
return true;  
default:  
return super.onOptionsItemSelected(item);  
}  
}  
}  
  
\* This source code was highlighted with [Source Code Highlighter](http://virtser.net/blog/post/source-code-highlighter.aspx).

**Часть 3 — Custom Suggestions**

  
  
Это заключительная статья по использованию Android Search Dialog (предыдущие находятся [здесь](http://habrahabr.ru/blogs/android_development/111475/) и [здесь](http://habrahabr.ru/blogs/android_development/111493/)). В ней я расскажу, как добавить в диалог динамические подсказки к поиску, а также, как интегрировать поиск по вашему приложению в системный Quick Search Box (QSB). Преимущество QSB в том, что с его помощью можно получать информацию из практически любого места в OS.

**Теория**

Подсказки к поиску создаются при помощи данных вашего приложения, по которым осуществляется поиск. Когда пользователь выбирает одну из них, то Search Manager посылает Intent к Activity, которое отвечает за поиск. Обычно, когда пользователь нажимает иконку поиска в диалоге, то отправляется Intent типа Search, однако, при выборе подсказки в данном случае можно определить другой тип Intent, так чтобы мы могли его перехватить и совершить соответствующие действия, например, создание нового диалога, или вызов Activity для отображения информации и т.д.  
Данные поискового запроса переносятся через Intent как и раньше, однако теперь мы будем использовать URI, чтобы определять тип запроса через контент-провайдер.  
  
Снова, нам не нужно производить никаких действий по отрисовке диалога, этим занимается Search Manager, всё, что от нас требуется — представить конфигурационный xml файл.  
  
Итак, когда Search Manager определяет наше Activity как отвечающее за поиск и обеспечивающее подсказки к поиску, то происходит следующая последовательность действий:

1. Когда Search Manager получает текст поискового запроса, то он отправляет свой запрос к контент-провайдеру, обеспечивающему подсказки.
2. Контент-провайдер возвращает курсор, указывающий на подсказки, которые совпадают с текстом поискового запроса.
3. Search Manager отображает подсказки, используя курсор

После того как список подсказок был отображен, может случиться следующее:

* Если пользователь изменяет текст запроса, то все вышеперечисленные шаги повторятся.
* Если пользователь запускает поиск, то подсказки игнорируются, и к Activity отправляется Intent типа Search.
* Если пользователь выбирает подсказку, то к Activity доставляется Intent другого типа (тип определяется в конфигурационном файле), переносящий URI в качестве данных. URI будет использоваться для поиска записи в таблице, соответствующей выбранной подсказке.

Итак, мы модифицируем наше приложение (то которое рассматривалось в [части 1](http://habrahabr.ru/blogs/android_development/111475/)) так, чтобы добавлялись динамические подсказки, причем, для отработки механизма, при выборе подсказки будем вызывать новое Activity, которое будет отображать информацию по запросу. Для реализации потребуется:

* Изменить конфигурационный файл диалога, добавив к нему информацию о контент-провайдере и типе Intent, используемом для подсказок
* Создать таблицу в БД SQLite, которая будет предоставлять столбцы, требуемые Search Manager'ом для подсказок
* Создать новый контент-провайдер, имеющий доступ к таблице подсказок, и определить его в манифесте
* Добавить Activity, которое будет отображать информацию при выборе подсказок

**Изменяем конфигурационный файл**

Напоминаю, что конфигурационный файл (res/xml/searchable.xml) требуется для отображения диалога и его изменения, например, для использования голосового поиска. Чтобы использовать динамические подсказки, необходимо добавить в файл параметр: android:searchSuggestAuthority. Он будет совпадать со строкой авторизации контент-провайдера. Кроме этого добавим параметр android:searchMode=«queryRewriteFromText», его значение указывает на то, что строка поиска в диалоге будет перезаписываться при навигации по подсказкам, например с помощью трекбола. Также добавим параметры, задающие оператор выборки, тип Intent отправляемого при выборе подсказки, и минимальное количество напечатанных символов в диалоге, необходимое для запроса к контент-провайдеру.  
  
**Файл res/xml/searchable.xml**

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

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

android:label="@string/app\_name"

android:hint="@string/search\_hint"

android:searchSettingsDescription="@string/settings\_description"

android:searchMode="queryRewriteFromText"

android:includeInGlobalSearch="true"

android:searchSuggestAuthority="com.example.search.SuggestionProvider"

android:searchSuggestIntentAction="android.intent.action.VIEW"

android:searchSuggestIntentData="content://com.example.search.SuggestionProvider/records"

android:searchSuggestThreshold="1"

android:searchSuggestSelection=" ?">

</searchable>

**Создаем контент-провайдер**

По сути наш контент-провайдер ничем не отличается от других. Но нужно сделать так, чтобы для каждой строки из таблицы подсказок выбирались нужные столбцы, те которые требует Search Manager. Мы будем запрашивать данные по подсказкам с помощью метода контент-провайдера query(). Причем вызываться он будет каждый раз, когда пользователь печатает новый символ в диалоге. Таким образом, метод query() должен возвращать курсор на записи в таблице, совпадающие с запросом, и тогда Search Manager сможет отобразить подсказки. Смотрите описание метода в комментариях к коду.  
Сам текст запроса будет дописываться к URI, так что с его получением проблем не будет, нужно просто использовать стандартный метод getLastPathSegment().

**Создание таблицы подсказок**

Когда Search Manager получает курсор, указывающий на записи, то он ожидает определенный набор столбцов для каждой записи. Обязательными являются два: \_ID — уникальный идентификатор каждой подсказки, и SUGGEST\_COLUMN\_TEXT\_1 — текст подсказки.  
Необязательных столбцов существует много, например используя SUGGEST\_COLUMN\_ICON\_1, вы можете определить для каждой записи иконку, отображаемую с левой стороны подсказки (очень удобно, например, для поиска по контактам).

**Определение типа данных для Intent**  
Так как мы передаем данные по запросу через URI, то нам нужен механизм для определения того, какая подсказка была выбрана. Тут есть два пути. Первый, заключается в том, чтобы определить отдельный столбец SUGGEST\_COLUMN\_INTENT\_DATA, в котором будут уникальные данные для каждой записи, тогда можно получать данные из Intent через getData() или getDataString(). Второй вариант — определить тип данных для всех Intent в конфигурационном файле (res/xml/searchable.xml) а потом дописывать к URI уникальные данные для каждого Intent, используя столбец SUGGEST\_COLUMN\_INTENT\_DATA\_ID.  
Мы будем использовать второй вариант, причем отдельных столбцов в таблице я не создавал, так как можно просто создать отображение из SUGGEST\_COLUMN\_INTENT\_DATA\_ID в rowId таблицы. Добавлю еще, что ради спортивного интереса в SQLite для поиска использовался [FTS3](http://www.sqlite.org/fts3.html), то есть пришлось создавать виртуальную таблицу, для которой нельзя накладывать ограничения на столбцы (constraints), такие как PRIMARY KEY или NULL/NOT NULL. Зато у виртуальных таблиц есть уникальный идентификатор строки, на него и установим отображение. То есть data для Intent будет иметь следующий вид: к URI будет дописываться "/" и rowId строки в таблице.

**Создание Activity для отображения информации**  
Интерфейс находится в res/layout/record\_activity.xml. Всё чем занимается Activity — получение данных из Intent, запрос курсора через контент-провайдер и отображение записи в текстовом поле.  
  
**Файл res/layout/record\_activity.xml**

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

android:orientation="vertical"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

android:padding="10dp">

<TextView

android:id="@+id/record\_header"

android:textSize="25dp"

android:textColor="?android:textColorPrimary"

android:layout\_width="wrap\_content"

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

</LinearLayout>

Теперь внесем информацию о контент-провайдере и новом Activity в манифест, также, так как у нас теперь два Activity, то укажем то, которое по умолчанию отвечает за поиск.  
  
**Файл AndroidManifest.xml**

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

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

package="com.example.search"

android:versionCode="1"

android:versionName="1.0">

<application android:icon="@drawable/icon" android:label="@string/app\_name">

<activity android:name=".Main"

android:label="@string/app\_name">

<intent-filter>

<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />

</intent-filter>

<intent-filter>

<action android:name="android.intent.action.SEARCH" />

</intent-filter>

<meta-data

android:name="android.app.searchable"

android:resource="@xml/searchable"

/>

</activity>

<activity android:name=".RecordActivity"

android:theme="@android:style/Theme.NoTitleBar" />

<provider android:name=".SuggestionProvider"

android:authorities="com.example.search.SuggestionProvider" />

<meta-data android:name="android.app.default\_searchable"

android:value=".Main" />

</application>

<uses-sdk android:minSdkVersion="5" />

</manifest>

**Перехват Intent в Activity, отвечающем за поиск**

После всех вышеперечисленных шагов нужно обработать Intent в главном Activity, которое отвечает за поиск. Так как мы определили тип Intent для подсказок как View, то нужно просто добавить проверку на него. В случае если условие выполнится, то запускается RecordActivity, используя Intent, в данные которого записывается URI + "/" + id подсказки в таблице.

**Интеграция с Quick Search Box**  
После того, как Вы модифицировали ваше приложение для использования custom suggestions, то можно добавить его к системному поиску. Для этого в файл searchable.xml нужно добавить два параметра:

1. android:includeInGlobalSearch=«true» — указывает на то, что QSB может осуществлять поиск по вашему приложению.
2. android:searchSettingsDescription="@string/settings\_description" — указывает на описание вашего приложения, которое отображается при настройке Quick Search Box. Эти настройки находятся в settings->search.

Эти параметры доступны с версии Android 1.6, то есть для версий ниже вы не сможете настроить ваше приложение для QSB.

**Исходный код**

Представляю полный исходный код всех необходимых классов. Main.java — главное Activity, отвечающее за поиск и за отправление запросов к контент-провайдеру, RecordActivity.java — получает Intent с данными по конкретной записи, получает ссылку на запись и отображает информацию. SuggestionProvider.java — контент-провайдер обрабатывающий запросы от Search Manager'a к таблице подсказок. RecordsDbHelper.java — отвечает за создание таблицы, её заполнение, установление необходимого отображения, и за сам «matching» записей.  
  
**Файл Main.java**

package com.example.search;

import android.app.ListActivity;

import android.app.SearchManager;

import android.content.Intent;

import android.database.Cursor;

import android.os.Bundle;

import android.view.Menu;

import android.view.MenuInflater;

import android.view.MenuItem;

import android.view.View;

import android.widget.Button;

import android.widget.EditText;

import android.widget.SimpleCursorAdapter;

import android.widget.Toast;

public class Main extends ListActivity {

private EditText text;

private Button add;

private RecordsDbHelper mDbHelper;

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.main);

mDbHelper = new RecordsDbHelper(this);

Intent intent = getIntent();

if (Intent.ACTION\_SEARCH.equals(intent.getAction())) {

//Берем строку запроса из экстры

String query = intent.getStringExtra(SearchManager.QUERY);

//Выполняем поиск

showResults(query);

} else if (Intent.ACTION\_VIEW.equals(intent.getAction())){

//Создаем Intent для открытия RecordActivity

Intent recordIntent = new Intent(this, RecordActivity.class);

recordIntent.setData(intent.getData());

startActivity(recordIntent);

finish();

}

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

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

add.setOnClickListener(new View.OnClickListener() {

public void onClick(View view) {

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

if (!data.equals("")) {

saveTask(data);

text.setText("");

}

}

});

}

private void saveTask(String data) {

mDbHelper.createRecord(data);

}

private void showResults(String query) {

//Запрашиваем у контент-провайдера курсор на записи

Cursor cursor = managedQuery(SuggestionProvider.CONTENT\_URI, null, null,

new String[] {query}, null);

if (cursor == null) {

Toast.makeText(this, "There are no results", Toast.LENGTH\_SHORT).show();

} else {

//Обновляем адаптер

String[] from = new String[] { RecordsDbHelper.KEY\_DATA };

int[] to = new int[] { R.id.text1 };

SimpleCursorAdapter records = new SimpleCursorAdapter(this, R.layout.record, cursor, from, to);

getListView().setAdapter(records);

}

}

//Создаем меню для вызова поиска (интерфейс в res/menu/main\_menu.xml)

public boolean onCreateOptionsMenu(Menu menu) {

MenuInflater inflater = getMenuInflater();

inflater.inflate(R.menu.main\_menu, menu);

return true;

}

public boolean onOptionsItemSelected(MenuItem item) {

switch (item.getItemId()) {

case R.id.search\_record:

onSearchRequested();

return true;

default:

return super.onOptionsItemSelected(item);

}

}

}

**Файл RecordActivity.java**

package com.example.search;

import android.app.Activity;

import android.database.Cursor;

import android.net.Uri;

import android.os.Bundle;

import android.view.Menu;

import android.view.MenuInflater;

import android.view.MenuItem;

import android.widget.TextView;

public class RecordActivity extends Activity {

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.record\_activity);

//Получаем URI с данными из Intent и запрашиваем данные через контент-провайдер

Uri uri = getIntent().getData();

Cursor cursor = managedQuery(uri, null, null, null, null);

if (cursor == null) {

finish();

} else {

//Устанавливаем данные в текстовое поле

cursor.moveToFirst();

TextView record = (TextView) findViewById(R.id.record\_header);

int rIndex = cursor.getColumnIndexOrThrow(RecordsDbHelper.KEY\_DATA);

record.setText(cursor.getString(rIndex));

}

}

//Создаем меню для вызова диалога поиска из этого активити

@Override

public boolean onCreateOptionsMenu(Menu menu) {

MenuInflater inflater = getMenuInflater();

inflater.inflate(R.menu.main\_menu, menu);

return true;

}

@Override

public boolean onOptionsItemSelected(MenuItem item) {

switch (item.getItemId()) {

case R.id.search\_record:

onSearchRequested();

return true;

default:

return false;

}

}

}

**Файл SuggestionProvider.java**

package com.example.search;

import android.app.SearchManager;

import android.content.ContentProvider;

import android.content.ContentResolver;

import android.content.ContentValues;

import android.content.UriMatcher;

import android.database.Cursor;

import android.net.Uri;

import android.provider.BaseColumns;

public class SuggestionProvider extends ContentProvider{

private RecordsDbHelper mDbHelper;

public static String AUTHORITY = "com.example.search.SuggestionProvider";

public static final Uri CONTENT\_URI = Uri.parse("content://" + AUTHORITY + "/records");

//MIME типы для getType()

public static final String RECORDS\_MIME\_TYPE = ContentResolver.CURSOR\_DIR\_BASE\_TYPE +"/vnd.example.search";

public static final String RECORD\_MIME\_TYPE = ContentResolver.CURSOR\_ITEM\_BASE\_TYPE +"/vnd.example.search";

//Для матчера разных URI

private static final int SEARCH\_RECORDS = 0;

private static final int GET\_RECORD = 1;

private static final int SEARCH\_SUGGEST = 2;

private static final UriMatcher sURIMatcher = makeUriMatcher();

@Override

public boolean onCreate() {

mDbHelper = new RecordsDbHelper(getContext());

return true;

}

/\*\*

\* Обрабатывает запросы от Search Manager'a.

\* Когда запрашивается конкретный элемент, то требуется только URI.

\* Когда запрашивается поиск по всей таблице, то первый элемент параметра selectionArgs содержит строку запроса.

\* Остальные параметры не нужны.

\*/

@Override

public Cursor query(Uri uri, String[] projection, String selection,

String[] selectionArgs, String sortOrder) {

//Используем UriMatcher, чтобы узнать какой тип запроса получен. Далее формируем соответствующий запрос к БД

switch (sURIMatcher.match(uri)) {

case SEARCH\_SUGGEST:

if (selectionArgs == null) {

throw new IllegalArgumentException(

"selectionArgs must be provided for the Uri: " + uri);

}

return getSuggestions(selectionArgs[0]);

case SEARCH\_RECORDS:

if (selectionArgs == null) {

throw new IllegalArgumentException(

"selectionArgs must be provided for the Uri: " + uri);

}

return search(selectionArgs[0]);

case GET\_RECORD:

return getRecord(uri);

default:

throw new IllegalArgumentException("Unknown Uri: " + uri);

}

}

private Cursor getSuggestions(String query) {

query = query.toLowerCase();

String[] columns = new String[] {

BaseColumns.\_ID,

RecordsDbHelper.KEY\_DATA,

SearchManager.SUGGEST\_COLUMN\_INTENT\_DATA\_ID};

return mDbHelper.getRecordMatches(query, columns);

}

private Cursor search(String query) {

query = query.toLowerCase();

String[] columns = new String[] {

BaseColumns.\_ID,

RecordsDbHelper.KEY\_DATA};

return mDbHelper.getRecordMatches(query, columns);

}

private Cursor getRecord(Uri uri) {

String rowId = uri.getLastPathSegment();

String[] columns = new String[] {

RecordsDbHelper.KEY\_DATA};

return mDbHelper.getRecord(rowId, columns);

}

/\*\*

\* Вспомогательный метод

\* нужен для сопоставления разным URI конкретных значений

\*/

private static UriMatcher makeUriMatcher() {

UriMatcher matcher = new UriMatcher(UriMatcher.NO\_MATCH);

// Для записей

matcher.addURI(AUTHORITY, "records", SEARCH\_RECORDS);

matcher.addURI(AUTHORITY, "records/#", GET\_RECORD);

// Для подсказок

matcher.addURI(AUTHORITY, SearchManager.SUGGEST\_URI\_PATH\_QUERY, SEARCH\_SUGGEST);

matcher.addURI(AUTHORITY, SearchManager.SUGGEST\_URI\_PATH\_QUERY + "/\*", SEARCH\_SUGGEST);

return matcher;

}

//Требуемые методы (наследуются от класса ContentProvider)

@Override

public String getType(Uri uri) {

switch (sURIMatcher.match(uri)) {

case SEARCH\_RECORDS:

return RECORDS\_MIME\_TYPE;

case SEARCH\_SUGGEST:

return SearchManager.SUGGEST\_MIME\_TYPE;

case GET\_RECORD:

return RECORD\_MIME\_TYPE;

default:

throw new IllegalArgumentException("Unknown URL " + uri);

}

}

@Override

public int update(Uri uri, ContentValues values, String selection,

String[] selectionArgs) {

throw new UnsupportedOperationException();

}

@Override

public Uri insert(Uri uri, ContentValues values) {

throw new UnsupportedOperationException();

}

@Override

public int delete(Uri uri, String selection, String[] selectionArgs) {

throw new UnsupportedOperationException();

}

}

**Файл RecordsDbHelper.java**

package com.example.search;

import java.util.HashMap;

import android.app.SearchManager;

import android.content.ContentValues;

import android.content.Context;

import android.database.Cursor;

import android.database.sqlite.SQLiteDatabase;

import android.database.sqlite.SQLiteOpenHelper;

import android.database.sqlite.SQLiteQueryBuilder;

import android.provider.BaseColumns;

import android.util.Log;

public class RecordsDbHelper {

//Единственный столбец в таблице - данные

public static final String KEY\_DATA = SearchManager.SUGGEST\_COLUMN\_TEXT\_1;

private static final String TAG = "RecordsDbHelper";

private DatabaseHelper mDbHelper;

private SQLiteDatabase mDb;

private static final String DATABASE\_NAME = "datas";

private static final String DATABASE\_TABLE = "records";

private static final int DATABASE\_VERSION = 2;

//Сценарий создания БД

private static final String DATABASE\_CREATE =

"CREATE VIRTUAL TABLE " + DATABASE\_TABLE +

" USING fts3 (" + KEY\_DATA + ");";

private static final HashMap<String,String> mColumnMap = buildColumnMap();

/\*\*

\* Возвращает курсор, указывающий на запись с rowId

\* @param rowId id возвращаемой записи

\* @param columns возвращаемые столбцы записи; если null, то все

\* @return курсор, указывающий на определенную запись, null - если не запись не найдена

\*/

public Cursor getRecord(String rowId, String[] columns) {

String selection = "rowid = ?";

String[] selectionArgs = new String[] {rowId};

return query(selection, selectionArgs, columns);

}

/\*\*

\* Возвращает курсор, указывающий на все записи, совпадающие с запросом

\* @param query текст поискового запроса

\* @param columns возвращаемые столбцы записи; если null, то все

\* @return курсор, указывающий на записи, совпадающие с запросом, null - если не записи не найдена

\*/

public Cursor getRecordMatches(String query, String[] columns) {

String selection = KEY\_DATA + " MATCH ?";

String[] selectionArgs = new String[] {query+"\*"};

return query(selection, selectionArgs, columns);

}

/\*\*

\* Создает отображение всевозможных запрашиваемых столбцов.

\* Будет установлено как проекция в SQLiteQueryBuilder.

\* Нужно для того, чтобы назначить для каждой записи уникальные значения SUGGEST\_COLUMN\_INTENT\_DATA\_ID

\* которые используются для получения конкретной записи по URI.

\*/

private static HashMap<String,String> buildColumnMap() {

HashMap<String,String> map = new HashMap<String,String>();

map.put(KEY\_DATA, KEY\_DATA);

map.put(BaseColumns.\_ID, "rowid AS " +

BaseColumns.\_ID);

map.put(SearchManager.SUGGEST\_COLUMN\_INTENT\_DATA\_ID, "rowid AS " +

SearchManager.SUGGEST\_COLUMN\_INTENT\_DATA\_ID);

return map;

}

/\*\*

\*

\* @param selection оператор выборки

\* @param selectionArgs аргументы, заменяющие "?" в запросе к БД

\* @param columns возвращаемые столбцы записи

\* @return курсор, указывающий на все записи, совпадающие с поисковым запросом

\*/

private Cursor query(String selection, String[] selectionArgs, String[] columns) {

/\* SQLiteBuilder предоставляет возможность создания отображения для всех

\* необходимых столбцов БД, что позволяет не сообщать контент-провайдеру

\* настоящие имена столбцов.

\*/

SQLiteQueryBuilder builder = new SQLiteQueryBuilder();

builder.setTables(DATABASE\_TABLE);

builder.setProjectionMap(mColumnMap);

Cursor cursor = builder.query(mDbHelper.getReadableDatabase(),

columns, selection, selectionArgs, null, null, null);

if (cursor == null) {

return null;

} else if (!cursor.moveToFirst()) {

cursor.close();

return null;

}

return cursor;

}

/\*\*

\*Создает/открывает БД

\*/

private static class DatabaseHelper extends SQLiteOpenHelper {

DatabaseHelper(Context context) {

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

}

@Override

public void onCreate(SQLiteDatabase db) {

db.execSQL(DATABASE\_CREATE);

}

@Override

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

Log.w(TAG, "Upgrading database from version " + oldVersion + " to "

+ newVersion + ", which will destroy all old data");

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

onCreate(db);

}

}

public RecordsDbHelper(Context context) {

mDbHelper = new DatabaseHelper(context);

}

/\*\*

\* Добавляет запись в таблицу

\* @param data данные, сохраняемые в таблицу

\* @return id записи, или -1, если добавление не удалось

\*/

public long createRecord(String data) {

mDb = mDbHelper.getWritableDatabase();

ContentValues initialValues = new ContentValues();

initialValues.put(KEY\_DATA, data);

return mDb.insert(DATABASE\_TABLE, null, initialValues);

}

}

Весь проект можно взять на [code.google.com](http://code.google.com/p/androidsearchexample)