

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ**

Брянский государственный технический университет

**Утверждаю**

**Ректор университета**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_О.Н.Федонин**

**«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2017 г.**

**Программирование МОБИЛЬНЫХ СИСТЕМ**

**ЗНАКОМСТВО ИСТОЧНИКАМИ ДАННЫХ В ANDROID**

**Методические указания**

**к выполнению лабораторной работы №5**

**для студентов очной формы обучения по направлениям подготовки 02.03.03 – «Математическое обеспечение и администрирование информационных систем», 09.03.01 – «Информатика и вычислительная техника» и 09.03.04 – «Программная инженерия»**

**Брянск 2017**

УКД 004.43

Программирование мобильных систем. Знакомство с базами данных и источниками данных в ОС Android. [Текст] + [Электронный ресурс]: Методические указания к выполнению лабораторной работы №5 для студентов очной формы обучения по направлениям подготовки 02.03.03 – «Математическое обеспечение и администрирование информационных систем», 09.03.01 – «Информатика и вычислительная техника» и 09.03.04 – «Программная инженерия». – Брянск: БГТУ, 2017. – 14 с.

Разработал:

Д.Н.Панус

ст.преп.

Рекомендовано кафедрой «Информатика и программное обеспечение» БГТУ (протокол №2 от 16.09.2016)

Научный редактор Д.А.Коростелев

Редактор издательства Л.И.Афонина

Компьютерный набор Д.Н.Панус

Темплан 2017 г., п.273

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подписано в печать 23.09.17. Формат 60х84 1/16 Бумага офсетная. Офсетная печать. Усл.печ.л. 1,8. Уч.-изд.л. 1,8 Тираж 1 экз. Заказ Бесплатно.

Издательство Брянского государственного технического университета

241035, Брянск, бульвар 50-летия Октября, 7, БГТУ. 58-82-49.

Лаборатория оперативной полиграфии БГТУ, ул. Институтская, 16.

**Цель работы**

Целью работы является ознакомление с особым механизмом предоставления данных между приложениями, основанными на базах данных, и других вариантах хранения данных.

Продолжительность работы – 5 часов.

**Порядок выполнения работы**

1. Ознакомиться с теоретическим материалом для лабораторной работы. Разобраться с теорией хранения пользовательской и системной информации приложениями в ОС Android.
2. Ознакомится с понятием "Источник данных"
3. Выбрать или предложить предметную область выполнения лабораторной работы.
4. Произвести проектирование и реализацию интерфейса приложения.
5. Реализовать в приложении требования к лабораторной работе.
6. Произвести сборку готового приложения в виде \*.apk файла приложения.
7. Продемонстрировать полученный результат преподавателю.

**Создание нового источника данных**

Источник данных (*Content Provider*) предоставляют интерфейс для публикации и потребления данных, основанный на простой адресной модели URI, используя схему: *сontent://*. Они позволяют отделить компоненты приложения, которые потребляют информацию, от исходных источников данных, предоставляя обобщенный механизм, благодаря которому приложения могут делиться своими данными или использовать чужие. Многие стандартные базы данных доступны в качестве Источников данных.

И могут использоваться сторонними приложениями. Сюда входят телефонные контакты, хранилище информации и др.

Для создания нового Источника данных наследуется абстрактный класс *ContentProvider*. Следует переопределить метод *onCreate*, чтобы создать (и инициализировать) базовый источник, который необходимо опубликовать.

|  |
| --- |
| **import** android.content.\*; **import** android.database.Cursor; **import** android.net.Uri; **import** android.database.SQLException; public **class** MyProvider **extends** ContentProvider { @Override  public boolean onCreate() { *//* ***TODO: Создать базу данных, с которой будет работать данный класс.* return true**;  } |

Нужно предусмотреть публичное статическое свойство *CONTENT\_URI*, которое возвращает полный URI для данного источника. URI Источника данных должен быть уникальным, поэтому лучше привязать его к имени вашего пакета. В общем виде определение URI Источника данных выглядит так:

*content://com.<CompanyName>.provider.<ApplicationName>/<DataPath>*

Пути URI могут представляться двумя способами. URI, показанный выше, — это запрос ко всем значениям определенного типа (в данным случае все элементы).

Чтобы получить запрос к конкретной строке, в конце нужно указать ее номер (в данном случае пятый элемент):

*сontent://com.paad.provider.myapp/elements/5*

**Регистрация Источника данных**

Источники данных должны быть зарегистрированы в манифесте вашего приложения, прежде чем их сможет обнаружить объект *ContentResolver*. Это делается с помощью тега *provider*, который включает в себя атрибут *name* и тег *authorities*. Полная версия должна соответствовать виду, представленному в следующем фрагменте XML-кода:

|  |
| --- |
| <provider android:name = « .MyContentProvider» android:authorities= «com.paad.seletondatabaseprovider»/> |

**Публикация адреса URI Источника данных**

Чтобы объект *ContentProvider* можно было обнаружить он должен объявлять о своем источнике с помощью публичного статистического свойства *CONTENT\_URI*, внутри которого находится путь к главной части содержимого:

|  |
| --- |
| public static final Uri CONTENT\_URI=  Uri.parse(«content:*// com.paad.seletondatabaseprovider/elements»);* |

Эти поля будут использоваться при доступе объекта *ContentResolve* к Источнику данных.

**Создание базы данных для Источника данных**

Чтобы инициализировать источник с данными, к которому вы планируете обращаться через объект СontentProvider, переопределите метод onCreate, как показано ниже в листинге. В результате базы данных создается и открывается только по мере необходимости.

|  |
| --- |
| **private** MySQLLiteOpenHelper myOpenHelper;  @Override  **public** Boolean onCreatr(){ *//Создайте исходную базу данных //Отложите открытие БД до тех пор, пока вам не понадобится выполнить запрос* myOpenHelper = **new** MySQLiteOpenHelper (getContext(), MySQLiteOpenHelper.DATABASE\_NAME, **null**, MySQLiteOpenHelper.DATABASE\_VERSION);  **return true**;  } |

**Выполнение запросов к Источнику данных**

Чтобы Источник данных поддерживал запросы, необходимо реализовать методы *query* и *getType*. Эти методы позволяют приложениям делиться данными друг с другом без необходимости публиковать отдельный интерфейс для каждого источника.

В общем случае использование объекта *ContentProvider* заключается в предоставлении доступа к БД.

Ниже показан каркас кода для реализации запросов внутри Источника данных с сипользованием исходной БД на основе SQLite.

|  |
| --- |
| @Override  **public** Cursor query (Uri uri, String [] projection, String selection, String[] selectionArgs, String sortOrder) {  *//откройте базу данных* SQLiteDatabase db;  **try** {  db=myOpenHelper.getWritableDatabase();  }  **catch** (SQLiteException ex)  {  Db = myOpenHelper.getReadableDatabase();  }  *//При необходимости подставьте сюда корректные SQL-выражения.* String groupBy = **null**;  String having = **null**;  *//Используйте SQLiteQueryBuilder для построения запроса к БД* SQLiteQueryBuilder queryBuilder = **new** SQLiteQueryBuilder();  *//Если это запрос к строке, ограничьте итоговую выборку переданным запросом.* **switch** (uriMatcher.match(uri)) {  Case SINGLE\_ROW :  String rowID = uri.getPathSegmets().get(1);  queryBuilder.appendWhere(KEY\_ID + “=” + rowID);  **default**: **break**;  }  *//Укажите таблицу, к которой будет выполнен запрос. Это может быть конкретная таблица или их совокупность.* queryBuilder.settables(MySQLiteOpenHelper.DATABASE\_TABLE);  *//Выполните запрос.* Cursor cursor = queryBuilder.query(db, projection, selection, selectionArgs, groupBy, having, sortOrder);  *//Верните результирующий курсор.* **return** cursor; } |

После выполнения запросов вы должны указать MIME-тип, чтобы идентифицировать возвращаемые данные. Для этого нудно переопределить метод *getType*, чтобы вернуть строку, которая уникальным образом описывает тип ваших данных.

Возвращаемый тип должен поддерживать одиночную запись и все записи целиком:

*vnd.android.cursor.item/vnd.<companyname>.<contenttype>* - одна запись.

*vnd.android.cursor.dir/vnd.<companyname>.<contenttype>* - все записи.

**Транзакции в Источниках данных**

Упростить доступ к запросам и транзакциям в Источнике данных можно, реализовав методы *delete*, *insert*, *update* и *query*.

Эти методы — интерфейс, используемый объектом *ContentResolver* для доступа к исходным данным. Они позволяют приложениям обмениваться данными в любой точке и не требуют разных интерфейсов для каждого источника данных.

|  |
| --- |
| @Override public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sort) { *// Если это запрос для получения строки, ограничьте результат одной строкой.* **switch** (uriMatcher.match(uri))   {  **case** SINGLE\_ROW :  *//* ***TODO: Изменять выборку, в зависимости от идентификатора строки, где: rowNumber = uri.getPathSegments().get(1));*** }  **return null**; }  @Override public Uri insert(Uri \_uri, ContentValues \_initialValues) {  long rowID = [ ... Добавляем новый элемент ... ]  *// Верните путь URI к последнему добавленному элементу.* **if** (rowID > 0)  { **return** ContentUris.withAppendedId(CONTENT\_URI, rowID);  }  **throw new** SQLException(**"Failed to add new item into "** + \_uri); }  @Override public int **delete**(Uri uri, String where, String[] whereArgs) {  **switch** (uriMatcher.match(uri)) {  **case** ALLROWS:  **case** SINGLE\_ROW:  **default**: **throw new** IllegalArgumentException(**"Unsupported URI:"** + uri);  } }  @Override public int update(Uri uri, ContentValues values, String where, String[]whereArgs) {  **switch** (uriMatcher.match(uri))  {  **case** ALLROWS:  **case** SINGLE\_ROW:  **default**: **throw new** IllegalArgumentException(**"Unsupported URI:"** + uri);  } } |

**Создание и использование Источников данных на примере работы с контактами**

Контакты в Android обладают встроенным API, который позволяет получать и изменять список контактов. Все контакты хранятся в базе данных SQLite, однако они не представляют единой таблицы. Для контактов отведено три таблицы, связанных отношением один-ко-многим: таблица для хранения информации о людях, таблица их телефонов и и таблица адресов их электронных почт. Но благодаря Android API мы можем абстрагироваться от связей между таблицами.

Общая форма получения контактов выглядит следующим образом:

|  |
| --- |
| ArrayList<String> **contacts** = **new** ArrayList<String>(); ContentResolver **contentResolver** = getContentResolver(); Cursor **cursor** = **contentResolver**.query(ContactsContract.Contacts.CONTENT\_URI, **null**, **null**, **null**, **null**);  **if**(cursor!=**null**){  **while** (cursor.moveToNext()){  *// получаем каждый контакт* String contact = cursor.getString(cursor.getColumnIndex(ContactsContract.Contacts.DISPLAY\_NAME\_PRIMARY));  *// добавляем контакт в список* contacts.add(contact);  }  cursor.close();  } |

Все контакты и сопутствующий функционал хранятся в специальных базах данных SQLite. Но нам не надо напрямую работать с ними. Мы можем воспользоваться объектом класса *Cursor*. Чтобы его получить, сначала вызывается метод *getContentResolver()*, который возвращает объект *ContentResolver*. Затем по цепочке вызывается метод *query()*. В этот метод передается ряд параметров, первый из которых представляет URI - ресурс, который мы хотим получить. Для обращения к базе данных контактов используется константа *ContactsContract.Contacts.CONTENT\_URI*.

Метод *contactsCursor.moveToNext()* позволяет последовательно перемещаться по записям контактов, считывая по одному контакту через вызов *contactsCursor.getString()*.

Итак, для доступа к контактам нам надо установить разрешение *android.permission.READ\_CONTACTS* в файле манифеста приложения:

|  |
| --- |
| *<?***xml version="1.0" encoding="utf-8"***?>* <**manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.example.eugene.contactsapp"**> <**uses-permission android:name="android.permission.READ\_CONTACTS"** /> <**application  android:allowBackup="true"  android:icon="@mipmap/ic\_launcher"  android:label="@string/app\_name"  android:supportsRtl="true"  android:theme="@style/AppTheme"**>  <**activity android:name=".MainActivity"**>  <**intent-filter**>  <**action android:name="android.intent.action.MAIN"** />   <**category android:name="android.intent.category.LAUNCHER"** />  </**intent-filter**>  </**activity**> </**application**> </**manifest**> |

Для вывода списка контактов в файле activity\_main.xml определим следующую разметку интерфейса:

|  |
| --- |
| <**LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  android:id="@+id/activity\_main"  android:orientation="vertical"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"**>  <**TextView  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:text="Контакты"**> </**TextView**>  <**ListView  android:id="@+id/contactList"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"**> </**ListView**> </**LinearLayout**> |

Для вывода списка контактов воспользуемся элементом *ListView*. И в классе *MainActivity* получим контакты:

|  |
| --- |
| **package** com.example.eugene.contactsapp; **import** android.content.ContentResolver; **import** android.content.pm.PackageManager; **import** android.database.Cursor; **import** android.provider.ContactsContract; **import** android.support.v4.app.ActivityCompat; **import** android.support.v4.content.ContextCompat; **import** android.support.v7.app.AppCompatActivity; **import** android.os.Bundle; **import** android.util.Log; **import** android.widget.ArrayAdapter; **import** android.widget.ListView; **import** android.Manifest; **import** android.widget.Toast; **import** java.util.ArrayList;  **public class** MainActivity **extends** AppCompatActivity {  **private static final int *REQUEST\_CODE\_READ\_CONTACTS***=1;  **private static boolean** *READ\_CONTACTS\_GRANTED* =**false**;  ListView **contactList**;  ArrayList<String> **contacts** = **new** ArrayList<String>();    @Override  **protected void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.activity\_main);  **contactList** = (ListView) findViewById(R.id.contactList);  *// получаем разрешения* **int** hasReadContactPermission = ContextCompat.checkSelfPermission(**this**, Manifest.permission.READ\_CONTACTS);  *// если устройство до API 23, устанавливаем разрешение* **if**(hasReadContactPermission == PackageManager.PERMISSION\_GRANTED){  *READ\_CONTACTS\_GRANTED* = **true**;  }  **else**{  *// вызываем диалоговое окно для установки разрешений* ActivityCompat.requestPermissions(**this**, **new** String[]{Manifest.permission.READ\_CONTACTS}, ***REQUEST\_CODE\_READ\_CONTACTS***);  }  *// если разрешение установлено, загружаем контакты* **if** (*READ\_CONTACTS\_GRANTED*){  loadContacts();  }  }   @Override  **public void** onRequestPermissionsResult(**int** requestCode, String[] permissions, **int**[] grantResults){   **switch** (requestCode){  **case *REQUEST\_CODE\_READ\_CONTACTS***:  **if**(grantResults.**length**>0 && grantResults[0] == PackageManager.PERMISSION\_GRANTED){  *READ\_CONTACTS\_GRANTED* = **true**;  }  }  **if**(*READ\_CONTACTS\_GRANTED*){  loadContacts();  }  **else**{  Toast.makeText(**this**, **"Требуется установить разрешения"**, Toast.LENGTH\_LONG).show();  }  }   **private void** loadContacts(){  ContentResolver contentResolver = getContentResolver();  Cursor cursor = contentResolver.query(ContactsContract.Contacts.CONTENT\_URI, **null**, **null**, **null**, **null**);  **if**(cursor!=**null**){  **while** (cursor.moveToNext()) {   *// получаем каждый контакт* String contact = cursor.getString(  cursor.getColumnIndex(ContactsContract.Contacts.DISPLAY\_NAME\_PRIMARY));  *// добавляем контакт в список* **contacts**.add(contact);  }  cursor.close();  }   *// создаем адаптер* ArrayAdapter<String> adapter = **new** ArrayAdapter<String>(**this**,  android.R.layout.***simple\_list\_item\_1***, **contacts**);  *// устанавливаем для списка адаптер* **contactList**.setAdapter(adapter);  } } |

**Задание на лабораторную работу**

Необходимо переработать механизм работы с данными в приложении, разработанном в 4 лабораторной работе, через реализацию собственного источника данных.

1. Приложение должно использовать реализованную, в рамках лабораторной работы №4, базу данных.
2. Реализовать на основе данной базы данных источник данных. В процессе работы источника данных должны принимать участие все разработанные таблицы. Возможен вариант реализации частей источника данных замещающего работу сложных комплексных SQL запросов.
3. Источник данных должен реализовывать полный список CRUD операций, а также все основные необходимые в его работе механизмы.
4. Реализованный источник данных должен быть доступен сторонним приложениям для взаимодействия.
5. В приложении все запросы для получения данных должны взаимодействовать только с вашим источником данных, т.е все запросы к БД должны проходить через источник данных.

**Контрольные вопросы**

1. Что такое источник данных.
2. Принципы работы с Базами данных в OS android. Класс Cursor.
3. Класс SQLiteOpenHelper.
4. Особенность выполнения запросов к базе данных при помощи методов query insert update и delete
5. Класс ContentValues. Его назначение.
6. Необходимые действия для создания источника данных.
7. Варианты backend при организации источника данных.
8. UriMatcher. Описание доступа к Источнику данных.
9. Методы источника данных и результат их выполнения.
10. Метод GetType источника данных. Его назначение.
11. Класс ContentResolver. Запросы к источнику данных.
12. Стандартные источники данных в Android. Перечислите.
13. Источник данных MediaStore
14. Источник данных CallLog и ContactsContract. Структура хранения данных.
15. Связь жизненного цикла курсора и activity. Как делается и зачем необходимо?

**Список рекомендуемой литературы**

1. Майер, Р. Android 4. Программирование приложений для планшетных компьютеров и смартфонов. / Пер. с англ. – М.: Эксмо, 2013. – 816 С.
2. Коматинени, С. Android 4 для профессионалов. Создание приложений для планшетных компьютеров и смартфонов./ Коматинени С., Маклин Д. / Пер. с англ. – М.: Вильямс, 2012. – 880 С.