Пример создания собственного ContentProvider для работы с SQLite БД часть 2



[Часть 1](http://www.nerdgrl.org/ru/programming/sqlite-contentprovider-1/)  
[Часть 3](http://www.nerdgrl.org/ru/programming/contentprovider-cursorloader-3/)

В [предыдущей части](http://www.nerdgrl.org/ru/programming/contentprovider-1/) я рассказывала что такое Contract Class и как его создать.  
Сегодня мы разберем из чего состоит ContentProvider, а также один из вариантов написания данного класса.

Как вы помните, обращаться к БД мы будем с помощью Content URI, которая несет в себе информацию о конкретном ContentProvider и о необходимой нам таблице. Вот один из примеров Content URI:  
content://org.nerdgrl.examples.contentproviderexample.provider.ContractClass/students  
Здесь мы видим схему данных (content://), уникальный идентификатор нашего провайдера — authority (org.nerdgrl.examples.contentproviderexample.provider.ContractClass), и имя таблицы (students).

В Android SDK есть класс UriMatcher, который хранит соответствие между Uri и неким заданным значение integer. Это значение можно использовать в операторе switch, чтобы описать поведение для каждого Content URI.

Для начала зададим четыре константы, соответствующие четырем возможным типам запроса к нашей БД:

private static final int STUDENTS = 1;

private static final int STUDENTS\_ID = 2;

private static final int CLASSES = 3;

private static final int CLASSES\_ID = 4;

Затем объявим переменную класса UriMatcher

private static final UriMatcher sUriMatcher;

И в static блоке подготовим ее к использованию:

static {

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

sUriMatcher.addURI(ContractClass.AUTHORITY, "students", STUDENTS);

sUriMatcher.addURI(ContractClass.AUTHORITY, "students/#", STUDENTS\_ID);

sUriMatcher.addURI(ContractClass.AUTHORITY, "classes", CLASSES);

sUriMatcher.addURI(ContractClass.AUTHORITY, "classes/#", CLASSES\_ID);

}

Также нам необходимо задать проекции для выборки столбцов в запросе, они пригодятся нам в методе query():

private static HashMap sStudentsProjectionMap;

private static HashMap sClassesProjectionMap;

Для проекции по умолчанию мы возьмем весь список столбцов:

static {

for(int i=0; i < ContractClass.Students.DEFAULT\_PROJECTION.length; i++) {

sStudentsProjectionMap.put(

ContractClass.Students.DEFAULT\_PROJECTION[i],

ContractClass.Students.DEFAULT\_PROJECTION[i]);

}

sClassesProjectionMap = new HashMap();

for(int i=0; i < ContractClass.Classes.DEFAULT\_PROJECTION.length; i++) {

sClassesProjectionMap.put(

ContractClass.Classes.DEFAULT\_PROJECTION[i],

ContractClass.Classes.DEFAULT\_PROJECTION[i]);

}

}

Теперь определим класс DBHelper, с помощью которого мы будем создавать базу данных и обращаться к ней в методах ContentProvider'a. Здесь мы зададим имя БД, имена таблиц, столбцов, а также запросы для создания таблиц и процедуры создания-обновления БД:

private static class DatabaseHelper extends SQLiteOpenHelper {

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

public static final String DATABASE\_TABLE\_STUDENTS = ContractClass.Students.TABLE\_NAME;

public static final String DATABASE\_TABLE\_CLASSES = ContractClass.Classes.TABLE\_NAME;

public static final String KEY\_ROWID = "\_id";

public static final String KEY\_FIRST\_NAME = "first\_name";

public static final String KEY\_SECOND\_NAME = "second\_name";

public static final String KEY\_CLASS\_LETTER = "class\_letter";

public static final String KEY\_FK\_CLASS\_ID = "fk\_class\_id";

public static final String KEY\_AVERAGE\_SCORE = "average\_score";

public static final String KEY\_CLASS\_NUMBER = "class\_number";

private static final String DATABASE\_CREATE\_TABLE\_STUDENTS =

"create table "+ DATABASE\_TABLE\_STUDENTS + " ("

+ KEY\_ROWID + " integer primary key autoincrement, "

+ KEY\_FIRST\_NAME + " string , "

+ KEY\_SECOND\_NAME + " string , "

+ KEY\_AVERAGE\_SCORE + " real , "

+ KEY\_FK\_CLASS\_ID + " integer, "

+" foreign key ("+KEY\_FK\_CLASS\_ID+") references "+DATABASE\_TABLE\_CLASSES+"("+KEY\_ROWID+"));";

private static final String DATABASE\_CREATE\_TABLE\_CLASSES =

"create table "+ DATABASE\_TABLE\_CLASSES + " ("

+ KEY\_ROWID + " integer primary key autoincrement, "

+ KEY\_CLASS\_NUMBER + " string , "

+ KEY\_CLASS\_LETTER + " string );";

private Context ctx;

DatabaseHelper(Context context) {

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

ctx = context;

}

@Override

public void onCreate(SQLiteDatabase db) {

db.execSQL(DATABASE\_CREATE\_TABLE\_STUDENTS);

db.execSQL(DATABASE\_CREATE\_TABLE\_CLASSES);

}

@Override

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

db.execSQL("DROP TABLE IF EXISTS " + DATABASE\_TABLE\_STUDENTS);

db.execSQL("DROP TABLE IF EXISTS " + DATABASE\_TABLE\_CLASSES);

onCreate(db);

}

}

Теперь подробнее о методах самого ContentProvider'a:

onCreate() - инициализирует ContentProvider. Провайдер будет создан как только вы обратитесь к нему с помощью ContentResolver'a  
query() - извлекает данные из БД, и возвращает их в виде Cursor  
insert() - добавляет новые данные в БД, возвращает uri новой записи  
update() - обновляет строки в БД согласно заданным условиям  
delete() - удаляет данные  
getType() - возвращает MIME-тип для заданной content URI

Следует помнить, что все перечисленные методы кроме onCreate() могут выполняться одновременно в нескольких потоках, и поэтому должны быть потоко-безопасными (thread-safe).

В методе onCreate() создадим наш DBHelper:

@Override

public boolean onCreate() {

dbHelper = new DatabaseHelper(getContext());

return true;

}

В реализации метода getType() мы просто будем возвращать тип данных из нашего ContractClass'a:

@Override

public String getType(Uri uri) {

switch (sUriMatcher.match(uri)) {

case STUDENTS:

return ContractClass.Students.CONTENT\_TYPE;

case STUDENTS\_ID:

return ContractClass.Students.CONTENT\_ITEM\_TYPE;

case CLASSES:

return ContractClass.Classes.CONTENT\_TYPE;

case CLASSES\_ID:

return ContractClass.Classes.CONTENT\_ITEM\_TYPE;

default:

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

}

}

Рассмотрим метод insert().

Добавлять строки можно только в таблицы, поэтому сделаем фильтр для тех Content Uri, которые не подходят:

if (sUriMatcher.match(uri) != STUDENTS &&

sUriMatcher.match(uri) != CLASSES) {

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

}

Далее получим экземпляр БД и создадим структуру для хранения данных новой строки таблицы:

SQLiteDatabase db = dbHelper.getWritableDatabase();

ContentValues values;

if (initialValues != null) {

values = new ContentValues(initialValues);

}

else {

values = new ContentValues();

}

И определим в какую таблицу нужно добавить новые данные:

switch (sUriMatcher.match(uri)) {

case STUDENTS:

rowId = db.insert(ContractClass.Students.TABLE\_NAME,

ContractClass.Students.COLUMN\_NAME\_FIRST\_NAME,

values);

if (rowId > 0) {

rowUri = ContentUris.withAppendedId(ContractClass.Students.CONTENT\_ID\_URI\_BASE, rowId);

getContext().getContentResolver().notifyChange(rowUri, null);

}

break;

case CLASSES:

...

break;

}

Здесь хочу обратить внимание на строку:

getContext().getContentResolver().notifyChange(rowUri, null);

Именно она отвечает за обновление данных в CursorAdapter(и, соответственно, в нашем ListView, где мы его будем использовать).

Полный код метода:

@Override

public Uri insert(Uri uri, ContentValues initialValues) {

if (

sUriMatcher.match(uri) != STUDENTS &&

sUriMatcher.match(uri) != CLASSES

) {

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

}

SQLiteDatabase db = dbHelper.getWritableDatabase();

ContentValues values;

if (initialValues != null) {

values = new ContentValues(initialValues);

}

else {

values = new ContentValues();

}

long rowId = -1;

Uri rowUri = Uri.EMPTY;

switch (sUriMatcher.match(uri)) {

case STUDENTS:

rowId = db.insert(ContractClass.Students.TABLE\_NAME,

ContractClass.Students.COLUMN\_NAME\_FIRST\_NAME,

values);

if (rowId > 0) {

rowUri = ContentUris.withAppendedId(ContractClass.Students.CONTENT\_ID\_URI\_BASE, rowId);

getContext().getContentResolver().notifyChange(rowUri, null);

}

break;

case CLASSES:

rowId = db.insert(ContractClass.Classes.TABLE\_NAME,

ContractClass.Classes.COLUMN\_NAME\_CLASS\_NUMBER,

values);

if (rowId > 0) {

rowUri = ContentUris.withAppendedId(ContractClass.Classes.CONTENT\_ID\_URI\_BASE, rowId);

getContext().getContentResolver().notifyChange(rowUri, null);

}

break;

}

return rowUri;

}

Теперь разберем метод query():

@Override

public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder) {

SQLiteQueryBuilder qb = new SQLiteQueryBuilder();

String orderBy = null;

switch (sUriMatcher.match(uri)) {

case STUDENTS:

qb.setTables(ContractClass.Students.TABLE\_NAME);

qb.setProjectionMap(sStudentsProjectionMap);

orderBy = ContractClass.Students.DEFAULT\_SORT\_ORDER;

break;

case STUDENTS\_ID:

qb.setTables(ContractClass.Students.TABLE\_NAME);

qb.setProjectionMap(sStudentsProjectionMap);

qb.appendWhere(ContractClass.Students.\_ID + "=" + uri.getPathSegments().get(ContractClass.Students.STUDENTS\_ID\_PATH\_POSITION));

orderBy = ContractClass.Students.DEFAULT\_SORT\_ORDER;

break;

case CLASSES:

qb.setTables(ContractClass.Classes.TABLE\_NAME);

qb.setProjectionMap(sClassesProjectionMap);

orderBy = ContractClass.Classes.DEFAULT\_SORT\_ORDER;

break;

case CLASSES\_ID:

qb.setTables(ContractClass.Classes.TABLE\_NAME);

qb.setProjectionMap(sClassesProjectionMap);

qb.appendWhere(ContractClass.Classes.\_ID + "=" + uri.getPathSegments().get(ContractClass.Classes.CLASSES\_ID\_PATH\_POSITION));

orderBy = ContractClass.Classes.DEFAULT\_SORT\_ORDER;

break;

default:

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

}

SQLiteDatabase db = dbHelper.getReadableDatabase();

Cursor c = qb.query(db, projection, selection, selectionArgs, null, null, orderBy);

c.setNotificationUri(getContext().getContentResolver(), uri);

return c;

}

Здесь используется объект класса SQLiteQueryBuilder для построения запроса. Методы setTables() и setProjectionMap() задают таблицу и набор столбцов для выборки. Для запроса к определенной строке используется appendWhere():

qb.appendWhere(ContractClass.Classes.\_ID + "=" + uri.getPathSegments().get(ContractClass.Classes.CLASSES\_ID\_PATH\_POSITION));

который добавляет условие WHERE к запросу. Как вы могли заметить, здесь мы как раз используем ContractClass.Classes.CLASSES\_ID\_PATH\_POSITION (в данном случае для таблицы Classes) - таким образом мы определяем, что номер требуемой строки идет на первой позиции Content Uri сразу после имени таблицы (content://<authority>/classes/1)

Метод update(), по своей структуре, аналогичен предыдущему:

@Override

public int update(Uri uri, ContentValues values, String where, String[] whereArgs) {

SQLiteDatabase db = dbHelper.getWritableDatabase();

int count;

String finalWhere;

String id;

switch (sUriMatcher.match(uri)) {

case STUDENTS:

count = db.update(ContractClass.Students.TABLE\_NAME, values, where, whereArgs);

break;

case STUDENTS\_ID:

id = uri.getPathSegments().get(ContractClass.Students.STUDENTS\_ID\_PATH\_POSITION);

finalWhere = ContractClass.Students.\_ID + " = " + id;

if (where !=null) {

finalWhere = finalWhere + " AND " + where;

}

count = db.update(ContractClass.Students.TABLE\_NAME, values, finalWhere, whereArgs);

break;

case CLASSES:

count = db.update(ContractClass.Classes.TABLE\_NAME, values, where, whereArgs);

break;

case CLASSES\_ID:

id = uri.getPathSegments().get(ContractClass.Classes.CLASSES\_ID\_PATH\_POSITION);

finalWhere = ContractClass.Classes.\_ID + " = " + id;

if (where !=null) {

finalWhere = finalWhere + " AND " + where;

}

count = db.update(ContractClass.Classes.TABLE\_NAME, values, finalWhere, whereArgs);

break;

default:

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

}

getContext().getContentResolver().notifyChange(uri, null);

return count;

}

Обратите внимание на следующую часть метода:

finalWhere = ContractClass.Students.\_ID + " = " + id;

if (where !=null) {

finalWhere = finalWhere + " AND " + where;

}

Здесь мы добавляем к уже имеющемуся условию запроса, условие равенства определенному id записи.

Код метода delete():

@Override

public int delete(Uri uri, String where, String[] whereArgs) {

SQLiteDatabase db = dbHelper.getWritableDatabase();

String finalWhere;

int count;

switch (sUriMatcher.match(uri)) {

case STUDENTS:

count = db.delete(ContractClass.Students.TABLE\_NAME,where,whereArgs);

break;

case STUDENTS\_ID:

finalWhere = ContractClass.Students.\_ID + " = " + uri.getPathSegments().get(ContractClass.Students.STUDENTS\_ID\_PATH\_POSITION);

if (where != null) {

finalWhere = finalWhere + " AND " + where;

}

count = db.delete(ContractClass.Students.TABLE\_NAME,finalWhere,whereArgs);

break;

case CLASSES:

count = db.delete(ContractClass.Classes.TABLE\_NAME,where,whereArgs);

break;

case CLASSES\_ID:

finalWhere = ContractClass.Classes.\_ID + " = " + uri.getPathSegments().get(ContractClass.Classes.CLASSES\_ID\_PATH\_POSITION);

if (where != null) {

finalWhere = finalWhere + " AND " + where;

}

count = db.delete(ContractClass.Classes.TABLE\_NAME,finalWhere,whereArgs);

break;

default:

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

}

getContext().getContentResolver().notifyChange(uri, null);

return count;

}

В [следующей части](http://www.nerdgrl.org/ru/programming/contentprovider-cursorloader-3/) я расскажу как использовать ContentProvider в связке с CursorLoader. Полный исходный код можно найти [здесь](http://www.nerdgrl.org/ru/wp-content/uploads/2015/06/ContentProviderExample.zip).