**Разработка приложений**

**Платформа Google Android** уже представляет собой заметное явление в области программного обеспечения для мобильных устройств. Новой платформой заинтересовались ведущие мировые производители мобильной электроники и сотовые операторы, и многие из них уже выпустили на рынок устройства, работающие под управлением **Android**.

В чем же заключается уникальность платформы Android? Основная идея Google состоит в том, что компания предлагает в открытый доступ исходные коды своей операционной системы, предлагает набор удобных инструментов для разработки и хорошо документированное SDK, что должно со временем привести к появлению большого количества программного обеспечения для этой платформы. В дальнейшем у Android есть все шансы стать самым успешным проектом для мобильных телефонов.

Аналитики и эксперты ИТ-рынка прогнозируют для платформы Google Android хорошие коммерческие перспективы. Android захватывает рынок мобильных телефонов, постепенно вытесняя с него общепризнанных лидеров. Google Android устанавливается не только на смартфоны, данная платформа подходит и для нетбуков.

Еще одним шагом в развитии Google Android стало открытие в октябре 2008 года онлайн-магазина приложений - **Android Market**, в котором можно приобрести программы и другой софт для устройств на базе новой платформы. Кроме того, теперь для разработчиков программного обеспечения появилась возможность брать плату за свои приложения в Android Market, что делает разработку приложений под эту платформу еще более привлекательной.

* [База данных SQLite и контентпровайдеры](http://androidfanclub.ru/programming/%D0%B1%D0%B0%D0%B7%D0%B0-%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85-sqlite-%D0%B8-%D0%BA%D0%BE%D0%BD%D1%82%D0%B5%D0%BD%D1%82%D0%BF%D1%80%D0%BE%D0%B2%D0%B0%D0%B9%D0%B4%D0%B5%D1%80%D1%8B)
* [Базовые виджеты](http://androidfanclub.ru/programming/%D0%B1%D0%B0%D0%B7%D0%BE%D0%B2%D1%8B%D0%B5-%D0%B2%D0%B8%D0%B4%D0%B6%D0%B5%D1%82%D1%8B)
* [Базовые сведения о платформе Android](http://androidfanclub.ru/programming/%D0%B1%D0%B0%D0%B7%D0%BE%D0%B2%D1%8B%D0%B5-%D1%81%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D1%8F-%D0%BE-%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B5-android)
* [Виджеты-списки и привязка данных](http://androidfanclub.ru/programming/%D0%B2%D0%B8%D0%B4%D0%B6%D0%B5%D1%82%D1%8B-%D1%81%D0%BF%D0%B8%D1%81%D0%BA%D0%B8-%D0%B8-%D0%BF%D1%80%D0%B8%D0%B2%D1%8F%D0%B7%D0%BA%D0%B0-%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)
* [Графика](http://androidfanclub.ru/programming/%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%B0)
* [Графический интерфейс пользователя](http://androidfanclub.ru/programming/%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9-%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81-%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8F)
* [Диалоговые окна](http://androidfanclub.ru/programming/%D0%B4%D0%B8%D0%B0%D0%BB%D0%BE%D0%B3%D0%BE%D0%B2%D1%8B%D0%B5-%D0%BE%D0%BA%D0%BD%D0%B0)
* [Компоненты Android-приложения](http://androidfanclub.ru/programming/%D0%BA%D0%BE%D0%BC%D0%BF%D0%BE%D0%BD%D0%B5%D0%BD%D1%82%D1%8B-android-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F)
* [Меню](http://androidfanclub.ru/programming/%D0%BC%D0%B5%D0%BD%D1%8E)
* [Приемники широковещательных намерений](http://androidfanclub.ru/programming/%D0%BF%D1%80%D0%B8%D0%B5%D0%BC%D0%BD%D0%B8%D0%BA%D0%B8-%D1%88%D0%B8%D1%80%D0%BE%D0%BA%D0%BE%D0%B2%D0%B5%D1%89%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D1%85-%D0%BD%D0%B0%D0%BC%D0%B5%D1%80%D0%B5%D0%BD%D0%B8%D0%B9-0)
* [Работа с файлами и сохранение пользовательских настроек](http://androidfanclub.ru/programming/%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0-%D1%81-%D1%84%D0%B0%D0%B9%D0%BB%D0%B0%D0%BC%D0%B8-%D0%B8-%D1%81%D0%BE%D1%85%D1%80%D0%B0%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5-%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D0%BA%D0%B8%D1%85-%D0%BD%D0%B0%D1%81%D1%82%D1%80%D0%BE%D0%B5%D0%BA)
* [Ресурсы, активы и локализация приложений](http://androidfanclub.ru/programming/%D1%80%D0%B5%D1%81%D1%83%D1%80%D1%81%D1%8B-%D0%B0%D0%BA%D1%82%D0%B8%D0%B2%D1%8B-%D0%B8-%D0%BB%D0%BE%D0%BA%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B9)
* [Службы](http://androidfanclub.ru/programming/%D1%81%D0%BB%D1%83%D0%B6%D0%B1%D1%8B-0)
* [Создание анимации](http://androidfanclub.ru/programming/%D1%81%D0%BE%D0%B7%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-%D0%B0%D0%BD%D0%B8%D0%BC%D0%B0%D1%86%D0%B8%D0%B8)
* [Создание приложения для Android](http://androidfanclub.ru/programming/%D1%81%D0%BE%D0%B7%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F-%D0%B4%D0%BB%D1%8F-android)
* [Среда разработки](http://androidfanclub.ru/programming/%D1%81%D1%80%D0%B5%D0%B4%D0%B0-%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8)
* [Уведомления](http://androidfanclub.ru/programming/%D1%83%D0%B2%D0%B5%D0%B4%D0%BE%D0%BC%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F)
* [Управление деятельностями](http://androidfanclub.ru/programming/%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5-%D0%B4%D0%B5%D1%8F%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D1%8F%D0%BC%D0%B8)

**База данных SQLite и контентпровайдеры**

В этом разделе рассматривается **работа со встроенной базой данных SQLite** и организация доступа к данным с помощью контентпровайдеров.

***Контентпровайдер*** — один из четырех фундаментальных компонентов **Android-приложений**, который обеспечивает информационное наполнение для приложений. Контентпровайдер нужен, если есть необходимость совместного использования данных между приложениями. Если вы не собираетесь давать данные другим приложениям, можете работать с данными непосредственно через классы, предоставляющие интерфейс для взаимодействия с**базой данных SQLite.**

* [База данных SQLite](http://androidfanclub.ru/programming/%D0%B1%D0%B0%D0%B7%D0%B0-%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85-sqlite)
* [Запросы к контент-провайдеру](http://androidfanclub.ru/programming/%D0%B7%D0%B0%D0%BF%D1%80%D0%BE%D1%81%D1%8B-%D0%BA-%D0%BA%D0%BE%D0%BD%D1%82%D0%B5%D0%BD%D1%82-%D0%BF%D1%80%D0%BE%D0%B2%D0%B0%D0%B9%D0%B4%D0%B5%D1%80%D1%83)
* [Контент-провайдеры](http://androidfanclub.ru/programming/%D0%BA%D0%BE%D0%BD%D1%82%D0%B5%D0%BD%D1%82-%D0%BF%D1%80%D0%BE%D0%B2%D0%B0%D0%B9%D0%B4%D0%B5%D1%80%D1%8B)
* [Практическое приложение для работы с базой данных](http://androidfanclub.ru/programming/%D0%BF%D1%80%D0%B0%D0%BA%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5-%D0%B4%D0%BB%D1%8F-%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D1%8B-%D1%81-%D0%B1%D0%B0%D0%B7%D0%BE%D0%B9-%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)
* [Создание контентпровайдера](http://androidfanclub.ru/programming/%D1%81%D0%BE%D0%B7%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-%D0%BA%D0%BE%D0%BD%D1%82%D0%B5%D0%BD%D1%82%D0%BF%D1%80%D0%BE%D0%B2%D0%B0%D0%B9%D0%B4%D0%B5%D1%80%D0%B0)

**База данных SQLite**

**Платформа Android** предоставляет функции управления базой данных, которые позволяют сохранять сложные коллекции данных. Android также поставляется с инструментом управления базой **данных sqlite3**, который дает возможность просматривать содержание таблиц, выполнять *команды SQL* и исполнять другие полезные функции на базах ***данных SQLite***.

Все базы данных, ***SQLite*** и другие, сохраняются на устройстве в каталоге***/data/data/package\_name/*** базы данных. **SQLite** хранит всю базу данных (*включая определения, таблицы, индексы и данные*) в единственном стандартном файле. ***Библиотеки Android*** обеспечивают набор классов для создания и управления базами данных SQLite.

* [Создание базы данных: класс SQLiteOpenHelper](http://androidfanclub.ru/programming/%D1%81%D0%BE%D0%B7%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-%D0%B1%D0%B0%D0%B7%D1%8B-%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85-%D0%BA%D0%BB%D0%B0%D1%81%D1%81-sqliteopenhelper)
* [Управление базой данных: класс SQLiteDatabase](http://androidfanclub.ru/programming/%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5-%D0%B1%D0%B0%D0%B7%D0%BE%D0%B9-%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85-%D0%BA%D0%BB%D0%B0%D1%81%D1%81-sqlitedatabase)

**Создание базы данных: класс SQLiteOpenHelper**

В библиотеке Android есть **класс SQLiteOpenHelper** для создания базы данных. **Класс SQLiteOpenHelper** содержит два абстрактных метода:

* **onCreate()** — вызывается при первом создании базы данных;
* **onUpgrade(**) — вызывается при модификации базы данных.

В приложении создается свой класс, наследуемый от ***SQLiteOpenHelper***. В этом классе необходимо реализовать вышеперечисленные методы, описав в них логику создания и модификации вашей базы.

В этом же классе принято объявлять открытые строковые константы для названия таблиц и полей создаваемой базы данных, которые клиенты могут использовать для определения столбцов при выполнении запросов к базе данных. Тщательно документируйте тип данных каждого столбца, т. к. клиенту требуется эта информация для обращения к данным. Например, так можно объявить константы для таблицы Contact:

public static final String TAbLE\_NAME = "contact";

public static final String NAME "first\_name";

public static final String PHONE "phone";

Ваш класс, расширяющий SQLiteOpenHelper, также неявно наследует **интерфейс BaseColumns**, в котором определена строковая **константа \_id**, представляющая имя поля для идентификаторов записей. В создаваемых таблицах базы данных **поле \_id** должно иметь**тип integer primary key autoincrement**. Описатель **autoincrement** является необязательным.

В методе обратного вызова **onCreate()** необходимо реализовать логику создания таблиц и при необходимости заполнить их начальными данными, например:

@Override

public void onCreate(SQLiteDatabase db){

db.execSQL("CREATE TAbLE + TAbLE\_NAME

+ (\_id INTEGER PRIMARY KEY AUTOINCREMENT,

+ COL\_NAME + TEXT, + COL\_PHONE + TEXT);");

Метод **onUpdate()** вызывается при установке обновлений программы с измененной структурой таблиц. В методе обратного вызова onUpgrade() можно, например, реализовать запрос в базу данных на уничтожение таблицы ( **drop table**), после чего вновь вызвать метод**onCreate()** для создания версии таблицы с обновленной структурой, например, так:

@Override

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

{db.execSQL("DROP TAbLE IF EXISTS + TAbLE\_NAME);

onCreate (db);

В реальном приложении изменение структуры базы данных и ее таблиц, конечно, должно происходить без потери пользовательских данных.

**Управление базой данных: класс SQLiteDatabase**

В библиотеке Android для управления базой данных SQLite существует **класс SQLiteDatabase**. В **классе SQLiteDatabase** определены методы для чтения, добавления, удаления, изменения данных.

Для чтения данных используют **вызов метода query()**:

Cursor query(

String table,

String[] columns,

String selection,

String[] selectionargs,

String groupBy,

String having,

String sortOrder)

В **метод query()** передают семь параметров:

1. **table** — имя таблицы, к которой передается запрос;
2. **columns**— список имен возвращаемых полей. При передаче null возвращаются все столбцы;
3. **selection**— параметр, формирующий выражение where (исключая сам оператор where) . Значение null возвращает все строки;
4. **selectionArgs** — значения аргументов фильтра;
5. **groupBy**—. параметр, формирующий выражение group ву (исключая сам оператор group ву). Если group ву не нужен, передается null;
6. **having**— параметр, формирующий выражение having (исключая сам оператор having) . Если не нужен, передается null;
7. **sortOrder**— параметр, форматирующий выражение order ву (исключая сам оператор order ву). При сортировке по умолчанию передается null.

**Объект Cursor,** возвращаемый **методом query()**, обеспечивает доступ к набору записей результирующей выборки. Для обработки возвращаемых данных **объект Cursor** имеет набор методов для чтения каждого типа данных —**getString(), getlnt() И getFloat().**

Для вставки новой записи в ***базу данных SQLite*** используется **метод insert()**:

long insert(String table,String nullColumnHack,ContentValues values)

В **метод insert()**необходимо передать три параметра:

* **table** — имя таблицы, в которую будет вставлена запись;
* **nullColumnHack**— в ***базе данных SQLite*** не разрешается вставлять полностью пустую строку, и если строка, полученная от клиента контент провайдера, будет пустой, то только этому столбцу явно будет назначено значение null;
* **values** — карта отображений (***класс Мар*** и его наследники), передаваемая клиентом контентпровайдера, которая содержит пары ключзначение. Ключи в карте должны быть названиями столбцов таблицы, значения — вставляемыми данными.

**Метод insert()** возвращает ***идентификатор \_id*** вставленной строки или -1 в случае ошибки.

Для обновления и удаления записей в базе данных используют соответственно **методы update() и delete():**

int update(String table,ContentValues values,

String whereClause,String[]whereArgs)

int delete (String table,String whereClause,String[]whereArgs)

В этих методах два последних параметра формируют ***SQLвыражение WHERE*** аналогично рассмотренному **методу query()** для чтения данных. Эти методы возвращают число модифицированных или удаленных строк.

Кроме вышеперечисленных методов, в этом классе также определены различные методы для выполнения других общих задач управления базой данных.

**Запросы к контент-провайдеру**

Чтобы сделать запрос к контент-провайдеру из клиентского приложения, необходимо три обязательных параметра: **URI**, который идентифицирует провайдера, имена запрашиваемых полей данных и типы данных для этих полей.

Если вы запрашиваете отдельную запись, также необходим идентификатор этой записи. Рассмотрим теперь процесс создания запросов к контент-провайдеру.

* [Добавление записей](http://androidfanclub.ru/programming/%D0%B4%D0%BE%D0%B1%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5-%D0%B7%D0%B0%D0%BF%D0%B8%D1%81%D0%B5%D0%B9)
* [Изменение записи](http://androidfanclub.ru/programming/%D0%B8%D0%B7%D0%BC%D0%B5%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5-%D0%B7%D0%B0%D0%BF%D0%B8%D1%81%D0%B8)
* [Позиционирование курсора](http://androidfanclub.ru/programming/%D0%BF%D0%BE%D0%B7%D0%B8%D1%86%D0%B8%D0%BE%D0%BD%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5-%D0%BA%D1%83%D1%80%D1%81%D0%BE%D1%80%D0%B0)
* [Удаление записей](http://androidfanclub.ru/programming/%D1%83%D0%B4%D0%B0%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5-%D0%B7%D0%B0%D0%BF%D0%B8%D1%81%D0%B5%D0%B9)
* [Чтение возвращаемых значений](http://androidfanclub.ru/programming/%D1%87%D1%82%D0%B5%D0%BD%D0%B8%D0%B5-%D0%B2%D0%BE%D0%B7%D0%B2%D1%80%D0%B0%D1%89%D0%B0%D0%B5%D0%BC%D1%8B%D1%85-%D0%B7%D0%BD%D0%B0%D1%87%D0%B5%D0%BD%D0%B8%D0%B9)

# Добавление записей

Чтобы добавить новую запись в контент-провайдер, сначала надо создать **объект Мар— карту** с парами ключ-значение в **объекте Contentvalues**, где каждый ключ соответствует имени столбца в контент-провайдере, а значение — конкретным данным для новой записи в этом столбце. После этого вызывается **метод ContentResolver.insert()**, которому надо передать **URI** провайдера и созданный **объект ContentValues**. Этот метод возвращает полный **URI**новой записи — т. е. ***URI контент-провайдера*** с добавленным в конце строки идентификатором для новой записи.

Вы можете использовать возвращаемый **URI**, чтобы сделать новый запрос и получить уже обновленную выборку данных. Например, код для добавления новой записи может выглядеть так:

ContentValues values new ContentValues(2);

формируем данные для вставки

values.put(ContactDbHelper.NAME,textName.getText().toString());

values.put(ContactDbHelper.PHONE,textPhone.getText().toString());

вставляем данные

getContentResolver().insert(ContactProvider.CONTENT\_URI, values);

обновляем выборку

cursor.requery() ;

**Изменение записи**

Содержащуюся в записи информацию можно изменить, используя вызов **метода update()**. Например, так:

ContentValues values new ContentValues(2);

values.put(ContactDbHelper.NAME,textName.getText().toString());

values.put(ContactDbHelper.PHONE,textPhone.getText().toString());

getContentResolver().update (

ContactProvider.CONTENT\_URI, values, \_ID=" id, null);

**Позиционирование курсора**

**Объект cursor**может использоваться для перемещений назад или вперед по выборке данных. У ***экземпляров типа cursor*** есть встроенное понятие позиции, похожей на Java-интерфейс **Iterator**. Чтобы получить требуемые записи из выборки, можно использовать несколько методов для позиционирования курсора:

* **rnoveToFirst()** — перемещает курсор в первую запись в выборке;
* **rnoveToLast()** — перемещает курсор в последнюю запись в выборке;
* **rnoveToNext()** — перемещает курсор в следующую запись и одновременно определяет, существует ли эта запись. Метод rnoveToNext() возвращает true, если курсор указывает на другую строку после перемещения, и false, если текущая запись была последней в выборке;
* **rnoveToPrevious()** — перемещает курсор в предыдущую запись;
* **rnoveToPosition()** — перемещает курсор в указанную позицию;
* **getPosition()** —возвращает текущий индекс позиции курсора.

Кроме вышеперечисленных методов у курсора есть еще и набор условных методов для определения позиции курсора в выборке:

* **isFirst {);**
* **isLast();**
* **isBeforeFirst();**
* **isAfterLast().**

Эти методы могут использоваться в программном коде для проверки местонахождения позиции курсора.

**Удаление записей**

Чтобы удалить единственную запись, вызовите **метод ContentResolver.delete()**с URI удаляемой строки. Например, так:

getContentResolver().delete(ContactProvider.CONTENT\_URI, \_ID=" + id, null);

cursor.requery();

Чтобы удалить множество записей, необходимо вызвать **метод contentResoiver.delete()** сSQLопределением where, в котором следует указать условие для удаления выбранных строк.

**Чтение возвращаемых значений**

Чтобы сделать запрос к контент-провайдеру, в клиентском приложении используют ***методы ContentResolver.query()*** или ***Activity.managedQuery()***:

Cursor query (Uri uri, String[] projection,

String selection, String[] selectionArgs, String sortOrder)

Cursor managedQuery (Uri uri, String[] projection,

String selection, String[] selectionArgs, String sortOrder)

Оба метода принимают один и тот же набор параметров и возвращают **объект Cursor**. Однако **метод managedQuery()** заставляет деятельность управлять циклом жизни **объекта Cursor** аналогично жизненному циклу самой деятельности.

Эти методы имеют одинаковый набор параметров. Первые два параметра являются обязательными:

* ***URI провайдера***— это **константа content\_uri**, которая идентифицирует конкретный**объект ContentProvider** и набор данных. Существуют некоторые вспомогательные методы для формирования **uri**, в частности **ContentUris.withAppendedld()** и**Uri.withAppendedPath()**, которые добавляют в конец **uri** идентификатор записи. Это статические **методы класса ContentUris**, которые возвращают объект Uri с добавленным в конце идентификатором записи;
* имена полей данных, которые вы хотите получить из БД.

Следующие два параметра в методе — это детализация фильтра для запроса, отформатированная как ***SQLпредложение where*** (исключая сам оператор where непосредственно). Значение **null** в этих параметрах возвращает все строки, если только **URI**сам не ограничивает запрос единственной записью. Чтобы ограничить запрос только одной конкретной записью, вы можете добавить значение id\_записи.

Запрос возвращает набор записей из базы данных. Имена столбцов, их типы и заданный по умолчанию порядок сортировки данных являются специфическими для каждого контентпровайдера. Но каждый контентпровайдер имеет **столбец \_id**, который содержит уникальный числовой идентификатор для каждой записи. Каждый провайдер может также сообщить о количестве возвращаемых записей через**поле \_count**.

Последний параметр определяет порядок сортировки записей, как в SQL oпepaтope**order ву**. Если в параметр передать значение **null**, выборка будет отсортирована по умолчанию. Порядок сортировки по умолчанию определяют заранее, в реализации **класса SQLiteOpenHelper**. Пример фрагмента кода для получения выборки данных:

// массив имен полей, которые мы хотим получить

private static final String[] rnContent new String[]

ContactDbHelper.\_ID,

ContactDbHelper.NAME,

ContactDbHelper.PHONE};

Cursor cursor

Cursor cursor managedQuery(

ContactProvider.CONTENT\_URI, rnContent, null, null, null);

**Контент-провайдеры**

**Контент-провайдеры** поддерживают стандартный синтаксис запросов для чтения, изменения, вставки и удаления данных. ***Контент-провайдеры*** — это единственный способ совместного использования данных между приложениями в***Android***.

**Если необходимо предоставить доступ к своим данных для других приложений, существует два варианта:**

* создать собственный контент-провайдер, как подкласс **класса ContentProvider**;
* добавить данные к существующему провайдеру — если уже есть провайдер, который управляет теми же данными и вы имеете разрешение для работы с этими данными.

Все контент-провайдеры предоставляют общий интерфейс для запросов данных клиентскими приложениями — на чтение, добавление, изменение и удаление данных. ***Объект contentProvider*** не используется напрямую. Клиентские приложения используют контент-провайдеры косвенно, обычно через ***объект ContentResolver***. Получить **contentResolver**можно через вызов ***метода getContentResolver()*** в классе деятельности или другого компонента ***Android приложения***:

ContentResolver resolver getContentResolver();

Вы можете использовать ***методы ContentResolver*** для взаимодействия с любыми контент-провайдерами, которые доступны и требуются для работы приложения с данными.

После инициализации запроса система Androidидентифицирует контент провайдер, который является адресатом запроса и его реализацией. Система сама инициализирует все объекты**ContentProvider**, и вам нет необходимости делать это самостоятельно.

Фактически, приложение никогда не имеет дело непосредственно с **объектами ContentProvider**. Как правило, есть только единственный экземпляр каждого типа**ContentProvider**, но он может связываться с многочисленными о**бъектами ContentResolver** в различных приложениях и процессах. Взаимодействие между процессами обрабатывается ***классами ContentProvider*** и  ***ContentResolver.***

**URI**

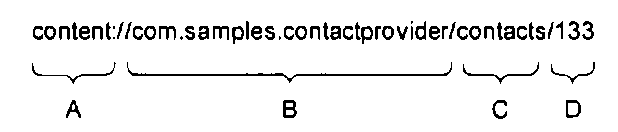
**Каждый контент-провайдер предоставляет открытый URI**(обернутый как объект им), что уникально идентифицирует его набор данных. Контент провайдер, который управляет множественными наборами данных (множественные таблицы), должен предоставлять отдельные URI для каждого набора данных. Все URI для провайдеров начинаются со строки***"content:***

При определении контентпровайдера, как правило, определяют константу для URI. Android определяет константы uri для всех провайдеров, которые идут с платформой. Вот, например,**uri** для таблицы телефонных звонков:

android.provider.CallLog.Calls.CONTENT\_URI

**Константа uri** используется во всех взаимодействиях с контентпровайдером. Каждый ***метод класса contentResolver*** берет **uri**как первый параметр **uri**— это то, что идентифицирует провайдера, к которому **объект ContentResolver** будет обращаться.

**Константа uri**состоит из четырех частей, которые показаны на рис.



**Части константы uri предоставляют следующие сведения:**

* **А**— стандартный префикс, указывающий, что данные предоставляются контентпровайдером. Этот префикс никогда не изменяется;
* **В** — часть uri, который идентифицирует контентпровайдер. Для сторонних приложений он должен быть именем пакета и класса (в нижнем регистре), чтобы гарантировать свою уникальность среди других им;
* **С**— путь, который контентпровайдер использует, чтобы определить требуемые наборы данных. Если контентпровайдер предоставляет только один тип данных, эта часть **uri**может отсутствовать. Если провайдер предоставляет данные нескольких типов, включая и подтипы, то эта часть **uri** будет, например, выглядеть так: ***contacts/photos*** или***contacts/birthday***;
* **D** — идентификатор конкретной записи. Это и есть **\_ID записи**. Если запрос не ограничивается одной записью, эта часть **uri**пропускается:**content://com.samples.contactprovider/contacts**

**Модель данных**

**Контент-провайдеры** представляют данные в виде плоской таблицы, где каждая строка — запись, а каждый столбец — данные специфического типа и значения. Каждая запись обязательно включает числовое ***поле \_id***, которое уникально идентифицирует запись в пределах таблицы. Идентификаторы могут использоваться для соответствия строк в связанных таблицах — например, находить номер телефона человека в одной таблице и фотографию того же человека в другой.

Запрос возвращает **объект Cursor**, который может перемещаться по записям, чтобы делать запись, и от столбца к столбцу, чтобы читать содержание каждого поля. Это специализировало методы для того, чтобы читать каждый тип данных. Так, чтобы читать поле, вы должны знать, какие данные поле содержит.

**Практическое приложение для работы с базой данных**

Реализуем приведенную ранее методику работы с данными в практическом приложении для редактирования контактов.

Создайте в **Eclipse** новый проект и в окне Create New Project введите следующие значения:

* Project name — ContactEditor;
* Application name — Contacts Sample;
* Package name — сот. samples. contacteditor;
* Create Activity — ListContactActivity.

В файле манифеста приложения необходимо зарегистрировать в элементе **<provider>** наш контент-провайдер.

**Файл AndroidManifest.xml**

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

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

package="com.samples.dbcontacts">

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

<provider

android:name=".ContactProvider"

android:authorities="com.samples.dbcontacts.contactprovider">

</provider>

<activity

android:name=".ContactActivity"

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

<intent-filter>

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

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

</intent-filter>

</activity>

</application>

</manifest>

**Файл разметки окна деятельности main.xml**

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

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

android:orientation="horizontal"

android:layout\_width="fill\_parent"

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

<TextView

android:id="@+id/name"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_alignParentLeft="true"

android:textSize="18sp"/>

<TextView

android:id="@+id/phone"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_alignParentRight="true"

android:textSize="18sp"

android:paddingRight="10px"/>

</RelativeLayout>

Диалоговые окна для добавления нового контакта и модификации существующего контакта представляют собой нестандартные диалоги, для которых требуется создание собственной разметки. В диалогах кроме кнопок **ОК** и **Cancel** добавлены текстовые поля для имени и телефона.

**Файл разметки диалога dialog.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="wrap\_content">

<LinearLayout

android:orientation="horizontal"

android:layout\_width="fill\_parent"

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

<TextView

android:text="@string/field\_name"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_alignParentLeft="true"/>

<EditText

android:id="@+id/name"

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

android:layout\_alignParentRight="true"/>

</LinearLayout>

<LinearLayout

android:orientation="horizontal"

android:layout\_width="fill\_parent"

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

<TextView

android:text="@string/field\_phone"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_alignParentLeft="true"/>

<EditText

android:id="@+id/phone"

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

android:layout\_alignParentRight="true"/>

</LinearLayout>

</LinearLayout>

В **файл strings.xml** вынесены все надписи для текстовых полей и кнопок, имеющихся в приложении (это приложение мы используем в следующей главе при создании локализованного приложения с русским языком интерфейса).

**Файл строковых ресурсов strings.xml**

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

<resources>

<string name="app\_name">Contacts from database sample</string>

<string name="btn\_ok">OK</string>

<string name="btn\_cancel">Cancel</string>

<string name="field\_name">Name:</string>

<string name="field\_phone">Phone:</string>

<string name="title\_add">Add new Contact</string>

<string name="title\_edit">Edit Contact</string>

<string name="title\_delete">Delete this Contact?</string>

<string name="menu\_add">Add</string>

<string name="menu\_edit">Edit</string>

<string name="menu\_delete">Delete</string>

<string name="toast\_notify">Please select Contact!</string>

</resources>

**Класс ContactDbHelper**, расширяющий **класс SQLiteOpenHelper**, представляет таблицу***Contact*** базы данных. В нем объявлена структура таблицы и в ***методе onCreate()***производится создание таблицы и заполнение ее текстовыми данными при первом запуске приложения на  устройстве.

**Файл класса ContactDbHelper.java**

package com.samples.dbcontacts;

import android.content.ContentValues;

import android.content.Context;

import android.database.sqlite.SQLiteDatabase;

import android.database.sqlite.SQLiteOpenHelper;

import android.provider.BaseColumns;

public class ContactDbHelper extends SQLiteOpenHelper

implements BaseColumns {

public static final String TABLE\_NAME = "contact";

public static final String NAME = "first\_name";

public static final String PHONE = "phone";

public ContactDbHelper(Context context) {

super(context, ContactProvider.DB\_CONTACTS, null, 1);

}

@Override

public void onCreate(SQLiteDatabase db) {

db.execSQL("CREATE TABLE " + TABLE\_NAME

+ " (\_id INTEGER PRIMARY KEY AUTOINCREMENT, "

+ NAME + " TEXT, " + PHONE + " TEXT);");

ContentValues values = new ContentValues();

values.put(NAME, "Jacob Anderson");

values.put(PHONE, "412412411");

db.insert(TABLE\_NAME, NAME, values);

values.put(NAME, "Emily Duncan");

values.put(PHONE, "161863187");

db.insert(TABLE\_NAME, NAME, values);

values.put(NAME, "Michael Fuller");

values.put(PHONE, "896443658");

db.insert(TABLE\_NAME, NAME, values);

values.put(NAME, "Emma Greenman");

values.put(PHONE, "964990543");

db.insert(TABLE\_NAME, NAME, values);

values.put(NAME, "Joshua Harrison");

values.put(PHONE, "759285086");

db.insert(TABLE\_NAME, NAME, values);

values.put(NAME, "Madison Johnson");

values.put(PHONE, "950285777");

db.insert(TABLE\_NAME, NAME, values);

values.put(NAME, "Matthew Cotman");

values.put(PHONE, "687699999");

db.insert(TABLE\_NAME, NAME, values);

values.put(NAME, "Olivia Lawson");

values.put(PHONE, "161863187");

db.insert(TABLE\_NAME, NAME, values);

values.put(NAME, "Andrew Chapman");

values.put(PHONE, "546599645");

db.insert(TABLE\_NAME, NAME, values);

values.put(NAME, "Daniel Honeyman");

values.put(PHONE, "876545644");

db.insert(TABLE\_NAME, NAME, values);

values.put(NAME, "Isabella Jackson");

values.put(PHONE, "907868756");

db.insert(TABLE\_NAME, NAME, values);

values.put(NAME, "William Patterson");

values.put(PHONE, "687699693");

db.insert(TABLE\_NAME, NAME, values);

values.put(NAME, "Joseph Godwin");

values.put(PHONE, "965467575");

db.insert(TABLE\_NAME, NAME, values);

values.put(NAME, "Samantha Bush");

values.put(PHONE, "907865645");

db.insert(TABLE\_NAME, NAME, values);

values.put(NAME, "Christopher Gateman");

values.put(PHONE, "896874556");

db.insert(TABLE\_NAME, NAME, values);

}

@Override

public void onUpgrade(

SQLiteDatabase db, int oldVersion, int newVersion) {

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

onCreate(db);

}

}

**Класс contactProvider**, расширяющий базовый ***класс contentProvider***, представляет логику доступа к содержимому базы данных **Contacts**.

**Файл класса провайдера ContactProvider.java**

package com.samples.dbcontacts;

import java.util.HashMap;

import android.content.ContentProvider;

import android.content.ContentUris;

import android.content.ContentValues;

import android.content.UriMatcher;

import android.database.Cursor;

import android.database.SQLException;

import android.database.sqlite.SQLiteDatabase;

import android.net.Uri;

import android.text.TextUtils;

public class ContactProvider extends ContentProvider {

public static final String DB\_CONTACTS = "contacts.db";

public static final Uri CONTENT\_URI = Uri.parse(

"content://com.samples.dbcontacts.contactprovider/contact");

public static final int URI\_CODE = 1;

public static final int URI\_CODE\_ID = 2;

private static final UriMatcher mUriMatcher;

private static HashMap<String, String> mContactMap;

private SQLiteDatabase db;

static {

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

mUriMatcher.addURI("com.samples.dbcontacts.contactprovider",

ContactDbHelper.TABLE\_NAME, URI\_CODE);

mUriMatcher.addURI("com.samples.dbcontacts.contactprovider",

ContactDbHelper.TABLE\_NAME + "/#", URI\_CODE\_ID);

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

mContactMap.put(ContactDbHelper.\_ID, ContactDbHelper.\_ID);

mContactMap.put(ContactDbHelper.NAME, ContactDbHelper.NAME);

mContactMap.put(ContactDbHelper.PHONE, ContactDbHelper.PHONE);

}

public String getDbName() {

return(DB\_CONTACTS);

}

@Override

public boolean onCreate() {

db = (new ContactDbHelper(getContext())).getWritableDatabase();

return (db == null) ? false : true;

}

@Override

public Cursor query(Uri url, String[] projection,

String selection, String[] selectionArgs, String sort) {

String orderBy;

if (TextUtils.isEmpty(sort)) {

orderBy = ContactDbHelper.NAME;

}

else {

orderBy = sort;

}

Cursor c = db.query(ContactDbHelper.TABLE\_NAME, projection, selection, selectionArgs,

null, null, orderBy);

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

return c;

}

@Override

public Uri insert(Uri url, ContentValues inValues) {

ContentValues values = new ContentValues(inValues);

long rowId = db.insert(ContactDbHelper.TABLE\_NAME, ContactDbHelper.NAME, values);

if (rowId > 0) {

Uri uri = ContentUris.withAppendedId(CONTENT\_URI, rowId);

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

return uri;

}

else {

throw new SQLException("Failed to insert row into " + url);

}

}

@Override

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

int retVal = db.delete(ContactDbHelper.TABLE\_NAME, where, whereArgs);

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

return retVal;

}

@Override

public int update(Uri url, ContentValues values,

String where, String[] whereArgs) {

int retVal = db.update(ContactDbHelper.TABLE\_NAME, values, where, whereArgs);

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

return retVal;

}

@Override

public String getType(Uri uri) {

return null;

}

}

**Класс деятельности NewContactActivity** предназначен для просмотра данных и имеет меню из трех пунктов — ***Add, Edit и Delete*** — для добавления, модификации и удаления данных соответственно

**Файл класса деятельности ContactActivity.java**

package com.samples.dbcontacts;

import android.app.AlertDialog;

import android.app.ListActivity;

import android.content.ContentValues;

import android.content.DialogInterface;

import android.database.Cursor;

import android.os.Bundle;

import android.view.LayoutInflater;

import android.view.Menu;

import android.view.MenuItem;

import android.view.View;

import android.widget.EditText;

import android.widget.ListAdapter;

import android.widget.SimpleCursorAdapter;

import android.widget.Toast;

public class ContactActivity extends ListActivity {

private static final int IDM\_ADD = 101;

private static final int IDM\_EDIT = 102;

private static final int IDM\_DELETE = 103;

private Cursor mCursor;

private ListAdapter mAdapter;

private static final String[] mContent = new String[] {

ContactDbHelper.\_ID, ContactDbHelper.NAME,

ContactDbHelper.PHONE};

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

mCursor = managedQuery(

ContactProvider.CONTENT\_URI, mContent, null, null, null);

mAdapter = new SimpleCursorAdapter(this,

R.layout.main, mCursor,

new String[] {ContactDbHelper.NAME, ContactDbHelper.PHONE},

new int[] {R.id.name, R.id.phone});

setListAdapter(mAdapter);

}

@Override

public boolean onCreateOptionsMenu(Menu menu) {

menu.add(Menu.NONE, IDM\_ADD, Menu.NONE, R.string.menu\_add)

.setIcon(R.drawable.ic\_menu\_add)

.setAlphabeticShortcut('a');

menu.add(Menu.NONE, IDM\_EDIT, Menu.NONE, R.string.menu\_edit)

.setIcon(R.drawable.ic\_menu\_edit)

.setAlphabeticShortcut('e');

menu.add(Menu.NONE, IDM\_DELETE, Menu.NONE, R.string.menu\_delete)

.setIcon(R.drawable.ic\_menu\_delete)

.setAlphabeticShortcut('d');

return(super.onCreateOptionsMenu(menu));

}

@Override

public boolean onOptionsItemSelected(MenuItem item) {

final long id = this.getSelectedItemId();

switch (item.getItemId()) {

case IDM\_ADD: {

CallAddContactDialog();

}

break;

case IDM\_EDIT:

if (id > 0) {

CallEditContactDialog(id);

}

else {

Toast.makeText(this, R.string.toast\_notify, Toast.LENGTH\_SHORT)

.show();

}

break;

case IDM\_DELETE:

if (id > 0) {

CallDeleteContactDialog(id);

}

else {

Toast.makeText(this, R.string.toast\_notify, Toast.LENGTH\_SHORT)

.show();

}

break;

}

return(super.onOptionsItemSelected(item));

}

private void CallAddContactDialog() {

LayoutInflater inflater = LayoutInflater.from(this);

View root = inflater.inflate(R.layout.dialog, null);

final EditText textName = (EditText)root.findViewById(R.id.name);

final EditText textPhone = (EditText)root.findViewById(R.id.phone);

AlertDialog.Builder b = new AlertDialog.Builder(this);

b.setView(root);

b.setTitle(R.string.title\_add);

b.setPositiveButton(

R.string.btn\_ok, new DialogInterface.OnClickListener() {

public void onClick(DialogInterface dialog, int whichButton) {

ContentValues values = new ContentValues(2);

values.put(ContactDbHelper.NAME, textName.getText().toString());

values.put(ContactDbHelper.PHONE, textPhone.getText().toString());

getContentResolver().insert(ContactProvider.CONTENT\_URI, values);

mCursor.requery();

}

});

b.setNegativeButton(

R.string.btn\_cancel, new DialogInterface.OnClickListener() {

public void onClick(DialogInterface dialog, int whichButton) {}

});

b.show();

}

private void CallEditContactDialog(final long id) {

LayoutInflater inflater = LayoutInflater.from(this);

View root = inflater.inflate(R.layout.dialog, null);

final EditText textName = (EditText)root.findViewById(R.id.name);

final EditText textPhone = (EditText)root.findViewById(R.id.phone);

mCursor.moveToPosition(this.getSelectedItemPosition());

textName.setText(mCursor.getString(1));

textPhone.setText(mCursor.getString(2));

AlertDialog.Builder b = new AlertDialog.Builder(this);

b.setView(root);

b.setTitle(R.string.title\_edit);

b.setPositiveButton(

R.string.btn\_ok, new DialogInterface.OnClickListener() {

public void onClick(DialogInterface dialog, int whichButton) {

ContentValues values = new ContentValues(2);

values.put(ContactDbHelper.NAME, textName.getText().toString());

values.put(ContactDbHelper.PHONE, textPhone.getText().toString());

getContentResolver().update(

ContactProvider.CONTENT\_URI, values, "\_ID=" + id, null);

mCursor.requery();

}

});

b.setNegativeButton(

R.string.btn\_cancel, new DialogInterface.OnClickListener() {

public void onClick(DialogInterface dialog, int whichButton) {}

});

b.show();

}

private void CallDeleteContactDialog(final long id) {

AlertDialog.Builder b = new AlertDialog.Builder(this);

b.setTitle(R.string.title\_delete);

b.setPositiveButton(

R.string.btn\_ok, new DialogInterface.OnClickListener() {

public void onClick(DialogInterface dialog, int whichButton) {

getContentResolver().delete(

ContactProvider.CONTENT\_URI, "\_ID=" + id, null);

mCursor.requery();

}

});

b.setNegativeButton(

R.string.btn\_cancel, new DialogInterface.OnClickListener() {

public void onClick(DialogInterface dialog, int whichButton) {}

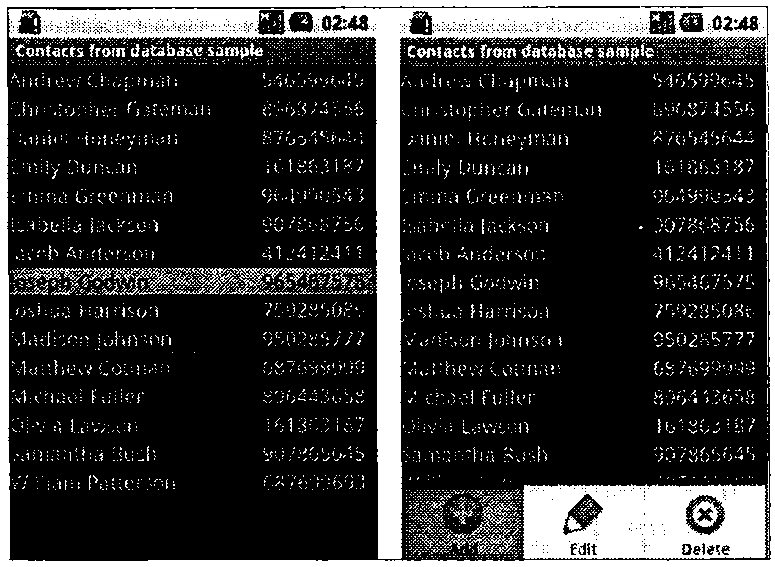
});

b.show();

}

}

Скомпилируйте проект и запустите его в эмуляторе мобильного устройства. При первом запуске приложения сработает метод обратного вызова **onCreate()**в классе**ContactDbHelper**, который создаст базу данных **Contacts** с одной таблицей и заполнит ее текстовыми данными, как показано на рис.



Приложение содержит меню, при выборе одного из пунктов которого отображаются диалоговые окна для добавления, модификации и удаления контакта:

