**Работаем с базой данных**

[Введение в базы данных SQLite](http://developer.alexanderklimov.ru/android/database.php#intro)  
[Создаём базу данных](http://developer.alexanderklimov.ru/android/database.php#createdb)  
[Работаем с базой данных](http://developer.alexanderklimov.ru/android/database.php#workdb)

**Введение в базы данных SQLite**

Android использует для работы с базами данных известную библиотеку SQLite. SQLite зарекомендовала себя в качестве чрезвычайно надёжной системы баз данных, которая используется во многих бытовых электронных устройствах и программах, включая некоторые MP3-проигрыватели, iPhone, iPod Touch, Mozilla Firefox и др.

С помощью SQLite вы можете создавать для своего приложения независимые реляционные базы данных. Android хранит базы данных в каталоге **/data/data/<имя\_вашего\_пакета>/databases** на эмуляторе, на устройстве путь может отличаться. По умолчанию все базы данных закрытые, доступ к ним могут получить только те приложения, которые их создали.

Каждая база данных состоит из двух файлов. Имя первого файла базы данных соответствует имени базы данных. Это основной файл баз данных SQLite, в нём хранятся все данные. Вы будете создавать его программно. Второй файл — файл журнала. Его имя состоит из имени базы данных и суффикса "-journal". В файле журнала хранится информация обо всех изменениях, внесенных в базу данных. Если в работе с данными возникнет проблема, Android использует данные журнала для отмены (или отката) последних изменений. Вы с ним не будете взаимодействовать, но если вы будете просматривать внутренности своего устройства, то будете знать, зачем этот файл там присутствует.

Те, кто имеет опыт работы со связкой PHP+MySQL, найдут много знакомых вещей и быстро разберутся с принципом работы. Для остальных постараюсь объяснить максимально доступно.

Прежде всего нужно запомнить, что в Android уже есть готовый класс **SQLiteOpenHelper**, от которого нужно наследоваться.

**Создаём базу данных**

Наша цель - создать базу данных, внести в неё данные и прочитать эти данные путём вывода результатов чтения на экран.

Создадим новый проект как обычно. Для взаимодействия с базой данных нам будет достаточно кнопки и текстовой метки **TextView**, в которой будем выводить результат считывания данных.

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

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical"

android:paddingBottom="@dimen/activity\_vertical\_margin"

android:paddingLeft="@dimen/activity\_horizontal\_margin"

android:paddingRight="@dimen/activity\_horizontal\_margin"

android:paddingTop="@dimen/activity\_vertical\_margin"

tools:context=".MainActivity">

<Button

android:id="@+id/button"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:onClick="onClick"

android:text="Запрос к базе данных" />

<TextView

android:id="@+id/textView"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:textAppearance="?android:attr/textAppearanceLarge" />

</LinearLayout>

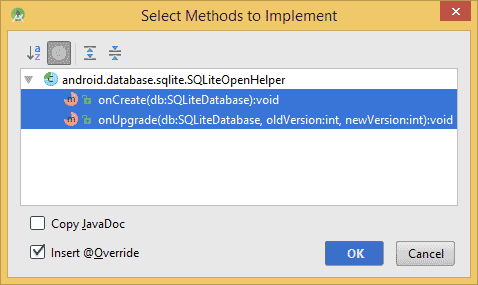
Пока ничего страшного. Разметка как разметка. Можно ещё добавить **ImageView** с изображением симпатичного котика, чтобы окончательно успокоиться.

Теперь чуть напрягитесь. При работе с базой данной принято создавать отдельный класс. Обычно, в проекте у нас уже есть один класс **MainActivity**, который относится к активности. А теперь будет два класса. Только второй класс будет наследоваться не от класса **Activity**, а от класса **SQLiteOpenHelper**.

Созданный класс будет работать с базой данных - добавлять, выбирать, удалять и прочие операции.

Щёлкаем правой кнопкой мыши на имени пакета в левой части студии и выбираем в меню **New** | **Java Class** и в диалоговом окне выбираем имя для нового класса, например, **DatabaseHelper**. Слово **Helper** обычно используют, чтобы показать, что класс является обёрткой (вспомогательным классом) какого-то абстрактного класса. Впрочем, вы можете придумать более замысловатое название, например, **ILoveNewYork** или **CatsForever**. Спустя год, когда вы вернётесь к своему примеру, это будет так увлекательно вспоминать, для чего был создан класс с таким красивым именем.

У нас появится заготовка. Наследуемся от **SQLiteOpenHelper**. Студия предложит создать два обязательных метода **onCreate()** и **onUpgrade()**, о которых поговорим позже.



После добавления методов студия по-прежнему ругается. Теперь ему подавай конструкторы. Получится такой код.:

package ru.alexanderklimov.database;

import android.content.Context;

import android.database.DatabaseErrorHandler;

import android.database.sqlite.SQLiteDatabase;

import android.database.sqlite.SQLiteOpenHelper;

public class DatabaseHelper extends SQLiteOpenHelper {

public DatabaseHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version) {

super(context, name, factory, version);

}

public DatabaseHelper(Context context, String name, SQLiteDatabase.CursorFactory factory,

int version, DatabaseErrorHandler errorHandler) {

super(context, name, factory, version, errorHandler);

}

@Override

public void onCreate(SQLiteDatabase db) {

}

@Override

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

}

}

Во многих примерах в интернете (и в официальной документации) я видел также, что в класс добавляется ещё один конструктор с двумя константами, который является упрощённым вариантом второго конструктора. Позже я покажу его удобство, а пока добавим новый код в класс.

// имя базы данных

private static final String DATABASE\_NAME = "mydatabase.db";

// версия базы данных

private static final int DATABASE\_VERSION = 1;

DatabaseHelper(Context context) {

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

}

Третий параметр **null** в суперклассе используется для работы с курсорами. Сейчас их не используем, поэтому оставим в покое.

Как вы уже догадались, константа **DATABASE\_NAME** отвечает за имя файла, в котором будет храниться база данных приложения. Можно придумать любое имя и обойтись без расширения. Но мне так привычнее.

Вторая константа **DATABASE\_VERSION** требует дополнительных объяснений. Она отвечает за номер версии базы. Принцип её работы схож с номером версий самого приложения. Когда мы видим, что вышла новая версия Chrome 33, то понимаем, что пора обновляться. Аналогично поступает и само приложение, когда замечает, что номер версии базы стал другим. Как только программа заметила обновление номера базы, она запускает метод**onUpgrade()**, который у нас сформировался автоматически. В этом методе необходимо разместить код, который должен сработать при обновлении базы.

Теперь нужно объяснить, зачем нужен этот метод **onUpgrade()**. Представьте ситуацию, что вы первоначально создали в базе таблицу, в которую заносятся имена котов и их электронные адреса и телефоны (продвинутые кошаки). Вроде всё замечательно. Если нужно поздравить усатых-полосатых с Международным днём кошек, который отмечается 1 марта, то проблем нет никаких. У вас есть список имён, по которому вы можете пройтись и лично написать каждому письмо. Пользователи, скачавшие ваше приложение, с удовольствием заполняют базу данных и дружно пишут письма мелким почерком. И вдруг до вас дошло, что совершили непростительную ошибку. Вы забыли добавить в базу данных даты рождения котов. А значит их никто не поздравит и не погладит (((.

Вы исправляете досадное упущение и выкладываете новую версию программы в открытый доступ. Новые пользователи, которые установят программу первый раз, радуются жизни - у них есть все необходимые данные для работы. Но что делать тем, кто уже работает со старой программой? Обновившись, они увидят дополнительное текстовое поле для ввода даты рождения, но в старой базе нет колонки для хранения новых данных. И ваша программа завершится с ошибкой. Полностью удалять и устанавливать новую версию программы тоже не выход - тогда пропадут старые данные, что тоже не желательно. Для таких случаев вы пишете код в методе **onUpgrade()**, чтобы при обновлении поменялась структура базы данных у старых пользователей. Мы позже попробуем смоделировать эту ситуацию.

Метод **onCreate()** вопросов не вызывает - здесь создаётся сама база данных с необходимыми данными для работы.

Когда ваше приложение будет готово, то в папке **data/data/имя\_пакета/databases** появится файл **mydatabase.db**(позже я вам покажу). Этот файл и будет вашей базой данных, в которой будет находиться созданная вами таблица. На данный момент в студии нет плагина для просмотра таблиц (в Eclipse есть). Надеюсь, со временем добавят поддержку. А пока вам придётся скачивать из устройства файл базы данных и просматривать его на компьютере специальными программами, работающими с SQLite на локальном компьютере.

Дальше идёт наполнение класса данными. Давайте для начала создадим совсем простую таблицу, которая будет содержать только имена котов. В первую очередь нам нужно определиться с именем таблицы в созданной базе данных **mydatabase.db**. Это обычная строка, поэтому создаём строковую константу для имени таблицы.

// имя таблицы

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

Также нужно указать имена столбцов для таблицы. Приложение будет обращаться к столбцам для извлечения данных, поэтому создадим публичные константы для столбцов с понятными именами. Наша таблица будет состоять из трёх столбцов: имя кота, его телефон и возраст.

// названия столбцов

public static final String CAT\_NAME\_COLUMN = "cat\_name";

public static final String PHONE\_COLUMN = "phone";

public static final String AGE\_COLUMN = "age";

Следующим приёмом редко пользуются, но я вам его покажу. Добавим реализацию интерфейса **BaseColumn**:

public class DatabaseHelper extends SQLiteOpenHelper **implements BaseColumns** {

Что это нам даёт? В большинстве случаев работа с базой данных происходит через специальные объекты **Cursor**, которые требуют наличия в таблице колонки с именем **\_id** (дополнительный четвёртый столбец в нашем примере). Вы можете создать столбец вручную в коде, а можно положиться на **BaseColumn**, который создаст столбец с нужным именем автоматически. Дело ваше. Если вы не будете работать с курсорами, то можете использовать и стандартное наименование **id** или вообще не использовать данный столбец, но не советую так поступать, чтобы не вырабатывать вредных привычек.

Теперь перейдём к методам и начнём с метода **onCreate()**, который вызывается, если в устройстве нет базы данных и наш класс должен создать его. Как мы помним, у метода есть параметр **db**, который относится к классу**SQLiteDatabase**. У класса есть специальный метод **execSQL()**, которому нужно передать запрос (SQL-скрипт) для создания таблицы. Для создания таблицы в SQL используется команда **CREATE TABLE ...**. Для удобства вынесем команду в отдельную строку. Аналогично поступим с командой **DROP TABLE**. Так как строка очень длинная и состоит из множества строковых переменных, которые нужно соединить в одну цепочку, то поступают следующим образом. Создаём ещё одну строковую константу для формирования скрипта и передадим её в метод.

private static final String DATABASE\_CREATE\_SCRIPT = "create table "

+ DATABASE\_TABLE + " (" + BaseColumns.\_ID

+ " integer primary key autoincrement, " + CAT\_NAME\_COLUMN

+ " text not null, " + PHONE\_COLUMN + " integer, " + AGE\_COLUMN

+ " integer);";

@Override

public void onCreate(SQLiteDatabase db) {

db.execSQL(DATABASE\_CREATE\_SCRIPT);

}

Основная сложность - не пропустить пробелы в запросе. Очень часто пропущенный пробел становится источником проблем и ваше приложение не может создать таблицу. Можете сначала написать сам скрипт создания таблицы, а уже потом заменять отдельные слова константами. Идентификатор **\_id** всегда должен использовать **integer primary key autoincrement**, остальные колонки на ваше усмотрение. В примере я специально использовал два разных варианта - **text** и **integer**.

Второй метод **onUpgrade()** вызывается при несовпадении версий. Часто в этом методе просто удаляют существующую таблицу и заменяют её на новую. Это самое простое и практичное решение. Впрочем, на первых порах, вам вряд ли придётся заниматься подобными делами, поэтому метод можно оставить даже пустым.

@Override

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

// Запишем в журнал

Log.w("SQLite", "Обновляемся с версии " + oldVersion + " на версию " + newVersion);

// Удаляем старую таблицу и создаём новую

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

// Создаём новую таблицу

onCreate(db);

}

Полностью код будет следующим:

package ru.alexanderklimov.database;

import android.content.Context;

import android.database.DatabaseErrorHandler;

import android.database.sqlite.SQLiteDatabase;

import android.database.sqlite.SQLiteOpenHelper;

import android.provider.BaseColumns;

import android.util.Log;

public class DatabaseHelper extends SQLiteOpenHelper implements BaseColumns {

// названия столбцов

public static final String CAT\_NAME\_COLUMN = "cat\_name";

public static final String PHONE\_COLUMN = "phone";

public static final String AGE\_COLUMN = "age";

// имя базы данных

private static final String DATABASE\_NAME = "mydatabase.db";

// версия базы данных

private static final int DATABASE\_VERSION = 1;

// имя таблицы

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

private static final String DATABASE\_CREATE\_SCRIPT = "create table "

+ DATABASE\_TABLE + " (" + BaseColumns.\_ID

+ " integer primary key autoincrement, " + CAT\_NAME\_COLUMN

+ " text not null, " + PHONE\_COLUMN + " integer, " + AGE\_COLUMN

+ " integer);";

DatabaseHelper(Context context) {

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

}

public DatabaseHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version) {

super(context, name, factory, version);

}

public DatabaseHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version, DatabaseErrorHandler errorHandler) {

super(context, name, factory, version, errorHandler);

}

@Override

public void onCreate(SQLiteDatabase db) {

db.execSQL(DATABASE\_CREATE\_SCRIPT);

}

@Override

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

// Запишем в журнал

Log.w("SQLite", "Обновляемся с версии " + oldVersion + " на версию " + newVersion);

// Удаляем старую таблицу и создаём новую

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

// Создаём новую таблицу

onCreate(db);

}

}

**Работаем с базой данных**

Переходим к главной активности и напишем код для кнопки:

private DatabaseHelper mDatabaseHelper;

public void onClick(View v) {

mDatabaseHelper = new DatabaseHelper(this, "mydatabase.db", null, 1);

SQLiteDatabase sdb;

sdb = mDatabaseHelper.getReadableDatabase();

}

Мы использовали второй конструктор из примера, но можно было написать и **new DatabaseHelper(this)** с использованием упрощённого конструктора, о котором я вам говорил выше. Таким образом, вы можете перенести логику создания базы в отдельный класс.

Запустите проект. Нажав на кнопку, вы создадите на устройстве базу данных. Убедиться в этом можно, если запустить **Android Device Monitor**. Выберите вкладку **File Explorer** и найдите своё приложение (на эмуляторе). Вы увидите, что появилась папка **data/data/имя\_пакета/databases** с файлом **mydatabase.db**. Метод**getReadableDatabase** создаёт или открывает базу данных.

Сейчас база данных пуста. Поступим следующим образом. Перенесём код для открытия базы данных в метод**onCreate()** и там же добавим новую запись. А при нажатии кнопки получим добавленную запись и выведем её в текстовом блоке.

Небольшое предупреждение. При работе с базой данных мы обращаемся к файлу. Если база данных очень большая, то запросы не будут мгновенными. Операции с файлами являются медленными, поэтому следует использовать многопоточность. Для наших примеров это не страшно, поэтому мы пока не будем усложнять код.

private DatabaseHelper mDatabaseHelper;

private SQLiteDatabase mSqLiteDatabase;

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

mDatabaseHelper = new DatabaseHelper(this, "mydatabase.db", null, 1);

mSqLiteDatabase = mDatabaseHelper.getWritableDatabase();

ContentValues values = new ContentValues();

// Задайте значения для каждого столбца

values.put(DatabaseHelper.CAT\_NAME\_COLUMN, "Рыжик");

values.put(DatabaseHelper.PHONE\_COLUMN, "4954553443");

values.put(DatabaseHelper.AGE\_COLUMN, "5");

// Вставляем данные в таблицу

mSqLiteDatabase.insert("cats", null, values);

}

public void onClick(View view) {

Cursor cursor = mSqLiteDatabase.query("cats", new String[] {DatabaseHelper.CAT\_NAME\_COLUMN, DatabaseHelper.PHONE\_COLUMN, DatabaseHelper.AGE\_COLUMN},

null, null,

null, null, null) ;

cursor.moveToFirst();

String catName = cursor.getString(cursor.getColumnIndex(DatabaseHelper.CAT\_NAME\_COLUMN));

int phoneNumber = cursor.getInt(cursor.getColumnIndex(DatabaseHelper.PHONE\_COLUMN));

int age = cursor.getInt(cursor.getColumnIndex(DatabaseHelper.AGE\_COLUMN));

TextView infoTextView = (TextView)findViewById(R.id.textView);

infoTextView.setText("Кот " + catName + " имеет телефон " + phoneNumber + " и ему " + age + " лет");

// не забываем закрывать курсор

cursor.close();

}

Запустите проект и убедитесь, что в текстовой метке выводится запись. Сразу предупрежу, что когда вы будете запускать программу, то каждый раз будет вставляться новая запись в базу данных. Поэтому после первого запуска можете закомментировать вставку новой записи в методе **onCreate()**. Или удаляйте программу и ставьте её заново.

Обратите внимание, что телефонный номер выводится совсем не такой, какой мы вводили в базу данных. Почему? Это вам в качестве домашнего задания.

**Вставка данных**

Теперь разберём по кусочкам, как делать основные операции с данными. Начнём с вставки.

1. Первый способ. ContentValues

Для вставки сначала подготавливаются данные с помощью класса **ContentValues**. Вы указываете имя колонки таблицы и значение для неё, т.е. работает по принципу "ключ-значение". Когда подготовите все данные во все столбцы, то вызывайте метод **insert()**, который сразу раскидает данные по столбцам. В методе **onCreate()** мы уже выполнили эту работу.

Способ очень удобен, требует мало кода и легко читаем. Вы создаёте экземпляр класса, а затем с помощью метода**put()** записываете в нужную колонку нужные данные. После чего вызывается метод **insert()**, который помещает подготовленные данные в таблицу.

Если колонок несколько, то вызывайте метод **put()** несколько раз:

// Задайте значения для каждого столбца

values.put(DatabaseHelper.CAT\_NAME\_COLUMN, "Рыжик");

values.put(DatabaseHelper.PHONE\_COLUMN, "4954553443");

values.put(DatabaseHelper.AGE\_COLUMN, "5");

У метода **insert()** три аргумента. В первом указывается имя таблицы, в которую будут добавляться записи. В третьем указывается объект **ContentValues**, созданный ранее. Второй аргумент используется для указания колонки. SQL не позволяет вставлять пустую запись, и если будет использоваться пустой **ContentValue**, то укажите во втором аргументе **null** во избежание ошибки. Проследите, чтобы колонка в таблице позволяла принимать **NULL**.

1. Второй способ. SQL-запрос

Существует также другой способ вставки через метод **execSQL()**, когда подготавливается нужная строка и запускается скрипт. Этот способ возможно понравится PHP-кодерам, которые привыкли к такому синтаксису.

В этом варианте используется традиционный SQL-запрос **INSERT INTO...**. Основное неудобство при этом способе - не запутаться в кавычках. Если что-то не вставляется, то смотрите логи сообщений.

// Абстрактный пример

db = new DatabaseHelper(this);

SQLiteDatabase sqdb = db.getWritableDatabase();

String insertQuery = "INSERT INTO " +

DatabaseHelper.DATABASE\_TABLE +

" (" + DatabaseHelper.CAT\_NAME\_COLUMN + ") VALUES ('Васька')";

sqdb.execSQL(insertQuery);

**Изменение данных**

Если запись уже существует, но вам нужно изменить какое-то значение, то вместо **insert()** используйте метод**update()**. В остальном принцип тот же. Предположим, что повторном осмотре котёнка выяснилось, что это кот, а не кошка. Если вы уже назвали котёнка Муркой, то логично назвать его теперь Мурзиком. Вызываем метод **put()**, а затем обновляем запись в базе данных.

ContentValues values = new ContentValues();

values.put(DatabaseHelper.CAT\_NAME\_COLUMN, "Мурзик");

mSqLiteDatabase.update(mDatabaseHelper.DATABASE\_TABLE,

values,

mDatabaseHelper.CAT\_NAME\_COLUMN + "= ?", new String[]{"Мурка"});

Первый параметр метода **update()** содержит имя таблицы. Второй параметр указывает, какие значения должны использоваться для обновления. Третий параметр задает условия отбора обновляемых записей (where). В приведенном примере "NAME = ?"означает, что столбец NAME должен быть равен некоторому значению. Символ **?**обозначает значение столбца, которое определяется содержимым последнего параметра. Если в двух последних параметрах метода передаётся значение **null**, будут обновлены ВСЕ записи в таблице.

Возможны и сложные условия.

db.update(mDatabaseHelper.TABLE\_NAME,

values,

"NAME = ? OR EMAIL = ?",

new String[] {"Васька", "vaska@cat.com"});

Если столбец не является строкой, то его нужно преобразовать в строку, чтобы использовать в качестве условий.

db.update(mDatabaseHelper.TABLE\_NAME,

values,

"\_id = ?",

new String[] {Integer.toString(1)});

Будьте осторожны с обновлениями. Если в последних двух параметрах передать значение **null**, то будут обновлены все записи в таблице, так как в запросе нет условий.

db.update(mDatabaseHelper.TABLE\_NAME,

values,

null, null);

**Удаление данных**

Метод **delete()** класса SQLiteDatabase работает по тому же принципу, как и метод **update()**. Он имеет следующую форму:

public int delete (String table,

String whereClause,

String[] whereArgs)

Разобравшись с базовым примером, вам будет проще понять другие примеры, связанные с базой данных. Существуют различные приёмы и техники, упрощающие работу с базой данных - вспомогательные классы-обёртки в качестве адаптеров, обёртки для методов вставки и извлечения данных и т.д. Но основа будет такая же, как у нашего примера.

**Чтение данных**

Считывать данные также можно двумя способами. В любом случае результат возвращается в виде объекта **Cursor**. Не путайте его с курсором мыши, который бегает у вас на экране.

1. Первый способ. Метод query()

А извлечение данных происходит через метод **query()**. Данные хранятся в наборе строк, которые можно представить в виде таблицы. Из этой таблицы вы уже можете извлечь конкретное значение.

У метода **query()** множество параметров. В первом параметре укажите имя таблицы, во втором - массив имён колонок, далее идут дополнительные условия. Пока везде оставим **null**. В нашем примере мы добавили одну запись и извлечь её просто.

Cursor cursor = mSqLiteDatabase.query("cats", new String[]{DatabaseHelper.CAT\_NAME\_COLUMN,

DatabaseHelper.PHONE\_COLUMN, DatabaseHelper.AGE\_COLUMN},

null, null,

null, null, null);

cursor.moveToFirst();

String catName = cursor.getString(cursor.getColumnIndex(DatabaseHelper.CAT\_NAME\_COLUMN));

long phoneNumber = cursor.getLong(cursor.getColumnIndex(DatabaseHelper.PHONE\_COLUMN));

int age = cursor.getInt(cursor.getColumnIndex(DatabaseHelper.AGE\_COLUMN));

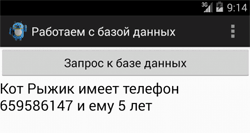
TextView infoTextView = (TextView) findViewById(R.id.textView);

infoTextView.setText("Кот " + catName + " имеет телефон " + phoneNumber + " и ему " +

age + " лет");

// не забываем закрывать курсор

cursor.close();



Мы переходим к первой записи (метод **moveToFirst()**) и считываем данные, которые выводим в текстовой метке.

Если в таблице записей много, то следует произвести последовательный перебор через цикл **while**.

// Абстрактный пример

Cursor cursor = mDatabase.query(mDatabaseHelper.TABLE\_NAME, new String[]{

DatabaseHelper.COLUMN\_ID, DatabaseHelper.CAT\_NAME\_COLUMN}, null,

null,

null,

null,

null

);

while (cursor.moveToNext()) {

int id = cursor.getInt(cursor.getColumnIndex(DatabaseHelper.COLUMN\_ID));

String name = cursor.getString(cursor

.getColumnIndex(DatabaseHelper.CAT\_NAME\_COLUMN));

Log.i("LOG\_TAG", "ROW " + id + " HAS NAME " + name);

}

cursor.close();

1. Второй способ. Метод rawQuery()

Второй способ использует сырой (raw) SQL-запрос. Сначала формируется строка запроса и скармливается методу**rawQuery()**.

// Абстрактный пример

// Метод 2: Сырой SQL-запрос

String query = "SELECT " + DatabaseHelper.COLUMN\_ID + ", "

+ DatabaseHelper.CAT\_NAME\_COLUMN + " FROM " + DatabaseHelper.TABLE\_NAME;

Cursor cursor2 = mDatabase.rawQuery(query, null);

while (cursor2.moveToNext()) {

int id = cursor2.getInt(cursor2

.getColumnIndex(DatabaseHelper.COLUMN\_ID));

String name = cursor2.getString(cursor2

.getColumnIndex(DatabaseHelper.CAT\_NAME\_COLUMN));

Log.i("LOG\_TAG", "ROW " + id + " HAS NAME " + name);

}

cursor2.close();

Итак, мы научились создавать базу данных, заносить данные и считывать данные. Изучите пример досконально, чтобы всё улеглось в голове. Пример очень базовый. Обычно создают отдельный класс для работы с запросами базы данных и класс-обёртку на основе **SQLiteOpenHelper** помещают в него. А затем создают в новом классе отдельные методы для добавления, извлечения, обновления данных - CRUD.

Естественно, мы можем выводить данные не только в лог, но и в адаптеры для **Spinner** или **ListView**. Об этом поговорим в других примерах.