RxJava (http://developer.alexanderklimov.ru/android/rx/)

Советы (http://developer.alexanderklimov.ru/android/tips-android.php)

Статьи (http://developer.alexanderklimov.ru/android/articles-android.php)

Книги (http://developer.alexanderklimov.ru/android/books.php)

Java (http://developer.alexanderklimov.ru/android/java/java.php)

Kotlin (http://developer.alexanderklimov.ru/android/kotlin/)

Дизайн (http://developer.alexanderklimov.ru/android/design/)

Отладка (http://developer.alexanderklimov.ru/android/debug/)

Open Source (http://developer.alexanderklimov.ru/android/opensource.php)

Полезные ресурсы (http://developer.alexanderklimov.ru/android/links.php)

SQLite и Android. Кошкин дом. Часть вторая

В первой части (cathouse.php) мы изучили возможности SQLite. Теперь нужно научиться подключать базу данных в приложении на Android.

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.

Интерфейс

Для начала создадим интерфейс программы. Для первой активности **MainActivity** выберем шаблон **Basic Activity**. Сразу же создадим вторую активность **EditorActivity** из шаблона **Empty Activity**.

В первой активности есть кнопка **Floating Action Button**, через которую будем попадать на вторую активность.

```
FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
fab.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Intent intent = new Intent(MainActivity.this, EditorActivity.class);
        startActivity(intent);
    }
});
```

Вторая активность предназначена для добавления новых гостей, которые поселяются в наш отель "Кошкин дом". Настроим его.

```
<?xml version="1.0" encoding="utf-8"?>
<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:padding="@dimen/activity_horizontal_margin"
    tools:context=".EditorActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <TextView
            style="@style/CategoryStyle"
            android:text="Общая информация" />

        <LinearLayout
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="2"
            android:orientation="vertical"
            android:paddingLeft="4dp">

            <EditText
                android:id="@+id/edit_guest_name"
                style="@style/EditorFieldStyle"
                android:hint="Имя"
                android:inputType="textCapWords" />

            <EditText
                android:id="@+id/edit_guest_city"
                style="@style/EditorFieldStyle"
                android:hint="Город"
                android:inputType="textCapWords" />
        </LinearLayout>
    </LinearLayout>

    <LinearLayout
        android:id="@+id/container_gender"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <TextView
            style="@style/CategoryStyle"
            android:text="Пол" />

        <LinearLayout
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="2"
            android:orientation="vertical">

            <Spinner
```

```
                android:id="@+id/spinner_gender"
                android:layout_width="wrap_content"
                android:layout_height="48dp"
                android:paddingRight="16dp"
                android:spinnerMode="dropdown" />
        </LinearLayout>
    </LinearLayout>

    <LinearLayout
        android:id="@+id/container_age"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <TextView
            style="@style/CategoryStyle"
            android:text="Возраст" />

        <RelativeLayout
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="2"
            android:paddingLeft="4dp">

            <EditText
                android:id="@+id/edit_guest_age"
                style="@style/EditorFieldStyle"
                android:hint="Возраст"
                android:inputType="number" />

        </RelativeLayout>
    </LinearLayout>
</LinearLayout>
```

Экран состоит из нескольких текстовых полей и одного выпадающего списка для выбора пола гостя.

Инициализируем текстовые поля и выпадающий список.

```
private EditText mNameEditText;
private EditText mCityEditText;
private EditText mAgeEditText;

private Spinner mGenderSpinner;

/**
 * Пол для гостя. Возможные варианты:
 * 0 для кошки, 1 для кота, 2 - не определен.
 */
private int mGender = 2;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_editor);

    mNameEditText = (EditText) findViewById(R.id.edit_guest_name);
    mCityEditText = (EditText) findViewById(R.id.edit_guest_city);
    mAgeEditText = (EditText) findViewById(R.id.edit_guest_age);
    mGenderSpinner = (Spinner) findViewById(R.id.spinner_gender);

    setupSpinner();
}

/**
 * Настраиваем spinner для выбора пола у гостя.
 */
private void setupSpinner() {

    ArrayAdapter genderSpinnerAdapter = ArrayAdapter.createFromResource(this,
            R.array.array_gender_options, android.R.layout.simple_spinner_item);

    genderSpinnerAdapter.setDropDownViewResource(android.R.layout.simple_dropdown_item_1lin
e);

    mGenderSpinner.setAdapter(genderSpinnerAdapter);
    mGenderSpinner.setSelection(2);

    mGenderSpinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
        @Override
        public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {
            String selection = (String) parent.getItemAtPosition(position);
            if (!TextUtils.isEmpty(selection)) {
                if (selection.equals(getString(R.string.gender_female))) {
                    mGender = 0; // Кошка
                } else if (selection.equals(getString(R.string.gender_male))) {
                    mGender = 1; // Кот
                } else {
                    mGender = 2; // Не определен
                }
            }
        }

        @Override
        public void onNothingSelected(AdapterView<?> parent) {
```

```
            mGender = 2; // Unknown
        }
    });
}
```

Добавим несколько строковых ресурсов.

```
<resources>
    <string name="app_name">CatHouse</string>
    <string name="action_settings">Settings</string>

    <string name="action_save">Сохранить</string>
    <string name="action_delete">Удалить</string>

    <string name="gender_unknown">Неизвестно</string>
    <string name="gender_male">Кот</string>
    <string name="gender_female">Кошка</string>

    <string-array name="array_gender_options">
        <item>@string/gender_female</item>
        <item>@string/gender_male</item>
        <item>@string/gender_unknown</item>
    </string-array>

</resources>
```

Подписываем контракт

Теперь можно заняться интеграцией базы данных в приложение.

При работе с базой данных принято создавать новый пакет **data** внутри основного пакета. Щёлкаем правой кнопкой мыши по имени пакета, выбираем **New | Package** и вводим новое имя.

В последних рекомендациях Гугла рекомендуется создавать класс-контракт. Будем придерживаться этого правила. Мы как бы подписываем контракт на работу с базой данных и предоставляем все нужные данные.

Внутри созданного пакета создаём новый класс **HotelContract**. Класс-контракт является контейнером для базы данных и может содержать несколько внутренних классов, которые представляют отдельные таблицы (не забывайте, что база данных может содержать несколько таблиц). Внутри класса создаём внутренний класс. В нашем случае будет один класс для таблицы **guests**.

Нам следует задать схему таблицы и константы для столбцов для удобства. Класс будет выглядеть так.

```
package ru.alexanderklimov.cathouse.data;

import android.provider.BaseColumns;

public final class HotelContract {

    private HotelContract() {
    };

    public static final class GuestEntry implements BaseColumns {
        public final static String TABLE_NAME = "guests";

        public final static String _ID = BaseColumns._ID;
        public final static String COLUMN_NAME = "name";
        public final static String COLUMN_CITY = "city";
        public final static String COLUMN_GENDER = "gender";
        public final static String COLUMN_AGE = "age";

        public static final int GENDER_FEMALE = 0;
        public static final int GENDER_MALE = 1;
        public static final int GENDER_UNKNOWN = 2;
    }
}
```

В классе используется реализация интерфейса **BaseColumn**:

```
public static final class GuestEntry implements 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**. 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.

После создания класса мы можем изменить код в **EditorActivity** в том месте, где происходит выбор пола гостя через выпадающий список.

```
if (selection.equals(getString(R.string.gender_female))) {
    mGender = HotelContract.GuestEntry.GENDER_FEMALE; // Кошка
} else if (selection.equals(getString(R.string.gender_male))) {
    mGender = HotelContract.GuestEntry.GENDER_MALE; // Кот
} else {
    mGender = HotelContract.GuestEntry.GENDER_UNKNOWN; // Не определен
}
```

SQLiteOpenHelper

Следующий шаг - создание класса в пакете **data**, который наследуется от специального класса **SQLiteOpenHelper** и непосредственно работает с базой данных. В классе создаются константы для удобной работы. Также реализуются методы **onCreate()** и **onUpgrade()**.

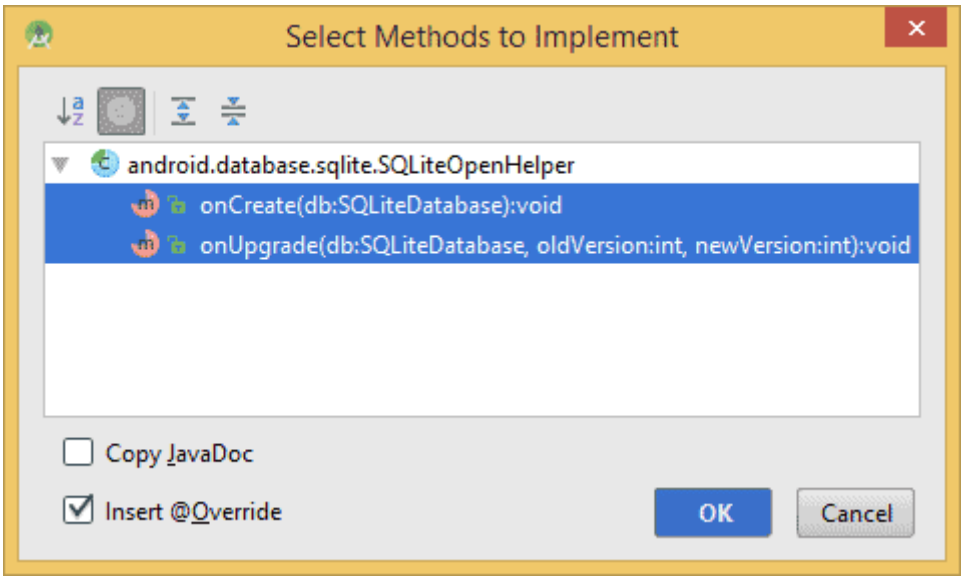
Созданный класс будет работать с базой данных - добавлять, выбирать, удалять записи и прочие операции.

Напомню, как выглядит схема нашей таблицы.

```
CREATE TABLE guests(_id INTEGER PRIMARY KEY AUTOINCREMENT,
                    name TEXT NOT NULL,
                    city TEXT NOT NULL,
                    gender INTEGER NOT NULL DEFAULT 3,
                    age INTEGER NOT NULL DEFAULT 0);
```

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, **HotelDbHelper**. 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.:

Класс **HotelDbHelper**.

```
package ru.alexanderklimov.cathouse.data;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

import ru.alexanderklimov.cathouse.data.HotelContract.GuestEntry;


public class HotelDbHelper extends SQLiteOpenHelper {

    public static final String LOG_TAG = HotelDbHelper.class.getSimpleName();

    /**
     * Имя файла базы данных
     */
    private static final String DATABASE_NAME = "hotel.db";

    /**
     * Версия базы данных. При изменении схемы увеличить на единицу
     */
    private static final int DATABASE_VERSION = 1;

    /**
     * Конструктор {@link HotelDbHelper}.
     *
     * @param context Контекст приложения
     */
    public HotelDbHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    /**
     * Вызывается при создании базы данных
     */
    @Override
    public void onCreate(SQLiteDatabase db) {
        // Строка для создания таблицы
        String SQL_CREATE_GUESTS_TABLE = "CREATE TABLE " + GuestEntry.TABLE_NAME + " ("
                + HotelContract.GuestEntry._ID + " INTEGER PRIMARY KEY AUTOINCREMENT, "
                + GuestEntry.COLUMN_NAME + " TEXT NOT NULL, "
                + GuestEntry.COLUMN_CITY + " TEXT NOT NULL, "
                + GuestEntry.COLUMN_GENDER + " INTEGER NOT NULL DEFAULT 3, "
                + GuestEntry.COLUMN_AGE + " INTEGER NOT NULL DEFAULT 0);";

        // Запускаем создание таблицы
        db.execSQL(SQL_CREATE_GUESTS_TABLE);
    }

    /**
     * Вызывается при обновлении схемы базы данных
     */
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
```

```
    }
}
```

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.

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

Метод 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.

```
@Override
public void onCreate(SQLiteDatabase db) {
    // Строка для создания таблицы
    String SQL_CREATE_GUESTS_TABLE = "CREATE TABLE " + GuestEntry.TABLE_NAME + " ("
            + HotelContract.GuestEntry._ID + " INTEGER PRIMARY KEY AUTOINCREMENT, "
            + GuestEntry.COLUMN_NAME + " TEXT NOT NULL, "
            + GuestEntry.COLUMN_CITY + " TEXT NOT NULL, "
            + GuestEntry.COLUMN_GENDER + " INTEGER NOT NULL DEFAULT 3, "
            + GuestEntry.COLUMN_AGE + " INTEGER NOT NULL DEFAULT 0);";

    // Запускаем создание таблицы
    db.execSQL(SQL_CREATE_GUESTS_TABLE);
}
```

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.

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.

Итак, 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);
}
```

Когда ваше приложение будет готово, то в папке **data/data/имя_пакета/databases** появится файл **hotel.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 готового плагина для просмотра таблиц (в Eclipse есть), но вроде уже видел плагин от сторонних разработчиков. А пока вам придётся скачивать из устройства файл базы данных и просматривать его на компьютере специальными программами, работающими с SQLite на локальном компьютере.

Работаем с записями базы данных

Чтобы проверить работоспособность базы данных, в главной активности поместим вспомогательный метод **displayDatabaseInfo()** для отображения информации.

```
package ru.alexanderklimov.cathouse;

import android.content.Intent;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.support.design.widget.FloatingActionButton;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.TextView;

import ru.alexanderklimov.cathouse.data.HotelContract.GuestEntry;
import ru.alexanderklimov.cathouse.data.HotelDbHelper;

public class MainActivity extends AppCompatActivity {

    private HotelDbHelper mDbHelper;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent intent = new Intent(MainActivity.this, EditorActivity.class);
                startActivity(intent);
            }
        });

        mDbHelper = new HotelDbHelper(this);
    }

    @Override
    protected void onStart() {
        super.onStart();
        displayDatabaseInfo();
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
```

```
            case R.id.action_insert_new_data:
                // Пока ничего не делаем
                return true;
            case R.id.action_delete_all_entries:
                // Пока ничего не делаем
                return true;
        }

        return super.onOptionsItemSelected(item);
    }

    private void displayDatabaseInfo() {
        // Создадим и откроем для чтения базу данных
        SQLiteDatabase db = mDbHelper.getReadableDatabase();

        // Зададим условие для выборки - список столбцов
        String[] projection = {
                GuestEntry._ID,
                GuestEntry.COLUMN_NAME,
                GuestEntry.COLUMN_CITY,
                GuestEntry.COLUMN_GENDER,
                GuestEntry.COLUMN_AGE };

        // Делаем запрос
        Cursor cursor = db.query(
                GuestEntry.TABLE_NAME,   // таблица
                projection,            // столбцы
                null,                  // столбцы для условия WHERE
                null,                  // значения для условия WHERE
                null,                  // Don't group the rows
                null,                  // Don't filter by row groups
                null);                   // порядок сортировки

        TextView displayTextView = (TextView) findViewById(R.id.text_view_info);

        try {
            displayTextView.setText("Таблица содержит " + cursor.getCount() + " гостей.\n
\n");
            displayTextView.append(GuestEntry._ID + " - " +
                    GuestEntry.COLUMN_NAME + " - " +
                    GuestEntry.COLUMN_CITY + " - " +
                    GuestEntry.COLUMN_GENDER + " - " +
                    GuestEntry.COLUMN_AGE + "\n");

            // Узнаем индекс каждого столбца
            int idColumnIndex = cursor.getColumnIndex(GuestEntry._ID);
            int nameColumnIndex = cursor.getColumnIndex(GuestEntry.COLUMN_NAME);
            int cityColumnIndex = cursor.getColumnIndex(GuestEntry.COLUMN_CITY);
            int genderColumnIndex = cursor.getColumnIndex(GuestEntry.COLUMN_GENDER);
            int ageColumnIndex = cursor.getColumnIndex(GuestEntry.COLUMN_AGE);

            // Проходим через все ряды
            while (cursor.moveToNext()) {
                // Используем индекс для получения строки или числа
                int currentID = cursor.getInt(idColumnIndex);
                String currentName = cursor.getString(nameColumnIndex);
                String currentCity = cursor.getString(cityColumnIndex);
                int currentGender = cursor.getInt(genderColumnIndex);
```

```
                int currentAge = cursor.getInt(ageColumnIndex);
                // Выводим значения каждого столбца
                displayTextView.append(("\n" + currentID + " - " +
                        currentName + " - " +
                        currentCity + " - " +
                        currentGender + " - " +
                        currentAge));
            }
        } finally {
            // Всегда закрываем курсор после чтения
            cursor.close();
        }
    }
}
```

Массив **projection** - это список столбцов, которые нас интересуют. В SQL-запросе мы их указываем в операторе **SELECT**:

```
SELECT name, city FROM guests;
```

В методе **query()** третий и четвёртый параметр определяют условие **WHERE**. Возьмём случай с выражением:

```
SELECT * FROM guests WHERE _id = 1;
```

В коде такое выражение выглядело бы так.

```
String selection = GuestEntry._ID + "=?";
String[] selectionArgs = {"1"};
```

Как видим, в знак вопроса подставляется нужное значение.

Посмотрите ещё несколько примеров.

Последний аргумент отвечает за сортировку по возрастанию или убыванию. Например, по возрасту.

```
SELECT name FROM guests WHERE _id > 1 BY age DESC;
```

```
// Зададим условие для выборки - список столбцов
String[] projection = {
        GuestEntry.COLUMN_NAME };

String selection = GuestEntry._ID + ">?";
String[] selectionArgs = {"1"};

Cursor cursor = db.query(
        GuestEntry.TABLE_NAME, // таблица
        projection,            // столбцы
        selection,             // столбцы для условия WHERE
        selectionArgs,         // значения для условия WHERE
        null,                  // Don't group the rows
        null,                  // Don't filter by row groups
        GuestEntry.COLUMN_AGE + " DESC");  // порядок сортировки
```

Чтение данных

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

Первый способ. Метод query()

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 = db.query(
        GuestEntry.TABLE_NAME,   // таблица
        projection,            // столбцы
        null,                  // столбцы для условия WHERE
        null,                  // значения для условия WHERE
        null,                  // Don't group the rows
        null,                  // Don't filter by row groups
        null);                   // порядок сортировки

// Проходим через все ряды
while (cursor.moveToNext()) {
    // Используем индекс для получения строки или числа
    int currentID = cursor.getInt(idColumnIndex);
    String currentName = cursor.getString(nameColumnIndex);
    String currentCity = cursor.getString(cityColumnIndex);
    int currentGender = cursor.getInt(genderColumnIndex);
    int currentAge = cursor.getInt(ageColumnIndex);
    // Выводим значения каждого столбца
    displayTextView.append(("\n" + currentID + " - " +
            currentName + " - " +
            currentCity + " - " +
            currentGender + " - " +
            currentAge));
}
```

Второй способ. Метод rawQuery()

Второй способ использует сырой (raw) SQL-запрос. Сначала формируется строка запроса и отдаётся 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();
```

Запустите проект. При запуске создаётся база 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 **hotel.db**. Метод **getReadableDatabase** создаёт или открывает базу данных.

Сейчас мы увидим, что пока у нас 0 гостей.

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.

Вставка данных для проверки

Рассмотрим, как вставлять новые данные. Добавим в меню главной активности пункт "Вставить данные". Для вставки данных применяется метод **ContentValues.put()**. В методе указываются ключ и значение. В качестве ключа выступает имя столбца таблицы, а его значением будет нужная информация о госте. Так как идентификатор будет вставляться автоматически, то его не используем. После того, как вы заполните все столбцы таблицы, вызывайте метод **insert()**, который и разместит данные в базе.

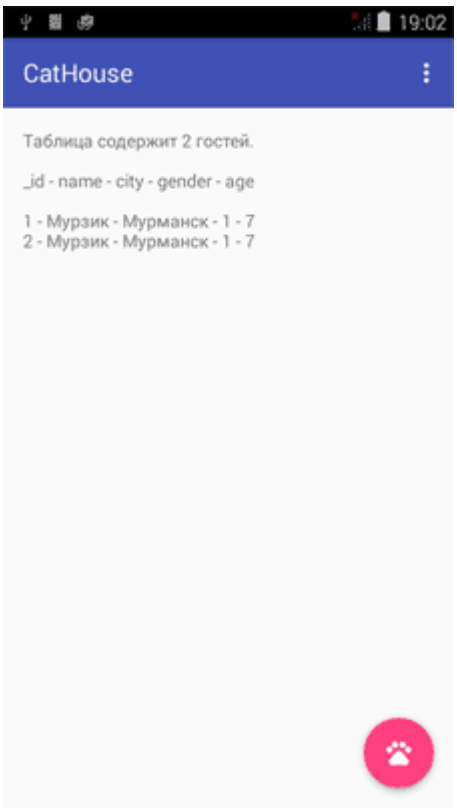
Напишем вспомогательный метод.

```
private void insertGuest() {

    // Gets the database in write mode
    SQLiteDatabase db = mDbHelper.getWritableDatabase();
    // Создаем объект ContentValues, где имена столбцов ключи,
    // а информация о госте является значениями ключей
    ContentValues values = new ContentValues();
    values.put(GuestEntry.COLUMN_NAME, "Мурзик");
    values.put(GuestEntry.COLUMN_CITY, "Мурманск");
    values.put(GuestEntry.COLUMN_GENDER, GuestEntry.GENDER_MALE);
    values.put(GuestEntry.COLUMN_AGE, 7);

    long newRowId = db.insert(GuestEntry.TABLE_NAME, null, values);
}
```

Вызовем метод в обработчике нажатия пункта меню.

```
case R.id.action_insert_new_data:
    insertGuest();
    displayDatabaseInfo();
    return true;
```

Сразу после вставки вызываем метод **displayDatabaseInfo()**, чтобы увидеть результат. Можно нажимать несколько раз. Так как данные жёстко заданы в коде, то увидим одинаковые данные, кроме увеличивающего значения идентификатора.



Вставка данных. Общая информация

Теперь разберём подробнее, как делать вставки.

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.

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.

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.

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);
```

Научившись вставлять данные, можно заняться второй активностью, которая и предназначена для этих целей.

Наполняем базу данных

Создадим вспомогательный метод для вставки записи в базу данных. Для этого считываем данные, которые вводятся в текстовые поля, а далее по предыдущему учебному примеру.

```
private void insertGuest() {
    // Считываем данные из текстовых полей
    String name = mNameEditText.getText().toString().trim();
    String city = mCityEditText.getText().toString().trim();
    String ageString = mAgeEditText.getText().toString().trim();
    int age = Integer.parseInt(ageString);

    HotelDbHelper mDbHelper = new HotelDbHelper(this);

    SQLiteDatabase db = mDbHelper.getWritableDatabase();

    ContentValues values = new ContentValues();
    values.put(GuestEntry.COLUMN_NAME, name);
    values.put(GuestEntry.COLUMN_CITY, city);
    values.put(GuestEntry.COLUMN_GENDER, mGender);
    values.put(GuestEntry.COLUMN_AGE, age);

    // Вставляем новый ряд в базу данных и запоминаем его идентификатор
    long newRowId = db.insert(GuestEntry.TABLE_NAME, null, values);

    // Выводим сообщение в успешном случае или при ошибке
    if (newRowId == -1) {
        // Если ID  -1, значит произошла ошибка
        Toast.makeText(this, "Ошибка при заведении гостя", Toast.LENGTH_SHORT).show();
    } else {
        Toast.makeText(this, "Гость заведён под номером: " + newRowId, Toast.LENGTH_SHORT).sh
ow();
    }
}
```

Метод вызывается в меню для значка с галочкой, которая выводится на панели действия активности.

```
case R.id.action_save:
    insertGuest();
    // Закрываем активность
    finish();
    return true;
```

Запускаем проект и проверяем работу кода.

Изменение данных

Обновление не реализовано в программе, проделайте это самостоятельно.

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(GuestEntry.COLUMN_NAME, "Мурзик");
db.update(GuestEntry.TABLE_NAME,
        values,
        GuestEntry.COLUMN_NAME + "= ?", 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(GuestEntry.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(GuestEntry.TABLE_NAME.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

Также не реализовано. Проделайте самостоятельно.

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)
```

Внедрение опасного кода

При работе с базой данной надо следить за безопасностью данных. Опытный пользователь может удалить базу. На своём устройстве он делать этого может и не будет, но на устройстве жертвы вполне.

Простой вариант атаки. Допустим у нас есть поле для ввода идентификатора, чтобы узнать информацию о госте. Нормальный пользователь введёт число "3" для поиска третьего гостя. В коде это будет следующим образом.

```
String selection = "GuestEntry._ID + " == " + userInput + ";";
```

Тогда идентификатор будет **_ID == 2;**.

Хакер может ввести следующую строку в текстовое поле:

```
1; DROP TABLE guests;
```

В коде это превратится в следующее:

```
_ID == 1; DROP TABLE guests;
```

Таким образом вредитель внедрил нежелательный код, который удалит таблицу.

Также вы можете скачать исходный код (../apk/cathouse_sqlite.zip).

Дополнительное чтение

Обсуждение статьи (http://forum.alexanderklimov.ru/viewtopic.php?id=33) на форуме.