**Простейшая база данных Часть третья.**

В [предыдущем примере](http://developer.alexanderklimov.ru/android/sqlite/simplydatabase.php2) использовалась таблица с контактами, которая включала в себя имя, номер телефона и возраст кота. Изменим таблицу, заменив колонку с возрастом на колонку с картинкой. На самом деле в этой колонке будет содержаться имя файла с картинкой, а сам файл будет находиться в определённой папке на внешнем накопителе, так как держать картинки в базе данных не рекомендуется.

Стандартная разметка **ListView** нам не подходит, поэтому создадим собственную разметку для элемента списка в файле **res/layout/list\_item.xml**:

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

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

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="horizontal">

<ImageView

android:id="@+id/imageViewPhoto"

android:layout\_width="wrap\_content"

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

<LinearLayout

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical">

<LinearLayout

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:orientation="horizontal">

<TextView

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Имя: "/>

<TextView

android:id="@+id/textViewName"

android:layout\_width="wrap\_content"

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

</LinearLayout>

<LinearLayout

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:orientation="horizontal">

<TextView

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Номер: "/>

<TextView

android:id="@+id/textViewPhone"

android:layout\_width="wrap\_content"

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

</LinearLayout>

</LinearLayout>

</LinearLayout>

Внесём небольшие изменения в класс **Contact**, убрав поле **mAge** и добавив поле **mPhotoFile**. Соответственно, изменим и параметры в конструкторах.

package ru.alexanderklimov.database;

public class Contact {

private int mId;

private String mName;

private String mPhoneNumber;

// private int mAge;

private String mPhotoFile;

// Пустой констуктор

public Contact() {

}

// Конструктор с параметрами

public Contact(int id, String name, String phoneNumber, String photoFile) {

this.mId = id;

this.mName = name;

this.mPhoneNumber = phoneNumber;

this.mPhotoFile = photoFile;

//this.mAge = age;

}

// Конструктор с параметрами

public Contact(String name, String phoneNumber, String photoFile) {

this.mName = name;

this.mPhoneNumber = phoneNumber;

this.mPhotoFile = photoFile;

//this.mAge = age;

}

// Создание геттеров-сеттеров

public int getID() {

return this.mId;

}

public void setID(int id) {

this.mId = id;

}

public String getName() {

return this.mName;

}

public void setName(String name) {

this.mName = name;

}

public String getPhoneNumber() {

return this.mPhoneNumber;

}

public void setPhoneNumber(String phoneNumber) {

this.mPhoneNumber = phoneNumber;

}

public String getPhotoFile() {

return this.mPhotoFile;

}

public void setPhotoFile(String photoFile) {

this.mPhotoFile = photoFile;

}

// public int getAge() {

// return this.mAge;

// }

//

// public void setAge(int age) {

// this.mAge = age;

// }

@Override

public String toString() {

return this.mName + " (" + this.mPhoneNumber + ")" + " " + this.mPhotoFile;

}

}

onUpgrade() и onDowngrade()

Здесь возникает проблема. Мы поменяли класс и нам нужно поменять структуру таблицы базы данных, добавив новую колонку, которая будет содержать информацию о пути к фотографии. Но когда пользователь обновится, то у него структура таблицы не изменится. И тогда произойдёт ошибка.

Чтобы определить, нуждается ли база данных в обновлении, класс **SQLiteOpenHelper** проверяет номер версии, который мы присваиваем через константу. В конструкторе он используется в суперклассе.

У баз данных SQLite имеется свой внутренний номер версии схемы базы данных, которым вы не можете управлять. Он увеличивается при любом изменении схемы базы данных. Вы указываете свой номер версии, который может как повышаться, так и понижаться.

private static final int DATABASE\_VERSION = 2;

Скорее всего вы будете повышать номер версии. Понижение встречается реже, когда нужно отменить изменения, внесённые при предыдущем неудачном обновлении.

Когда пользователь устанавливает новую версию приложения и приложение в первый раз обращается к базе данных, то класс сравнивает свой номер версии с номером версии базы данных на устройстве. Если номер версии в коде класса выше внутреннего номера версии базы, вызывается метод **onUpgrade()**. Соответственно, если номер версии ниже, то вызывается противоположный метод **onDowngrade()**. После выполнения одного из этих методов внутренний номер версии базы данных заменяется номером версии из кода вашего класса.

Если пользователь установил предыдущую версию приложения и обращался к базе данных, то база данных уже существует. При существующей базе данных класс не будет создавать её заново.

Метод **onUpgrade()** получает три параметра — базу данных SQLite, номер версии самой базы и номер версии, переданный суперклассу **SQLiteOpenHelper**.

По номерам версий можно определить, какие изменения должны вноситься в базу данных в зависимости от того, какая версия базы уже установлена на устройстве пользователя. Это важно для сложных проектов, в которых структура менялась несколько раз.

Общий вид кода для метода **onUpgrade()** будет следующим.

@Override

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

if (oldVersion == 1) {

//Код, выполняемый для версии 1

}

if (oldVersion < 3) {

//Код, выполняемый для версии 1 или 2

}

}

Метод **onDowngrade()** работает аналогичным образом, но выполняется при понижении версии.

Посмотрим, как этот способ применим к нашему примеру. Мы решили обновить структуру таблицы, добавив новый столбец **COLUMN\_PHOTO** в готовую таблицу **DATABASE\_TABLE**. Изменения должны коснуться как новых, так и существующих пользователей, поэтому соответствующий код должен быть и в методе **onCreate()** и в методе **onUpgrade()**. Метод **onCreate()**гарантирует, что новый столбец будет присутствовать у всех новых пользователей, а метод **onUpgrade()** добавит столбец у всех существующих пользователей.

Для удобства создадим отдельный метод **updateDatabase()**, чтобы не повторять одинаковый код в двух местах. Метод будет вызываться из **onCreate()** и **onUpgrade()**. Код из **onCreate()**переместим в новый метод **updateDatabase()**, и вдобавок напишем код создания дополнительного столбца.

@Override

public void onCreate(SQLiteDatabase db) {

// db.execSQL(DATABASE\_CREATE\_SCRIPT);

// переносим в наш метод

updateDatabase(db, 0, DATABASE\_VERSION);

}

@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);

// код в нашем методе

updateDatabase(db, oldVersion, newVersion);

}

private void updateDatabase(SQLiteDatabase db, int oldVersion, int newVersion) {

if (oldVersion < 1) {

// раньше этот код был в onCreate()

db.execSQL(DATABASE\_CREATE\_SCRIPT);

}

if (oldVersion < 2) {

//Код добавления нового столбца

// Этот код будет выполняться

// в том случае, если у пользователя уже установлена версия 2 базы данных.

db.execSQL("ALTER TABLE " + DATABASE\_TABLE + " ADD COLUMN " + COLUMN\_PHOTO + " TEXT;");

}

if (oldVersion < 7) {

Contact newContact = new Contact("Кот версии 6", "7777777", "cat7.jpg");

ContentValues values = new ContentValues();

values.put(COLUMN\_CAT\_NAME, newContact.getName());

values.put(COLUMN\_PHONE, newContact.getPhoneNumber());

values.put(COLUMN\_PHOTO, newContact.getPhotoFile());

db.insert(DATABASE\_TABLE, null, values);

}

}

В старом коде мы удаляли таблицу и создавали её заново при обновлении версии:

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

Для демонстрации примеров этот способ подойдёт. Но представьте себе, что пользователь уже создал много своих записей для любимых котов. Если он обновит своё приложение, то все записи его исчезнут. Поэтому нам нужно не удалить таблицу, а добавить в неё новый столбец. Для этой операции существует команда SQL **ALTER TABLE <ИМЯ\_ТАБЛИЦЫ ADD COLUMN <ИМЯ\_СТОЛБЦА> TYPE**.

db.execSQL("ALTER TABLE " + DATABASE\_TABLE + " ADD COLUMN " + COLUMN\_PHOTO + " TEXT;");

Команда **ALTER TABLE** может использоваться для переименования таблиц. Например, если вы захотите переименовать таблицу CATS в KITTENS, это делается так:

ALTER TABLE CATS RENAME TO KITTENS

Когда вы меняете структуру таблицы, то будьте внимательны. Вам придётся пройтись по всему классу и поменять код под новые требования. Очень часто какой-то кусок пропускают и потом приходится искать источник ошибок. Например, меняется номер индекса. И нужно не забыть поменять числа в коде, так как мы не удаляли старые колонки, а добавляли новые.

package ru.alexanderklimov.database;

import android.content.ContentValues;

import android.content.Context;

import android.database.Cursor;

import android.database.sqlite.SQLiteDatabase;

import android.database.sqlite.SQLiteOpenHelper;

import android.provider.BaseColumns;

import android.util.Log;

import java.util.ArrayList;

import java.util.List;

public class DatabaseHelper extends SQLiteOpenHelper implements BaseColumns {

// названия столбцов

public static final String COLUMN\_ID = BaseColumns.\_ID;

public static final String COLUMN\_CAT\_NAME = "cat\_name";

public static final String COLUMN\_PHONE = "phone";

//public static final String COLUMN\_AGE = "age";

public static final String COLUMN\_PHOTO = "photo";

// имя таблицы

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

// имя базы данных

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

// версия базы данных

//private static final int DATABASE\_VERSION = 1;

private static final int DATABASE\_VERSION = 5;

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

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

+ " integer primary key autoincrement, " + COLUMN\_CAT\_NAME

+ " text not null, " + COLUMN\_PHONE + " integer, " + COLUMN\_PHOTO

+ " text not null);";

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);

updateDatabase(db, 0, DATABASE\_VERSION);

}

@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);

updateDatabase(db, oldVersion, newVersion);

}

private void updateDatabase(SQLiteDatabase db, int oldVersion, int newVersion) {

if (oldVersion < 1) {

// раньше этот код был в onCreate()

db.execSQL(DATABASE\_CREATE\_SCRIPT);

}

if (oldVersion < 2) {

//Код добавления нового столбца

// Этот код будет выполняться

// в том случае, если у пользователя уже установлена версия 2 базы данных.

db.execSQL("ALTER TABLE " + DATABASE\_TABLE + " ADD COLUMN " + COLUMN\_PHOTO + " TEXT;");

}

if (oldVersion < 7) {

Contact newContact = new Contact("Кот версии 6", "7777777", "cat7.jpg");

ContentValues values = new ContentValues();

values.put(COLUMN\_CAT\_NAME, newContact.getName());

values.put(COLUMN\_PHONE, newContact.getPhoneNumber());

values.put(COLUMN\_PHOTO, newContact.getPhotoFile());

db.insert(DATABASE\_TABLE, null, values);

}

}

// Добавляем новый контакт

public void addContact(Contact contact) {

SQLiteDatabase db = this.getWritableDatabase();

ContentValues values = new ContentValues();

values.put(COLUMN\_CAT\_NAME, contact.getName());

values.put(COLUMN\_PHONE, contact.getPhoneNumber());

//values.put(COLUMN\_AGE, contact.getAge());

values.put(COLUMN\_PHOTO, contact.getPhotoFile());

// Вставляем строку в таблицу

db.insert(DATABASE\_TABLE, null, values);

db.close();

}

// Получить контакт по идентификатору

public Contact getContact(int id) {

SQLiteDatabase db = this.getReadableDatabase();

Cursor cursor = db.query(DATABASE\_TABLE, new String[]{COLUMN\_ID,

COLUMN\_CAT\_NAME, COLUMN\_PHONE, COLUMN\_PHOTO}, COLUMN\_ID + "=?",

new String[]{String.valueOf(id)}, null, null, null, null);

if (cursor != null)

cursor.moveToFirst();

// Contact contact = new Contact(Integer.parseInt(cursor.getString(0)),

// cursor.getString(1), cursor.getString(2), Integer.parseInt(cursor.getString(3)));

Contact contact = new Contact(Integer.parseInt(cursor.getString(0)),

cursor.getString(1), cursor.getString(2), cursor.getString(4));

cursor.close();

return contact;

}

// Получить все контакты

public List<Contact> getAllContacts() {

List<Contact> contactList = new ArrayList<>();

// Выбираем всю таблицу

String selectQuery = "SELECT \* FROM " + DATABASE\_TABLE;

SQLiteDatabase db = this.getWritableDatabase();

Cursor cursor = db.rawQuery(selectQuery, null);

// Проходим по всем строкам и добавляем в список

if (cursor.moveToFirst()) {

do {

Contact contact = new Contact();

contact.setID(Integer.parseInt(cursor.getString(0)));

contact.setName(cursor.getString(1));

contact.setPhoneNumber(cursor.getString(2));

contact.setPhotoFile(cursor.getString(4));

//contact.setAge(Integer.parseInt(cursor.getString(3)));

contactList.add(contact);

} while (cursor.moveToNext());

}

cursor.close();

return contactList;

}

// Получить число контактов

public int getContactsCount() {

String countQuery = "SELECT \* FROM " + DATABASE\_TABLE;

SQLiteDatabase db = this.getReadableDatabase();

Cursor cursor = db.rawQuery(countQuery, null);

int count = cursor.getCount();

cursor.close();

return count;

}

public int updateContact(Contact contact) {

SQLiteDatabase db = this.getWritableDatabase();

ContentValues values = new ContentValues();

values.put(COLUMN\_CAT\_NAME, contact.getName());

values.put(COLUMN\_PHONE, contact.getPhoneNumber());

//values.put(COLUMN\_AGE, contact.getAge());

values.put(COLUMN\_PHOTO, contact.getPhotoFile());

// обновляем строку

return db.update(DATABASE\_TABLE, values, COLUMN\_ID + " = ?",

new String[]{String.valueOf(contact.getID())});

}

public void deleteContact(Contact contact) {

SQLiteDatabase db = this.getWritableDatabase();

db.delete(DATABASE\_TABLE, COLUMN\_ID + " = ?",

new String[]{String.valueOf(contact.getID())});

db.close();

}

}

Теперь попробуем получить записи в списке, пока без вывода изображений. Мы не будем добавлять новые записи, а проверим, как отобразятся старые записи при обновлении программы. Напомню, что мы убрали колонку с возрастом и добавили колонку для фото:

// Если этот код работает, его написал Александр Климов,

// а если нет, то не знаю, кто его писал.

package ru.alexanderklimov.database;

import android.os.Bundle;

import android.support.v7.app.AppCompatActivity;

import android.util.Log;

import android.view.View;

import android.widget.ArrayAdapter;

import android.widget.ListView;

import java.util.List;

public class MainActivity extends AppCompatActivity {

private DatabaseHelper mDatabaseHelper;

//private StringBuilder mStringBuilder = new StringBuilder();

//private List<Contact> mContacts;

private ListView mCatsListView;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

mCatsListView = (ListView) findViewById(R.id.listViewCats);

mDatabaseHelper = new DatabaseHelper(this);

// Вставляем контакты

// mDatabaseHelper.addContact(new Contact("Test", "44", "catOne.jpg"));

}

public void onClick(View view) {

// Считываем все контакты

Log.d("Reading", "Reading all contacts..");

List<Contact> contacts = mDatabaseHelper.getAllContacts();

ArrayAdapter<Contact> adapter = new ArrayAdapter<>(this,

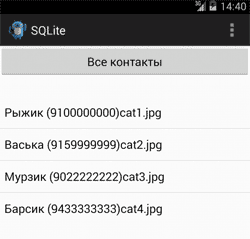
android.R.layout.simple\_list\_item\_1, android.R.id.text1, contacts);

mCatsListView.setAdapter(adapter);

}

}

Мы видим, что всё работает и в списке выводятся имена файлов к картинкам.



Осталось написать адаптер к списку, который бы позволял задействовать **ImageView** для картинок. Создадим свой адаптер на основе **ArrayAdapter**.

package ru.alexanderklimov.database;

import android.content.Context;

import android.util.Log;

import android.view.LayoutInflater;

import android.view.View;

import android.view.ViewGroup;

import android.widget.ArrayAdapter;

import android.widget.ImageView;

import android.widget.TextView;

import java.util.List;

public class ImageAdapter extends ArrayAdapter<Contact> {

private final List<Contact> mContactList;

private final Context mContext;

public ImageAdapter(Context context, List<Contact> list) {

super(context, R.layout.list\_item, list);

this.mContext = context;

this.mContactList = list;

}

@Override

public View getView(int position, View convertView, ViewGroup parent) {

LayoutInflater inflater = (LayoutInflater) mContext

.getSystemService(Context.LAYOUT\_INFLATER\_SERVICE);

View rowView = inflater.inflate(R.layout.list\_item, parent, false);

TextView nameTextView = (TextView) rowView.findViewById(R.id.textViewName);

TextView phoneNumberTextView = (TextView) rowView.findViewById(R.id.textViewPhone);

ImageView photoImageView = (ImageView) rowView.findViewById(R.id.imageViewPhoto);

nameTextView.setText(mContactList.get(position).getName());

phoneNumberTextView.setText(mContactList.get(position).getPhoneNumber());

// String s = list.get(position).toString();

photoImageView.setImageResource(R.mipmap.ic\_launcher);

String imagePath = mContactList.get(position).getPhotoFile();

Log.i("Test", imagePath);

return rowView;

}

}

Мне лень было копировать картинки на SD-карту, поэтому в адаптере я просто присвоил компоненту **ImageView** рисунок из ресурсов. В логах вы увидите, имена файлов для картинок. Вам надо составить полный путь к файлу и вывести изображение в **ImageView** (ваше домашнее задание).

Наполним список через созданный адаптер.

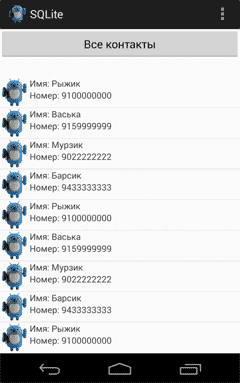
List<Contact> contacts = mDatabaseHelper.getAllContacts();

ArrayAdapter<Contact> adapter = new ImageAdapter(this, contacts);

mCatsListView.setAdapter(adapter);

Я вставлял в базу одни и те же данные, поэтому пример получился однообразным. В реальном примере записи будут уникальными со своими индивидуальными картинками.

Это был тренировочный пример с упрощённым кодом.



В примере использовался **ArrayAdapter**, работающий с массивом. Для небольшой базы данных это не критично. Но если приложение работает с очень большим объемом информации, её чтение из базы данных займет некоторое время. Кроме того, для хранения массива потребуется много памяти. Поэтому для извлечения данных используются другие адаптеры, использующие курсоры. Общим предком для таких адаптеров является класс **CursorAdapter**.