**Связанные таблицы**

Иногда одной таблицы мало. Создать несколько таблиц не проблема, но важно связать их между собой. Представьте себе, чтобы создаём социальную сеть для котов. В первой таблице Контакты мы храним стандартные данные: идентификатор, имя кота, его телефон. Во второй таблице Профиль содержится идентификатор, характеристика и специальное поле, которое связывает профиль с определённым котом из первой таблицы. Любой невский маскарадный кот может иметь несколько профилей, например он может указать в своём профиле, что он Храбрый, Умный, Красивый и т.д. И британский голубой (это не то, что вы подумали) кот может вступить с ним в  переписку: "Уйди, прАтивный".

Мы знаем, как создавать таблицу для базы данных. Вторая таблица создаётся в том же классе. Смотрим код.

package ru.alexanderklimov.test;

import java.util.ArrayList;

import java.util.List;

import android.content.ContentValues;

import android.content.Context;

import android.database.Cursor;

import android.database.sqlite.SQLiteDatabase;

import android.database.sqlite.SQLiteOpenHelper;

public class MultiTableDataBaseHelper extends SQLiteOpenHelper {

// версия базы данных

private static final int DATABASE\_VERSION = 1;

// имя базы данных

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

// имя таблиц

private static final String TABLE\_CONTACTS = "contacts"; // 1-я таблица

private static final String TABLE\_PROFILES = "profiles"; // 2-я таблица

// Имена колонок

private static final String KEY\_ID = "\_id";

private static final String KEY\_NAME = "name";

private static final String KEY\_PHONE = "phone\_number";

private static final String KEY\_DESCRIPTION = "description";

private static final String KEY\_BELONGTO = "belongto";

public MultiTableDataBaseHelper(Context context) {

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

}

@Override

public void onCreate(SQLiteDatabase db) {

// TODO Auto-generated method stub

// Создаём первую таблицу

String CREATE\_CONTACTS\_TABLE = "CREATE TABLE " + TABLE\_CONTACTS + "("

+ KEY\_ID + " INTEGER PRIMARY KEY autoincrement," + KEY\_NAME

+ " TEXT not null," + KEY\_PHONE + " TEXT not null" + ")";

db.execSQL(CREATE\_CONTACTS\_TABLE);

// Создаём вторую таблицу

String CREATE\_PROFILE\_TABLE = "CREATE TABLE " + TABLE\_PROFILES

+ "(" + KEY\_ID + " INTEGER PRIMARY KEY autoincrement,"

+ KEY\_BELONGTO + " INTEGER," + KEY\_DESCRIPTION + " TEXT not null"

+ ")";

db.execSQL(CREATE\_PROFILE\_TABLE);

}

@Override

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

// TODO Auto-generated method stub

// Удаляем старые таблицы

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

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

// и создаём их снова

onCreate(db);

}

// Добавляем новый контакт

void addContact(Contact contact) {

SQLiteDatabase db = this.getWritableDatabase();

ContentValues values = new ContentValues();

values.put(KEY\_NAME, contact.getName());

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

db.insert(TABLE\_CONTACTS, null, values);

db.close();

}

// Добавляем новое описание в профиль

void addProfileDescription(Profile profile) {

SQLiteDatabase db = this.getWritableDatabase();

ContentValues values = new ContentValues();

values.put(KEY\_BELONGTO, Integer.toString(profile.getBelongTo()));

values.put(KEY\_DESCRIPTION, profile.getDescription());

db.insert(TABLE\_PROFILES, null, values);

db.close();

}

// Получаем данные о контакте

Contact getContact(int id) {

SQLiteDatabase db = this.getReadableDatabase();

Cursor cursor = db.query(TABLE\_CONTACTS, new String[] { KEY\_ID,

KEY\_NAME, KEY\_PHONE }, KEY\_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));

return contact;

}

// Получим все контакты

public List<Contact> getAllContacts() {

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

String selectQuery = "SELECT \* FROM " + TABLE\_CONTACTS;

SQLiteDatabase db = this.getWritableDatabase();

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

if (cursor.moveToFirst()) {

do {

Contact contact = new Contact(Integer.parseInt(cursor

.getString(0)), cursor.getString(1),

cursor.getString(2));

contactList.add(contact);

} while (cursor.moveToNext());

}

return contactList;

}

// Обновляем данные в контакте

public int updateContact(Contact contact) {

SQLiteDatabase db = this.getWritableDatabase();

ContentValues values = new ContentValues();

values.put(KEY\_NAME, contact.getName());

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

return db.update(TABLE\_CONTACTS, values, KEY\_ID + " = ?",

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

}

// Удаляем контакт

public void deleteContact(Contact contact) {

SQLiteDatabase db = this.getWritableDatabase();

db.delete(TABLE\_CONTACTS, KEY\_ID + " = ?",

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

db.close();

}

// Подсчитываем число контактов

public int getContactsCount() {

String countQuery = "SELECT \* FROM " + TABLE\_CONTACTS;

SQLiteDatabase db = this.getReadableDatabase();

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

cursor.close();

return cursor.getCount();

}

// Все профили для контакта

public List<Profile> getAllProfilesWithContact(int contactID) {

List<Profile> profileList = new ArrayList<Profile>();

String selectQ = "SELECT \* FROM " + TABLE\_PROFILES + " WHERE "

+ KEY\_BELONGTO + "=" + String.valueOf(contactID) + ";";

SQLiteDatabase db = this.getWritableDatabase();

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

// Альтернативный вариант

// Cursor cursor2 = db.query(TABLE\_PROFILES, new String[] { KEY\_ID,

// KEY\_BELONGTO, KEY\_NAME }, KEY\_BELONGTO + "=?",

// new String[] { String.valueOf(contactID) }, null, null, null,

// null);

if (cursor.moveToFirst()) {

do {

Profile profile = new Profile(Integer.parseInt(cursor

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

profileList.add(profile);

} while (cursor.moveToNext());

}

return profileList;

}

// Все профили

public List<Profile> getAllProfiles() {

List<Profile> profileList = new ArrayList<Profile>();

String selectQ = "SELECT \* FROM " + TABLE\_PROFILES;

SQLiteDatabase db = this.getWritableDatabase();

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

if (cursor.moveToFirst()) {

do {

Profile profile = new Profile(Integer.parseInt(cursor

.getString(cursor.getColumnIndex(KEY\_ID))),

Integer.parseInt(cursor.getString(cursor

.getColumnIndex(KEY\_BELONGTO))),

cursor.getString(cursor.getColumnIndex(KEY\_DESCRIPTION)));

profileList.add(profile);

} while (cursor.moveToNext());

}

return profileList;

}

}

Структуру таблиц можно увидеть на скриншотах в конце статьи. В коде использователь два класса **Contact** и **Profile**. Создадим их.

**Contact**

package ru.alexanderklimov.test;

public class Contact {

private int mId;

private String mName;

private String mPhoneNumber;

public Contact(int id, String name, String phone\_number) {

this.mId = id;

this.mName = name;

this.mPhoneNumber = phone\_number;

}

public Contact(String name, String phone\_number) {

this.mName = name;

this.mPhoneNumber = phone\_number;

}

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 phone\_number) {

this.mPhoneNumber = phone\_number;

}

public String toString() {

return this.getName() + " с номером: " + this.getPhoneNumber() + "\n";

}

}

**Profile**

package ru.alexanderklimov.test;

public class Profile {

private int mId;

private int mBelongTo;

private String mDescription;

public Profile(int id, int belongTo, String name) {

this.mId = id;

this.mBelongTo = belongTo;

this.mDescription = name;

}

public Profile(int belongTo, String name) {

this.mBelongTo = belongTo;

this.mDescription = name;

}

public int getID() {

return mId;

}

public void setID(int id) {

this.mId = id;

}

public int getBelongTo() {

return mBelongTo;

}

public void setBelongTo(int belongTo) {

this.mBelongTo = belongTo;

}

public String getDescription() {

return mDescription;

}

public void setDescription(String name) {

this.mDescription = name;

}

@Override

public String toString() {

return this.getDescription() + " связан с "

+ Integer.toString(this.getBelongTo()) + "\n";

}

}

Добавим в разметку компонент **ListView** (activity\_main.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="vertical" >

<ListView

android:id="@+id/listView1"

android:layout\_width="match\_parent"

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

</ListView>

</LinearLayout>

Для наглядности создадим разметку для отдельного элемента списка, состоящего из двух **TextView**, в которых будет отображаться имя и описание из профиля (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="wrap\_content"

android:orientation="horizontal" >

<TextView

android:id="@+id/tvContact"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_gravity="center"

android:layout\_weight="1" />

<TextView

android:id="@+id/tvProfile"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_gravity="center"

android:layout\_weight="1" />

</LinearLayout>

В коде основной активности добавим записи в базу данных, а затем выведем какие-нибудь данные в список и в LogCat.

package ru.alexanderklimov.test;

import java.util.List;

import android.os.Bundle;

import android.app.Activity;

import android.content.Context;

import android.util.Log;

import android.view.LayoutInflater;

import android.view.View;

import android.view.ViewGroup;

import android.widget.BaseAdapter;

import android.widget.ListView;

import android.widget.TextView;

public class MainActivity extends Activity {

private List<Profile> mListOfProfile;

private ListView listView;

private MultiTableDataBaseHelper db;

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

setTitle("Две таблицы");

db = new MultiTableDataBaseHelper(this);

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

if (contacts.size() == 0) {

// Добавляем контакты в базу

db.addContact(new Contact("Васька", "111-11-11"));

db.addContact(new Contact("Барсик", "222-22-22"));

db.addContact(new Contact("Рыжик", "333-33-33"));

db.addContact(new Contact("Мурзик", "4444-44-44"));

contacts = db.getAllContacts();

}

// Выводим список контактов

Log.i("Contacts", contacts.toString());

mListOfProfile = db.getAllProfiles();

if (mListOfProfile.size() == 0) {

// Характеристики для первого кота

db.addProfileDescription(new Profile(contacts.get(0).getID(),

"Загадочный"));

db.addProfileDescription(new Profile(contacts.get(0).getID(),

"Хитрый"));

db.addProfileDescription(new Profile(contacts.get(0).getID(),

"Умный"));

// Характеристики для второго кота

db.addProfileDescription(new Profile(contacts.get(1).getID(),

"Гордый"));

db.addProfileDescription(new Profile(contacts.get(1).getID(),

"Добрый"));

db.addProfileDescription(new Profile(contacts.get(1).getID(),

"Ласковый"));

// Характеристики для третьего кота

db.addProfileDescription(new Profile(contacts.get(2).getID(),

"Отважный"));

db.addProfileDescription(new Profile(contacts.get(2).getID(),

"Беззаботный"));

// Характеристики для четвертого кота

db.addProfileDescription(new Profile(contacts.get(3).getID(),

"Неотразимый"));

// Выводим все характеристики для первого кота

mListOfProfile = db.getAllProfilesWithContact(contacts.get(0)

.getID());

// mListOfProfile = db.getAllProfiles();

} else {

// Если данные в базе уже есть, то выводим все характеристики для второго кота

mListOfProfile = db.getAllProfilesWithContact(contacts.get(1)

.getID());

}

// Выводим результат в LogCat

Log.i("Profile", mListOfProfile.toString());

listView = (ListView) findViewById(R.id.listView1);

listView.setAdapter(myAdapter);

}

BaseAdapter myAdapter = new BaseAdapter() {

@Override

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

if (convertView != null)

return convertView;

LayoutInflater inflater = (LayoutInflater) getSystemService(Context.LAYOUT\_INFLATER\_SERVICE);

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

TextView tvContact = (TextView) rowView.findViewById(R.id.tvContact);

TextView tvProfile = (TextView) rowView.findViewById(R.id.tvProfile);

tvContact.setText(db.getContact(

mListOfProfile.get(position).getBelongTo()).getName());

tvProfile.setText(mListOfProfile.get(position).getDescription());

return rowView;

}

@Override

public long getItemId(int position) {

return 0;

}

@Override

public Object getItem(int position) {

return null;

}

@Override

public int getCount() {

return mListOfProfile.size();

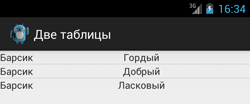
}

};

}

Итак, мы добавили в базу контакты четырёх котов. Затем каждый кот добавил о себе нужные характеристики. Первые два кота добавили к себе в профиль по три описания, третий код добавил два описания, а четвёртый кот Мурзик оказался самым скромным и добавил в профиль только одно описание.

При первом запуске вы увидите список характеристик первого кота, при повторном запуске - список для второго кота.



Таким образом вы должны понять принцип связывания двух таблиц. Поле **belongto** во второй таблице содержит числа, которые совпадают с идентификаторами котов из первой таблицы. В методе **getAllProfilesWithContact(int contactID)** вы передаёте идентификатор контакта и запрос возвращает все профили конкретного кота.

