Лабораторная работа № 8 по теме:
«Работа с СУБД SQLite»
Теоретическая часть

SQLite — встраиваемая система управления базами данных (СУБД), доступная всем Android-приложениям. SQLite не использует парадигму клиент-сервер, это означает, что движок SQLite не является отдельно работающим процессом, с которым взаимодействует программа. Вместо этого предоставляется библиотека, с которой приложение компонуется, и СУБД становится составной частью программы.

К преимуществам SQLite относится также то, что исходный код этой СУБД передан в общественное достояние, а значит ею можно пользоваться бесплатно.

На платформе Android доступен пакет android.database.sqlite, содержащий API, необходимые для использования СУБД SQLite.

Одним из основных принципов баз данных (БД) SQL является использование схемы, т.е. формального объявления того, как БД организована. Схема отражена в операторах SQL, использующихся при создании БД. Поэтому бывает полезно создать сопутствующий класс, известный как *класс контракта*, в котором явным образом определяется макет схемы БД.

Контрактный класс (англ. contract class) — это контейнер для констант, которые определяют имена URI, таблиц и столбцов. Данный класс позволяет использовать одни и те же константы во всех других классах одного пакета. Это позволяет изменить имя таблицы или столбца в одном месте и распространить это изменение по всему вашему коду.

В листинге 1 приведён пример определения имени таблицы и названий столбцов базы данных контактов.

Листинг 1. Класс контракта

```
package com.example.sqliteapp;
import android.provider.BaseColumns;

public final class DBContract {
    private DBContract() {}
    public static class DBEntry implements BaseColumns {
        public static final String TABLE_NAME = "people";
        public static final String COLUMN_NAME_NAME = "name";
        public static final String COLUMN_NAME_PHONE = "phone";
    }
}
```

После создания контрактного класса, необходимо реализовать методы, которые создают и управляют базой данных и таблицами.

SQLite хранит всю базу данных, включая таблицы, индексы и данные в одном стандартном файле с расширением db. На платформе Android файлы баз данных SQLite сохраняются в файловой системе мобильного устройства в личных папках конкретных приложений (в каталогах /data/data/package_name/databases, где package_name — имя пакета приложения, которое управляет этой базой данных).

Эти данные по умолчанию недоступны для других приложений или пользователей. Если требуется, чтобы сторонние приложения могли получить доступ к вашей базе данных, то необходимо использовать компонент **поставщик контента** (англ. Content Provider)

Для создания и управления базой данных на платформе Android имеется класс SQLiteOpenHelper, содержащий полезный набор API-интерфейсов. При использовании этого класс система выполняет потенциально длительные операции по созданию и обновлению базы данных только при необходимости, а не во время запуска приложения. Для этого нужно вызвать метод getWritableDatabase () или getReadableDatabase ().

Метод getWritableDatabase () создаёт и (или) открывает базу данных, которая будет использоваться для чтения и записи. После успешного открытия база данных кэшируется, поэтому можно вызывать этот метод

каждый раз, когда требуется писать в базу данных. После завершения работы с базой необходимо вызывать метод close () для освобождения ресурсов.

Метод getReadableDatabase () также создаёт и (или) открывает базу данных. Возвращает тот же объект, что и getWritableDatabase (), если только из-за какой-то проблемы, например, полного диска, не потребуется открытие базы в режиме только для чтения. В этом случае будет возвращен объект базы данных, доступный только для чтения.

Чтобы использовать SQLiteOpenHelper, необходимо создать собственный класс, наследуемый от SQLiteOpenHelper и переопределяющий методы обратного вызова onCreate () и onUpgrade (). Метод onCreate () вызывается при первом создании базы данных, а метод onUpgrade () — при модификации базы данных. В этих методах описывается логика создания и модификации базы.

В классе, наследуемом от SQLiteOpenHelper, можно также объявлять открытые строковые константы для названия таблиц и столбцов создаваемой базы данных, если не используется контрактный класс. Необходимо тщательно документировать тип данных каждого столбца, т. к. клиенту требуется эта информация для обращения к данным.

Также как контрактный класс, класс, расширяющий SQLiteOpenHelper, неявно наследует интерфейс BaseColumns, в котором определена строковая константа _id, представляющая имя поля для идентификаторов записей. В создаваемых таблицах базы данных поле _id должно иметь тип INTEGER PRIMARY KEY AUTOINCREMENT, т.е. поле _id является первичным ключом для таблицы. Описатель AUTOINCREMENT является необязательным.

На платформе Android для управления базой данных SQLite используется класс SQLiteDatabase. В этом классе есть набор методов для создания, удаления и управления базой данных SQL. Объект класса

SQLiteDatabase представляет базу данных и передается в качестве входного параметра в методы onCreate () и onUpgrade () класса, расширяющего SQLiteOpenHelper.

Создание и наполнение базы данных

Логика создания таблиц реализуется в методе обратного вызова onCreate () класса, расширяющего SQLiteOpenHelper (листинг 2). Там же можно заполнить таблицы начальными данными.

Листинг 2. Создание таблицы в БД

```
package com.example.sqliteapp;
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.provider.BaseColumns;

class DatabaseHelper extends SQLiteOpenHelper implements BaseColumns {
    public static final String DB_CONTACTS = "contacts.db";

    public DatabaseHelper(Context context) {
        super(context, DB_CONTACTS, null, 1);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL("CREATE TABLE " + DBContract.DBEntry.TABLE_NAME
                + " (_id INTEGER PRIMARY KEY AUTOINCREMENT, "
                + DBContract.DBEntry.COLUMN_NAME_NAME + " TEXT, " +
DBContract.DBEntry.COLUMN_NAME_PHONE + " TEXT);");
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion) {
    }
}
```

Для выполнения SQL-команды создания таблицы (CREATE TABLE) в листинге 2 используется метод execSQL (). Этот метод принимает строку с одной SQL-командой и выполняет её. При этом данный метод ничего не возвращает, поэтому его нельзя использовать, например, для выполнения запросов к базе данные (команда SELECT). Обратите внимание на то, что в строку с SQL-командой вставляются значения названия таблицы и её столбцов, которые были объявлены в классе контракта. При создании

строки с SQL-командой нужно соблюдать правила синтаксиса запросов на языке SQL.

Для вставки данных в таблицу рекомендуется использовать метод insert (), в который передаётся объект класса ContentValues. Данный класс используется, чтобы хранить данные в виде пар «ключ — значение». В качестве ключа указывают название столбца. Для добавления каждой пары используется метод put (), в который передаются два параметра: ключ и значение.

В метод insert () передаются три параметра:

1. имя таблицы;
2. второй параметр определяет, что делать в случае, если объект ContentValues пуст (т.е. не были указаны никакие значения). Если указывается имя столбца, платформа вставляет строку и устанавливает значение этого столбца равным null. Если указано значение null, платформа не будет вставлять строку при отсутствии значений;
3. объект класса ContentValues.

Метод insert () возвращают идентификатор созданной строки или -1, если при вставке данных произошла ошибка, например, если возник конфликт с уже существующими данными в базе.

В листинге 3 приведен пример добавления двух строк в таблицу.

Листинг 3. Добавление данных

```
ContentValues values = new ContentValues();

values.put(DBContract.DBEntry.COLUMN_NAME_NAME, "Иван Иванов");
values.put(DBContract.DBEntry.COLUMN_NAME_PHONE, "8-999-111-11-11");
db.insert(DBContract.DBEntry.TABLE_NAME,
DBContract.DBEntry.COLUMN_NAME_NAME, values);

values.put(DBContract.DBEntry.COLUMN_NAME_NAME, "Пётр Петров");
values.put(DBContract.DBEntry.COLUMN_NAME_PHONE, "8-999-222-22-22");
db.insert(DBContract.DBEntry.TABLE_NAME,
DBContract.DBEntry.COLUMN_NAME_NAME, values);
```

После создания таблиц в базе и, по необходимости, заполнения её начальными значениями в методе onCreate (), необходимо также определить действия при обновлении базы. Метод onUpgrade () вызывается при установке обновлений программы с измененной структурой таблиц в базе. В методе обратного вызова onUpgrade () можно, например, реализовать запрос в базу данных на уничтожение таблицы (DROP TABLE), после чего вновь вызвать метод onCreate () для создания версии таблицы с обновленной структурой (листинг 4).

Листинг 4. Пример метода onUpgrade ()

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
{
    db.execSQL("DROP TABLE IF EXISTS " + DBContract.DBEntry.TABLE_NAME);
    onCreate(db);
}
```

После написания класса, расширяющего SQLiteOpenHelper, в классе активити необходимо создать экземпляр этого класса и вызвать метод getWritableDatabase() или getReadableDatabase() тогда, когда потребуется работа с базой. Например, можно создать кнопку, по нажатию на которую база будет создаваться и наполняться начальными значениями (листинг 5).

Листинг 5. Код MainActivity.java

```
package com.example.sqliteapp;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.database.sqlite.SQLiteDatabase;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        final Button buttonCreate = (Button)findViewById(R.id.button);

        buttonCreate.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View arg0) {
                SQLiteDatabase sql_db = new
```

```
DatabaseHelper(getApplicationContext()).getWritableDatabase();
                if (sql_db != null) {
                    Toast.makeText(getApplicationContext(),
                            "БД контактов создана!",
Toast.LENGTH_LONG).show();
                }
                else {
                    Toast.makeText(getApplicationContext(),
                            "Ошибка при создании БД!",
Toast.LENGTH_LONG).show();
                }
            }
        });
    }
}
```

Чтобы убедиться, что база данных была успешно создана, можно открыть окно Device File Explorer (при запущенном эмуляторе виртуального устройства или при подключенном реальном устройстве). Вкладка Device File Explorer располагается в нижнем правом углу Android Studio. В каталоге data/data/com.example.sqliteapp/databases будет находиться файл contacts.db (рисунок 1).

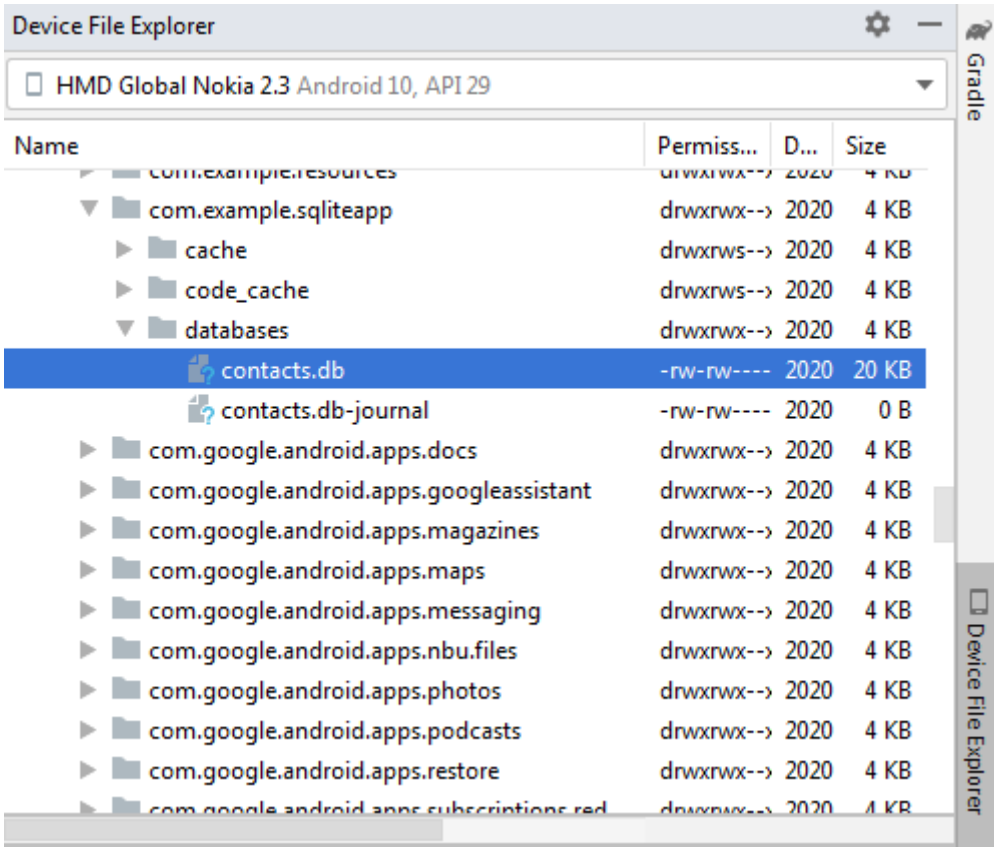

Рисунок 1. Вкладка Device File Explorer.

Чтение данных из базы

Для чтения данных из базы используется метод query (), в который передаются критерии выбора и желаемые столбцы. Результаты запроса возвращаются в объекте Cursor ─ курсоре. Метод query () принимает следующие параметры:

1. Имя таблицы, к которой выполняется запрос.
2. Массив желаемых столбцов (если нужны все столбцы, передаётся null).
3. Столбцы для запроса WHERE (условия отбора).
4. Значения для запроса WHERE (условия отбора).
5. Параметры для оператора GROUP BY, т.е. по какому столбцу будут группироваться строки (если null, группировка выполняться не будет).
6. Параметры для оператора HAVING (если null, не фильтровать).
7. Параметры для оператора ORDER BY, т.е. порядок сортировки.

Третий и четвертый параметры объединяются для создания условия отбора WHERE. Поскольку параметры предоставляются отдельно от запроса выборки, они экранируются перед объединением. Это делает запросы невосприимчивыми к SQL-инъекции.

Когда запрос query () выполнен, чтобы просмотреть строку в курсоре, используется один из методов перемещения курсора, который всегда необходимо вызывать перед началом чтения значений. Поскольку курсор начинается с позиции -1, вызов moveToNext () помещает «позицию чтения» в первую запись в результатах и возвращает, прошел ли курсор уже за последнюю запись в наборе результатов. Для каждой строки вы можете прочитать значение столбца, вызвав один из методов получения курсора, например, getString () или getLong (). Для каждого из методов get вы должны передать позицию индекса нужного столбца, которую вы можете получить, вызвав метод getColumnIndex () или getColumnIndexOrThrow (), который

выбрасывает исключение, если столбец с таким именем не существует. По завершении перебора результатов вызовите close () для курсора, чтобы освободить его ресурсы.

В листинге 6 показано, как получить все значения столбцов с именем и телефоном из базы контактов, хранящиеся в курсоре, и добавить их в текстовое поле.

Листинг 6. Извлечение данных

```
String[] projection = {
      DBContract.DBEntry.COLUMN_NAME_NAME,
      DBContract.DBEntry.COLUMN_NAME_PHONE
 };

SQLiteDatabase db = new
DatabaseHelper(getApplicationContext()).getReadableDatabase();
Cursor cursor = db.query(
         DBContract.DBEntry.TABLE_NAME,
         projection,
         null,
         null,
         null,
         null,
         null
 );

 int name = cursor.getColumnIndex(DBContract.DBEntry.COLUMN_NAME_NAME);
 int phone = cursor.getColumnIndex(DBContract.DBEntry.COLUMN_NAME_PHONE);
 TextView text = (TextView) findViewById(R.id.text);
 while (cursor.moveToNext()) {
     String Name = cursor.getString(name);
     String Phone = cursor.getString(phone);
     text.append("\n" +  Name + " \n " +  Phone + " \n ");
 }
cursor.close();

DatabaseHelper.close();
```

Удаление и обновление данных в базе

Чтобы удалить строки из таблицы, необходимо в методе delete () указать критерии, по которым будут отобраны строки для удаления. Механизм отбора работает так же, как условия отбора для метода query (). Метод delete () невосприимчив к SQL-инъекции.

В листинге 7 приведён пример удаления данных из базы контактов. Имя человека, данные о котором необходимо удалить из базы, вводится в элемент Edit Text — поле ввода для данных ET_name. В переменную selection вводятся столбцы для условия удаления, а в selectionArgs — значения для удаления. Если необходимо удалить все данные из таблицы в качестве этих параметров передаётся null, например, db.delete(DBContract.DBEntry.TABLE_NAME, null, null).

Листинг 7. Удаление данных.

```
String input_name = ET_name.getText().toString();
SQLiteDatabase db = new
DatabaseHelper(getApplicationContext()).getWritableDatabase();

String selection = DBContract.DBEntry.COLUMN_NAME_NAME + "=?";
String[] selectionArgs = {input_name};
db.delete(DBContract.DBEntry.TABLE_NAME, selection, selectionArgs);
```

Если требуется изменить подмножество значений базы данных, следует использовать метод update (). Обновление таблицы объединяет синтаксис ContentValues функции insert () с синтаксисом WHERE функции delete (). Метод update () возвращает количество затронутых строк в базе данных.

Задание

Разработаем приложение, позволяющее создавать базу данных, заполнять начальными значениями, считывать данные из базы, сохранять новые записи, удалять записи и очищать базу.

В базе будет одна таблица — список контактов, содержащий идентификатор, имя и телефон человека.

Компоновка будет содержать четыре кнопки, поля для ввода имени и телефона, а также элемент ScrollView — полосу прокрутки, содержащую TextView. Код компоновки представлен в листинге 8.

Листинг 8. Компоновка приложения.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Показать БД"
        android:id="@+id/buttonShow"/>
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Добавить запись"
        android:id="@+id/buttonAdd"/>
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Удалить запись"
        android:id="@+id/buttonDel"/>
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Очистить БД"
        android:id="@+id/buttonClear"/>

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="24sp"
        android:id="@+id/ET_name"/>
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="24sp"
        android:id="@+id/ET_phone"/>
    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent">
        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="vertical" >
            <TextView
                android:id="@+id/text"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:paddingLeft="20dp"
                android:paddingTop="10dp"
                android:textSize="20sp" />
        </LinearLayout>
    </ScrollView>
</LinearLayout>
```

Файл контактного класса приведён в листинге 9.

Листинг 9. Контрактный класс.

```
package com.example.sqliteapp;
import android.provider.BaseColumns;

public final class DBContract {
    private DBContract() {}
    public static class DBEntry implements BaseColumns {
        public static final String TABLE_NAME = "people";
        public static final String COLUMN_NAME_NAME = "name";
        public static final String COLUMN_NAME_PHONE = "phone";
    }
}
```

После создания контактного класса необходимо создать класс, наследующий от SQLiteOpenHelper. Его код приведён в листинге 10.

Листинг 10. Класс, наследующий от SQLiteOpenHelper.

```
package com.example.sqliteapp;
import android.content.ContentValues;
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.provider.BaseColumns;

class DatabaseHelper extends SQLiteOpenHelper implements BaseColumns {
    public static final String DB_CONTACTS = "contacts.db";
    public static final int DATABASE_VERSION = 1;
    public DatabaseHelper(Context context) {
        super(context, DB_CONTACTS, null, DATABASE_VERSION);
    }
    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL("CREATE TABLE " + DBContract.DBEntry.TABLE_NAME
                + " (_id INTEGER PRIMARY KEY AUTOINCREMENT, "
                + DBContract.DBEntry.COLUMN_NAME_NAME + " TEXT, " +
DBContract.DBEntry.COLUMN_NAME_PHONE + " TEXT);");

        ContentValues values = new ContentValues();
        values.put(DBContract.DBEntry.COLUMN_NAME_NAME, "Иван Иванов");
        values.put(DBContract.DBEntry.COLUMN_NAME_PHONE, "8-999-111-11-
11");
        db.insert(DBContract.DBEntry.TABLE_NAME,
DBContract.DBEntry.COLUMN_NAME_NAME, values);

        values.put(DBContract.DBEntry.COLUMN_NAME_NAME, "Пётр Петров");
        values.put(DBContract.DBEntry.COLUMN_NAME_PHONE, "8-999-222-22-
22");
        db.insert(DBContract.DBEntry.TABLE_NAME,
```

```
DBContract.DBEntry.COLUMN_NAME_NAME, values);

        values.put(DBContract.DBEntry.COLUMN_NAME_NAME, "Иван Петров");
        values.put(DBContract.DBEntry.COLUMN_NAME_PHONE, "8-999-333-33-
33");
        db.insert(DBContract.DBEntry.TABLE_NAME,
DBContract.DBEntry.COLUMN_NAME_NAME, values);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion) {
        db.execSQL("DROP TABLE IF EXISTS " +
DBContract.DBEntry.TABLE_NAME);
        onCreate(db);
    }
}
```

И, наконец, необходимо дополнить код MainActivity.java (листинг 11).

Листинг 11. MainActivity.java

```
package com.example.sqliteapp;
import androidx.appcompat.app.AppCompatActivity;
import android.content.ContentValues;
import android.database.Cursor;
import android.os.Bundle;
import android.database.sqlite.SQLiteDatabase;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity implements
View.OnClickListener{
    Button buttonDel, buttonClear, buttonShow, buttonAdd;
    EditText ET_name, ET_phone;
    TextView text;
    DatabaseHelper DatabaseHelper;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        buttonAdd = (Button)findViewById(R.id.buttonAdd);
        buttonAdd.setOnClickListener(this);
        buttonShow = (Button)findViewById(R.id.buttonShow);
        buttonShow.setOnClickListener(this);
        buttonDel = (Button)findViewById(R.id.buttonDel);
        buttonDel.setOnClickListener(this);
        buttonClear = (Button)findViewById(R.id.buttonClear);
        buttonClear.setOnClickListener(this);
        ET_name = (EditText)findViewById(R.id.ET_name);
        ET_phone = (EditText)findViewById(R.id.ET_phone);
```

```
        text = (TextView)findViewById(R.id.text);

        DatabaseHelper = new DatabaseHelper(this);
    }

    @Override
    public void onClick(View v) {
        String input_name = ET_name.getText().toString();
        String input_phone = ET_phone.getText().toString();

        SQLiteDatabase db = DatabaseHelper.getWritableDatabase();
        ContentValues values = new ContentValues();

        switch (v.getId()) {
            case R.id.buttonShow:
                text.setText("");
                String[] projection = {
                        DBContract.DBEntry.COLUMN_NAME_NAME,
                        DBContract.DBEntry.COLUMN_NAME_PHONE
                };
                Cursor cursor = db.query(
                        DBContract.DBEntry.TABLE_NAME,
                        projection,
                        null,
                        null,
                        null,
                        null,
                        null
                );
                int index_name =
cursor.getColumnIndex(DBContract.DBEntry.COLUMN_NAME_NAME);
                int index_phone =
cursor.getColumnIndex(DBContract.DBEntry.COLUMN_NAME_PHONE);
                while (cursor.moveToNext()) {
                    String value_name = cursor.getString(index_name);
                    String value_phone = cursor.getString(index_phone);
                    text.append("\n" +  value_name + " \n " +
value_phone + " \n ");
                }
                cursor.close();
                break;

            case R.id.buttonAdd:
                values.put(DBContract.DBEntry.COLUMN_NAME_NAME,
input_name);
                values.put(DBContract.DBEntry.COLUMN_NAME_PHONE,
input_phone);
                db.insert(DBContract.DBEntry.TABLE_NAME, null, values);
                buttonShow.callOnClick();
                break;

            case R.id.buttonDel:
                String selection = DBContract.DBEntry.COLUMN_NAME_NAME +
"=?";
                String[] selectionArgs = {input_name};
```

```
                db.delete(DBContract.DBEntry.TABLE_NAME, selection,
selectionArgs);
                buttonShow.callOnClick();
                break;

            case R.id.buttonClear:
                db.delete(DBContract.DBEntry.TABLE_NAME, null, null);
                buttonShow.callOnClick();
                break;
        }
        DatabaseHelper.close();
    }
}
```

На рисунке 2 приведены скриншоты работы приложения.

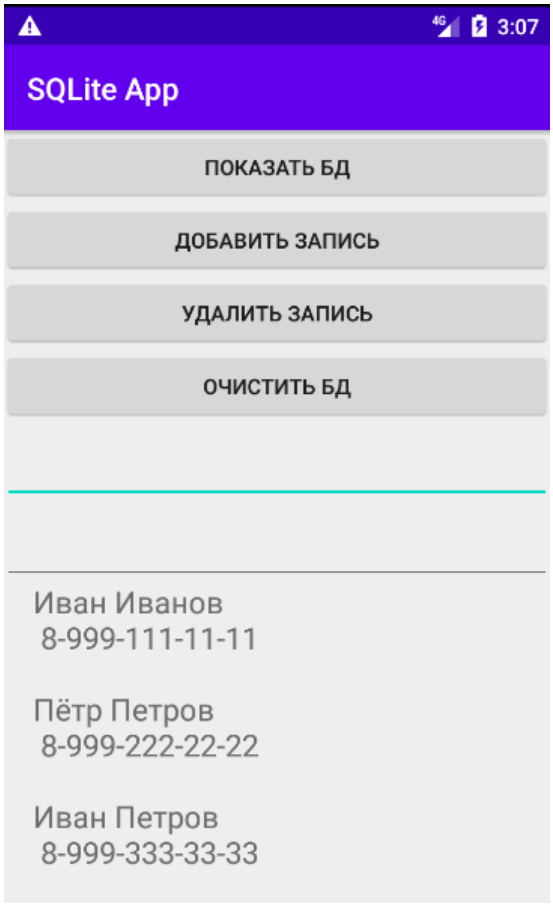

Рисунок 2. Результаты работы приложения.

Самостоятельно дополните приложение, добавьте новые столбцы в таблицу контактов, доработайте интерфейс приложения.

Контрольные вопросы:

1. Для чего используется класс SQLiteOpenHelper?
2. Для чего используется класс SQLiteDatabase?
3. Для чего используется класс ContentValues?
4. Для чего используется класс Cursor?
5. Каково назначение метода getWritableDatabase?
6. Каково назначение метода getReadableDatabase?
7. Каково назначение метода execSQL?
8. Каково назначение метода put при работе с БД?
9. Каково назначение метода query при работе с БД?
10. Каково назначение метода insert при работе с БД?
11. Каково назначение метода delete при работе с БД?
12. Каково назначение метода update при работе с БД?
13. Когда происходит вызов метода onCreate для объекта класса, наследующего от SQLiteOpenHelper?
14. Когда происходит вызов метода onUpgrade для объекта класса, наследующего от SQLiteOpenHelper?