Цель лабораторной работы:

Разработка Android приложения, демонстрирующего возможности работы с базой данных SQLite.

Задачи лабораторной работы:

- создать приложение;
- настроить интерфейс приложения;
- реализовать логику приложения.

1 Введение

Для достижения цели, поставленной в лабораторной работе, сформулируем требования к разрабатываемому приложению. Приложение демонстрирует возможности работы с базой данных, предполагает реализацию следующих действий:

- добавление записей в базу данных;
- считывание строк и вывод на экран;
- удаление базы данных.

2 Создание приложения

Создадим новое Android приложение:

Project Name: Lab7_1-SQLite;

Package Name: com.example.lab7_1_sqlite;

Activity Files: SQLiteActivity.java, activity_sqlite.xml.

3 Настройка интерфейса

Разрабатываемое приложение является учебно-тренировочным, предполагает демонстрацию возможностей работы с базой данных: создание, добавление записей, просмотр записей, удаление базы данных. Поэтому интерфейс будет максимально простым: нам понадобятся два поля для ввода данных (в таблице базы данных два столбца), поле для вывода записей базы данных и три кнопки, по одной на каждое действие: добавление записей, вывод записей, удаление базы.

В нашем случае активность приложения имеет вид, показанный на рис. 17.1. Разумеется, не обязательно в точности воспроизводить предложенный внешний вид приложения, а на самом деле даже имеет смысл настроить интерфейс по своему усмотрению, чтобы потренировать навыки работы с компонентами и настройками пользовательского интерфейса.

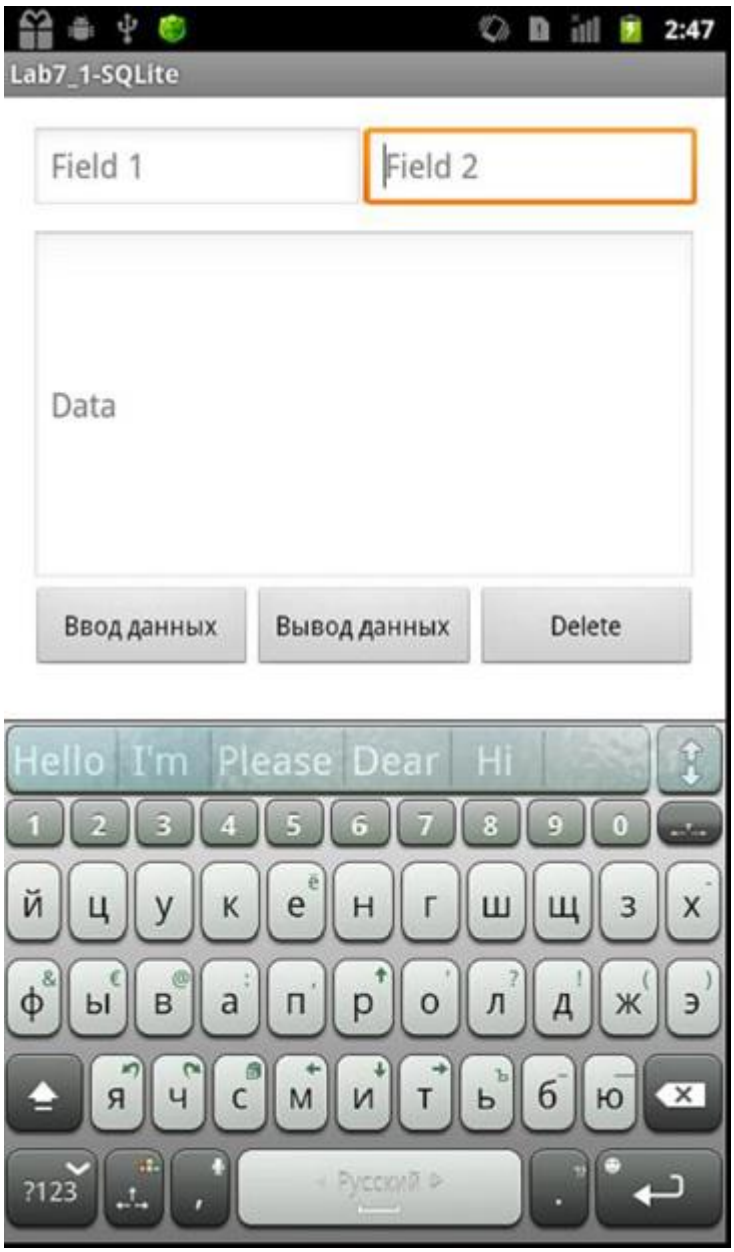

Рис. 1. Внешний вид приложения для демонстрации основных возможностей базы данных SQLIte в системе Android

17.4 Реализация логики

Для создания и обновления базы данных, Android предоставляет класс SQLiteOpenHelper. Для работы с этим классом в приложении необходимо создать класс-наследник, в котором обязательно реализовать методы: onCreate() и onUpgarde().

Создадим новый класс, назовем его, например, MyOpenHelper:

```
public class MyOpenHelper extends SQLiteOpenHelper{...}
```

Реализуем метод onCreate():

```
public void onCreate(SQLiteDatabase DB) {
```

```
  String query="create table " + TABLE_NAME + " (_id integer primary key
autoincrement, " + field_name_1 + "
TEXT, " + field_name_2 + " TEXT)";
  DB.execSQL(query);
}
```

Параметром метода onCreate() является объект класса SQLiteDatabase, позволяющего работать с базой данных напрямую.

В строке query формируется запрос на создание таблицы, где

TABLE_NAME	- константа, содержащая имя таблицы,
FIELD_NAME_1	- константа, содержащая имя первого столбца,
FIELD_NAME_2	- константа, содержащая имя второго столбца.

Все эти константы объявлены в классе MyOpenHelper:

```
public String TABLE_NAME="first_table";
public String FIELD_NAME_1="first_field";
public String FIELD_NAME_2="second_field";
```

Метод execSQL() класса SQLiteDatabase запускает запрос на выполнение.

Еще необходимо реализовать метод onUpgrade(), в нашем случае он ничего существенного не делает, но необходимо прописать реализацию этого метода даже, если это будет пустая реализация:

```
public void onUpgrade(SQLiteDatabase DB, int oldVersion,
        int newVersion) {
  Log.d("myLogs","| Upgrade |"+DB.toString());
}
```

Система потребует создать конструктор класса MyOpenHelper, т. к. для его предка конструктор без параметров не определен, поэтому добавим в код класса следующие строчки:

```
MyOpenHelper(Context ct, String nm,
        SQLiteDatabase.CursorFactory cf, int vs){
  super(ct, nm, cf, vs);
}
```

Полученный конструктор не делает ничего особенного, просто вызывает конструктор предка, т. е. класса SQLiteOpenHelper.

Мы описали создание вспомогательного класса, необходимого нам для создания и открытия базы данных, полный код этого класса представлен в листинге 17.1.

Пришло время описать реализацию заявленных функций приложения. Далее будем работать с классом активности, описание которого находится в файле **SQLiteActivity.java**.

Выполним необходимую подготовку, зададим переменные, как поля класса активности:

```
EditText field1, field2, result;
MyOpenHelper myHelper = null;
SQLiteDatabase DB;
```

Вторая строка определяет объект класса MyOpenHelper, а третья - объект класса SQLiteDatabase, далее в программе все взаимодействия с базой данных после ее создания будут выполняться через этот объект.

Далее в методе onCreate() активности создадим экземпляр класса MyOpenHelper, для создания, открытия и возможно управления базой данных, имя которой myDB:

```
myHelper=new MyOpenHelper(this, "myDB", null, 1);
```

Для работы приложения необходимо задать действия, которые будут выполняться при нажатии на кнопки, для этого настроим свойство On Click кнопок:

- для кнопки **Ввод данных** свойству On Click присвоим значение insertIntoDatabase, при нажатии на кнопку будет вызываться метод с этим именем, реализованный в классе активности;
- для кнопки **Вывод данных** свойству On Click присвоим значение readDatabase, при нажатии на кнопку будет вызываться метод с этим именем, реализованный в классе активности;
- для кнопки **Delete** свойству On Click присвоим значение deleteDatabase, при нажатии на кнопку будет вызываться метод с этим именем, реализованный в классе активности.

Рассмотрим реализацию метода insertIntoDatabase().

В этом методе после проверки того, что в поля введены значения, получаем экземпляр базы данных с разрешением на запись:

```
DB = myHelper.getWritableDatabase();
```

Далее формируем запись, добавляемую в таблицу:

```
ContentValues CV = new ContentValues()
CV.put(myHelper.FIELD_NAME_1,field1.getText().toString());
CV.put(myHelper.FIELD_NAME_2,field2.getText().toString());
```

После этого добавляем запись в таблицу и закрываем экземпляр базы данных:

```
DB.insert(myHelper.TABLE_NAME,null,CV);
DB.close();
```

Полный код рассмотренного метода можно найти в листинге 17.2.

Рассмотрим реализацию метода readDatabase().

В этом методе сначала получаем экземпляр базы данных с разрешением на чтение:

```
DB = myHelper.getReadableDatabase();
```

Далее формируем и выполняем запрос к базе данных на получение всех значений из базы, результат запроса помещается в объект класса Cursor:

```
String columns[]={"_id",myHelper.FIELD_NAME_1,
        myHelper.FIELD_NAME_2};
Cursor cursor=DB.query(myHelper.TABLE_NAME, columns, null,
        null, null, null, "_id");
```

После этого выведем содержимое класса Cursor в поле на форме, предназначенное для вывода информации:

```
if(cursor!=null){
  cursor.moveToFirst();
  if (cursor.moveToFirst()) {
    do {
      result.setText(result.getText().toString()+"\n" +cursor.getString(0)+")
"+cursor.getString(1)+",
      "+cursor.getString(2));
    } while (cursor.moveToNext());
  }
}
```

В этом кусочке кода основное внимание можно обратить на работу с классом курсор, очень похоже на работу со списками. Начиная с начала списка получаем значения элементов и передвигаемся на следующий элемент, пока не достигнем конца.

После всех манипуляций с базой данных необходимо ее закрыть:

```
DB.close();
```

Полный код рассмотренного метода можно найти в листинге 17.2.

Рассмотрим реализацию метода deleteDatabase.

В этом методе сначала получаем экземпляр базы данных с разрешением на запись:

```
DB = myHelper.getWritableDatabase();
```

Удалим таблицу, с помощью метода delete() класса SQLiteDatabase:

```
DB.delete(myHelper.TABLE_NAME, null, null);
```

После всего закроем базу:

```
DB.close();
```

Полный код рассмотренного метода можно найти в листинге 17.2.

17.5 Заключение

В работе на примере простого приложения рассмотрели выполнение основных операций с базами данных: создание, добавление записей, просмотр всех записей таблицы, удаление. Для выполнения выборок из таблицы, необходимо познакомиться с основами построения запросов в SQLite и сформировать запрос для метода query() класса SQLiteDatabase.

```
package com.example.lab7_1_sqlite;

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

public class MyOpenHelper extends SQLiteOpenHelper {

  public String TABLE_NAME="first_table";
  public String FIELD_NAME_1="first_field";
  public String FIELD_NAME_2="second_field";

  MyOpenHelper(Context ct, String nm, SQLiteDatabase.CursorFactory cf, int
vs){
    super(ct, nm, cf, vs);
  }

  @Override
  public void onUpgrade(SQLiteDatabase DB, int oldVersion, int newVersion) {
    Log.d("myLogs","| Upgrade |"+DB.toString());
  }

  @Override
  public void onCreate(SQLiteDatabase DB) {
    Log.d("myLogs","| Create |"+DB.toString());
    String query="create table " + TABLE_NAME + " ( _id integer primary key
      autoincrement, " + FIELD_NAME_1 + " TEXT, " + FIELD_NAME_2 + " TEXT)";
    DB.execSQL(query);
  }
}
```

Листинг 17.1. Класс MyOpenHelper

```
package com.example.lab7_1_sqlite;

import android.os.Bundle;
import android.app.Activity;
import android.content.ContentValues;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.util.Log;
import android.view.View;
import android.widget.EditText;

public class SQLiteActivity extends Activity {

  MyOpenHelper myHelper = null;
  EditText field1, field2, result;
  SQLiteDatabase DB;

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

```
    myHelper=new MyOpenHelper(this, "myDB", null, 1);

    field1=(EditText)findViewById(R.id.field1);
    field2=(EditText)findViewById(R.id.field2);
    result=(EditText)findViewById(R.id.dbResult);
  }

  public void insertIntoDatabase(View v){

    if(!field1.getText().toString().equals("") &&
        !field2.getText().toString().equals("")){
      Log.d("myLogs","Insert INTO DB ("+field1.getText().toString()+ "," +
field2.getText().toString()+")");

      DB = myHelper.getWritableDatabase();
      String query="create table if not exist " + myHelper.TABLE_NAME +
      " (_id integer primary key autoincrement, " + myHelper.FIELD_NAME_1 + "
TEXT, " + myHelper.FIELD_NAME_2 + " TEXT)";

      ContentValues CV = new ContentValues();
      CV.put(myHelper.FIELD_NAME_1,field1.getText().toString());
      CV.put(myHelper.FIELD_NAME_2,field2.getText().toString());
      DB.insert(myHelper.TABLE_NAME,null,CV);
      DB.close();
      field1.setText("");
      field2.setText("");
    }
  }

  public void readDatabase(View v){

    result.setText("");

    Log.d("myLogs","READ FROM DB");
    DB = myHelper.getReadableDatabase();

    String columns[]={"_id",myHelper.FIELD_NAME_1, myHelper.FIELD_NAME_2};
    Cursor cursor=DB.query(myHelper.TABLE_NAME, columns, null, null, null,
    null, "_id");
    if(cursor!=null){
      cursor.moveToFirst();
      if (cursor.moveToFirst()) {
        do {
          result.setText(result.getText().toString()+ "\n" +
                cursor.getString(0) + ") " + cursor.getString(1) + "," +
cursor.getString(2));
        } while (cursor.moveToNext());
      }
    }

    DB.close();
  }

  public void deleteDatabase(View v){
    Log.d("myLogs","Delete Database");
    DB = myHelper.getWritableDatabase();

    DB.delete(myHelper.TABLE_NAME, null, null);
    DB.close();
  }
}
```

Листинг 17.2. Класс SQLiteActivity