**Министерство образования и науки Российской Федерации**

**Федеральное государственное автономное образовательное**

**учреждение высшего профессионального образования  
 «Севастопольский государственный университет»**

# 

**Методические указания**

**к выполнению лабораторной работы №5**

**по дисциплине**

**«Мобильные информационные технологии»**

для студентов всех форм обучения направления подготовки 09.03.02 «Информационные системы и технологии»

**Севастополь**

**2018**

Методические указания к выполнению лабораторной работы № 4 по дисциплине «Мобильные информационные технологии» для студентов всех форм обучения направления подготовки 09.03.02 «Информационные системы и технологии» /Сост. В.Е. Шишкевич ‒ Севастополь: СевГУ, 2018. ‒ 16 с.

**Лабораторная работа №5**

**Исследование взаимодействия android-приложения с базой данных**

**1. Цель работы**

Освоить взаимодействие Android-приложения с SQLite.

**2. Краткие теоретические сведения**

**SQLite** – это встраиваемая кроссплатформенная реляционная база данных, которая поддерживает достаточно полный набор команд SQL, с открытым исходным кодом.

Позиция функциональности SQLite где-то между MySQL и PostgreSQL. Однако, на практике, SQLite нередко оказывается в 2-3 раза (и даже больше) быстрее. Такое возможно благодаря высокоупорядоченной внутренней архитектуре и устранению необходимости в соединениях типа «сервер-клиент» и «клиент-сервер». Используя высоко эффективную инфраструктуру, SQLite может работать в крошечном объёме выделяемой для неё памяти, гораздо меньшем, чем в любых других системах БД. Это делает SQLite очень удобным инструментом с возможностью использования практически в любых задачах, возлагаемых на базу данных.

Движок БД представляет библиотеку, с которой программа компонуется и SQLite становится составной частью программы. Вся БД хранится в единственном стандартном файле на машине, на которой исполняется программа.

Процедурный интерфейс к SQLite почти такой же, как у MySQL и других расширений БД. По большей части переход к SQLite потребует только изменять префикс функций, например, с mysql на sqlite.

**Преимущества SQLite:**

– очень популярна;

– очень надежна;

– существует консольная утилита для работы с базами;

– поддерживает динамическое типизирование данных;

– открытые исходные коды.

**Недостатки SQLite:**

– нет хранимых процедур.

– нет встроенной поддержки UNICODE

– не подходит для приложений, которые часто обращаются в БД.

**Возможные типы полей:**

1) INTEGER;

2) REAL;

3) TEXT;

4) BLOB.

**Создание, наполнение и модификация БД в Android Studio**

В приложении при подключении к БД указываются имя версия БД. При этом могут возникнуть следующие ситуации:

1) БД не существует. Это может быть, например, в случае первичной установки программы. В этом случае приложение должно само создать БД и все таблицы в ней.

2) БД существует, но ее версия устарела. Это может быть в случае обновления программы. Например новой версии программы нужны дополнительные поля в старых таблицах или новые таблицы. В этом случае приложение должно обновить существующие таблицы и создать новые, если это необходимо.

3) БД существует и ее версия актуальна. В этом случае приложение успешно подключается к БД и работает.

Для обработки описанных выше ситуаций необходимо создать класс, являющийся наследником для SQLiteOpenHelper. Назовем его DBHelper. Этот класс предоставит методы для создания или обновления БД в случаях ее отсутствия или устаревания.

**onCreate** – метод, который будет вызван, если БД, к которой производится подключение, не существует.

**onUpgrade** –метод, который будет вызван в случае, если происходит попытка подключения к БД более новой версии, чем существующая.

Чтобы освоить приемы работы с БД, создадим приложение – справочник контактов, которое будет хранить имя и email. Вводить данные будем на экране приложения, а для отображения информации будем использовать логи. Обычно для этого используется List.

Создадим проект:

**Projectname**: P0341\_SimpleSQLite

**BuildTarget**: Android 2.3.3

**Applicationname**: SimpleSQLite

**Packagename**: ru.startandroid.develop.p0341simplesqlite

**CreateActivity**: MainActivity

Нарисуем экран для ввода записей и очистки таблицы. Разместим полей для ввода *ID*, *name* и *email* и кнопки для:

– добавления записи;

– вывода существующих записей;

– обновления записей;

– удаления записей;

– очистки таблицы.

Для этого в файле main.xml напишем следующий код:

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

<LinearLayout

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

    android:layout\_width="fill\_parent"

    android:layout\_height="fill\_parent"

    android:orientation="vertical">

    <LinearLayout

        android:id="@+id/linearLayout4"

        android:layout\_width="match\_parent"

        android:layout\_height="wrap\_content"

        android:layout\_marginTop="5dp">

        <TextView

            android:layout\_width="wrap\_content"

            android:layout\_height="wrap\_content"

            android:text="ID"

            android:layout\_marginLeft="5dp"

            android:layout\_marginRight="25dp">

        </TextView>

        <EditText

            android:id="@+id/etID"

            android:layout\_width="70dp"

            android:layout\_height="wrap\_content"

            android:layout\_marginTop="2dp">

        </EditText>

        <Button

            android:id="@+id/btnUpd"

            android:layout\_width="wrap\_content"

            android:layout\_height="wrap\_content"

            android:text="Update">

        </Button>

        <Button

            android:id="@+id/btnDel"

            android:layout\_width="wrap\_content"

            android:layout\_height="wrap\_content"

            android:text="Delete">

        </Button>

    </LinearLayout>

    <LinearLayout

        android:id="@+id/linearLayout1"

        android:layout\_width="match\_parent"

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

        <TextView

            android:layout\_width="wrap\_content"

            android:layout\_height="wrap\_content"

            android:text="Name"

            android:layout\_marginLeft="5dp"

            android:layout\_marginRight="5dp">

        </TextView>

        <EditText

            android:id="@+id/etName"

            android:layout\_width="wrap\_content"

            android:layout\_height="wrap\_content"

            android:layout\_weight="1">

            <requestFocus>

            </requestFocus>

        </EditText>

    </LinearLayout>

    <LinearLayout

        android:id="@+id/linearLayout3"

        android:layout\_width="match\_parent"

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

        <TextView

            android:id="@+id/textView2"

            android:layout\_width="wrap\_content"

            android:layout\_height="wrap\_content"

            android:text="Email"

            android:layout\_marginLeft="5dp"

            android:layout\_marginRight="5dp">

        </TextView>

        <EditText

            android:id="@+id/etEmail"

            android:layout\_width="wrap\_content"

            android:layout\_height="wrap\_content"

            android:layout\_weight="1">

        </EditText>

    </LinearLayout>

    <LinearLayout

        android:id="@+id/linearLayout2"

        android:layout\_width="match\_parent"

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

        <Button

            android:id="@+id/btnAdd"

            android:layout\_width="wrap\_content"

            android:layout\_height="wrap\_content"

            android:text="Add">

        </Button>

        <Button

            android:id="@+id/btnRead"

            android:layout\_width="wrap\_content"

            android:layout\_height="wrap\_content"

            android:text="Read">

        </Button>

        <Button

            android:id="@+id/btnClear"

            android:layout\_width="wrap\_content"

            android:layout\_height="wrap\_content"

            android:text="Clear">

        </Button>

    </LinearLayout>

</LinearLayout>

Ниже приведен скрипт MainActivity.java, в котором реализовано взаимодействие с БД, а после него приведено описание всех функций.

package ru.startandroid.develop.p0341simplesqlite;

import ru.startandroid.develop.p0341simpelsqlite.R;

import android.app.Activity;

import android.content.ContentValues;

import android.content.Context;

import android.database.Cursor;

import android.database.sqlite.SQLiteDatabase;

import android.database.sqlite.SQLiteOpenHelper;

import android.os.Bundle;

import android.util.Log;

import android.view.View;

import android.view.View.OnClickListener;

import android.widget.Button;

import android.widget.EditText;

public class MainActivity extends Activity implements OnClickListener {

  final String LOG\_TAG = "myLogs";

  Button btnAdd, btnRead, btnClear, btnUpd, btnDel;

  EditText etName, etEmail, etID;

  DBHelper dbHelper;

  /\*\* Called when the activity is first created. \*/

  public void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);

    setContentView(R.layout.main);

    btnAdd = (Button) findViewById(R.id.btnAdd);

    btnAdd.setOnClickListener(this);

    btnRead = (Button) findViewById(R.id.btnRead);

    btnRead.setOnClickListener(this);

    btnClear = (Button) findViewById(R.id.btnClear);

    btnClear.setOnClickListener(this);

    btnUpd = (Button) findViewById(R.id.btnUpd);

    btnUpd.setOnClickListener(this);

    btnDel = (Button) findViewById(R.id.btnDel);

    btnDel.setOnClickListener(this);

    etName = (EditText) findViewById(R.id.etName);

    etEmail = (EditText) findViewById(R.id.etEmail);

    etID = (EditText) findViewById(R.id.etID);

    // создаем объект для создания и управления версиями БД

    dbHelper = new DBHelper(this);

  }

  public void onClick(View v) {

    // создаем объект для данных

    ContentValues cv = new ContentValues();

    // получаем данные из полей ввода

    String name = etName.getText().toString();

    String email = etEmail.getText().toString();

    String id = etID.getText().toString();

    // подключаемся к БД

    SQLiteDatabase db = dbHelper.getWritableDatabase();

    switch (v.getId()) {

    case R.id.btnAdd:

      Log.d(LOG\_TAG, "--- Insert in mytable: ---");

      // подготовим данные для вставки в виде пар: наименование столбца -

      // значение

      cv.put("name", name);

      cv.put("email", email);

      // вставляем запись и получаем ее ID

      long rowID = db.insert("mytable", null, cv);

      Log.d(LOG\_TAG, "row inserted, ID = " + rowID);

      break;

    case R.id.btnRead:

      Log.d(LOG\_TAG, "--- Rows in mytable: ---");

      // делаем запрос всех данных из таблицы mytable, получаем Cursor

      Cursor c = db.query("mytable", null, null, null, null, null, null);

      // ставим позицию курсора на первую строку выборки

      // если в выборке нет строк, вернется false

      if (c.moveToFirst()) {

        // определяем номера столбцов по имени в выборке

        int idColIndex = c.getColumnIndex("id");

        int nameColIndex = c.getColumnIndex("name");

        int emailColIndex = c.getColumnIndex("email");

        do {

          // получаем значения по номерам столбцов и пишем все в лог

          Log.d(LOG\_TAG,

              "ID = " + c.getInt(idColIndex) + ", name = "

                  + c.getString(nameColIndex) + ", email = "

                  + c.getString(emailColIndex));

          // переход на следующую строку

          // а если следующей нет (текущая - последняя), то false -

          // выходим из цикла

        } while (c.moveToNext());

      } else

        Log.d(LOG\_TAG, "0 rows");

      c.close();

      break;

    case R.id.btnClear:

      Log.d(LOG\_TAG, "--- Clear mytable: ---");

      // удаляем все записи

      int clearCount = db.delete("mytable", null, null);

      Log.d(LOG\_TAG, "deleted rows count = " + clearCount);

      break;

    case R.id.btnUpd:

      if (id.equalsIgnoreCase("")) {

        break;

      }

      Log.d(LOG\_TAG, "--- Update mytable: ---");

      // подготовим значения для обновления

      cv.put("name", name);

      cv.put("email", email);

      // обновляем по id

      int updCount = db.update("mytable", cv, "id = ?",

          new String[] { id });

      Log.d(LOG\_TAG, "updated rows count = " + updCount);

      break;

    case R.id.btnDel:

      if (id.equalsIgnoreCase("")) {

        break;

      }

      Log.d(LOG\_TAG, "--- Delete from mytable: ---");

      // удаляем по id

      int delCount = db.delete("mytable", "id = " + id, null);

      Log.d(LOG\_TAG, "deleted rows count = " + delCount);

      break;

    }

    // закрываем подключение к БД

    dbHelper.close();

  }

  class DBHelper extends SQLiteOpenHelper {

    public DBHelper(Context context) {

      // конструктор суперкласса

      super(context, "myDB", null, 1);

    }

    public void onCreate(SQLiteDatabase db) {

      Log.d(LOG\_TAG, "--- onCreate database ---");

      // создаем таблицу с полями

      db.execSQL("create table mytable ("

          + "id integer primary key autoincrement,"

          + "name text,"

          + "email text" + ");");

    }

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

    }

  }

}

В методе Activity – **onCreate** мы определяем объекты, присваиваем обработчики и создаем объект dbHelper класса DBHelper для управления БД. Сам класс будет описан ниже.

Далее смотрим метод Activity – **onClick**, в котором мы обрабатываем нажатия на кнопки.

Класс **ContentValues** используется для указания полей таблицы и значений, которые мы в эти поля будем вставлять. Мы создаем объект **cv**, и позже его используем. Далее мы записываем в переменные значения из полей ввода. Затем, с помощью метода **getWritableDatabase** подключаемся к БД и получаем объект **SQLiteDatabase**. Он позволит нам работать с БД. Мы будем использовать его методы **insert** – вставка записи, **query** – чтение, **delete** – удаление. У них много разных параметров на вход, но мы пока используем самый минимум.

Далее смотрим, какая кнопка была нажата:

**btnAdd** – добавление записи в таблицу *mytable*. Мы заполняем объект **cv** парами: **имя** поля и **значение**. И (при вставке записи в таблицу) в указанные поля будут вставлены соответствующие значения. Мы заполняем поля *name* и *email*. *id* у нас заполнится автоматически (primary key autoincrement). Вызываем метод **insert** – передаем ему имя таблицы и объект *cv* с вставляемыми значениями. Второй аргумент метода используется, при вставке в таблицу пустой строки. Нам это сейчас не нужно, поэтому передаем null. Метод *insert* возвращает *ID* вставленной строки, мы его сохраняем в *rowID* и выводим в лог.

**btnRead** – чтение всех записей из таблицы *mytable*. Для чтения используется метод **query**. На вход ему подается имя таблицы, список запрашиваемых полей, условия выборки, группировка, сортировка. Т.к. нам нужны все данные во всех полях без сортировок и группировок - мы используем везде null. Только имя таблицы указываем. Метод возвращает нам объект класса **Cursor**. Его можно рассматривать как таблицу с данными. Метод **moveToFirst** – делает первую запись в *Cursor* активной и заодно проверяет, есть ли вообще записи в нем (т.е. выбралось ли что-либо в методе *query*). Далее мы получаем порядковые номера столбцов в Cursor по их именам с помощью метода **getColumnIndex**. Эти номера потом используем для чтения данных в методах **getInt** и **getString** и выводим данные в лог. С помощью метода **moveToNext** мы перебираем все строки в *Cursor* пока не добираемся до последней. Если же записей не было, то выводим в лог соответствующее сообщение – 0 rows. В конце закрываем курсор (освобождаем занимаемые им ресурсы) методом *close*, т.к. далее мы его нигде не используем.

**btnClear** – очистка таблицы. Метод **delete** удаляет записи. На вход передаем имя таблицы и null в качестве условий для удаления, а значит удалится все. Метод возвращает кол-во удаленных записей.

После этого закрываем соединение с БД методом **close**.

 В этой теме важно понять, что для работы с БД мы использовали два класса:

– **DBHelper**, наследующий **SQLiteOpenHelper**. В его **конструкторе**мы вызываем конструктор супер-класса и указываем имя и версию БД. Метод **getWritableDatabase**выполняет подключение к базе данных и возвращает нам объект **SQLiteDatabase**для работы с ней. Метод **close**закрывает подключение к БД. В случае, когда БД отсутствует или устарела, класс предоставляет нам самим реализовать создание или обновление в методах **onCreate**и **onUpgrate**.

– **SQLiteDatabase**. Содержит методы для работы с данными – т.е. **вставка**, **обновление**, **удаление**и **чтение**.

**btnUpd** – обновление записи в *mytable*. Проверяем, что значение *id* не пустое, заполняем cv данными для апдейта и обновляем запись. Для этого используется метод **update**. На вход ему подается имя таблицы, заполненный *ContentValues* с значениями для обновления, строка условия (Where) и массив аргументов для строки условия. В строке условия использован знак ‘?’. При запросе к БД вместо этого знака будет подставлено значение из массива аргументов, в нашем случае это – значение переменной id. Если знаков ‘?’ в строке условия несколько, то им будут сопоставлены значения из массива по порядку. Метод update возвращает нам кол-во обновленных записей, которое мы выводим в лог.

**btnDel** – удаление записи из *mytable*. Проверяем, что *id* не пустое и вызываем метод **delete**. На вход передаем имя таблицы, строку условия и массив аргументов для условия. Метод *delete* возвращает кол-во удаленных строк, которое мы выводим в лог.

В методе **onCreate** этого класса мы используем метод *execSQL* объекта *SQLiteDatabase* для выполнения SQL-запроса, который создает таблицу. Этот метод вызывается, если БД не существует и ее надо создавать. По запросу видно, что мы создаем таблицу mytable с полями id, name и email.

Все сохраним и запустим приложение. Будем работать с БД и смотреть логи, которые покажут, какие методы выполняются, и что в них происходит.

Введем данные в поля ввода и нажмем **Add** (рисунок 1).

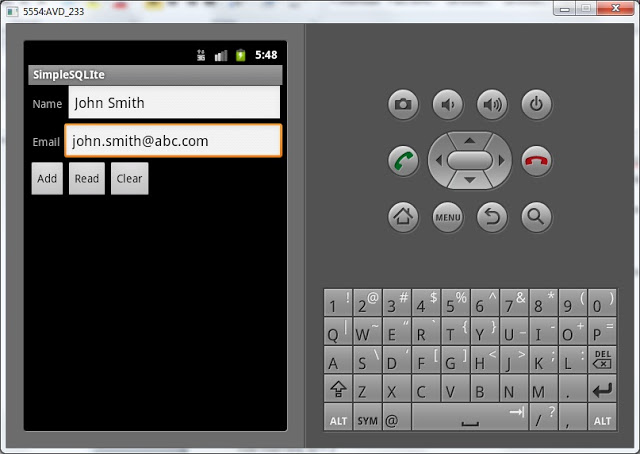


Рисунок 1 – Ввод данных в БД

Смотрим лог:

--- onCreate database ---

--- Insert in mytable: ---

row inserted, ID = 1

Мы видим, что вызывался метод **onCreate**в классе **DBHelper**, а значит выполнялся скрипт по **созданию**таблицы. Это произошло потому, что это **первый**запуск приложения и БД еще **не была создана**. Теперь БД существует и с ней можно работать.

Далее видим, что вызывался метод вставки записи и вернул **ID**= 1.

 Вставим еще какую-нибудь запись (рисунок 2).

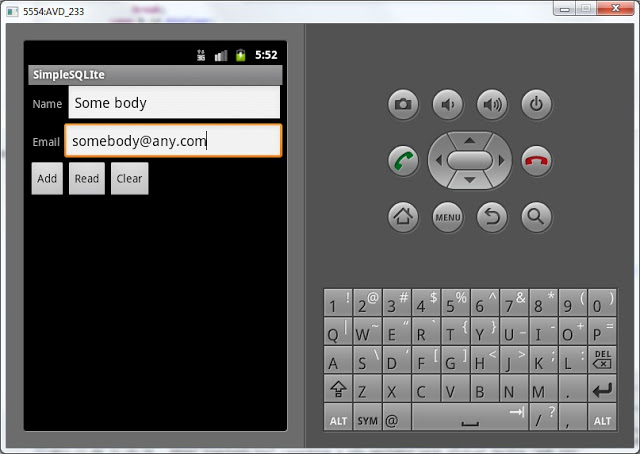


Рисунок 2 – Ввод данных в БД

Смотрим лог:

--- Insert in mytable: ---

row inserted, ID = 2

На этот раз **onCreate не вызывался**, т.к. БД уже **существует**. Вставилась запись с ID = 2.

 Давайте посмотрим содержимое таблицы - нажмем кнопку **Read**и посмотрим лог:

*--- Rows in mytable: ---*

*ID = 1, name = John Smith, email =*[*john.smith@abc.com*](mailto:john.smith@abc.com)

*ID = 2, name = Some body, email =*[*somebody@any.com*](mailto:somebody@any.com)

Мы видим записи, которые вставляли. Тут все верно.

 Теперь очистим таблицу - нажмем **Clear**. Смотрим лог:

*--- Clear mytable: ---*

*deleted rows count = 2*

Удалено две записи, все верно. Если теперь посмотрим содержимое таблицы – кнопка Read, то увидим, что их нет:

*--- Rows in mytable: ---*

*0 rows*

Теперь добавим пару записей и нажмем Read. Смотрим лог:

*ID = 3, name = Ivan Petrov, email = ipetrov @abc.com*

*ID = 4, name = Anton Sidorov, email = asidorov @def.com*

Теперь попробуем обновить запись с *ID*=3. Для этого вводим 3 в поле*ID* и новые данные в поля*Name*и *Email* (рисунок 3).

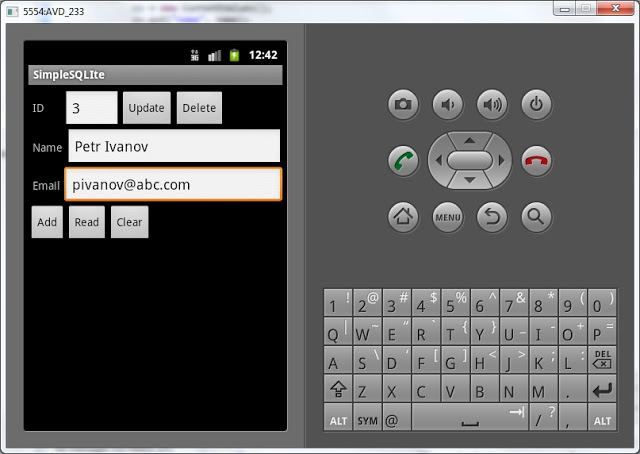


Рисунок 3 – Обновление записи в БД

Жмем **Update**, смотрим лог:

*-- Update mytable: ---*

*updated rows count = 1*

Обновилась одна запись, все верно.

Нажмем **Read**и убедимся в этом. Лог:

*ID = 3, name = Petr Ivanov, email = pivanov @abc.com*

*ID = 4, name = Anton Sidorov, email = asidorov @def.com*

Запись действительно обновилась.

Теперь давайте удалим запись с **ID**= 4. Вводим 4 в поле **ID** (рисунок 4).

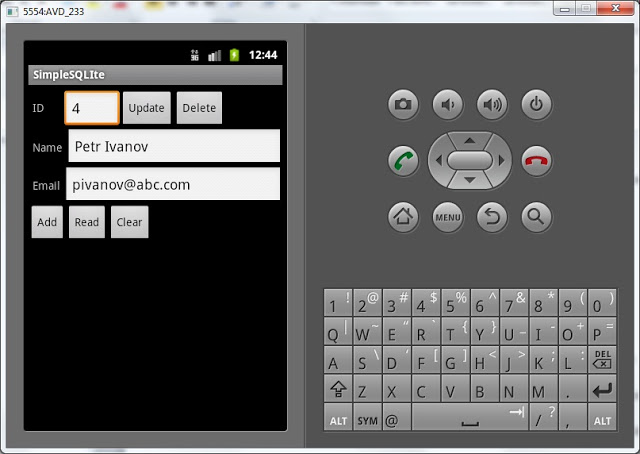


Рисунок 4 – Удаление записи из БД

Жмем **Delete**, смотрим лог:

*--- Delete from mytable: ---*

*deleted rows count = 1*

Одна запись удалена.

Жмем **Read**, смотрим лог:

*--- Rows in mytable: ---*

*ID = 3, name = Petr Ivanov, email = pivanov @abc.com*

Осталась одна запись.

**3. Постановка задачи**

3.1 Ознакомиться с теоретическими сведениями.

3.2 Разработать приложение, взаимодействующее с базой данных в соответствии с номером варианта (таблица 3.1). Основное активити должно содержать 4 кнопки:

1-я кнопка – для вывода информации из таблицы в удобном для восприятия формате;

2-я кнопка – для ввода новой записи в таблицу;

3-я кнопка – для обновления записи по одному из полей;

4-я кнопка – для очистки таблицы.

При запуске приложения необходимо:

1. Создать БД, если ее не существует.

2. Создать таблицу, содержащую соответствующие поля.

3. Реализовать возможность добавления и удаления записей.

4. Реализовать возможность очистки таблицы.

3.3 Создать новое отдельное приложение на основе приложения, созданного в пункте 3.2. Переопределить функцию onUpgrade. При изменении версии БД необходимо удалить таблицу, создать новую таблицу, содержащую поля с учетом разделения полей. Изменить версию базы данных.

3.4 Продемонстрировать результаты работы.

3.5 Сделать выводы по работе.

Таблица 3.1 – Варианты заданий

|  |  |
| --- | --- |
| № варианта | Схема БД |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 |  |
| 9 |  |
| 10 |  |
| 11 |  |
| 12 |  |

**4. Содержание отчета**

Отчет должен содержать:

– название и цель работы;

– листинги xml- и java-кода;

– скриншоты работы приложения;

– выводы по работе.

**5. Контрольные вопросы**

5.1 Особенности библиотеки SQLite.

5.2 Преимущества и недостатки SQLite.

5.3 Как создавать новые базы данных SQLite?

5.4 Как взаимодействовать с базами данных для вставки, обновления и удаления строк?

5.5 Какие ситуации нужно рассматривать при создании базы данных?

5.6 Какой класс нужно использовать для обработки описанных выше ситуаций и для чего он нужен?